Algoritma & Struktur Data

Dr. Anto Satriyo Nugroho, M.Eng
Email: asnugroho@gmail.com HP: 0888-673-1891
URL: http://asnugroho.net/lecture/ds.html

Name Anto Satriyo Nugroho

Birthday October 1970

Education received B.Eng(S1), M.Eng (S2) and Dr.Eng (S3) from Nagoya Institute of Technology, Japan (Electrical & Computer Eng.) in 1995, 2000 dan 2003.

Scholarship from Pemerintah RI (S1) & Japanese government Monbukagakusho (S2 & S3)

Core competence

- Pattern Recognition, Artificial Intelligence & Datamining
- Biomedical Engineering & Bioinformatics/Computational Biology
- Research activities & publications can be seen from http://asnugroho.net/

Work experiences

- 1990-now BPP Teknologi (Pusat Teknologi Informasi & Komunikasi)
- 2003-2004 Visiting Professor at School of Computer & Cognitive Sciences, Chukyo University, Japan
- 2004-2007 Visiting Professor at School of Life System Science & Technology, Chukyo University, Japan
- 2003-2007 Researcher at Institute for Advanced Studies of Artificial Intelligence, Chukyo University Japan
- 2007-now Vice president of Indonesian Society for Softcomputing
- 2007-now Lecturer at Swiss German University, Al Azhar Univ. Indonesia

Referensi

1. Donald E. Knuth, The Art of Computer Programming: Vol.1-3 Fundamental Algorithms, 3rd Ed., Addison Wesley, 1997
2. Thomas Cormen, Charles Leiserson, Introduction to Algorithms (2nd ed.), MIT Press, 2001
3. Hiroyuki Kondoh, Teihon C puroguramaa no tameno arugorizumu to deta kouzou, Softbank Publishing, 1998

   Berbahasa Jepang, buku teks struktur data dan algoritma yang sangat terkenal di Jepang. Saat masih mengajar di Chukyo Univ., saya memakai buku ini sebagai buku teks. Slide saya buat berdasarkan penjelasan yang diberikan pada buku ini
4. Bohyoh Shibata, Meikai C Gengo Nyuumon-hen, Softbank publishing, 2004

   Dr.Shibata menerima penghargaan untuk buku teks C++ yang ditulisnya. Sebagaimana halnya dengan buku teks C++, buku pengantar C di atas ditulis dengan bahasa yang mudah difahami dengan contoh-contoh yang memudahkan seseorang memahami bahasa C. Beberapa contoh yang saya tampilkan pada kuliah ini merujuk pada buku ini.
5. Thompson Susabda Ngoen, Pengantar Algoritma dengan Bahasa C, Penerbit Salemba Teknika, 2006

Cara Belajar

1. Datang tepat waktu. Toleransi keterlambatan adalah 15 menit. Siswa yang datang terlambat lebih dari 15 menit tetap berhak mengikuti kuliah, tetapi dihitung sebagai tidak hadir.
2. Apabila ada kendala yang menyebabkan keterlambatan tidak dapat dihindari, informasikan ke dosen lewat SMS (nama & alasan). Apabila alasan dapat diterima, walau terlambat sekalipun absensi tetap dihitung hadir.
3. Membuat catatan: misalnya dengan mem-print slide dan membuat catatan pada slide
4. Ketinggalan sekali saja, akan berakibat fatal. Cukup berat memahami kuliah berikutnya.
5. Kalau tidak faham segera bertanya
6. Kalau terlalu cepat, jangan segan minta agar diperlambat
7. Kalau terlalu lambat, jangan segan minta agar dipercepat
8. Mengulang materi yang diajarkan di rumah
9. Melatih SENDIRI kemampuan programming dari materi yang diajarkan

UJIAN LISAN

10. Di akhir tiap kuliah akan diberikan beberapa soal untuk latihan/PR
11. Minggu berikutnya diadakan UJIAN LISAN berdasarkan soal yang diberikan sebagai PR
12. Nilai ujian lisan akan berpengaruh pada penentuan nilai akhir
13. Saya ingin anda semua memperoleh 100 untuk ujian LISAN. Karena itu silakan perbaiki nilai ujian lisan anda. Perbaikan nilai ujian lisan dapat dilakukan di sebarang waktu (konsultasikan waktu anda dengan saya via email asnugroho@gmail.com)
14. Tidak ada Ujian Ulang ! Struktur Data dan Algoritma bukan matakuliah yang dapat difahami dalam semalam !

Apakah Algoritma & Struktur Data itu ?

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Ketentuan & Asumsi

- Menguasai pemakaian bahasa C, terutama
  - pointer
  - structure

- Materi kuliah dapat didownload dari
  http://asnugroho.net/lecture/ds.html

Agenda

1	Pendahuluan: apakah algoritma & struktur data itu ?
	Array: satu dimensi & multidimensi
2	Array & Pointer & Structure
	Latihan array, pointer & structure
3	Computational Complexity
	Latihan array, pointer & structure
4	Linear list, stack & queue (1)
	Latihan
5	Linear list, stack & queue (2)
	Latihan
6	Doubly-linked list & tree structure (1)
	Latihan

Agenda

7	Tree structure (2)
	Latihan
8	Hash
	Latihan
9	Sequential & Binary Search
	Latihan
10	Selection sort & Insertion sort
	Latihan pemrograman selection sort & insertion sort
11	Quick sort & Merge sort
	Latihan pemrograman quick & merge sort
12	Review
	Review

Tahapan dalam Pemrograman

1. Analisa masalah
2. Memilih algoritma dan struktur data untuk menyelesaikan masalah itu
3. Coding

Programming tidak hanya sekedar mengetik di keyboard

Definisi

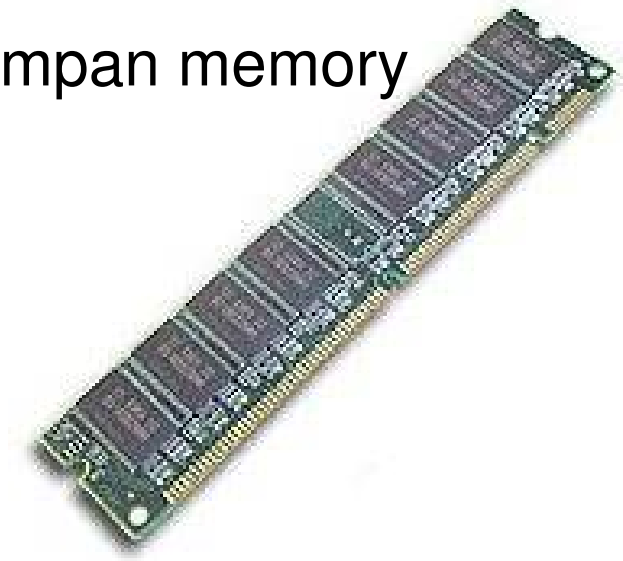
Algoritma:

prosedur terbatas yang terdiri beberapa operasi menyelesaikan suatu masalah (Ibaraki)

program : algoritma yang diimplementasikan dalam bahasa pemrograman tertentu

Struktur data

cara pengaturan data agar bisa disimpan memory komputer secara efisien

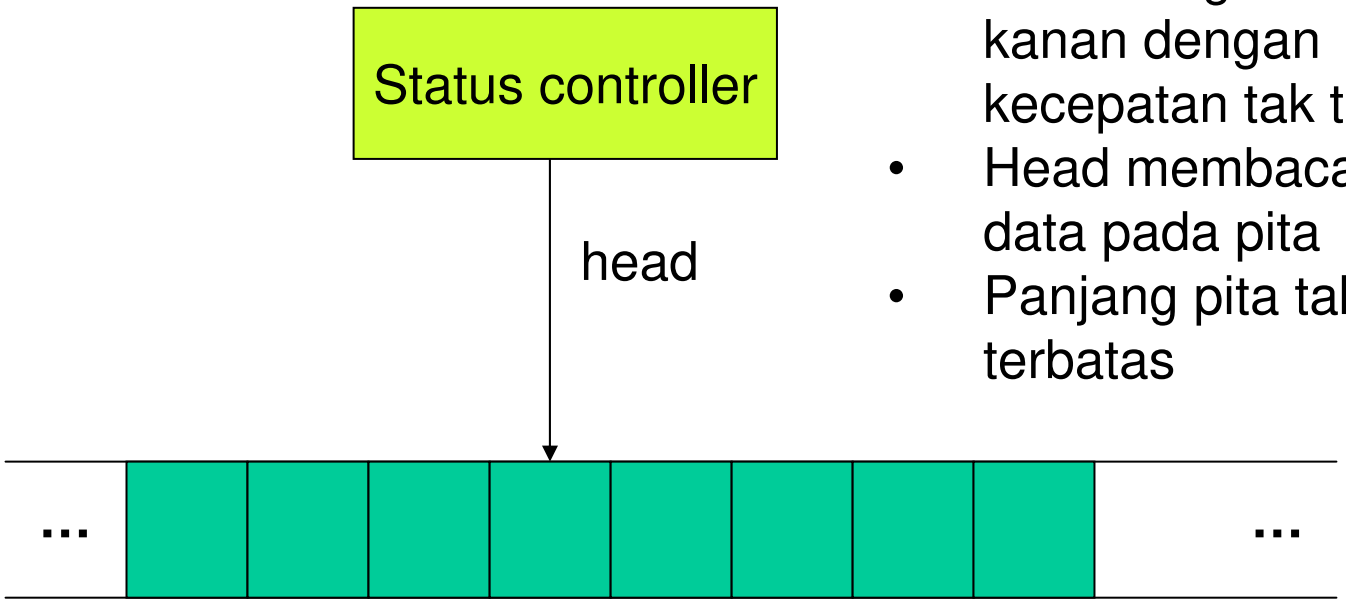


Hubungan antara Algoritma & Struktur Data

Wirth: algoritma + struktur data = program

- Algoritma →penyelesaian satu masalah secara matematis
- Turing machine
  - Model matematika komputer
  - Memiliki tape dengan panjang tak terhingga sebagai alat penyimpan ingatan (memory)
  - Running time, kecepatan tak terhingga
- Padahal komputer yang ada sangat terbatas baik memory maupun kecepatannya

Turing Machine

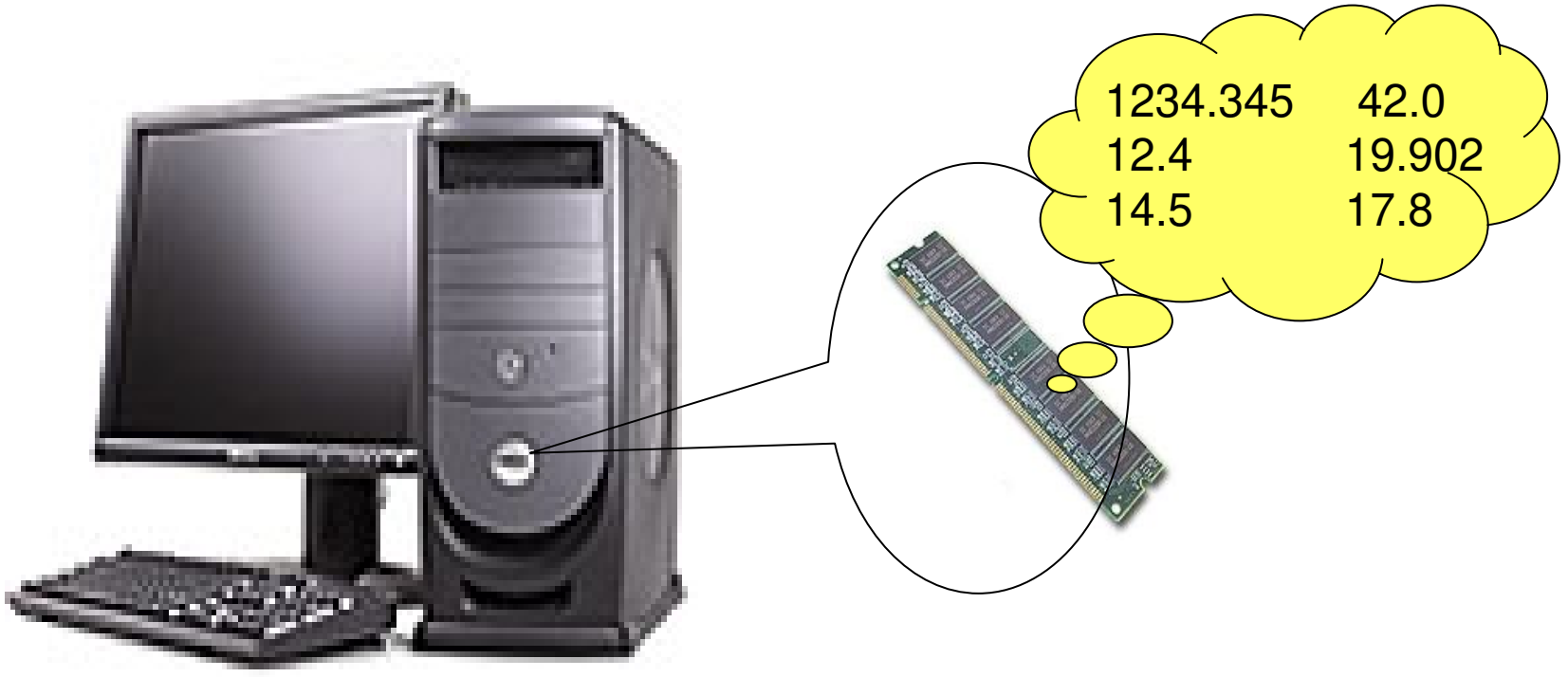


- Head bergeser ke kiri-kanan dengan kecepatan tak terbatas
- Head membaca/menulis data pada pita
- Panjang pita tak terbatas

Pita 1 dimensi dengan panjang tak terhingga

Hubungan antara Algoritma & Struktur Data

Di dunia nyata, kemampuan komputer maupun memory sangat terbatas



Hubungan antara Algoritma & Struktur Data

- Performa algoritma yang ideal
  - Memory yang diperlukan kecil, running time singkat
- Tradeoff antara waktu dan ruang (memory)

Memory

Running time

	Besar	Kecil
Lama		●
Singkat	●	●

Hubungan antara Algoritma & Struktur Data

- Bagaimana data harus direpresentasikan saat membuat program ?

Struktur data

List, stack, queue, tree, dll

- Algoritma yang baik + Struktur data yang tepat = program yang baik
  - Pemilihan algoritma dan struktur data yang tepat harus mempertimbangkan skala data, CPU, memori, dsb.
  - Perlu pengetahuan algoritma dan struktur apa saja yang ada dan mungkin dipakai

Contoh Algoritma: BUBBLE SORT

banyaknya data: n

Data diurutkan/disorting dari yang bernilai besar

Proses

step 1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	10	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	10	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	10	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	8	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	8	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4
Step-4	10	8	7	5	4

Perhatikan !

- Apakah yang anda peroleh setelah satu iterasi dalam bubble sort selesai ?
  - Setelah iterasi pertama selesai, elemen dengan nilai terbesar akan muncul di ujung paling kiri
  - Setelah iterasi kedua selesai, elemen dengan nilai terbesar kedua akan muncul pada urutan kedua dari kiri
  - dst.

QUIZ

1. Urutkan deret angka berikut dengan bubble sort

13 14 10 4 18 20 25 17

2. Tuliskan hasil tiap langkah (step).

Bubble Sort

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Contoh Algoritma: BUBBLE SORT

banyaknya data: n

Data diurutkan/disorting dari yang bernilai besar

Proses

step 1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	10	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	10	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	10	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	8	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	8	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4
Step-4	10	8	7	5	4

Perhatikan !

- Apakah yang anda peroleh setelah satu iterasi dalam bubble sort selesai ?
  - Setelah iterasi pertama selesai, elemen dengan nilai terbesar akan muncul di ujung paling kiri
  - Setelah iterasi kedua selesai, elemen dengan nilai terbesar kedua akan muncul pada urutan kedua dari kiri
  - dst.

C Program untuk Bubble Sort

```
void bubble_sort(int x[ ], int datanum)
{
    int i,j;
    int tmp;

    for(i=0;i<datanum-1;i++)
            for(j=datanum-1;j>i;j--) {
                    if( x[j-1] < x[j]) {
                            tmp=x[j];
                            x[j]=x[j-1];
                            x[j-
    1]=tmp;
                    }
                        }
}
```

```
#include	<stdio.h>

bubble_sort(int x[],int datanum)
{
	int i,j;
	int tmp;

	for(i=0;i<datanum-1;i++)
		for(j=datanum-1;j>i;j--)
			if( x[j-1] < x[j]) {
				tmp=x[j];
				x[j]=x[j-1];
				x[j-1]=tmp;
		}

}
int main(void)
{
	int i;
	int data[10]= {10,3,2,1,4,7,6,9,5,8};
	bubble_sort(data,10);
	for(i=0;i<10;i++){
		printf("%d %d\n",i,data[i]);
	}
}
```

Computational Complexity

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Agenda

1	Pendahuluan: apakah algoritma & struktur data itu ?
	Array & Pointer & Structure (1): satu dimensi & multidimensi
2	Array & Pointer & Structure (2)
	Array & Pointer & Structure (3)
3	Computational Complexity
	Array & Pointer & Structure (4)
4	Linear list, stack & queue (1)
	Latihan
5	Linear list, stack & queue (2)
	Latihan
6	Doubly-linked list & tree structure (1)
	Latihan

Agenda

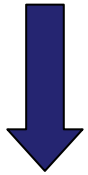
7	Tree structure (2)
	Latihan
8	Hash
	Latihan
9	Sequential & Binary Search
	Latihan
10	Selection sort & Insertion sort
	Latihan pemrograman selection sort & insertion sort
11	Quick sort & Merge sort
	Latihan pemrograman quick & merge sort
12	Review
	Review

Apakah program yang baik itu ?

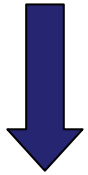
Bagaimanakah cara membuat program yang baik ?

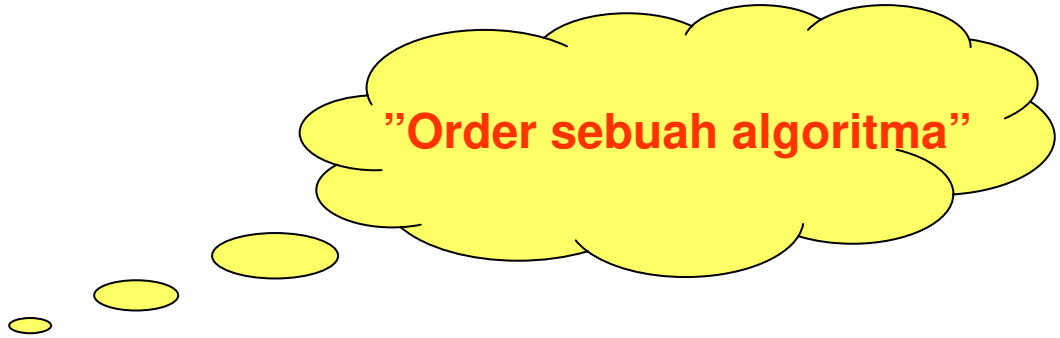
Algoritma yang tepat/sesuai dengan masalah

+pemilihan struktur data yang tepat



Bagaimana mengukur baik-tidaknya sebuah algoritma ?





Computational complexity sebuah algoritma

Bagaimana mengukur kualitas sebuah algoritma ?

Sortinglah deretan angka di bawah ini :

1 5 3 2 100 12 25 17 23 13 …

Algoritma sorting sangat bervariasi.
Bagaimana memilih algoritma yang terbaik ?

Apakah WAKTU merupakan parameter tepat untuk mengukur kualitas sebuah algoritma ?

Cepat tidaknya jalannya sebuah program sangat bergantung kepada arsitektur komputer

Komputer tua/lambat

1 jam

Komputer baru/cepat

1 detik

Waktu bukan parameter yang baik,
karena sangat tergantung arsitektur komputer

Apakah WAKTU merupakan parameter tepat untuk mengukur kualitas sebuah algoritma ?

Waktu yang diperlukan untuk menjalankan sebuah program juga sangat tergantung kualitas code (kemampuan programmer), walaupun dijalankan pada komputer yang sama

Program yang ditulis A

Program yang ditulis B

10 s

30 s

Standar evaluasi Algoritma

- Pada prinsipnya diukur berdasarkan waktu eksekusi/menjalankan program tsb.
- Masalah:
  1. Sangat tergantung pada komputer yang dipakai untuk menjalankan program
  2. Sangat tergantung pada kualitas programmer dalam menuliskan program

Evaluasi memakai computational Complexity (Big Oh notation)

Complexity

- Definisi :

  Waktu yang diperlukan untuk menjalankan sebuah algoritma pada sebuah komputer virtual, yang direpresentasikan secara “kasar” sebagai fungsi dari n (n adalah banyaknya input data)

- Karakteristik: tidak dipengaruhi oleh hardware komputer maupun kemampuan programming

- O : representasi worst-case complexity

  $O(n^2)$ $\longrightarrow$ Complexity algoritma berbanding lurus terhadap n^2 (n adalah banyaknya input data)

Worst case vs Average complexity

- Worst case complexity: mengasumsikan input data dalam kondisi terburuk
- Average complexity : sulit untuk dihitung
- Umumnya algoritma dievaluasi berdasarkan worst case complexity

- Contoh perkecualian

  Quicksort

  Worst case complexity : $O(n^2)$

  Average complexity : $O(n \log n)$

Karakteristik worst case complexity O

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

Dua operasi yang berturutan dengan complexity masing-masing dalam fungsi f dan g, maka complexity total adalah yang terbesar antara f dan g

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

Bila sebuah operasi dengan complexity O(f(n)) dijalankan pada O(g(n)) kali, maka complexity total operasi tersebut adalah f x g

Catatan Penting

- Big Oh notation merepresentasikan evaluasi algoritma terhadap input data sebanyak n
- Nilai yang berada dalam tanda kurung () adalah 1, log n, n, n^2, n pangkat bilangan bulat yang lain
- Notasi ini mengabaikan koefisien berupa konstanta seperti 2, 100, 0.1, atau berupa variabel a

  Contoh: O(2n) ditulis sebagai O(n), karena 2 diabaikan

Contoh

$$
\begin{aligned}
O(3n^2) + O(2n) &= O(n^2) + O(n) \\
&= O(\max(n^2, \quad n)) \\
&= O(n^2)
\end{aligned}
$$

$$
\begin{aligned}
O(n^2) + O(n) &= O(\max(n^2, n)) \\
&= O(n^2)
\end{aligned}
$$

$$
\begin{aligned}
O(n^2)O(n) &= O(n^2.n) \\
&= O(n^3)
\end{aligned}
$$

Relasi nilai n, log (n), n^2 dst

$$1 \quad \log n \quad n \quad n\log n \quad n^2 \quad n^3 \quad \cdots \quad n^k \quad 2^n$$

kecil ⟶ besar

n	log n	n log n	n*n	n*n*n	2^n
1	0.00	0.00	1	1	2
2	1.00	2.00	4	8	4
3	1.58	4.75	9	27	8
4	2.00	8.00	16	64	16
5	2.32	11.61	25	125	32
6	2.58	15.51	36	216	64
7	2.81	19.65	49	343	128
8	3.00	24.00	64	512	256
9	3.17	28.53	81	729	512
10	3.32	33.22	100	1000	1024
11	3.46	38.05	121	1331	2048

Hafalkan urutan di atas !

Catatan dalam memilih algoritma

1. Compexity saja tidak cukup

   Contoh: Hash memiliki complexity yang kecil. Tetapi memiliki beberapa kelemahan seperti hilangnya informasi urutan registrasi data, masalah pada memory dsb.

2. Untuk data berskala kecil, tidak selamanya algoritma yang ordernya kecil selalu lebih baik daripada algoritma yang berorder besar (algoritma yang berorder kecil, bisa jadi memiliki koefisien konstanta bernilai besar)

3. Algoritma yang berorder kecil kadangkala memerlukan waktu yang lebih lama dalam proses coding (penulisan program)

   program yang sering dipakai: pilihlah algoritma yang berorder complexitykecil

   program yang hanya sekali pakai (jarang): pilihlah yang paling mudah dalam proses coding

4. Trade of antara waktu dan memory

Complexity Bubble Sort

Hitunglah complexity bubble sort !

```
1.  void bubble_sort(int x[ ], int datanum)
2.  {
3.          int i,j;
4.          int tmp;
5.          for(i=0;i<datanum-1;i++)
6.                  for(j=datanum-1;j>i;j--)
7.                          if( x[j-1] < x[j]) {
8.                                  tmp=x[j];
9.                                  x[j]=x[j-1];
10.                                 x[j-1]=tmp;
11.         }
12.}
```

Complexity Bubble Sort

1. Baris ke-5: area kuning
   array scanning dari kiri ke kanan diulang sebanyak (n-1) kali
   Complexity: O(n)
2. Baris ke-6: area hijau
   array scanning dari kanan ke kiri, tiap 2 buah data dibandingkan dan jika urutan besar-kecilnya tidak benar, kedua data di swap.
   Rata-rata: n/2
   Complexity: O(n)
3. Baris ke 7-11: area abu-abu
   tidak ada loop
   complexity: O(1)

Total complexity: $O(n) \times O(n) \times O(1) = O(n^2)$

Latihan

Sederhanakan notasi berikut

$$O(3n^2 + \log n) + O(n \log n + 3n) =$$

$$O(n \log n + 4n)O(4n) =$$

$$O(2^n) + O(10n \log n + 2n^2 + \log n) + O(30 \log n + 3n) =$$

List, Stack & Queue

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Beberapa Jenis Struktur Data

1. Array
   1. Linear List
   2. Stack
   3. Queue

   1. Apa ?
   2. Bagaimana cara implementasinya ?

2. List
   1. Connected List
   2. Circular List
   3. Doubly-linked List
   4. Multi list structure

3. Tree Structure

Linear List

Apakah Linear List itu ?

- Sekumpulan elemen yang diatur secara terurut

$$x[1], x[2], \cdots, x[k-1], x[k], x[k+1], \cdots, x[n]$$

- Linear List tidak sama dengan Connected-List

Operasi pada Linear List

No.	Operasi
1	Menambahkan sebuah elemen sebelum elemen ke-k
2	Menghapus elemen ke-k
3	Membaca/menulis isi elemen ke-k
4	Mencari elemen dengan key tertentu
5	Menggabungkan beberapa list menjadi satu
6	Memecah sebuah list ke beberapa buah
7	Mengcopy sebuah list
8	Menghitung banyaknya elemen dalam sebuah list

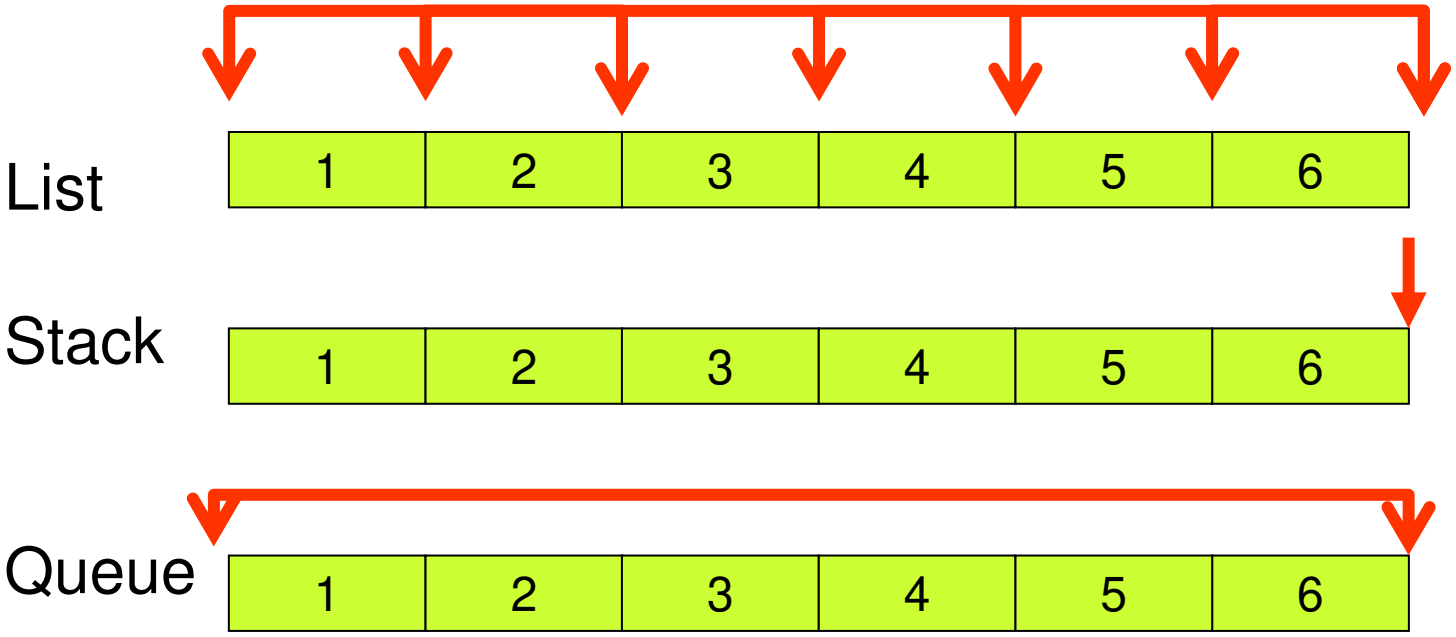
List, Stack & Queue

- Tidak semua operasi list diperlukan pada setiap program
  - Penentuan struktur data didasarkan pada operasi yang diperlukan saja agar bisa berjalan dengan efisien
- Pada sebuah Linear List, penyisipan dan penghapusan elemen dapat dijalankan di sebarang posisi
- Bentuk khusus linear list:
  Penambahan elemen dan penghapusannya dilakukan di posisi terdepan atau posisi terbelakang saja
  - Stack
  - Queue

Stack dan Queue juga merupakan salah satu jenis list

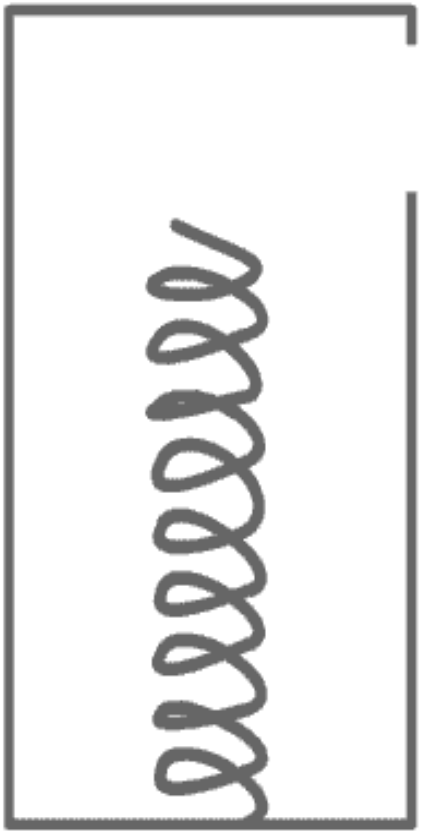
List, Stack & Queue

- Pada sebuah Linear List, penyisipan dan penghapusan elemen dapat dijalankan di sebarang posisi
- Penambahan dan penghapusan elemen pada stack/queue dilakukan di posisi terdepan atau posisi terbelakang saja



Stack

Apakah Stack itu ?



Apakah Stack itu ?

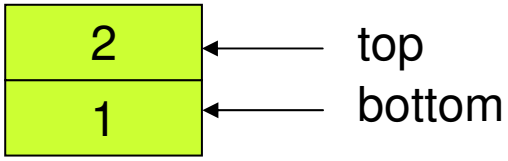
- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack

PUSH

Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack

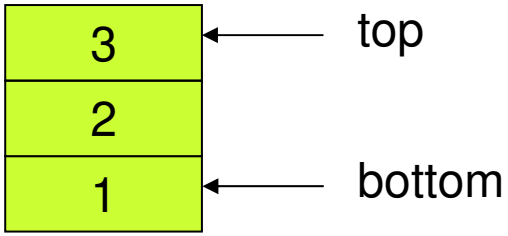
PUSH



Apakah Stack itu ?

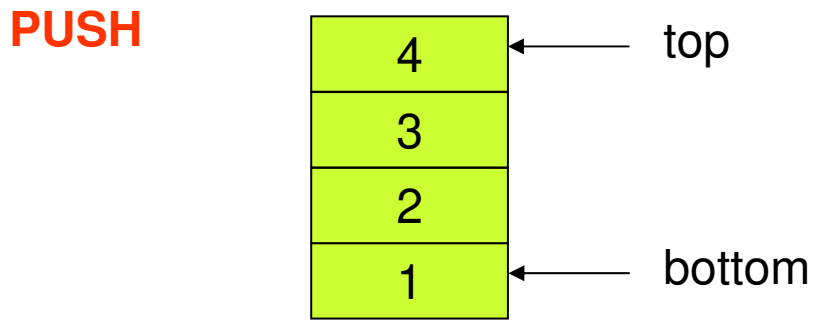
- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack

PUSH



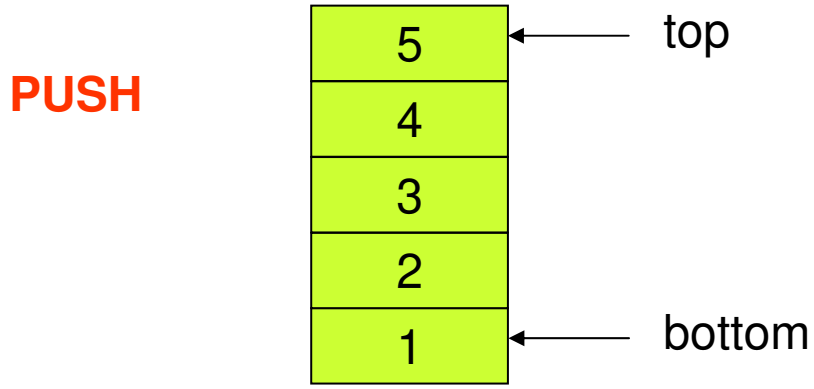
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack



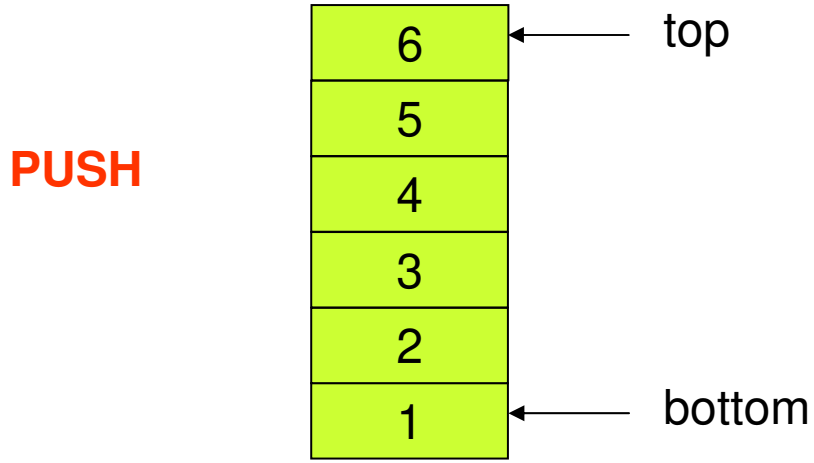
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack



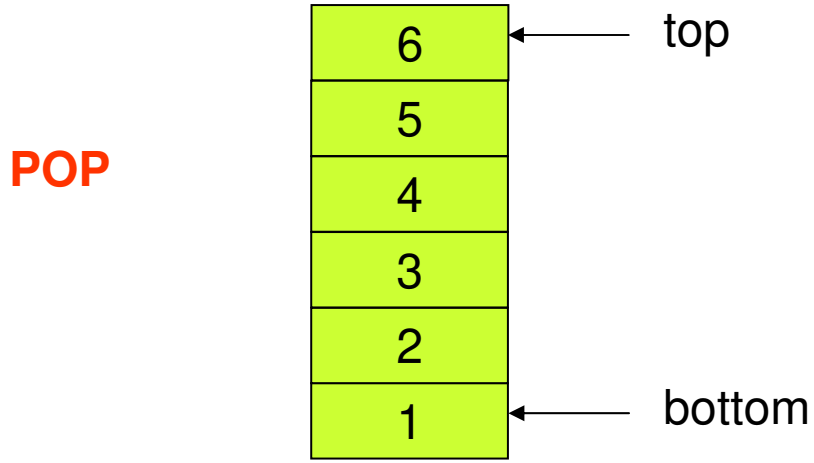
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **PUSH** : Menambahkan elemen pada sebuah stack



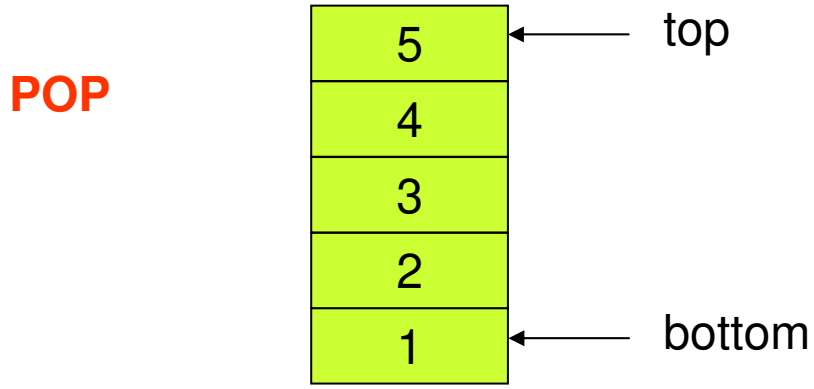
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack



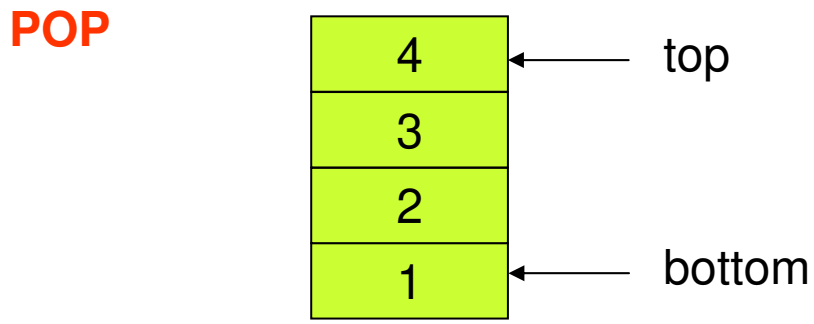
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack



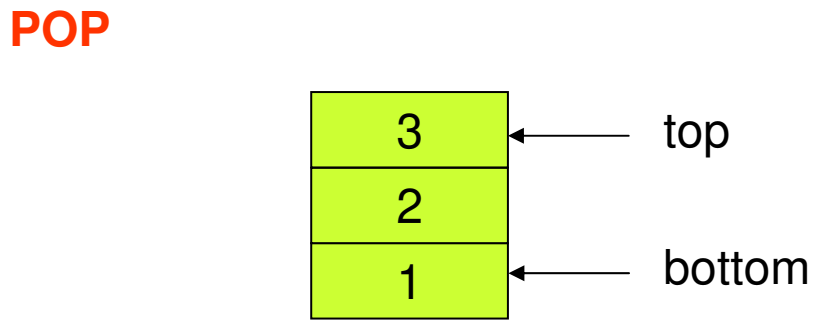
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack



Apakah Stack itu ?

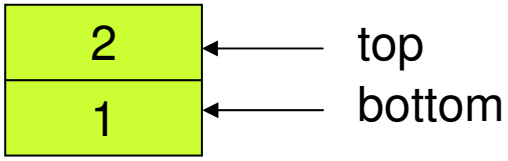
- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack



Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack

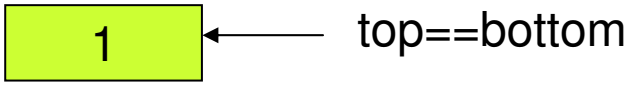
POP



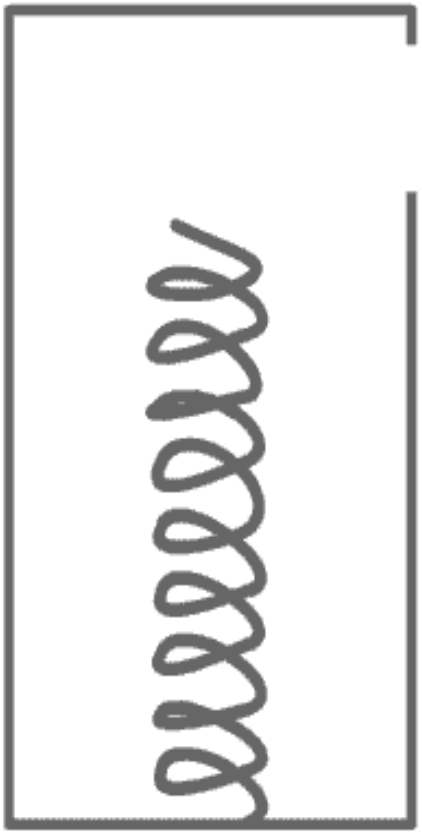
Apakah Stack itu ?

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan
- Yang dihapus adalah elemen yang paling terakhir ditambahkan
- Nama lain : LIFO (Last In First Out)
- Operasi **POP** : Menghapus sebuah elemen dari sebuah stack

POP



Apakah Stack itu ?



PUSH dan POP

Stack Overflow & Stack Underflow

- Stack Overflow

  Menambahkan data pada sebuah stack yang telah penuh
- Stack Underflow

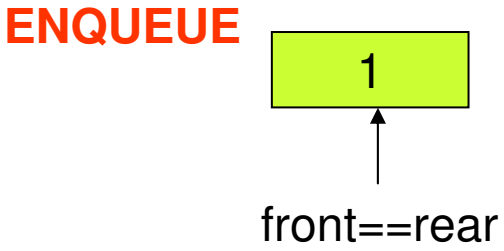
  Menghapus data dari sebuah stack yang sudah kosong

Queue

Apakah Queue itu ?

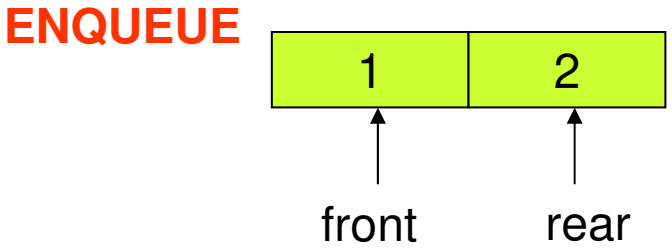
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



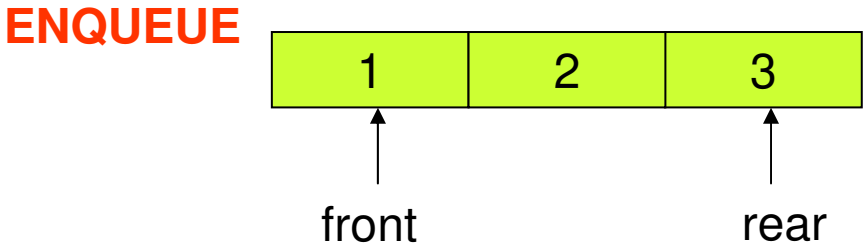
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



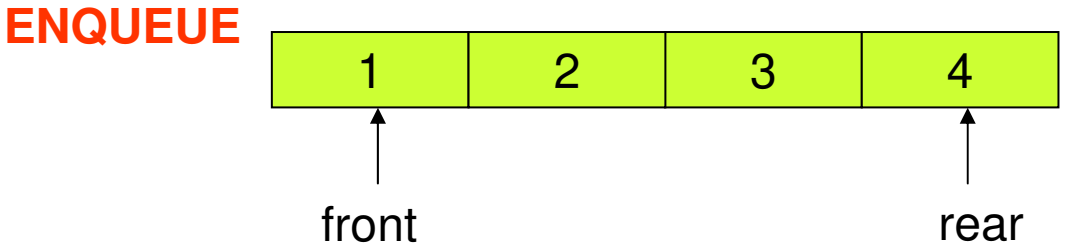
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



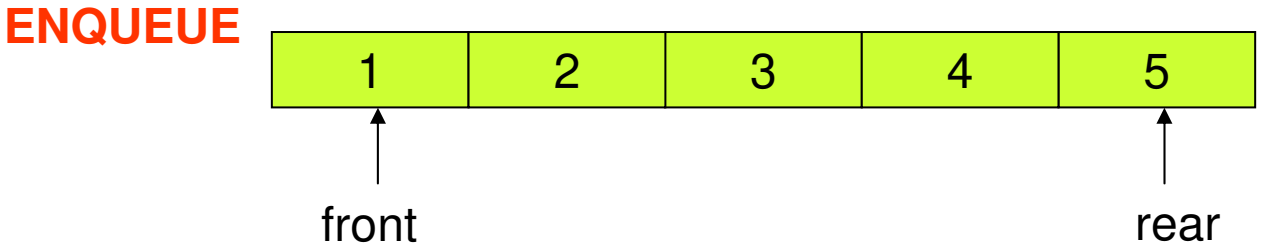
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



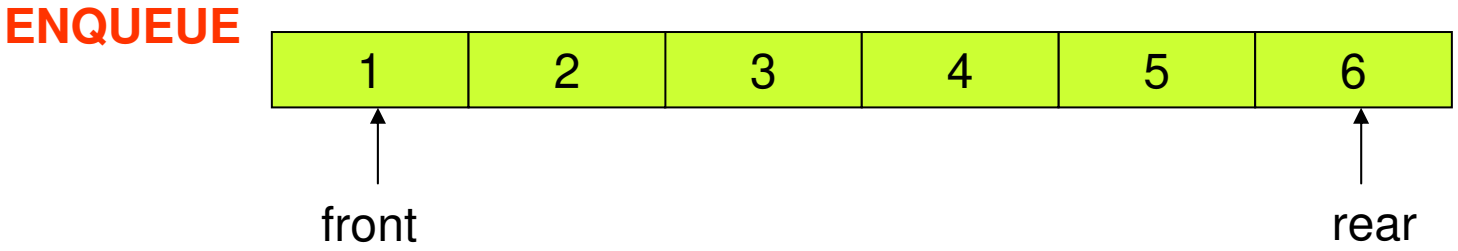
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



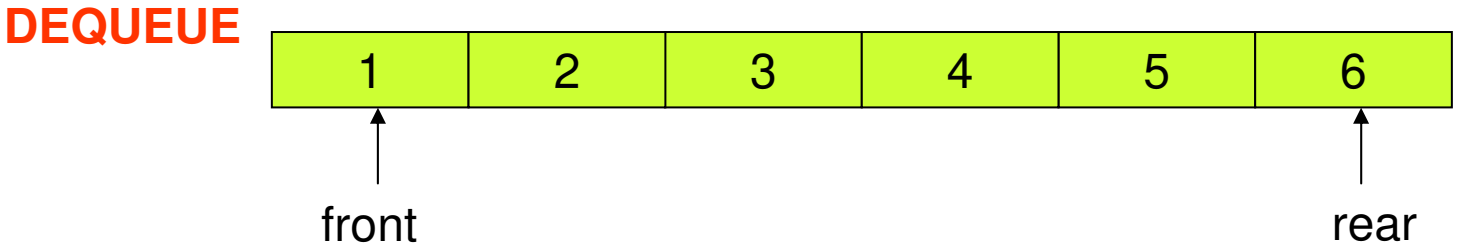
Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **ENQUEUE**: menambahkan data pada sebuah list



Apakah Queue itu ?

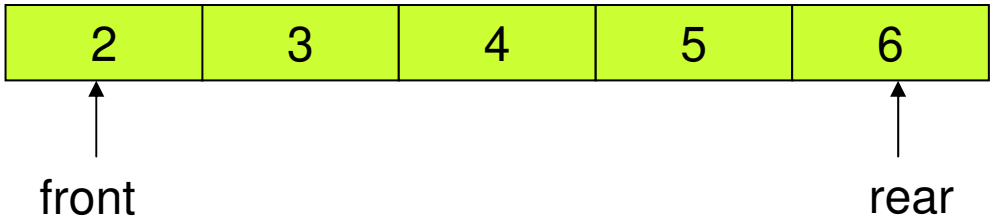
- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list



Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list

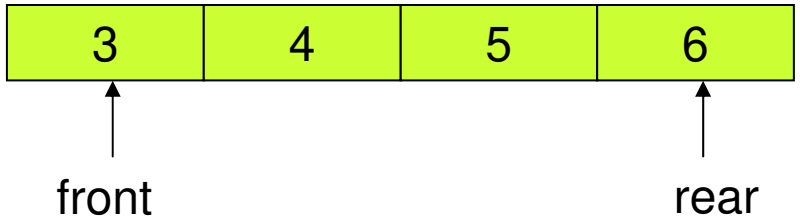
DEQUEUE



Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list

DEQUEUE



Apakah Queue itu ?

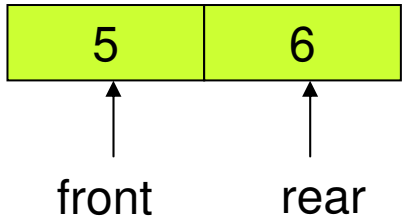
- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list

DEQUEUE

Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list

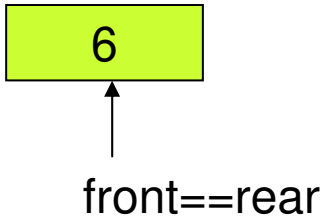
DEQUEUE



Apakah Queue itu ?

- Penambahan data dilakukan pada sebuah ujung sebuah list, sedangkan penghapusan data dilakukan pada ujung yang lain
- Data yang dihapus adalah data yang paling awal ditambahkan
- Nama lain : FIFO (First In First Out)
- Operasi **DEQUEUE**: menghapus data pada sebuah list

DEQUEUE



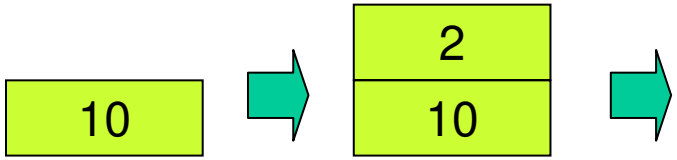
Animasi Queue

ENQUEUE dan DEQUEUE

Latihan 1

- Gambarkan kondisi stack setelah dilakukan operasi berikut:

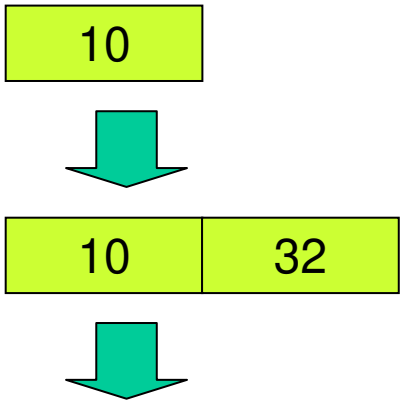
```
push(10);
push(2);
pop();
push(20);
pop();
push(15);
push(5);
```



Latihan 2

- Gambarkan kondisi queue setelah dilakukan operasi berikut:

```
enqueue(10);
enqueue(32);
enqueue(5);
dequeue();
enqueue(10);
dequeue();
dequeue();
```



Beberapa jenis struktur data

1. Array
   1. Linear List
   2. Stack
   3. Queue

   1. Apa ?
   2. Bagaimana cara implementasinya ?

2. List
   1. Connected List
   2. Circular List
   3. Doubly-linked List
   4. Multi list structure

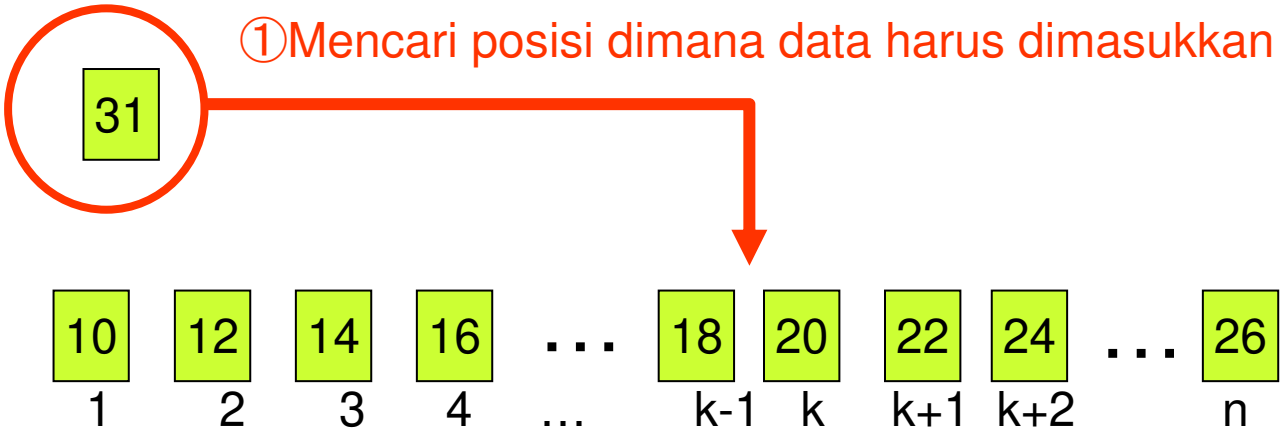
3. Tree Structure

Implementasi Linear List

- List size (banyaknya elemen) : n

$$x[1], x[2], \cdots, x[k-1], x[k], x[k+1], \cdots, x[n]$$

- Memasukkan (menyisipkan) data baru sebelum data no.k

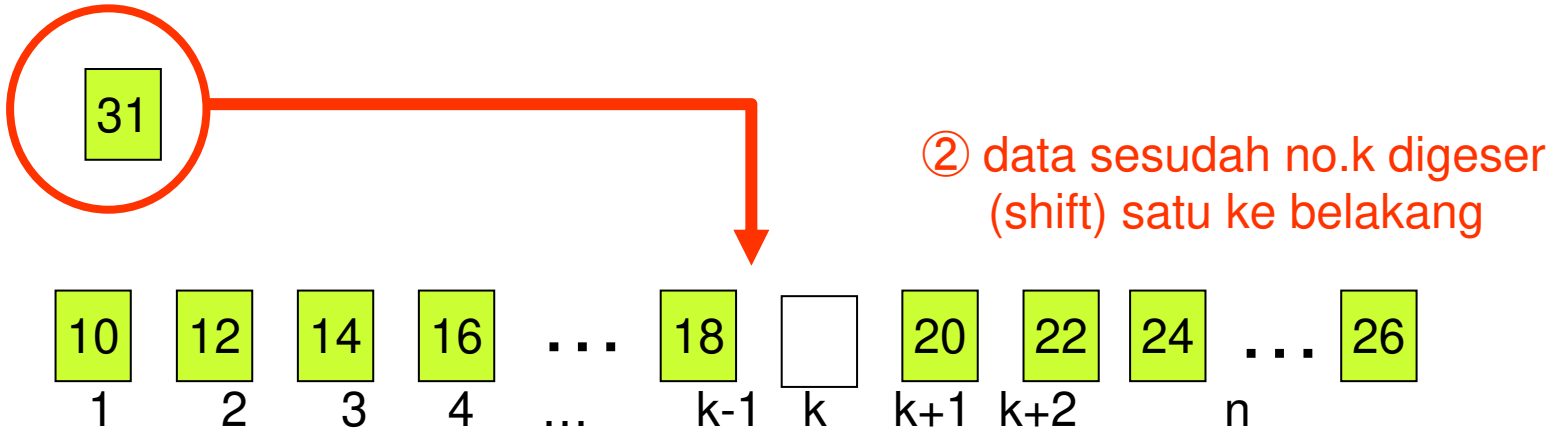


Implementasi Linear List

- List size (banyaknya elemen) : n

$$x[1], x[2], \cdots, x[k-1], x[k], x[k+1], \cdots, x[n]$$

- Memasukkan (menyisipkan) data baru sebelum data no.k



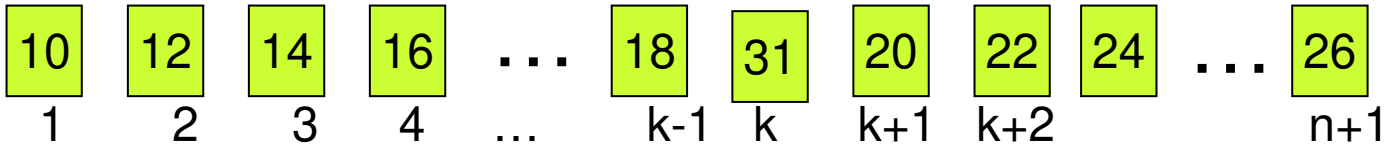
Implementasi Linear List

- List size (banyaknya elemen) : n

$$x[1], x[2], \cdots, x[k-1], x[k], x[k+1], \cdots, x[n]$$

- Memasukkan (menyisipkan) data baru sebelum data no.k
- Rata-rata shift data: n/2 ⟶ Computational complexity:

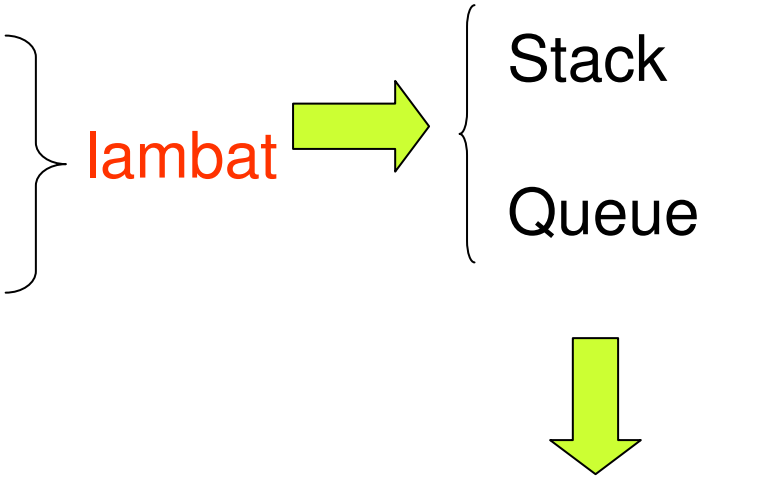
$$O(n)$$



Implementasi Linear List

- Membaca/menulis konten sebuah data: O(1) random access

- Penambahan data : O(n)
- Penghapusan data: O(n)



efisien dalam komputasi

Beberapa jenis struktur data

1. Array
   1. Linear List
   2. Stack
   3. Queue

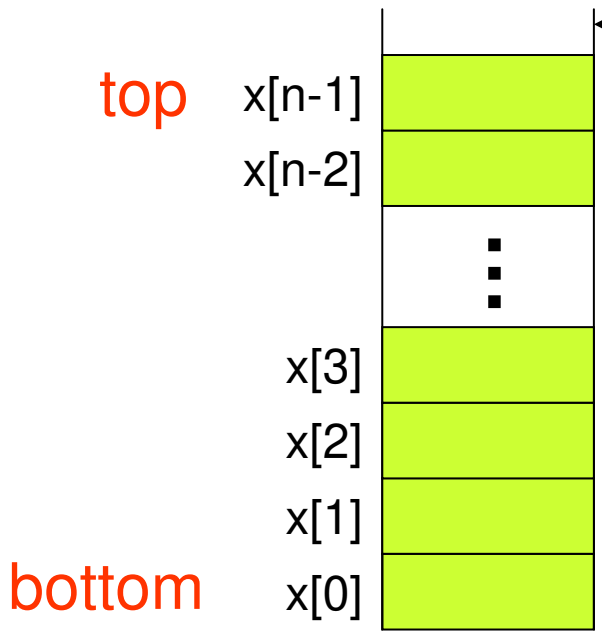
   1. Apa ?
   2. Bagaimana cara implementasinya ?

2. List
   1. Connected List
   2. Circular List
   3. Doubly-linked List
   4. Multi list structure

3. Tree Structure

Implementasi Stack memakai Array

- Penambahan dan penghapusan elemen dilakukan pada elemen list yang terletak di paling depan



n (stack pointer)
Menunjuk ke satu posisi di atas top

Menambahkan data ke stack (PUSH)

x[n++] = data;

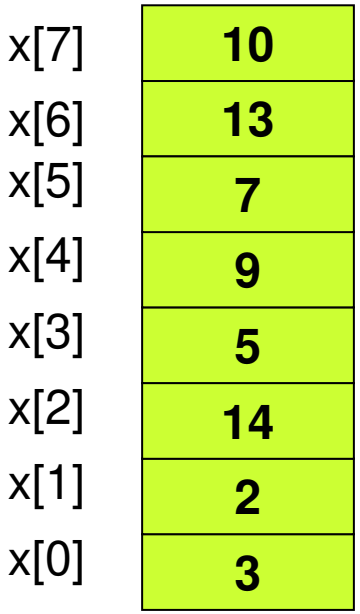
Menghapus data dari stack (POP)

data =x[--n];

Saat PUSH/POP dilakukan, jangan lupa mengupdate stack pointer

Suplemen（cara pemakaian ++ dan --）

n++	setelah diproses n = n + 1	++n	sebelum diproses n = n + 1
n --	setelah diproses n = n – 1	-- n	sebelum diproses n = n – 1




```
n=3;
printf("%d\n",x[n]);    5
printf("%d\n", n);      3
printf("%d\n",x[++n]); 9
printf("%d\n", n);      4
printf("%d\n",x[n++]); 9
printf("%d\n", n);      5
printf("%d\n",x[n--]); 7
printf("%d\n", n);      4
printf("%d\n",x[--n]); 5
printf("%d\n", n);      3
```

Implementasi Stack

- 4 operasi utama dalam implementasi stack
    1. init : inisialisasi stack (mengosongkan stack)
    2. push : menambahkan data pada stack
    3. pop : menghapus data dari stack
    4. empty : memeriksa apakah stack sudah kosong atau tidak

Reverse Polish Notation (RPN)

- Reverse Polish Notation: menaruh operator di belakang
- Nama lain : postfix notation
- Implementasi RPN memakai stack
  1. Jika “angka”, tambahkan pada stack
  2. Jika “operator”, turunkan (POP) dua buah data dari stack, lakukan perhitungan, dan tambahkan (PUSH) hasilnya pada stack

Reverse Polish Notation (RPN)

- Downloadlah program dari situs kuliah, compile dan jalankan

- Cara penulisan

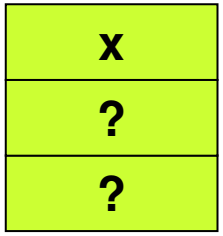
A + B	**A B +**
A – B	**A B –**
A / B	**A B /**
A * B	**A B ***

Reverse Polish Notation (RPN)

```
while ((c = getchar()) != EOF) {
    if (isdigit(c)) {
        ungetc(c, stdin);
        scanf("%ld", &x);
        push(x);
    }
else {
        switch (c) {
        case '+':
            b = pop(); a = pop();
            push(a + b);
            break;
        case '-':
            …
         ….
            }
    }
}
```

Setelah baris ke-67

Jika input data adalah angka, baca ulang Data tsb dan PUSH-lah data itu pada stack

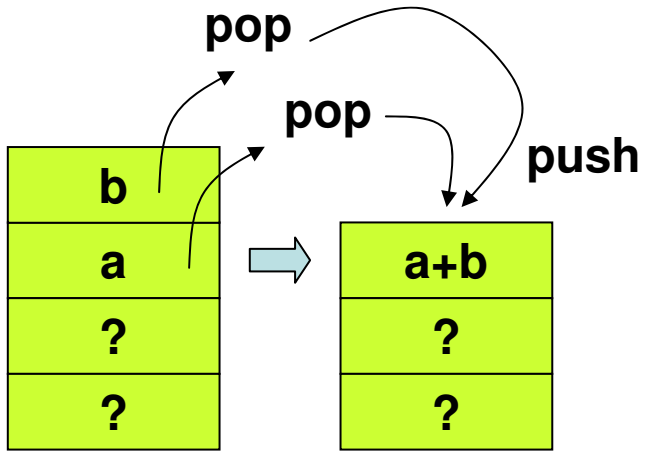


Jika input data itu bukan angka, lakukan perhitungan sesuai dengan operasi yang ditunjukkan oleh input data tsb.（+,-,*,/）

```
switch (c) {
case '+':
    b = pop(); a = pop();
    push(a + b);
    break;
case '-':
    b = pop(); a = pop();
    push(a - b);
    break;
case '*':
    b = pop(); a = pop();
    push(a * b);
    break;
case '/':
    b = pop(); a = pop();
    push(a / b);
    break;
case '\n':
    if (! empty())
        printf("Jawab: %ld \n", pop());
    init();
    break;
```

Jika operator itu '+'

Jika operator itu '-'



Jika operator itu '*'

Jika operator itu '/'

Jika return key

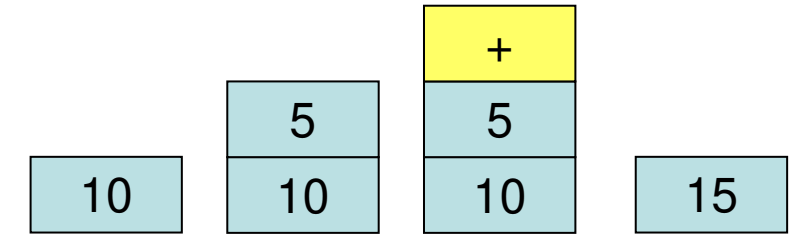
Reverse Polish Notation (RPN)

Contoh-1 10 + 5

10 5 +

Data di PUSH ke stack

Jika operator, lakukan operasi pada dua data terakhir yang di PUSH ke stack



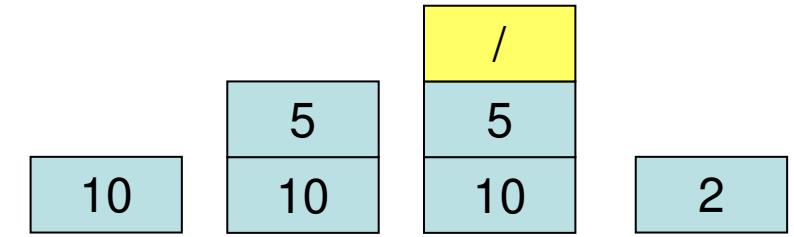
Reverse Polish Notation (RPN)

Contoh-2 10 / 5

10 5 /

Data di PUSH ke stack

Jika operator, lakukan operasi pada dua data terakhir yang di PUSH ke stack



Reverse Polish Notation (RPN)

Contoh-3
(banyaknya input > 2)

10 * (5 +4)

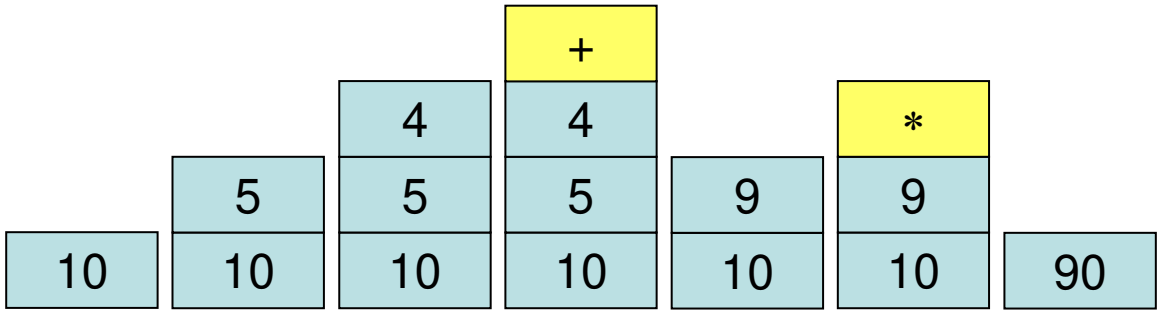
A ∗ B → A B ∗

10, (5+4) , *

A + B → A B +

10 , (5, 4, +), *

10 5 4 + *



Reverse Polish Notation (RPN)

Contoh-4

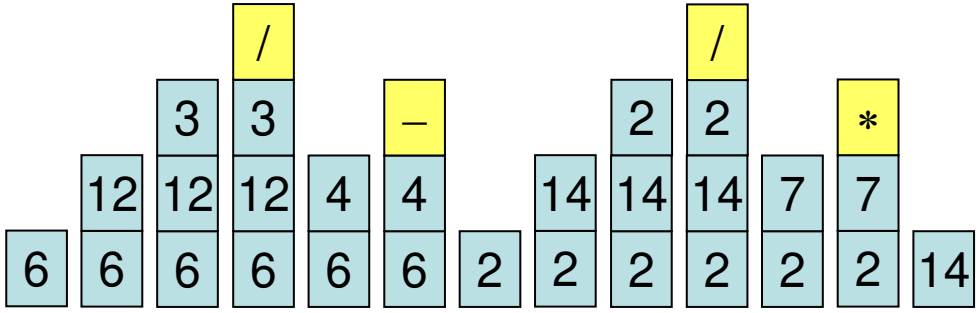
$$\left(6-\frac{12}{3}\right)*\frac{14}{2}$$

$$\left(6-\frac{12}{3}\right),\left(\frac{14}{2}\right),*$$

$$\left(6,\left(\frac{12}{3}\right),-\right),(14,2,/),*$$

$$(6,(12,3,/),-),(14,2,/),*$$

$$6 \quad 12 \quad 3 \quad / \quad - \quad 14 \quad 2 \quad / \quad *$$



Latihan 1

a) Ubahlah perhitungan di bawah ke dalam RPN

$$\frac{\frac{12}{3} * \left(25 - \frac{16}{4}\right)}{\frac{40}{10} + 2}$$

b) Implementasikan perhitungan di atas dengan stack, dan gambarkan kondisi stack tiap tahap

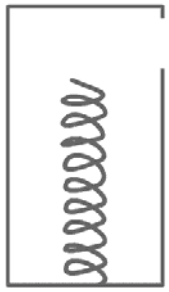
Beberapa jenis struktur data

1. Array
   1. Linear List
   2. Stack
   3. Queue

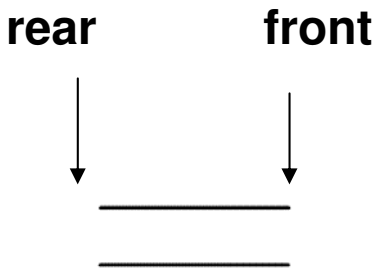
   1. Apa ?
   2. Bagaimana cara implementasinya ?
2. List
   1. Connected List
   2. Circular List
   3. Doubly-linked List
   4. Multi list structure
3. Tree Structure

Implementasi Queue

- Implementasi queue lebih sulit daripada stack. Pada stack, penambahan dan penghapusan data hanya dilakukan pada satu ujung saja, sehingga cukup mengubah posisi pointer sesuai dengan penambahan/pengurangan data
- Implementasi queue, harus mengubah posisi DUA buah pointer, yaitu pointer yang menunjuk ke FRONT, dan pointer yagn menunjuk ke REAR
- Ada dua cara implementasi queue:



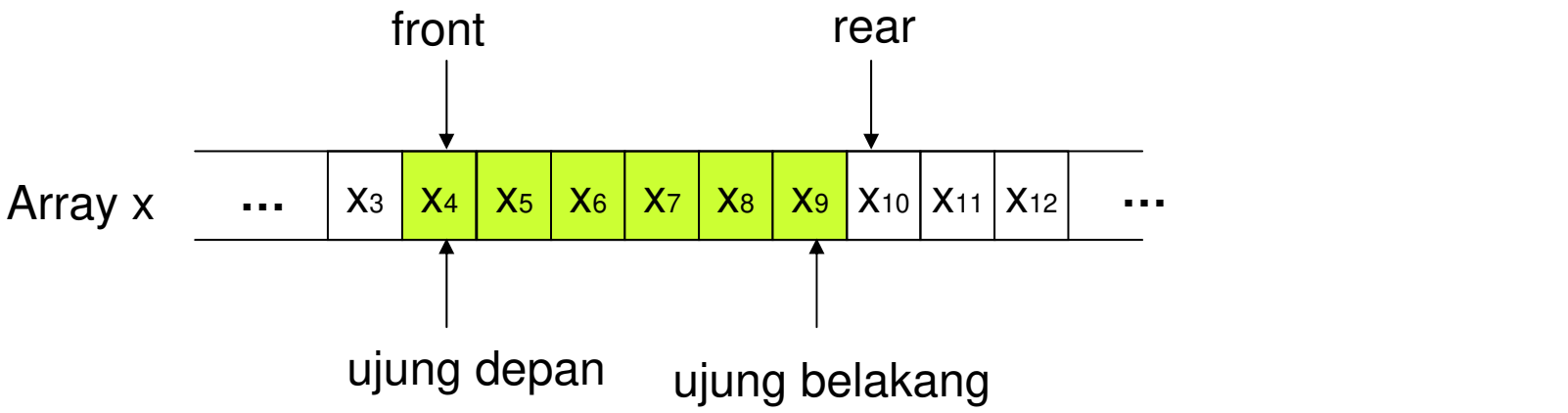
stack



queue

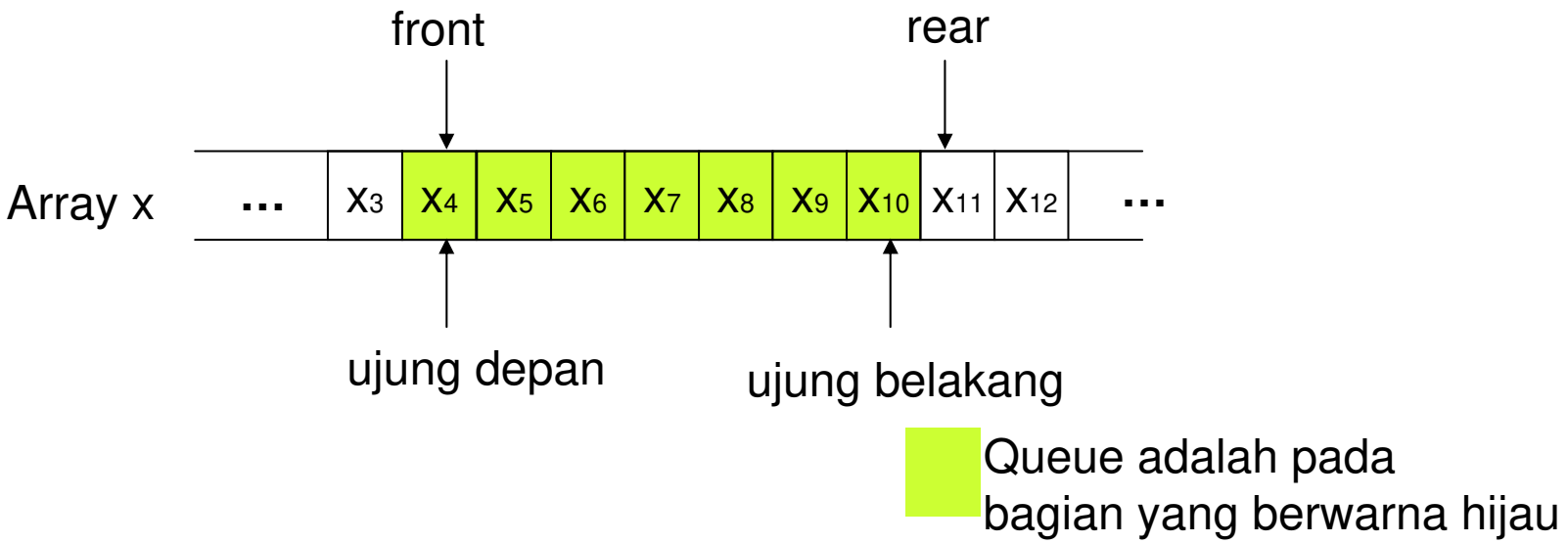
Cara 1

- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- Saat data dihapus dari queue, naikkan posisi pointer FRONT



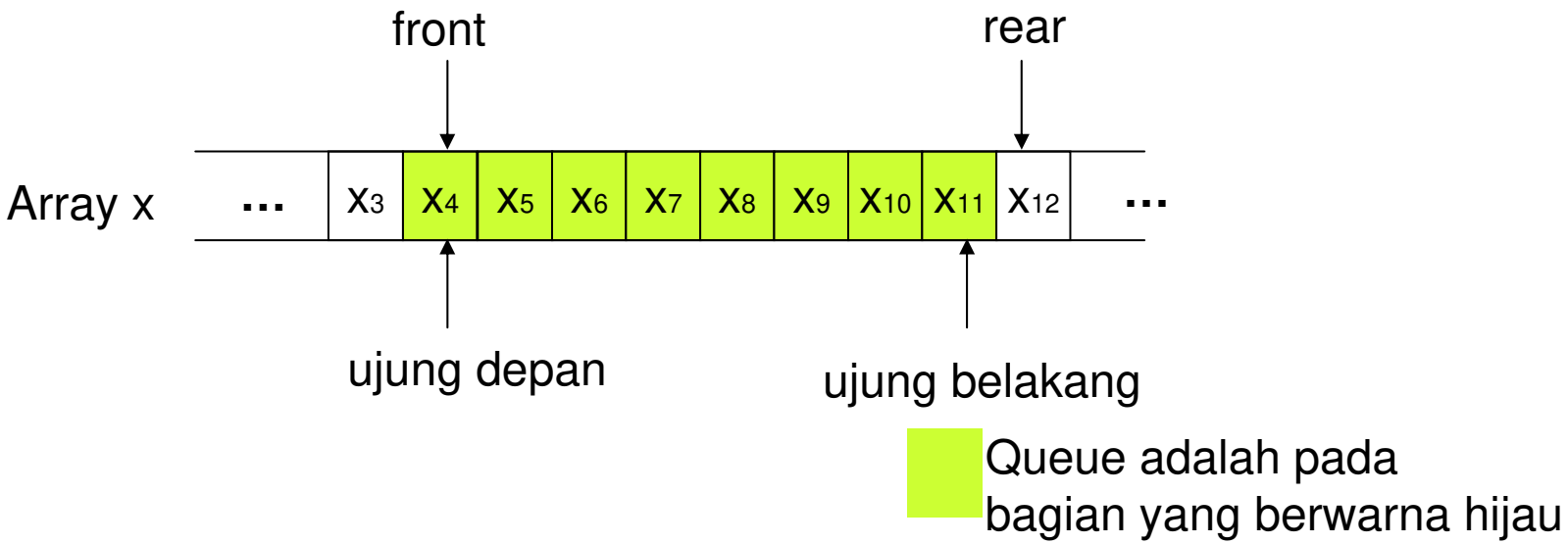
Cara 1

- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- **Saat data ditambahkan ke queue, naikkan posisi pointer REAR**
- Saat data dihapus dari queue, naikkan posisi pointer FRONT



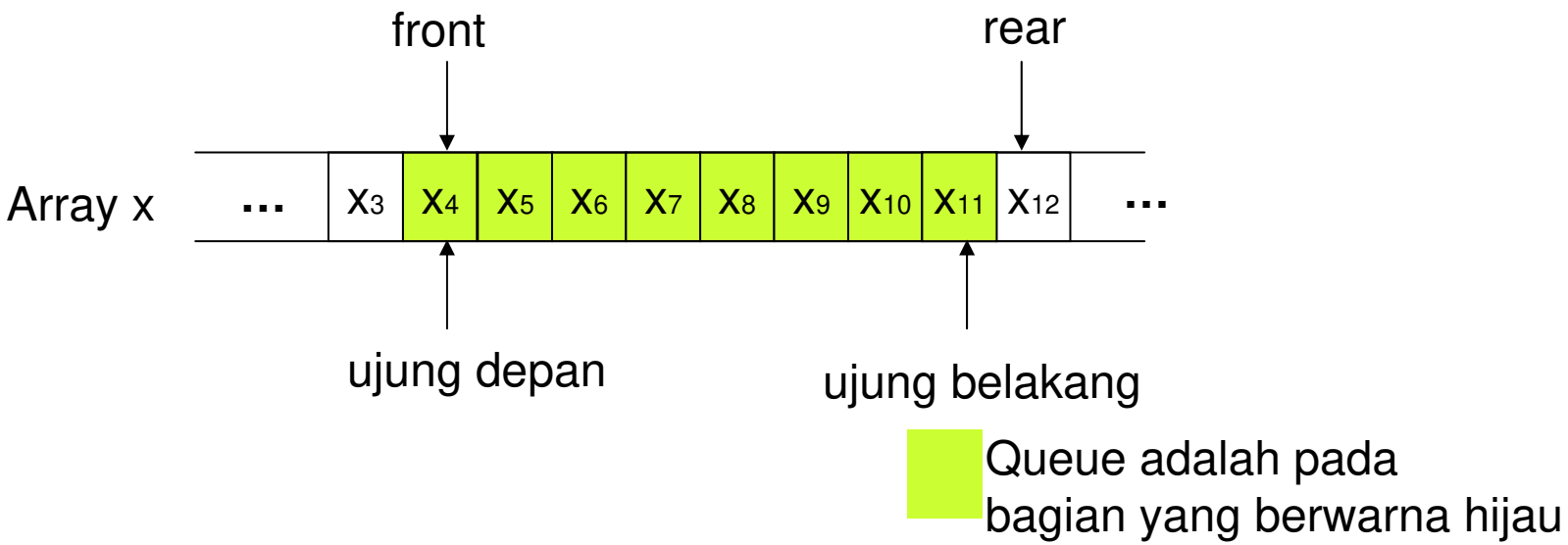
Cara 1

- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- **Saat data ditambahkan ke queue, naikkan posisi pointer REAR**
- Saat data dihapus dari queue, naikkan posisi pointer FRONT



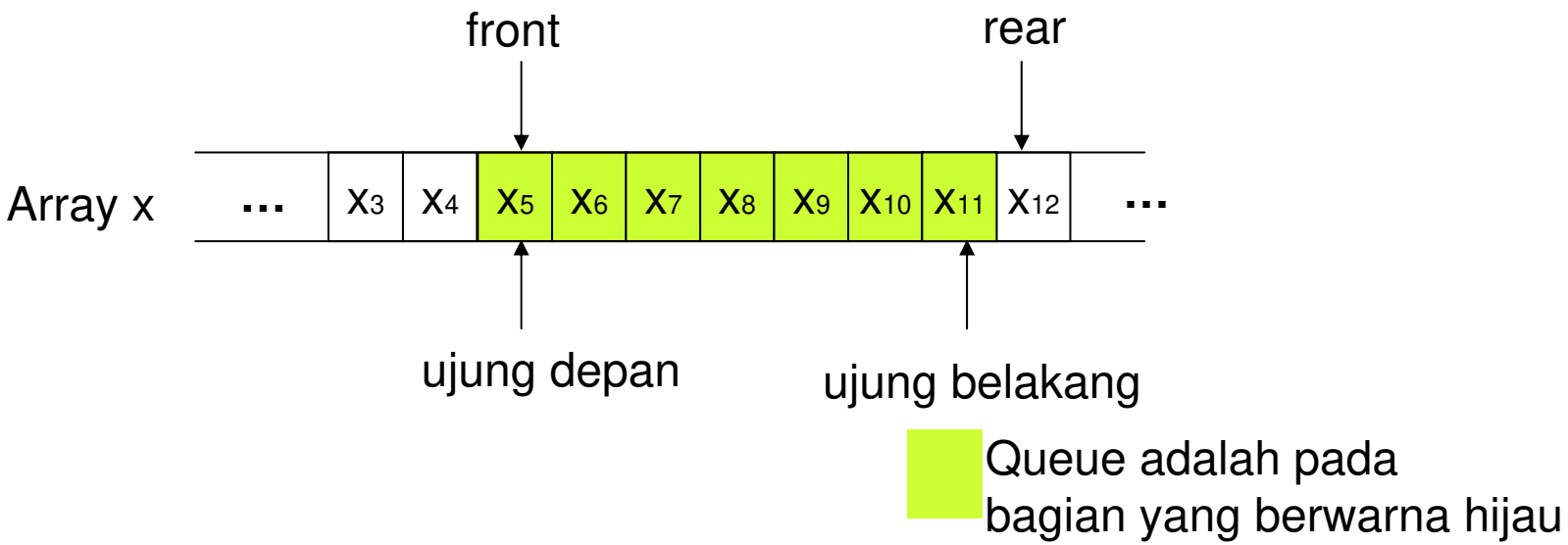
Cara 1

- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



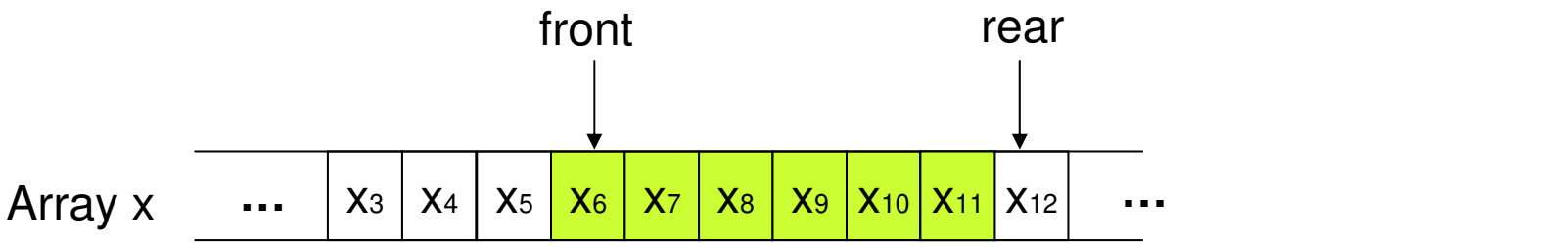
Cara 1

- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Cara 1

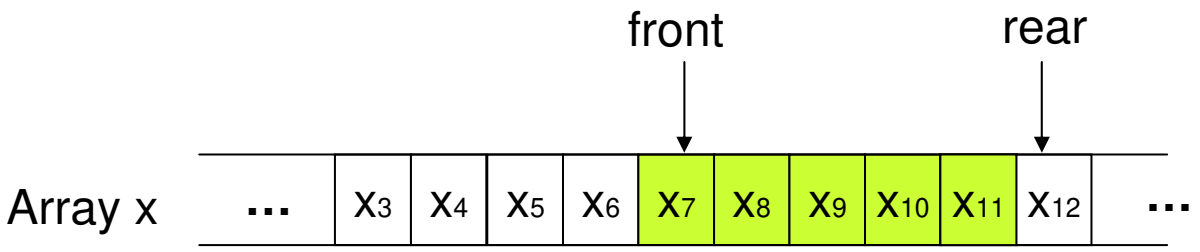
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

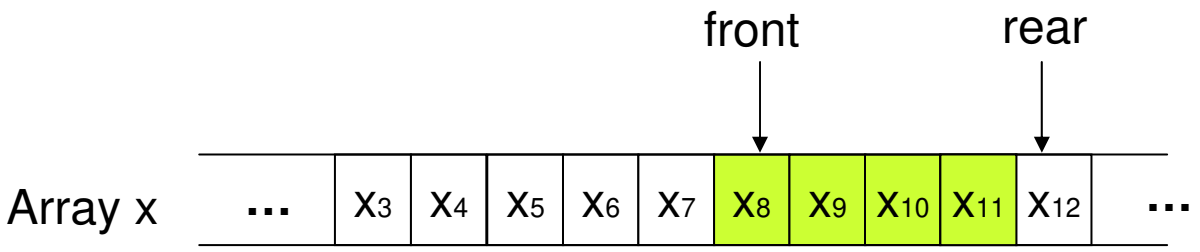
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

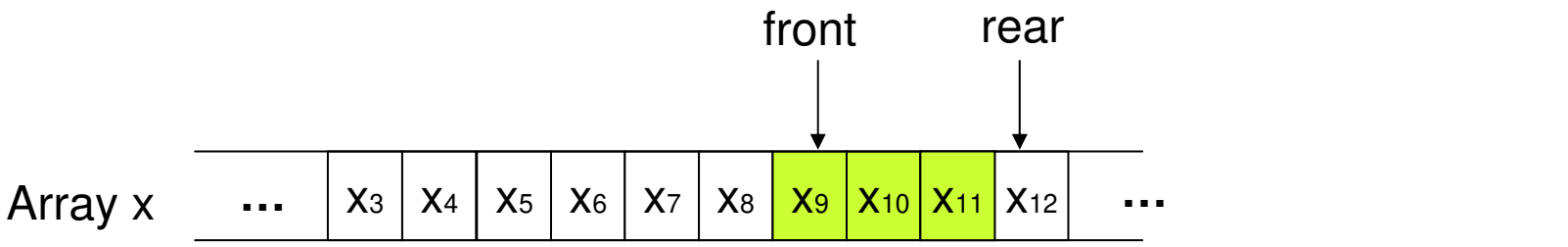
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

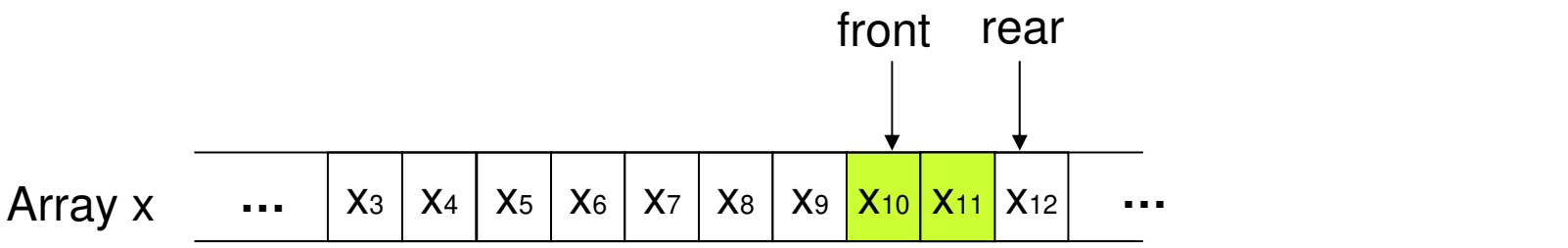
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

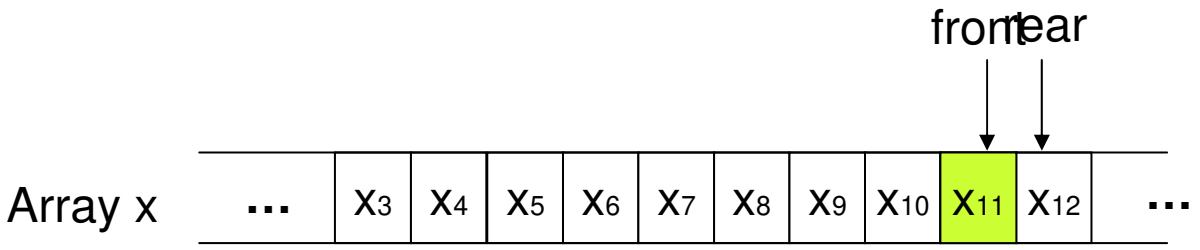
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

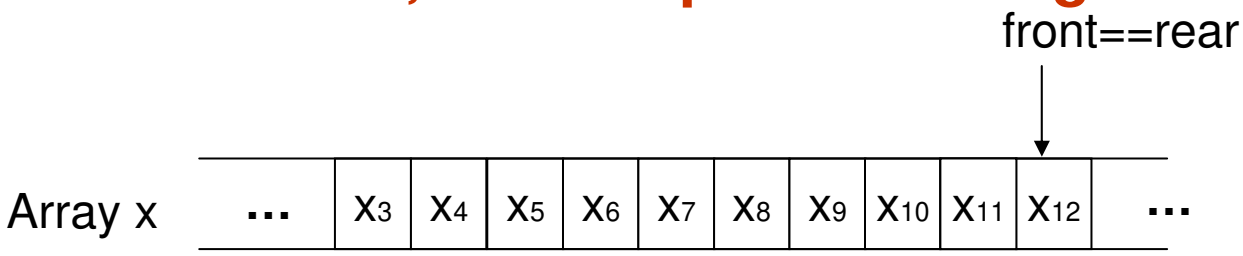
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- **Saat data dihapus dari queue, naikkan posisi pointer FRONT**



Queue adalah pada bagian yang berwarna hijau

Cara 1

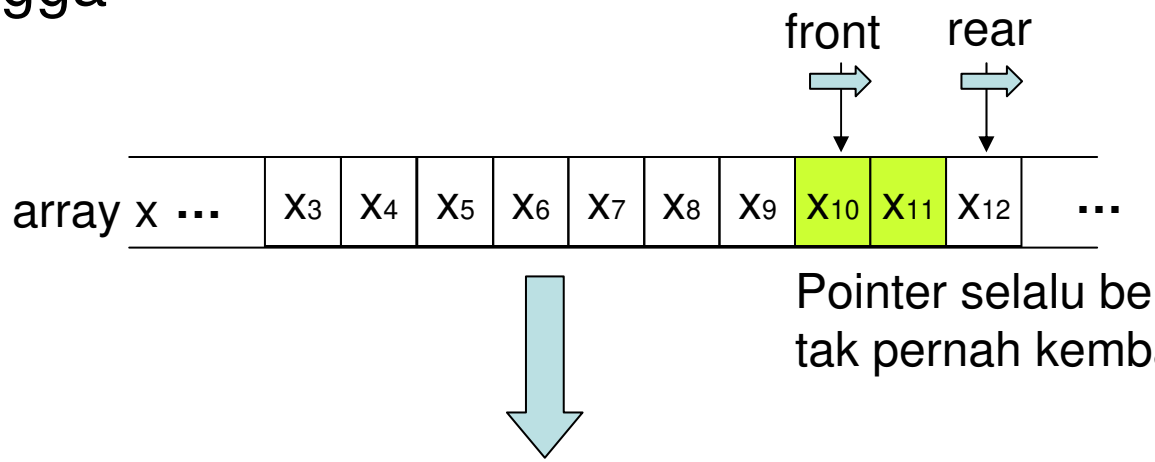
- Tempatkan data pada sebuah array, dan setlah pointer agar menunjuk ke posisi FRONT dan REAR
- Saat data ditambahkan ke queue, naikkan posisi pointer REAR
- Saat data dihapus dari queue, naikkan posisi pointer FRONT
- **Saat front==rear, berarti queue kosong**



Queue adalah pada bagian yang berwarna hijau

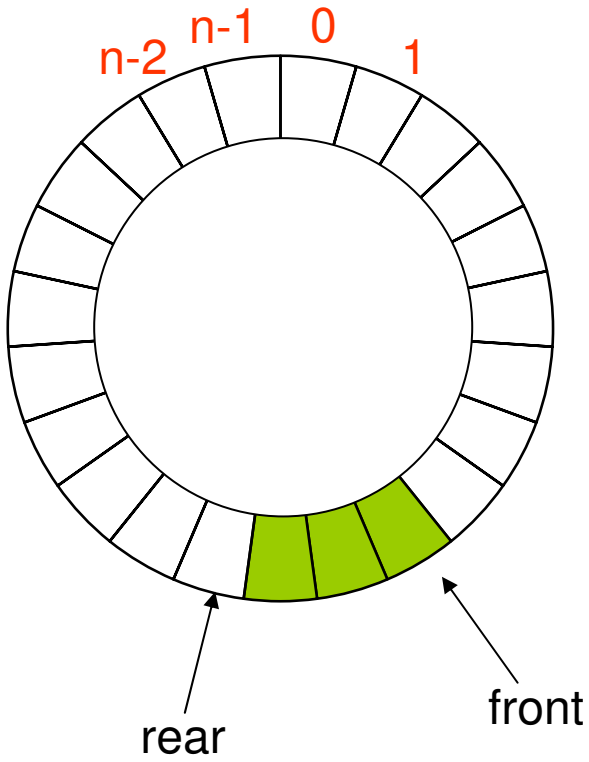
Masalah yang timbul pada Cara 1

Front dari rear selalu bertambah secara motononik, sehingga memerlukan array dengan ukuran tak terhingga



Menyambungkan FRONT dan REAR dari array itu sehingga membentuk cincin (RING BUFFER)

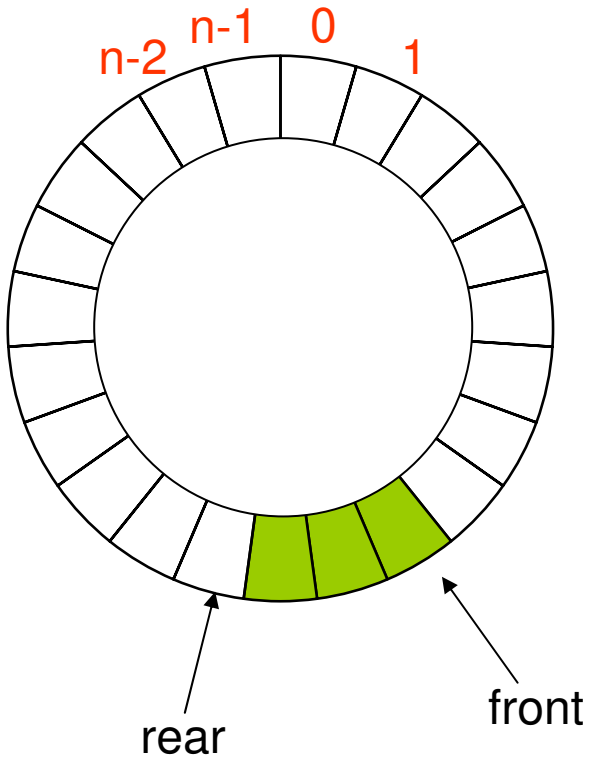
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- Menghapus data dari queue: nilai front dinaikkan satu
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

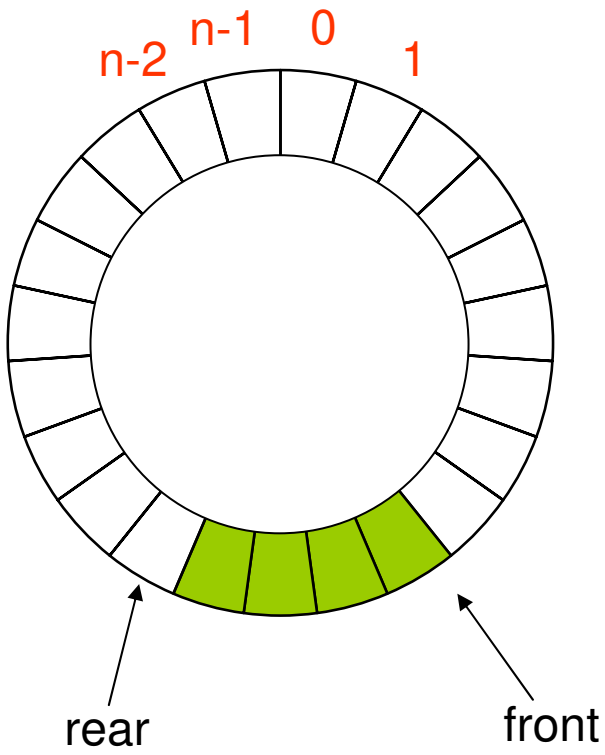
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- **Menambahkan data ke queue: nilai rear dinaikkan satu**
- Menghapus data dari queue: nilai front dinaikkan satu
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

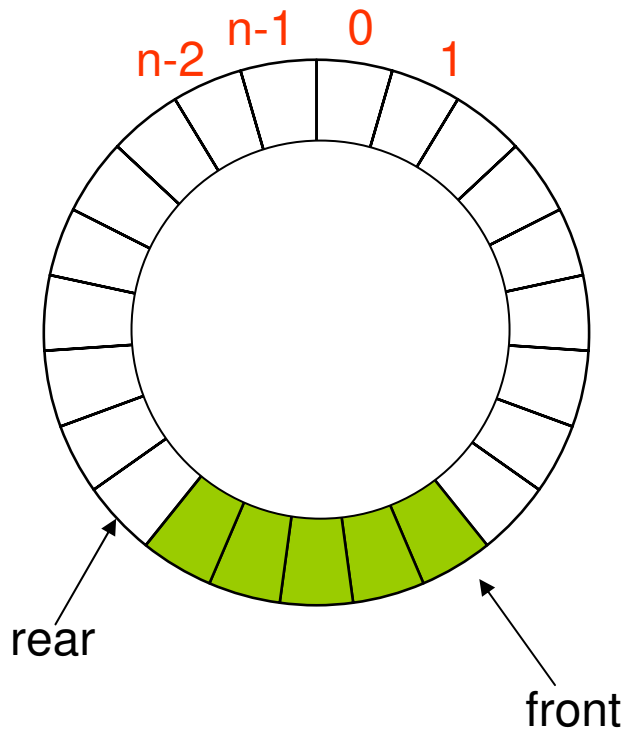
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- **Menambahkan data ke queue: nilai rear dinaikkan satu**
- Menghapus data dari queue: nilai front dinaikkan satu
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

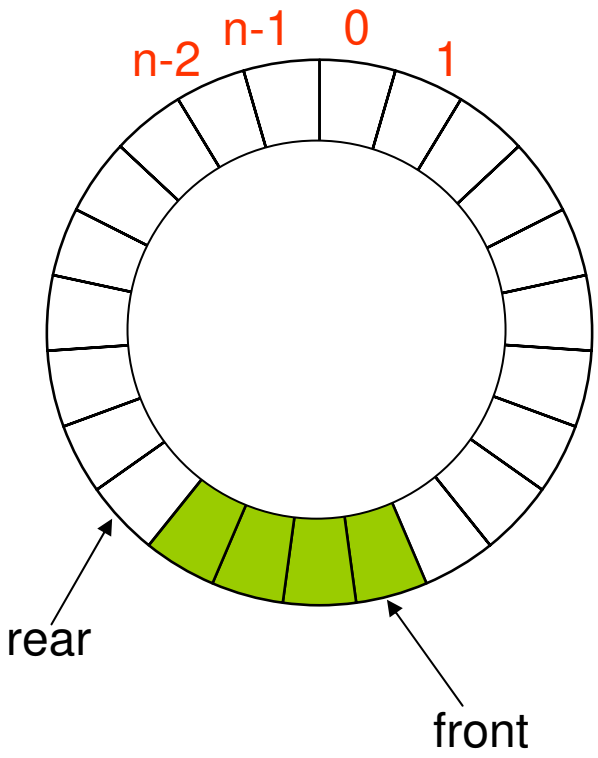
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- **Menambahkan data ke queue: nilai rear dinaikkan satu**
- Menghapus data dari queue: nilai front dinaikkan satu
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

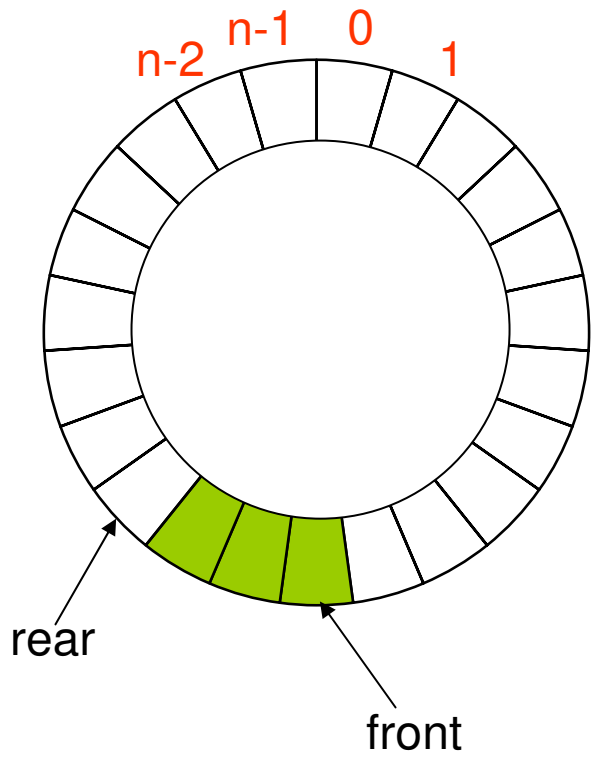
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- **Menghapus data dari queue: nilai front dinaikkan satu**
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

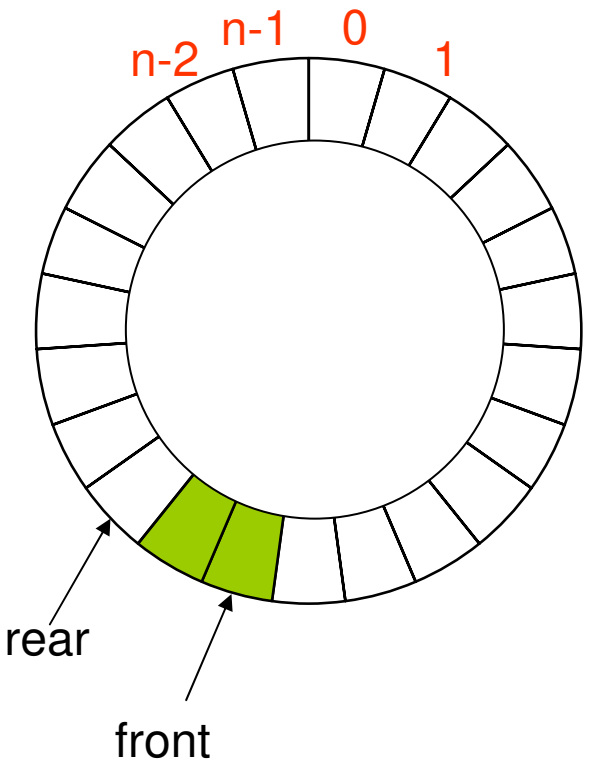
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- **Menghapus data dari queue: nilai front dinaikkan satu**
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

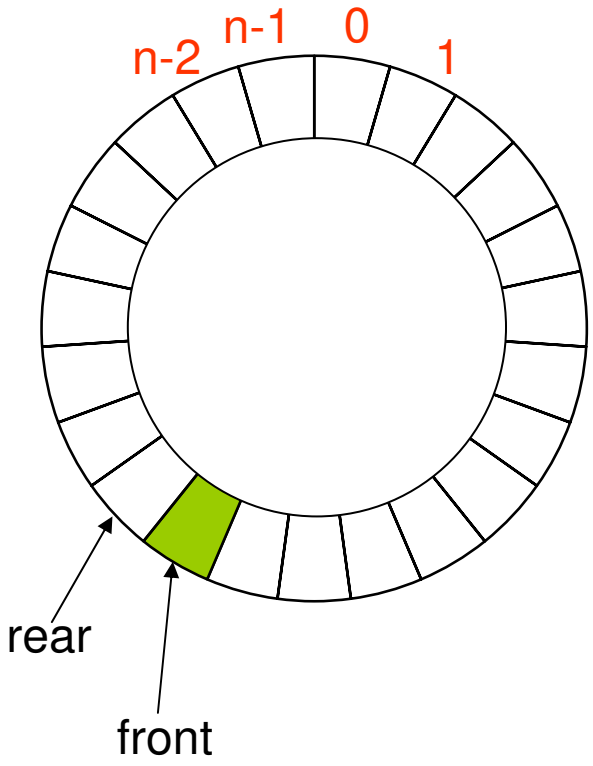
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- **Menghapus data dari queue: nilai front dinaikkan satu**
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

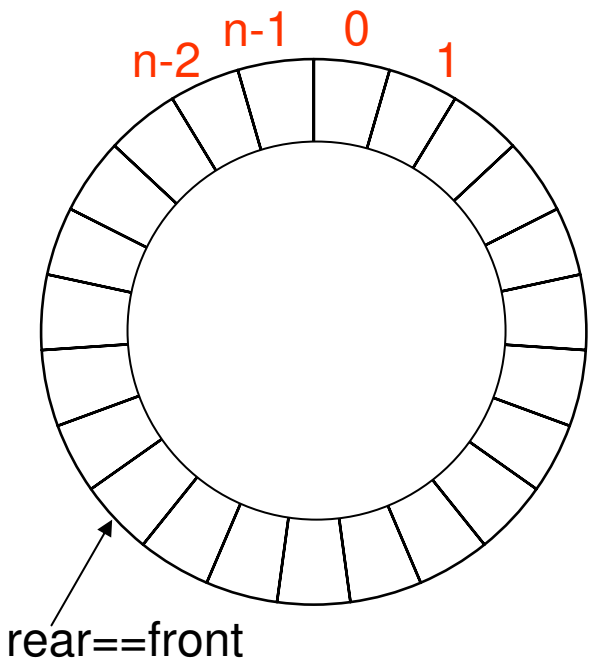
Cara 1



- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- **Menghapus data dari queue: nilai front dinaikkan satu**
- Queue kosong pada saat front==rear

Queue adalah pada bagian yang berwarna hijau

Cara 1

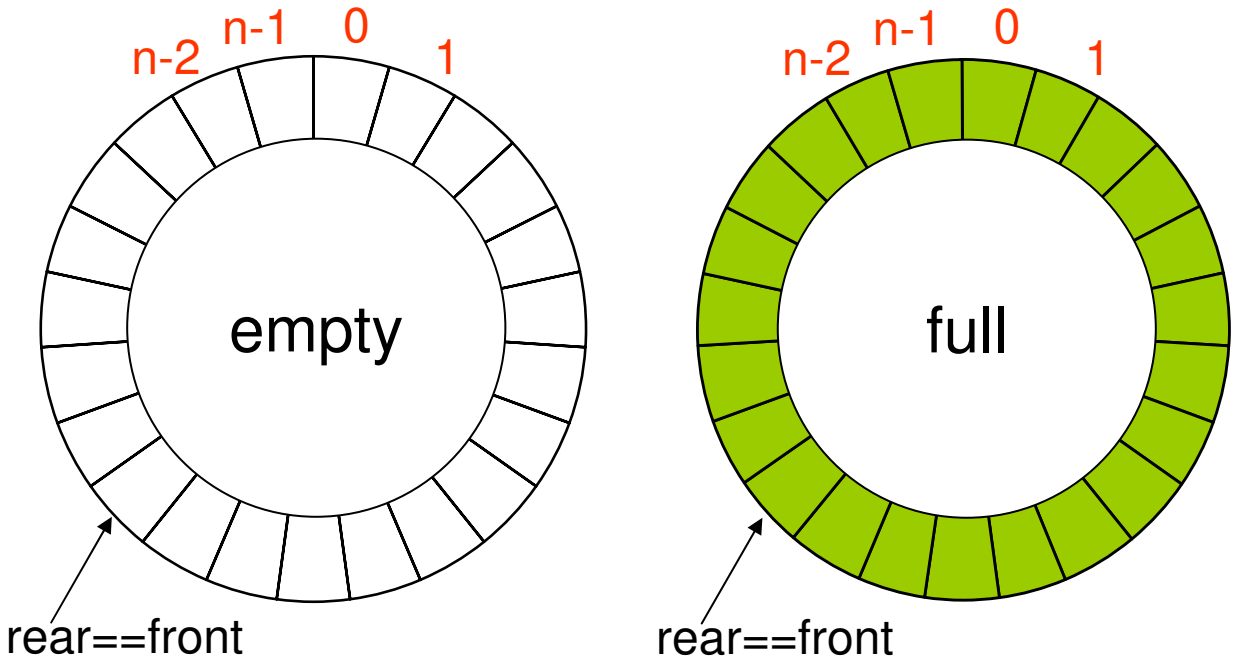


- Array dibuat seperti cincin, dimana elemen terakhir disambungkan dengan elemen pertama
- Menambahkan data ke queue: nilai rear dinaikkan satu
- Menghapus data dari queue: nilai front dinaikkan satu
- **Queue kosong pada saat front==rear**

Queue adalah pada bagian yang berwarna hijau

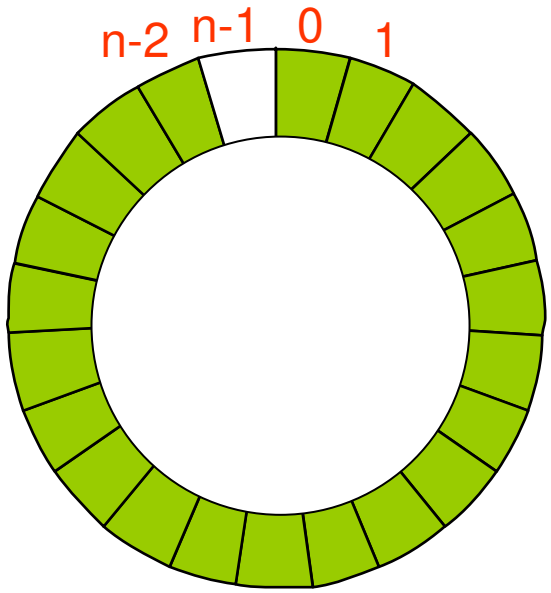
Masalah pada Ring Buffer

- Pada saat rear==front , ada dua interpretasi yang muncul, “queue kosong” ataukah “queue full”



Masalah pada Ring Buffer

- Solusi
  - Menyiapkan sebuah FLAG untuk memberi tanda kalau queue kosong
  - Queue dijaga agar tidak pernah full, dengan menyisakan sebuah elemen yang kosong



Tugas

- Downloadlah program stack.c dan selesaikan fungsi push dan pop !
- Jelaskan operasi yang dilakukan pada tiap baris program stack.c itu

Tugas

- Downloadlah program simulasi QUEUE: **queue.c** dan selesaikan fungsi-fungsi sbb.
  - void enqueue(char *x)
  - void dequeue()
  - Void queue_print()

- Implementasi memakai RING BUFFER, dengan selalu menyisakan sebuah elemen kosong
- Queue size: 5 (fixed)
- Input yang dimasukkan berupa character string

Contoh hasil eksekusi

```
[1] tambahkan task  [2] eksekusi [3] selesai     1
    Isi task :    baca buku
    Kondisi QUEUE :
          [0] baca buku

[1] tambahkan task  [2] eksekusi [3] selesai     1
    Isi task :    membuat program
    Kondisi QUEUE :
          [0] baca buku    [1] membuat program

[1] tambahkan task  [2] eksekusi [3] selesai     2
    Menjalankan task : baca buku
    Kondisi QUEUE :
          [1] membuat program
```

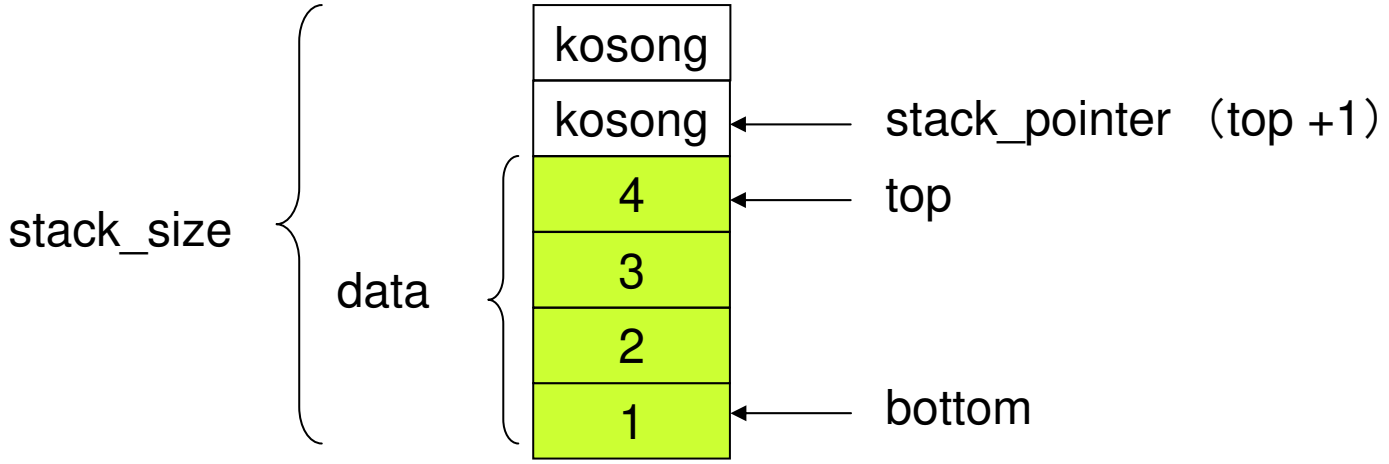
Hint

- Dalam fungsi enqueue/dequeue, posisi pointer front dan rear harus diupdate sehingga menunjuk ke posisi yang benar
  - Pengubahan posisi tsb. Memakai fungsi next()
  - Pada fungsi next(), sisa pembagian dihitung dengan cara membaginya dengan QUEUE_SIZE, sehingga jika nilai itu melebih QUEUE_SIZE, dia akan kembali ke ujung depan queue.
  - Posisi berikutnya dihitung dengan

    posisiberikutnya =next(posisi saat ini)

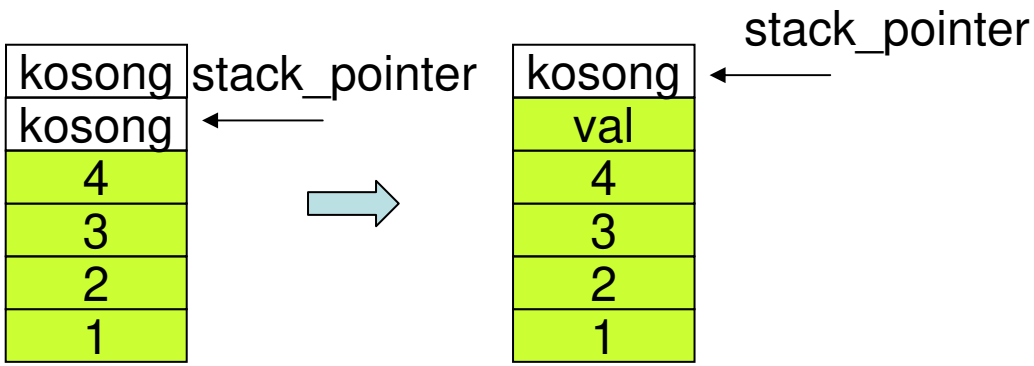
Pembahasan STACK.C

- Implementasi program STACK memakai array



push()

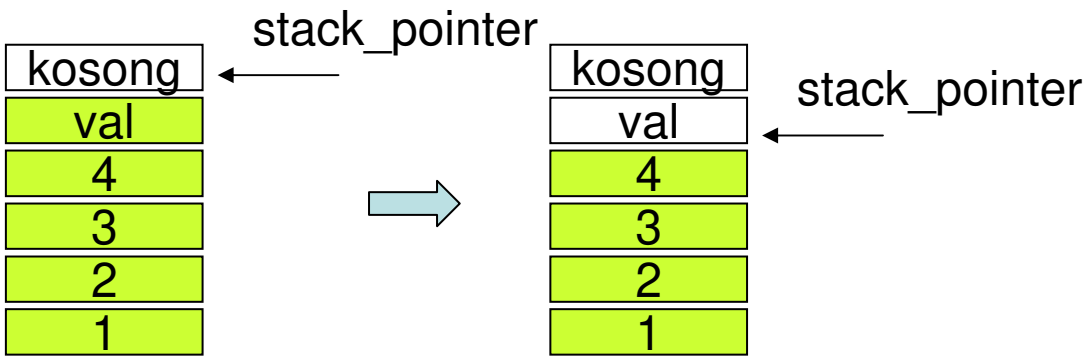
```
void push(int *x,int val)
{

    if(stack_pointer > stack_size-1) {
            printf("Stack overflow\n");  //memeriksa apakah overflow atau
    tidak
    }
    else {
            x[stack_pointer++]=val;//tambahkan data pada stack
    }


}
```



pop()

```
void pop(int *x)
{

    if(stack_pointer <= 0) {
            printf("Stack underflow\n");
    }
    else {
            stack_pointer--;  // menurunkan stack pointer

    }
}
```

Pada program ini, penghapusan data tidak perlu dilakukan. Cukup dengan mengubah posisi stack_pointer



Tugas: QUEUE.C

- Downloadlah program simulasi QUEUE: **queue.c** dan selesaikan fungsi-fungsi sbb.
  - void enqueue(char *x)
  - void dequeue()
  - Void queue_print()

- Implementasi memakai RING BUFFER, dengan selalu menyisakan sebuah elemen kosong
- Queue size: 5 (fixed)
- Input yang dimasukkan berupa character string

Contoh hasil eksekusi

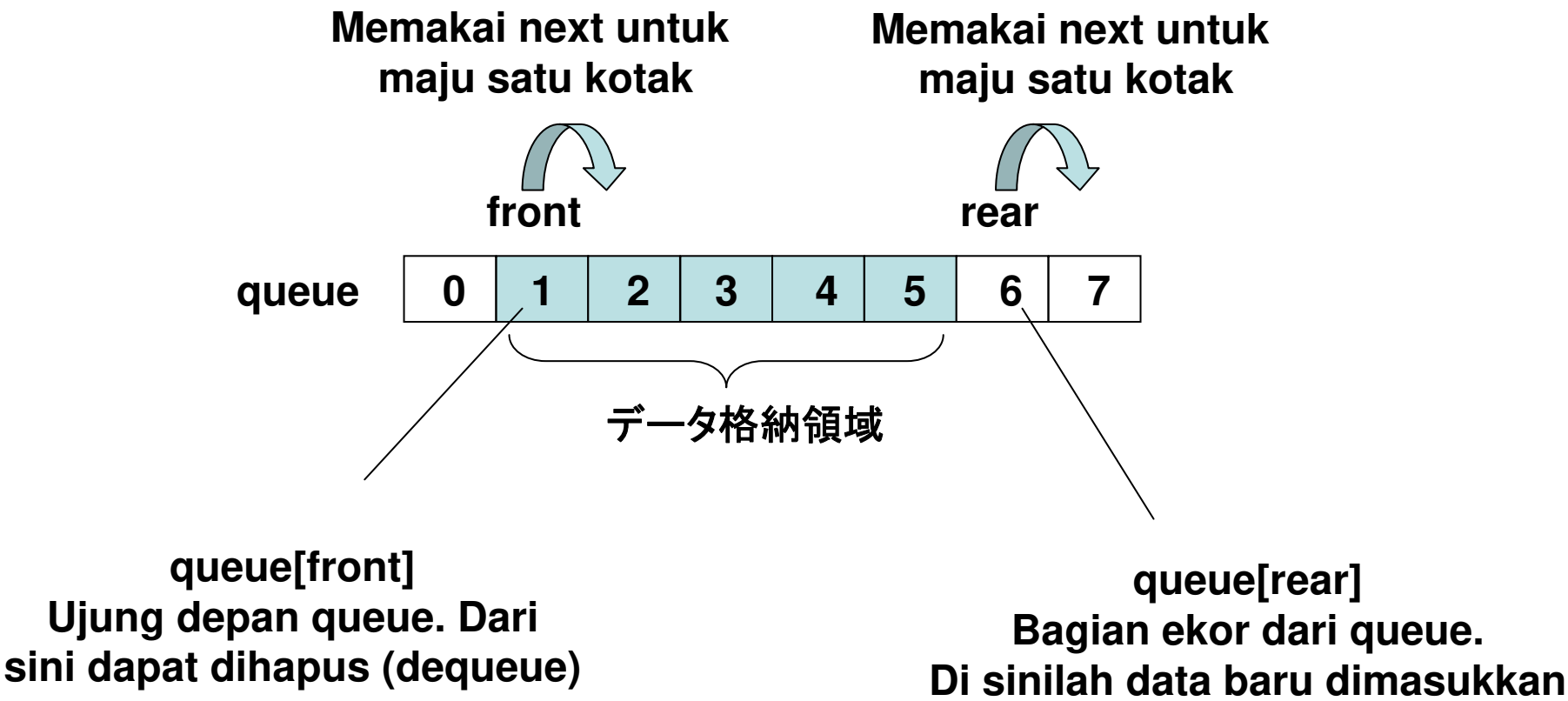
```
[1] tambahkan task  [2] eksekusi [3] selesai     1
    Isi task :    baca buku
    Kondisi QUEUE :
          [0] baca buku

[1] tambahkan task  [2] eksekusi [3] selesai     1
    Isi task :    membuat program
    Kondisi QUEUE :
          [0] baca buku    [1] membuat program

[1] tambahkan task  [2] eksekusi [3] selesai     2
    Menjalankan task : baca buku
    Kondisi QUEUE :
          [1] membuat program
```

Hint

- Dalam fungsi enqueue/dequeue, posisi pointer front dan rear harus diupdate sehingga menunjuk ke posisi yang benar
  - Pengubahan posisi tsb. Memakai fungsi next()
  - Pada fungsi next(), sisa pembagian dihitung dengan cara membaginya dengan QUEUE_SIZE, sehingga jika nilai itu melebih QUEUE_SIZE, dia akan kembali ke ujung depan queue.
  - Posisi berikutnya dihitung dengan

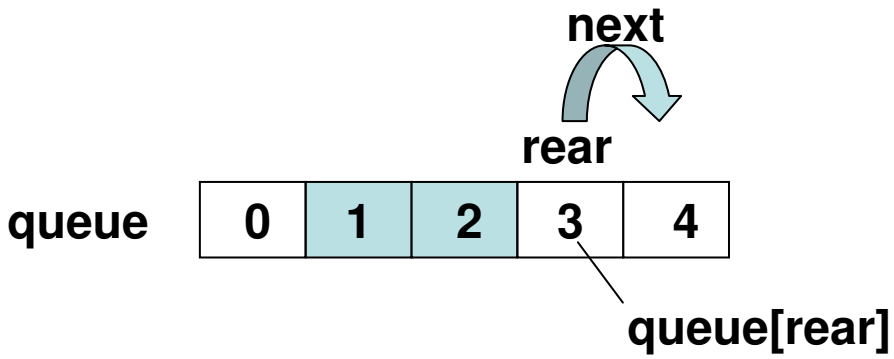
    posisiberikutnya =next(posisi saat ini)

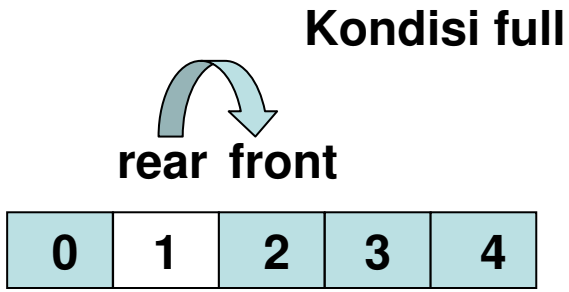
queue,front,rear



Fungsi next akan bergerak maju terus, dan setelah
sampai ujung kanan akan berpindah ke kotak 0

enqueue()

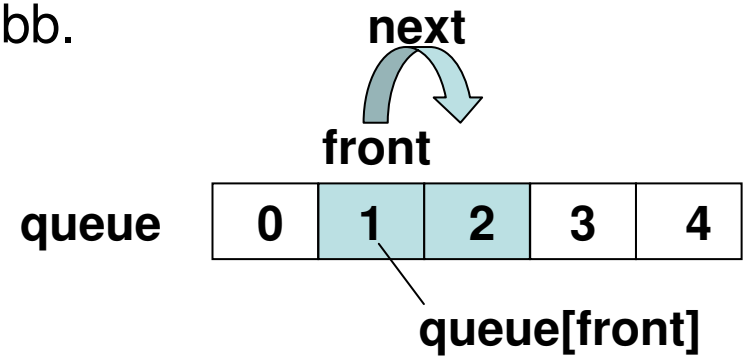




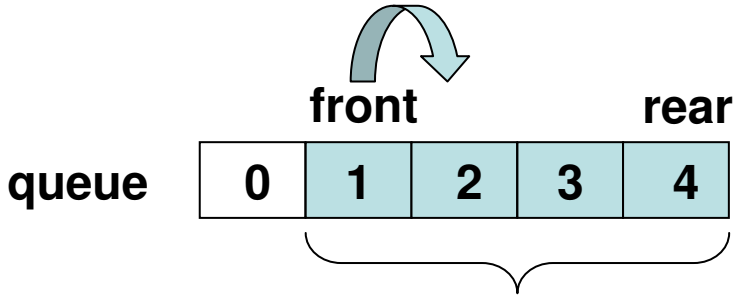

```
void enqueue(char *x)
{
// Periksalah terlebih dahulu, apakah queue itu full atau tidak
// Jika tidak full, tambahkan baru ke queue
// Selesaikan fungsi ini
        if (next(rear) == front)
                error("Queue sudah penuh\n");
        else {
                sprintf(queue[rear],x);
                rear=next(rear);
        }
}
```

dequeue()

```
// DEQUEUE sebuah task dari queue, dan menjalankannya
void dequeue()
{
        char *x;

// Periksalah apakah queue itu kosong atau tidak
// Jika tidak kosong, DEQUEUE-lah sebuah elemen
// dari queue, dan tampilkan pada layar baris sbb.
// "Menjalankan task: (isi task)"
// Selesaikan fungsi ini
        if (empty())
                error("Queue kosong\n");
        else {
                x=queue[front];
                front=next(front);
                printf("\tMenjalankan task: %s\n",x);
        }
}
```



queue_print()



Pada queue[i]
i digerakkan dari front sampai ke rear

```
// Menampilkan isi QUEUE

void queue_print()
{

// Selesaikan fungsi ini
        int i;

        printf("Kondisi QUEUE:\n");
        for(i=front;i != rear; i=next(i))
                printf("\t[%d] %s ", i, queue[i]);
        printf("\n");
}
```

Linked List

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

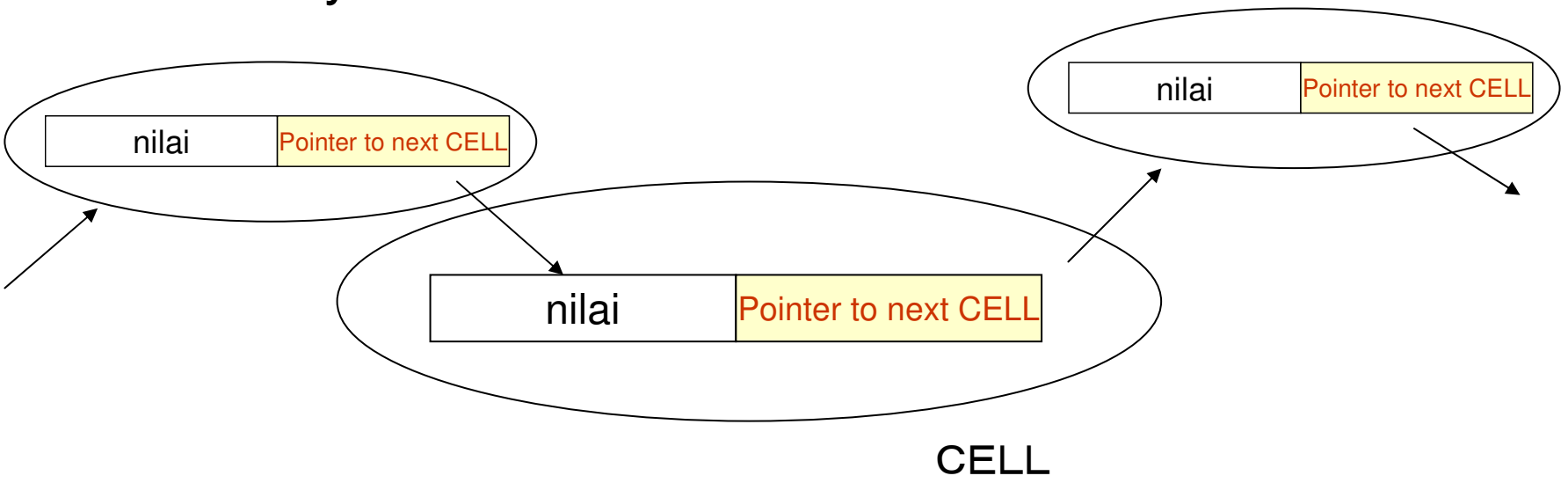
Beberapa jenis struktur data

1. Array
    1. Linear List
    2. Stack
    3. Queue
2. List
    1. Linked List
    2. Circular List
    3. Doubly-linked List
    4. Multi list structure

    1. Apa ?
    2. Bagaimana cara implementasinya ?
3. Tree Structure

Apakah Linked List itu ?

- Elemen (disebut dengan CELL, atau SEL dalam bahasa Indonesia) yang mungkin terletak terpisah-pisah di memory, disambungkan dengan pointer.
- Tiap sel berisi dua informasi : nilai dan pointer ke sel berikutnya



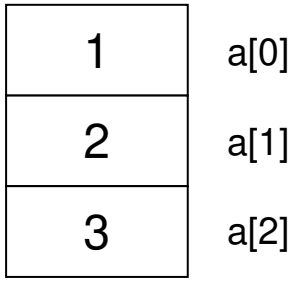
Mengapa memakai Linked List ?

1. Mudah untuk menambahkan dan menghapus elemen （pada array tidak mungkin menambahkan elemen, karena banyaknya elemen sudah ditentukan dari awal）
2. Panjang list bisa diubah dengan bebas (panjang array fixed)
3. Mudah untuk menyambungkan beberapa list, maupun memutuskannya (array tidak bisa)
4. Memungkinkan user mendesain struktur data yang kompleks

Array vs Linked List

	Array	Linked List
Penambahan dan penghapusan elemen	Tidak mungkin	Mungkin
Panjang list	Fixed	Bisa diubah
Akses ke elemen	$O(1)$ **Random access** cepat	$O(n)$ **Sequential access** (harus mengikuti pointer satu demi satu) lambat

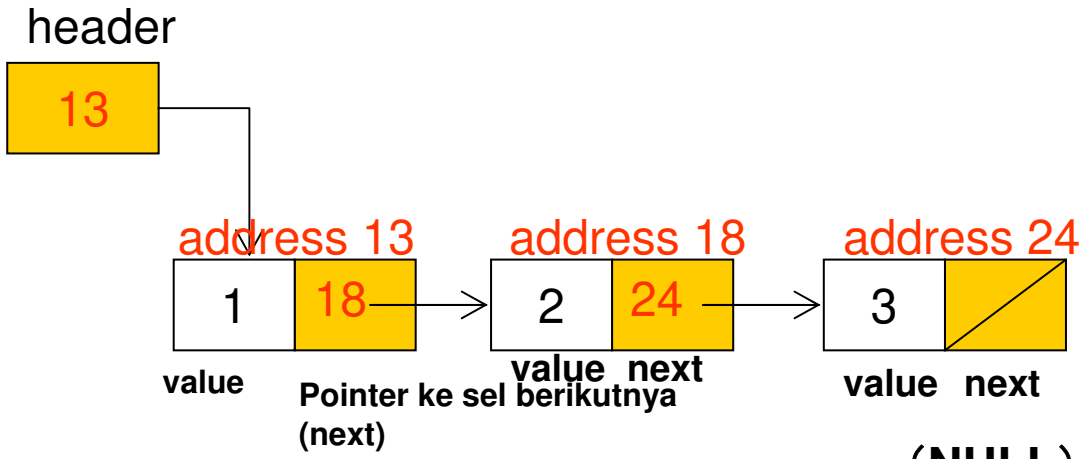
Array vs Linked List




```
int  a[3];
int  n;
```

Array

Address tiap sel berurutan



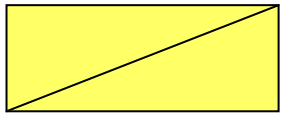

```
struct CELL {
    struct CELL *next;
     int              value;
} *header;
```

Linked List

Address tidak berurutan
Akses ke tiap sel dimulai dari header

NULL POINTER

- Nilai yang dimiliki sebuah pointer adalah address pada memory dimana data tersimpan
- Pointer yang tidak menunjuk ke address manapun disebut dengan NULL pointer. Maksudnya, satu kondisi khusus dimana pointer itu belum diset dengan sebuah address tertentu
- Pada stdio.h biasanya didefinisikan dengan nilai 0
- Saat fungsi fopen,malloc dieksekusi, jika terdapat error, maka nilai yang dikembalikan adalah NULL



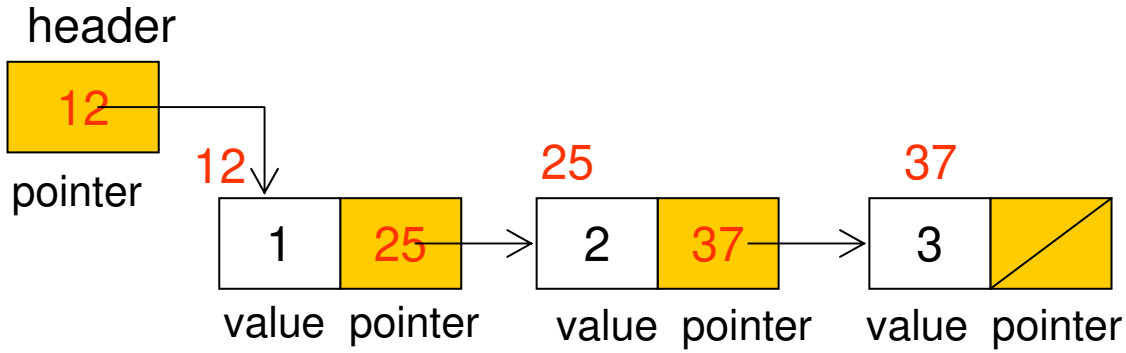
Pada kuliah ini, NULL disimbolkan dengan kotak yang diberi garis diagonal

AKSES SEKUENSIAL

- Sel berisi (value + pointer ke sel berikutnya)
- Akses hanya bisa dilakukan searah saja, dari ujung depan ke arah belakang. Akses berlawanan arah tidak dapat dilakukan.

Cara menampilkan isi seluruh list :

```
struct CELL  *p;
for(p=header;p!=NULL;p=p->next)    printf("%d\n",p->value);
```

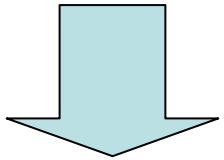


結果:
1
2
3

Pointer p digerakkan ke kanan dengan cara p=p->next

Cara menampilkan isi sel tertentu

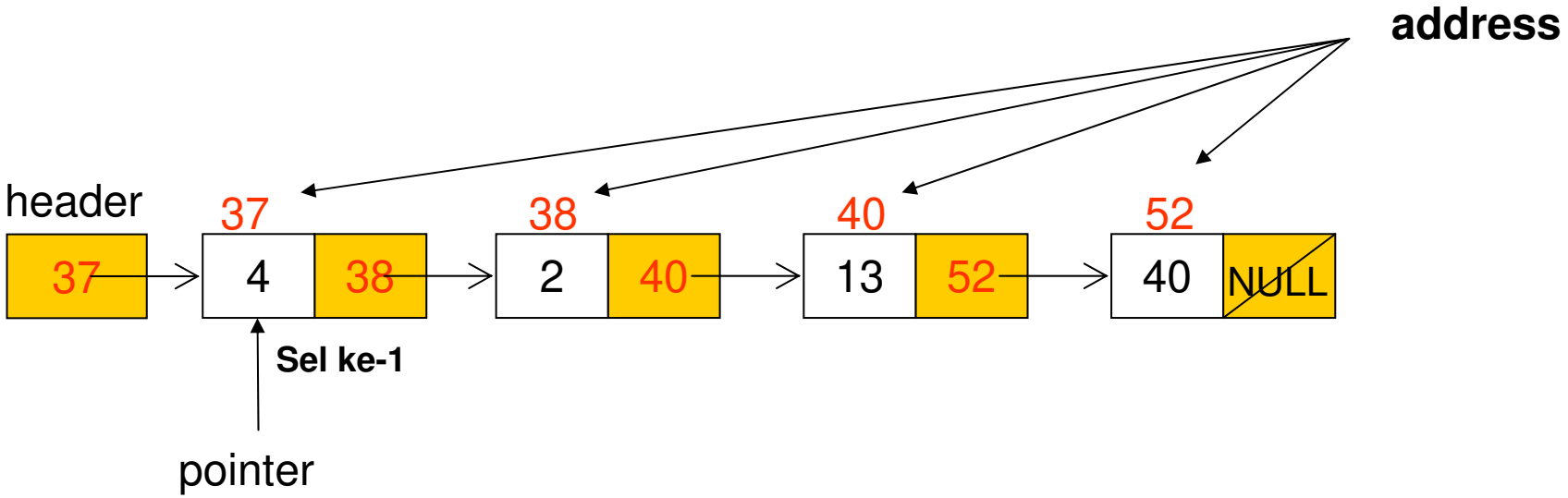
- Pada array (misalnya nama array: a), isi sel tertentu dapat ditampilkan dengan cara a[nomer urut sel keberapa]. Misalnya a[5] akan menampilkan isi sel ke-6. Hal ini karena satu sel dengan sel yang lain terletak pada posisi yang berurutan di memory.
- Pada linked list, kita tidak tahu secara langsung, sel itu terletak dimana dalam memory.



Akses harus dilakukan satu persatu, urut mulai dari sel terdepan

Cara menampilkan isi sel tertentu

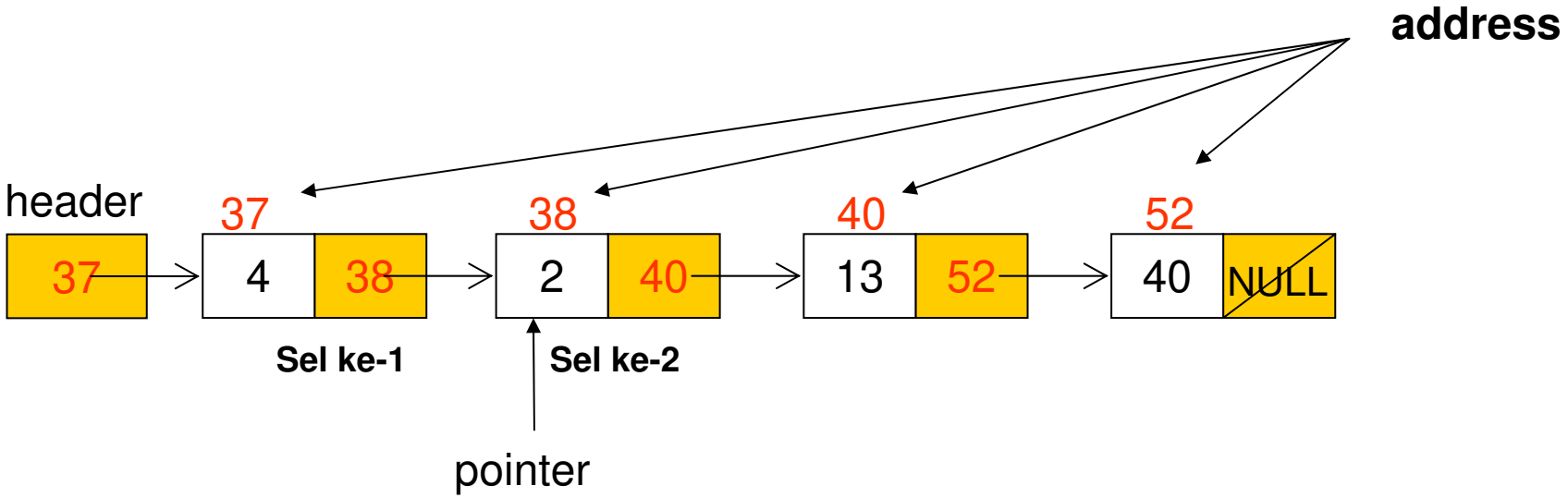
Tampilkan isi (value) sel ke-3 !



Sel ke-1→ isi : value=4
address sel berikutnya=38

Cara menampilkan isi sel tertentu

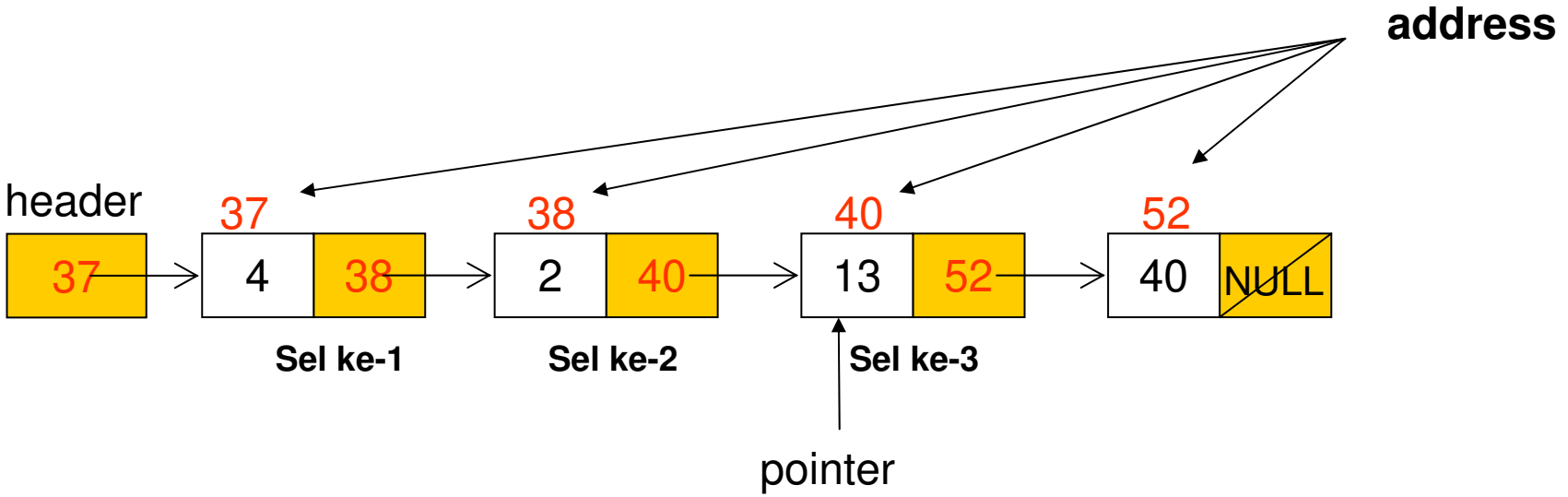
Tampilkan isi (value) sel ke-3 !



Sel ke-2→ isi : value=2
address sel berikutnya=40

Cara menampilkan isi sel tertentu

Tampilkan isi (value) sel ke-3 !



Sel ke-3→ isi : value=13
address sel berikutnya=52

Sampai di sini sudah bisa menampilkan isi sel ke-3

Operasi pada Linked List

1. Menambahkan data pada linked list
    1. Menambahkan sel baru yang terletak setelah sebuah sel tertentu
    2. Menambahkan sel baru pada posisi terdepan sebuah linked list
2. Menghapus data pada linked list
    1. Menghapus sebuah sel yang terletak setelah sebuah sel tertentu
    2. Menghapus sel pada posisi terdepan sebuah linked list
3. Syarat-syarat pada perbatasan

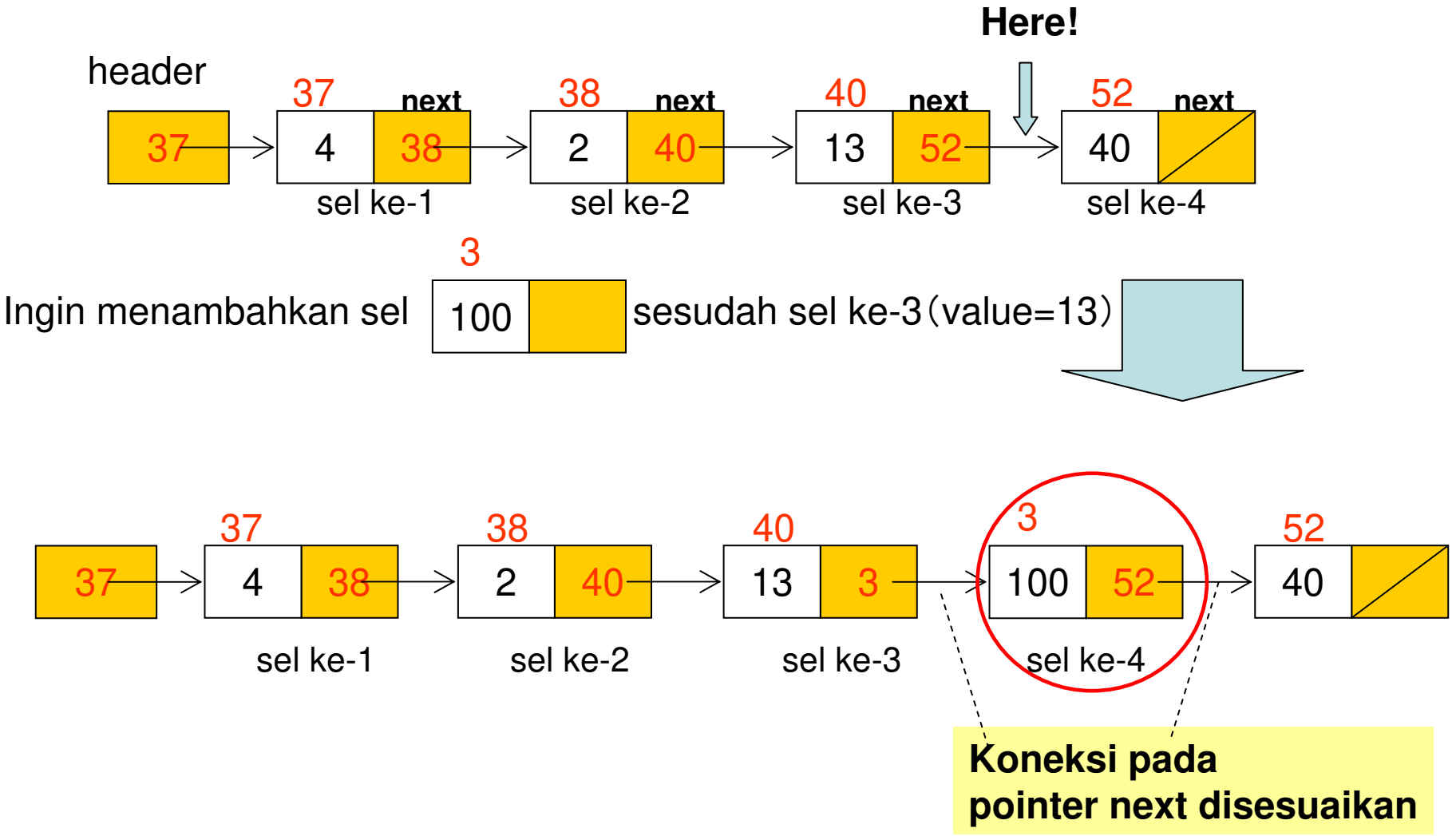
Menambahkan sel baru

1. Membuat sel yang akan ditambahkan (memakai malloc)
2. Mengeset “value” pada sel itu
3. Mengeset pointer “next” yang menunjukkan koneksi antara satu sel dengan yang lain

Catatan penting:

Pada linked list, penambahan sebuah sel baru hanya bisa dilakukan pada posisi sesudah sel tertentu, tetapi tidak dapat ditambahkan pada posisi sebelum sel tsb.

Contoh penambahan sel baru



Menambahkan sel baru setelah sel yang ditunjuk oleh pointer x

(1)Sediakan pointer p yang menunjuk ke sebuah sel

struct CELL *p;

(2)Buatlah sebuah sel baru dengan malloc() (alokasi memory) yang dimulai dengan address yang ditunjukkan oleh pointer p

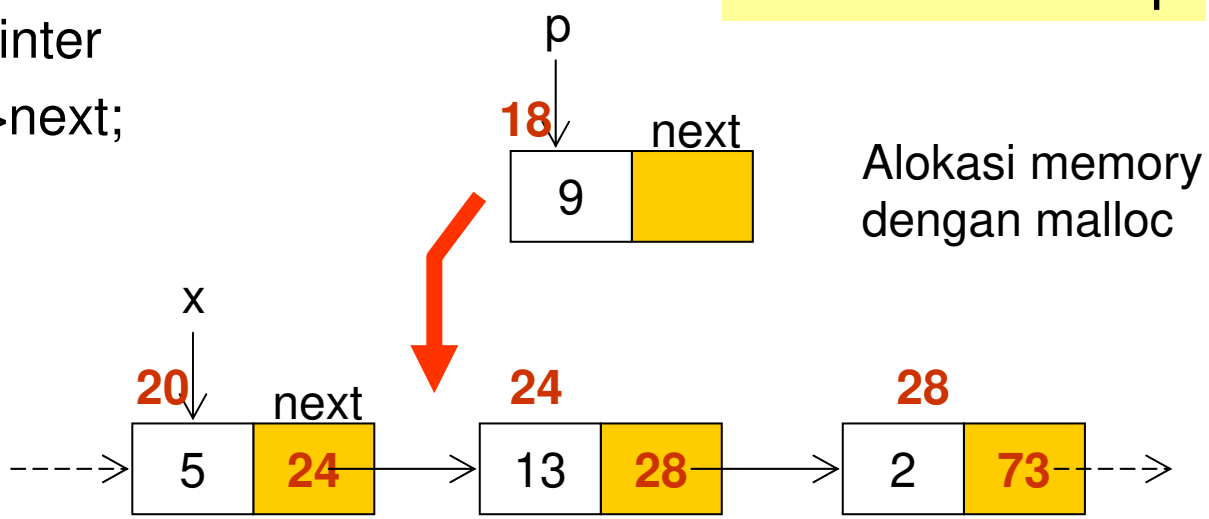
(3)Set-lah value pada sel p

(4)Modifikasi pointer

p->next = x->next;

x->next = p;

Masukkan sel p



Menambahkan sel baru setelah sel yang ditunjuk oleh pointer x

(1)Sediakan pointer p yang menunjuk ke sebuah sel

struct CELL *p;

(2)Buatlah sebuah sel baru dengan malloc() (alokasi memory) yang dimulai dengan address yang ditunjukkan oleh pointer p

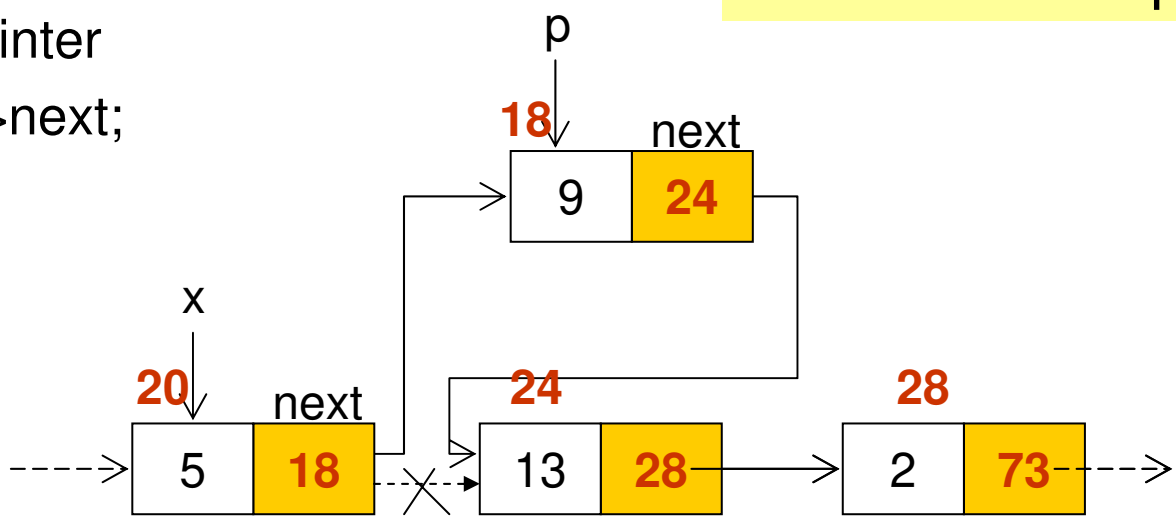
(3)Set-lah value pada sel p

(4)Modifikasi pointer

p->next = x->next;

x->next = p;

Masukkan sel p



Operasi pada Linked List

1. Menambahkan data pada linked list
    1. Menambahkan sel baru yang terletak setelah sebuah sel tertentu
    2. Menambahkan sel baru pada posisi terdepan sebuah linked list
2. Menghapus data pada linked list
    1. Menghapus sebuah sel yang terletak setelah sebuah sel tertentu
    2. Menghapus sel pada posisi terdepan sebuah linked list
3. Syarat-syarat pada perbatasan

Menambahkan sel baru pada posisi terdepan sebuah linked list

- Cara menambahkan sel baru pada posisi terdepan sebuah linked list berbeda dengan cara menambahkan sel ke tengah list tsb.

- Memakai pointer header yang menunjukkan posisi terdepan sebuah linked-list

Menambahkan sel baru pada posisi terdepan sebuah linked list

(1)Sediakan pointer p yang menunjuk ke sebuah sel

struct CELL *p;

(2)Buatlah sebuah sel baru dengan malloc() (alokasi memory) yang dimulai dengan address yang ditunjukkan oleh pointer p

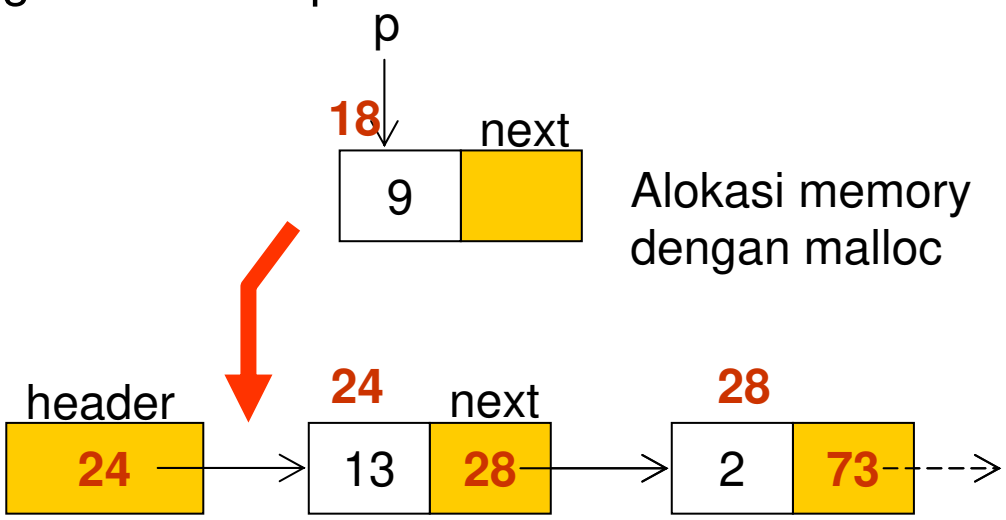
(3)Set-lah value pada sel p

(4)Substitusikan address pada header ke bagian next pada p

p->next = header;

(5) Updatelah nilai header dengan address p

header = p;



Menambahkan sel baru pada posisi terdepan sebuah linked list

(1)Sediakan pointer p yang menunjuk ke sebuah sel

struct CELL *p;

(2)Buatlah sebuah sel baru dengan malloc() (alokasi memory) yang dimulai dengan address yang ditunjukkan oleh pointer p

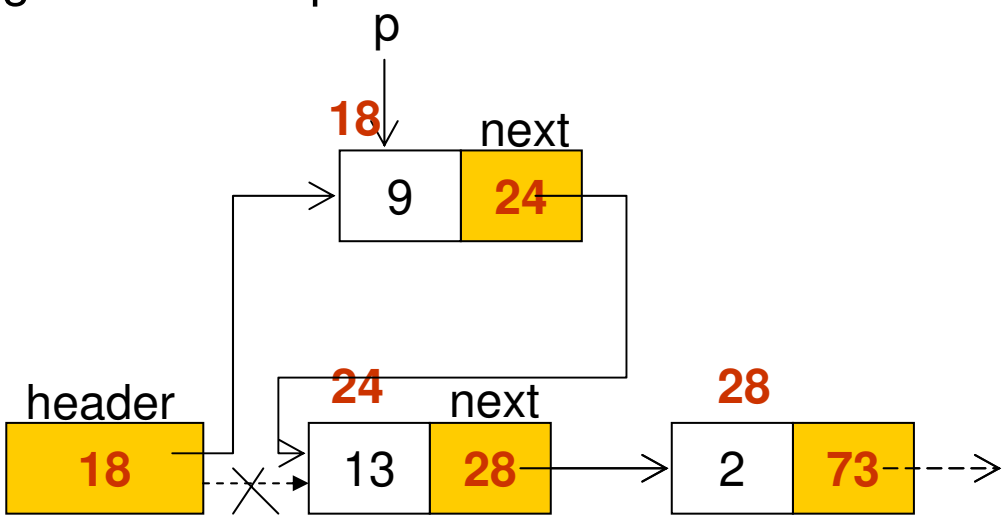
(3)Set-lah value pada sel p

(4)Substitusikan address pada header ke bagian next pada p

p->next = header;

(5) Updatelah nilai header dengan address p

header = p;



Menambahkan sel baru pada posisi terdepan sebuah linked list

- Cara lain menambahkan sel baru pada posisi terdepan sebuah list
- Header bukan pointer melainkan sel ke-0
- Memakai cara yang sama dengan list (1)（menambahkan elemen pada tengah-tengah sebuah list）

Menambahkan sel baru pada posisi terdepan sebuah linked list (2)

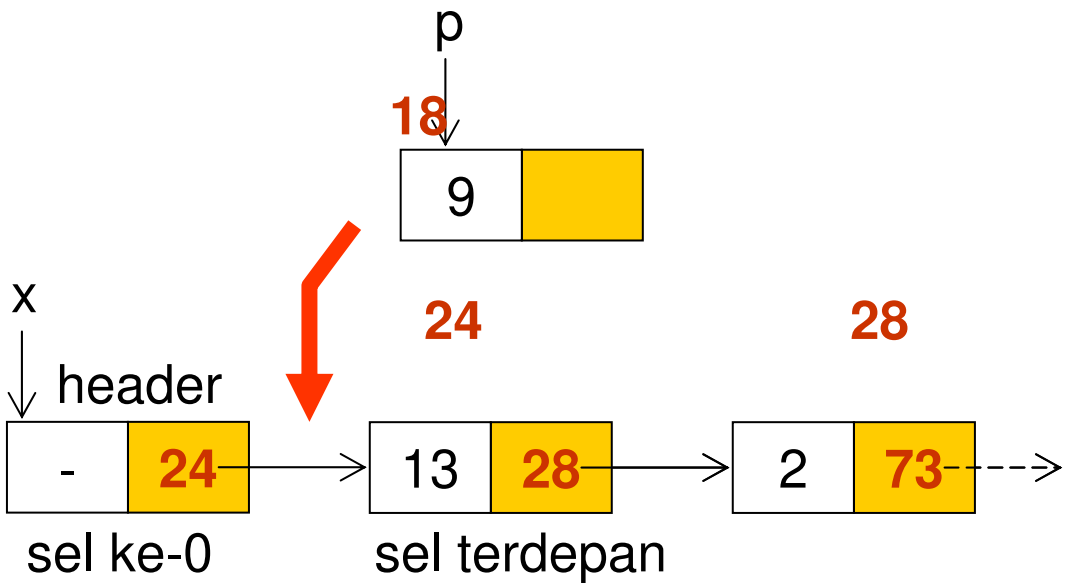
(1) Header tidak didefinisikan sebagai sebuah pointer, melainkan sebuah sel

struct CELL header;

(2)Tempatkan next pada header agar menunjuk ke posisi terdepan pada list, dan header dianggap sebagai sel ke-0

(3)Masukkan elemen p pada posisi sesudah elemen ke-0

(4)Selanjutnya sama dengan proses penambahan sel baru pada penjelasan sebelumnya (modifikasi next)



Menambahkan sel baru pada posisi terdepan sebuah linked list (2)

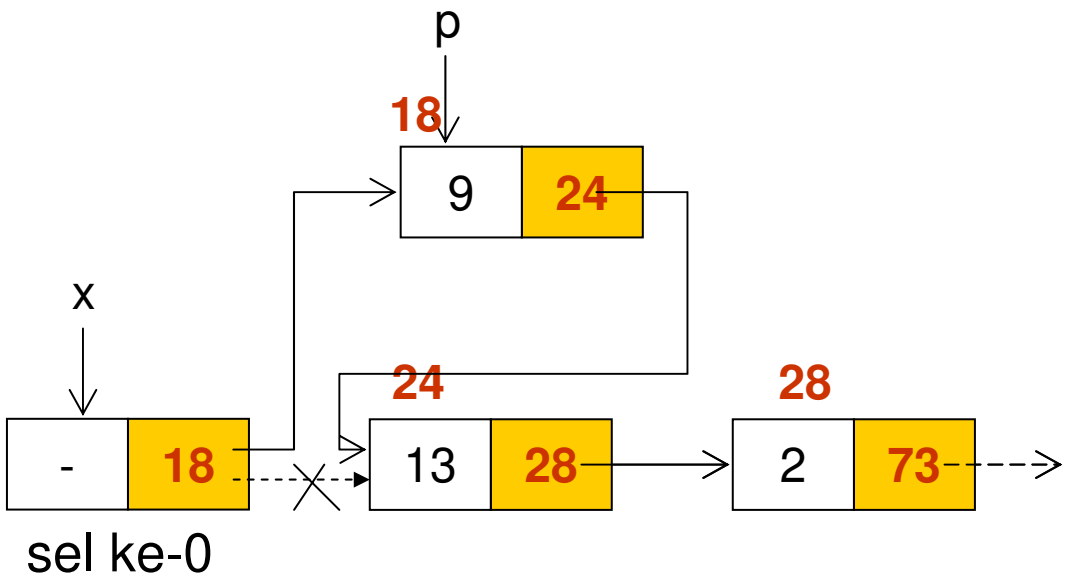
(1) Header tidak didefinisikan sebagai sebuah pointer, melainkan sebuah sel

struct CELL header;

(2)Tempatkan next pada header agar menunjuk ke posisi terdepan pada list, dan header dianggap sebagai sel ke-0

(3)Masukkan elemen p pada posisi sesudah elemen ke-0

(4)Selanjutnya sama dengan proses penambahan sel baru pada penjelasan sebelumnya (modifikasi next)



Operasi pada Linked List

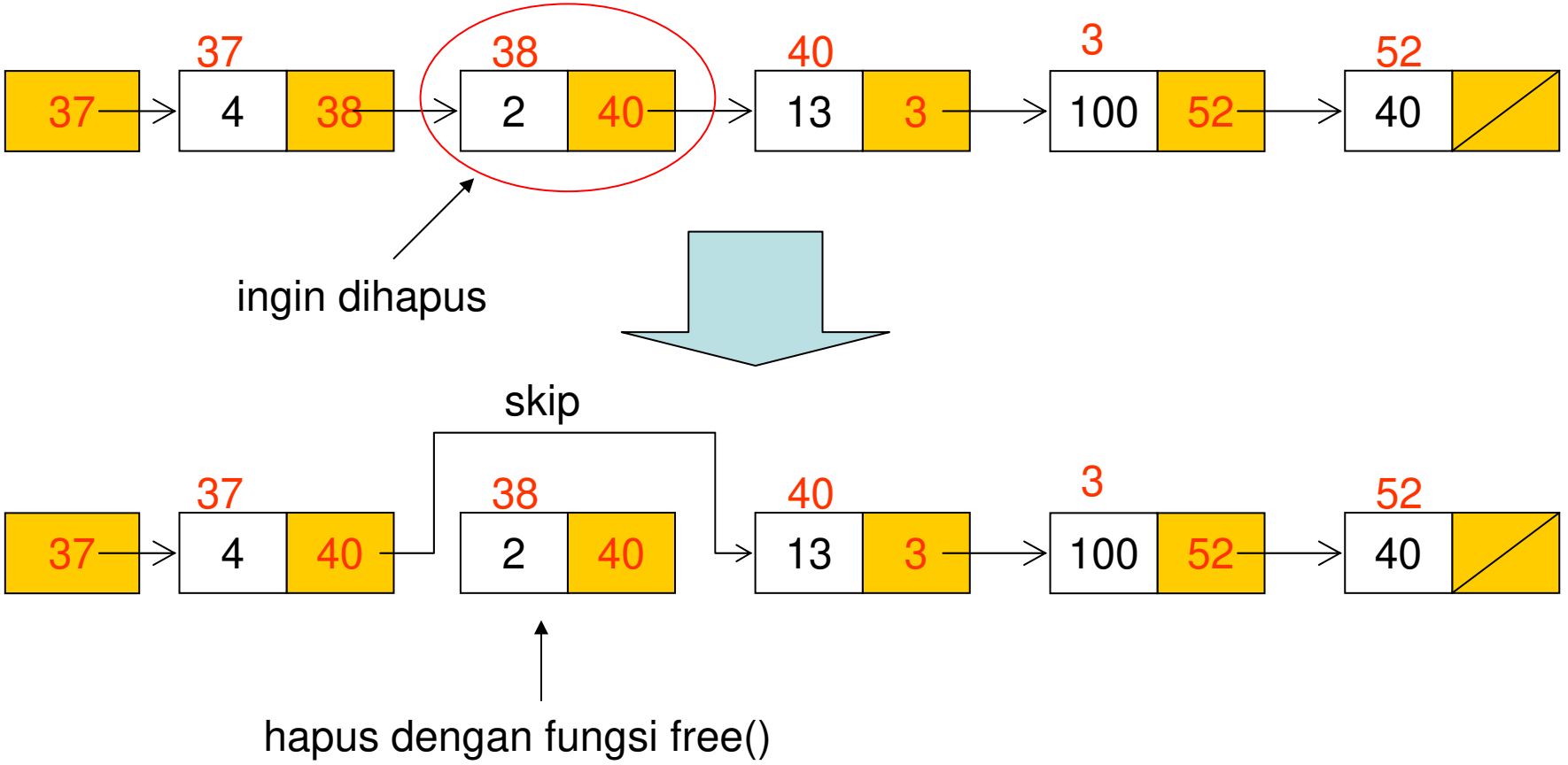
1. Menambahkan data pada linked list
    1. Menambahkan sel baru yang terletak setelah sebuah sel tertentu
    2. Menambahkan sel baru pada posisi terdepan sebuah linked list
2. Menghapus data pada linked list
    1. Menghapus sebuah sel yang terletak setelah sebuah sel tertentu
    2. Menghapus sel pada posisi terdepan sebuah linked list
3. Syarat-syarat pada perbatasan

Menghapus sebuah sel

1. Periksa pointer NULL
2. Mengubah pointer next（skip 1 sel）
3. Mengeluarkan isi sel yang dihapus (optional)
4. Bebaskan memory yang dialokasikan memakai fungsi free()

Catatan penting
Pada linked list, kita bisa menghapus sebuah sel yang terletak pada posisi sesudah sel tertentu. Tetapi kita tidak dapat menghapus sebuah sel yang ditunjuk oleh pointer itu sendiri !

Contoh penghapusan sebuah sel



Cara menghapus sel sesudah sel x

(1)siapkan pointer yang akan menunjuk ke sel yang akan dihapus

```
struct CELL *p;
```

(2)Periksalah apakah sel yang dihapus itu NULL atau tidak

```
if(x->next == NULL)
    fatal_error("sel tidak dapat dihapus karena tidak ada sel lagi sesudahnya");
```

(3)Set-lah agar pointer p menunjuk ke sel yang ingin dihapus

```
p = x->next;
```

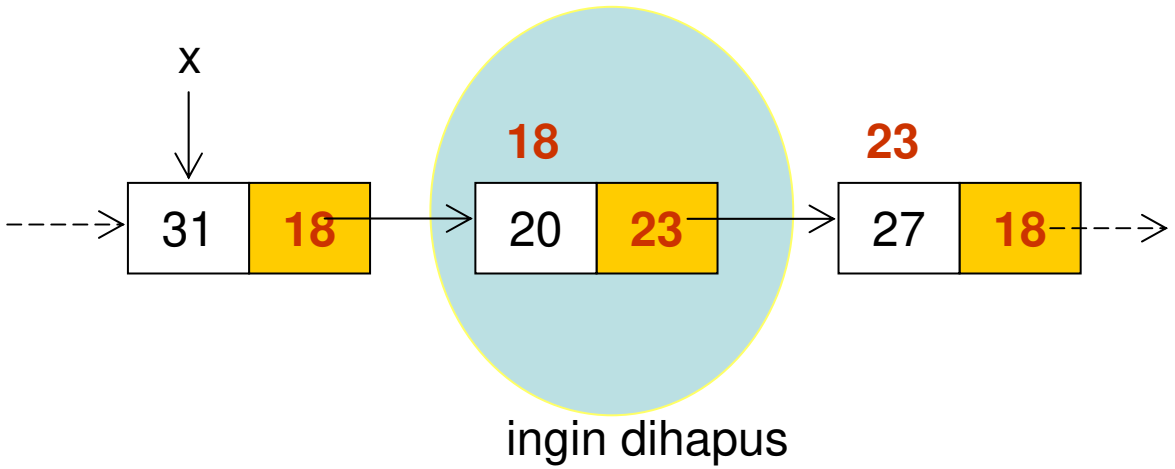
(4)Sesuaikan next dari sel sel x (skip p)

```
x->next = p->next;
```

(5)Print-lah isi sel yang ditunjuk oleh pointer p

(6)Bebaskan memory yang ditunjuk oleh pointer p

```
free(p);
```



Cara menghapus sel sesudah sel x

(1)siapkan pointer yang akan menunjuk ke sel yang akan dihapus

```
struct CELL *p;
```

(2)Periksalah apakah sel yang dihapus itu NULL atau tidak

```
if(x->next == NULL)
    fatal_error("sel tidak dapat dihapus karena tidak ada sel lagi sesudahnya");
```

(3)Set-lah agar pointer p menunjuk ke sel yang ingin dihapus

```
p = x->next;
```

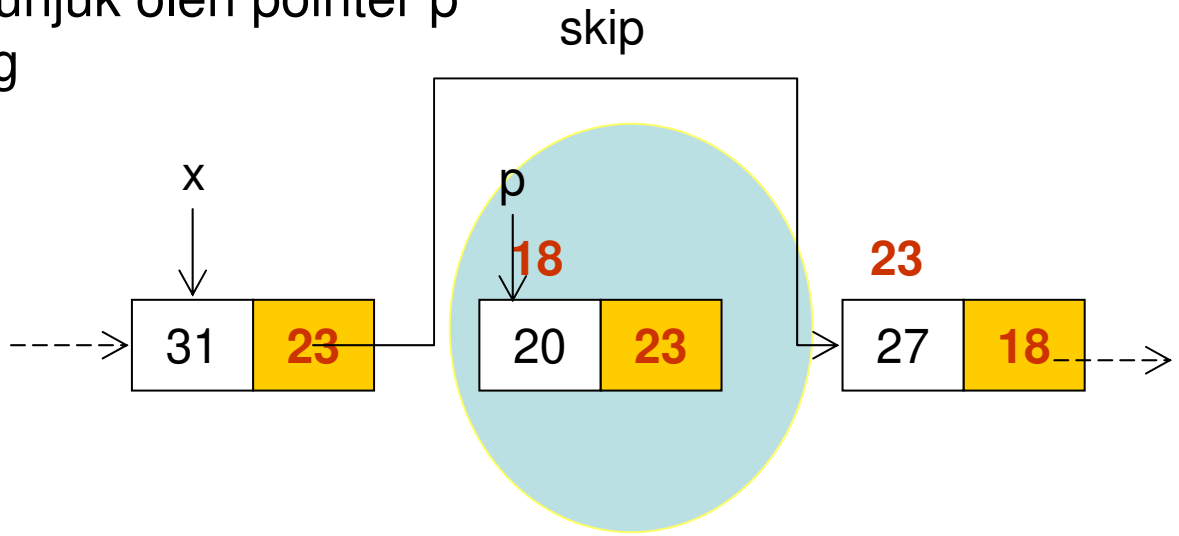
(4)Sesuaikan next dari sel sel x (skip p)

```
x->next = p->next;
```

(5)Print-lah isi sel yang ditunjuk oleh pointer p

(6)Bebaskan memory yang ditunjuk oleh pointer p

```
free(p);
```



Cara menghapus sel sesudah sel x

(1)siapkan pointer yang akan menunjuk ke sel yang akan dihapus

```
struct CELL *p;
```

(2)Periksalah apakah sel yang dihapus itu NULL atau tidak

```
if(x->next == NULL)
    fatal_error("sel tidak dapat dihapus karena tidak ada sel lagi sesudahnya");
```

(3)Set-lah agar pointer p menunjuk ke sel yang ingin dihapus

```
p = x->next;
```

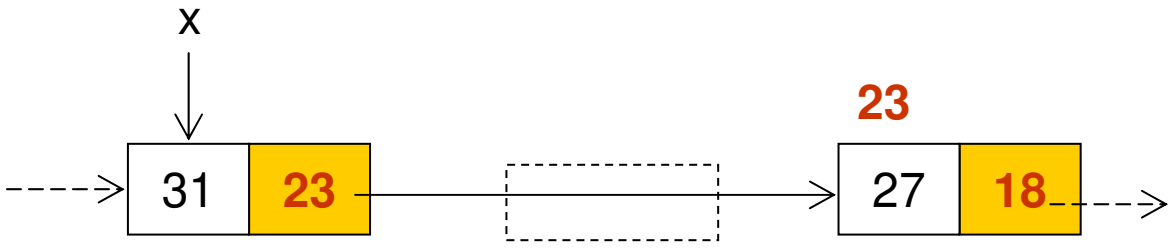
(4)Sesuaikan next dari sel sel x (skip p)

```
x->next = p->next;
```

(5)Print-lah isi sel yang ditunjuk oleh pointer p

(6)Bebaskan memory yang ditunjuk oleh pointer p

```
free(p);
```



dihapus dengan free()

Operasi pada Linked List

1. Menambahkan data pada linked list
    1. Menambahkan sel baru yang terletak setelah sebuah sel tertentu
    2. Menambahkan sel baru pada posisi terdepan sebuah linked list
2. Menghapus data pada linked list
    1. Menghapus sebuah sel yang terletak setelah sebuah sel tertentu
    2. Menghapus sel pada posisi terdepan sebuah linked list
3. Syarat-syarat pada perbatasan

Menghapus sel pada posisi terdepan sebuah linked list

- Memakai cara yang lain dengan cara menghapus sel di tengah-tengah sebuah linked list
- Memanfaatkan pointer header

Menghapus sel pada posisi terdepan sebuah linked list

(1)Siapkan pointer p

```
struct CELL *header, *p;
```

(2)Periksa apakah header NULL atau tidak

```
if(header == NULL)
        fatal_error("List kosong, sehingga penghapusan sel tidak dapat dilakukan");
```

(3)Set-lah address elemen terdepan ke p

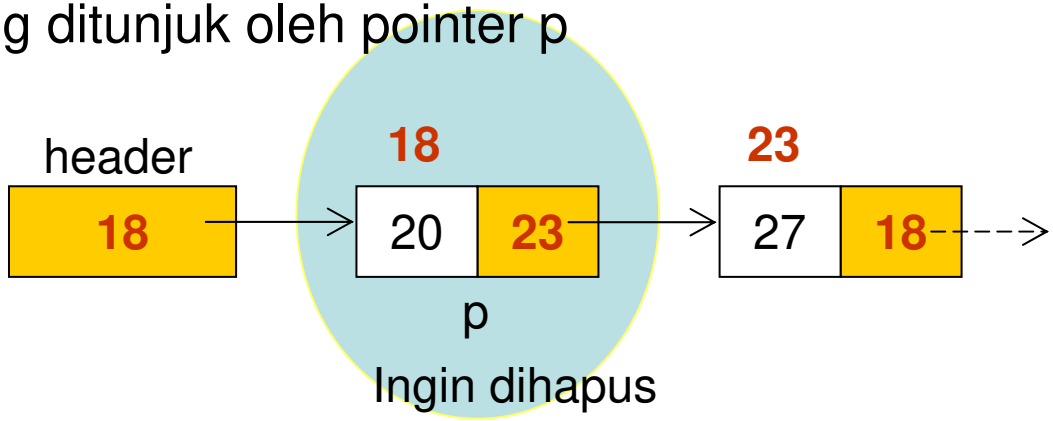
```
p = header;
```

(4)Update-lah pointer header

```
header = p->next;
```

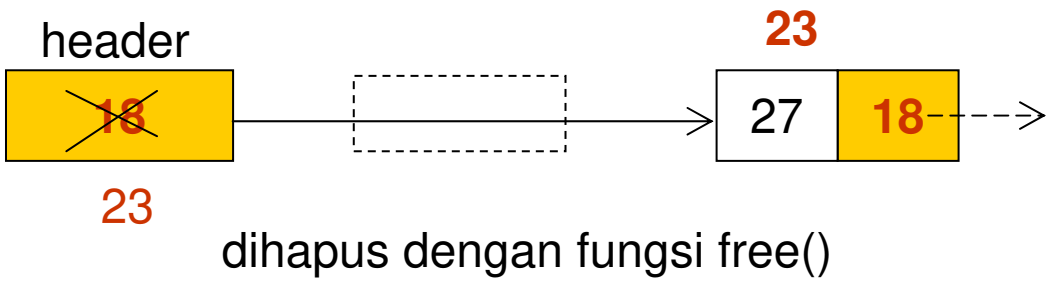
(5)Tampilkan isi sel yang ditunjuk oleh p (optional)

(6) Bebaskan memory yang ditunjuk oleh pointer p



Menghapus sel pada posisi terdepan sebuah linked list

(1)Siapkan pointer p
struct CELL *header, *p;
(2)Periksa apakah header NULL atau tidak
if(header == NULL)
fatal_error(”List kosong, sehingga penghapusan sel tidak dapat dilakukan”);
(3)Set-lah address elemen terdepan ke p
p = header;
(4)Update-lah pointer header
header = p->next;
(5)Tampilkan isi sel yang ditunjuk oleh p (optional)
(6) Bebaskan memory yang ditunjuk oleh pointer p



Menghapus sel pada posisi terdepan sebuah linked list

- Memakai cara sama seperti saat menghapus sel pada tengah-tengah sebuah list
- Header bukan pointer, melainkan sel ke-0

Menghapus sel pada posisi terdepan sebuah linked list (2)

(1)Header dideklarasikan sebagai sebuah sel.

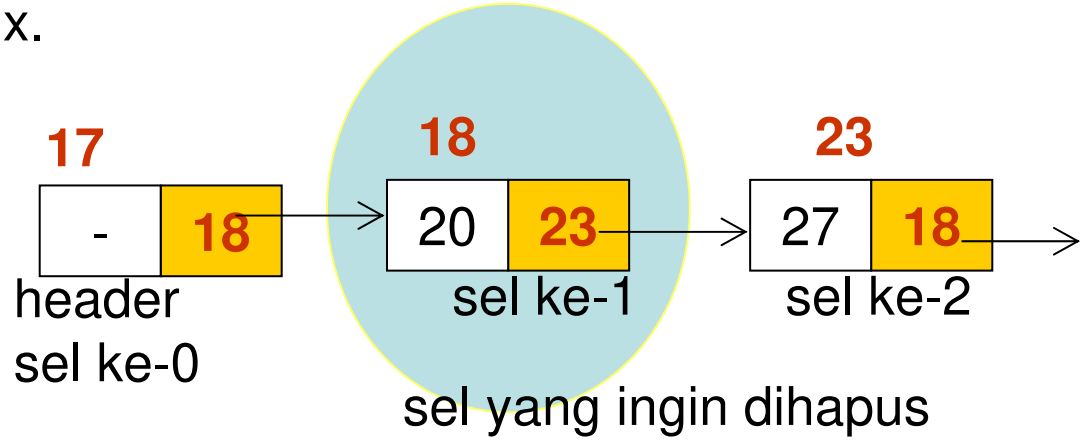
p dideklarasikan sebagai pointer yang menunjuk ke sel yang ingin dihapus

struct CELL header, *p;

(2)Substitusikan address header ke pointer x

x=&header

(3)Selanjutnya pergunakan cara yang sama dengan cara menghapus sel yang berada di tengah-tengah sebuah list. Yaitu menghapus sel yang terletak sesudah sel yang ditunjuk oleh pointer x.



Menghapus sel pada posisi terdepan sebuah linked list (2)

(1)Header dideklarasikan sebagai sebuah sel.

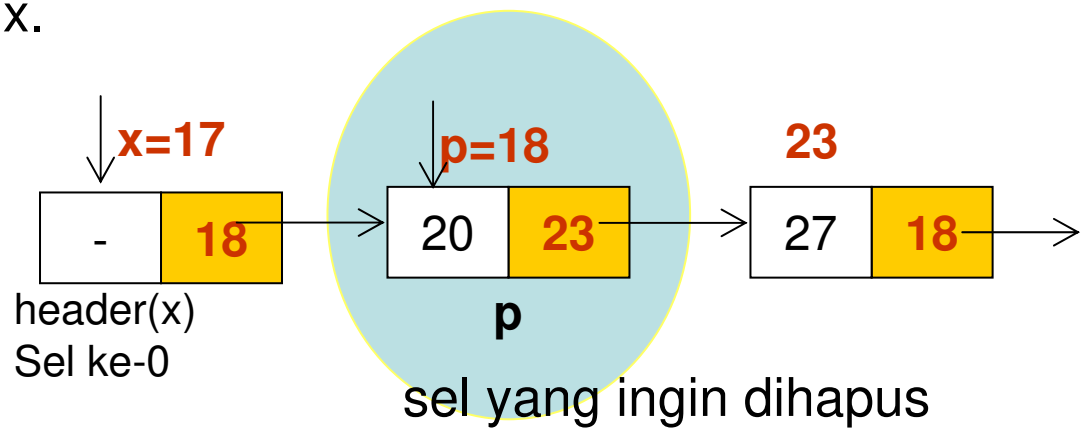
p dideklarasikan sebagai pointer yang menunjuk ke sel yang ingin dihapus

struct CELL header, *p;

(2)Substitusikan address header ke pointer x

x=&header

(3)Selanjutnya pergunakan cara yang sama dengan cara menghapus sel yang berada di tengah-tengah sebuah list. Yaitu menghapus sel yang terletak sesudah sel yang ditunjuk oleh pointer x.



Menghapus sel pada posisi terdepan sebuah linked list (2)

(1)Header dideklarasikan sebagai sebuah sel.

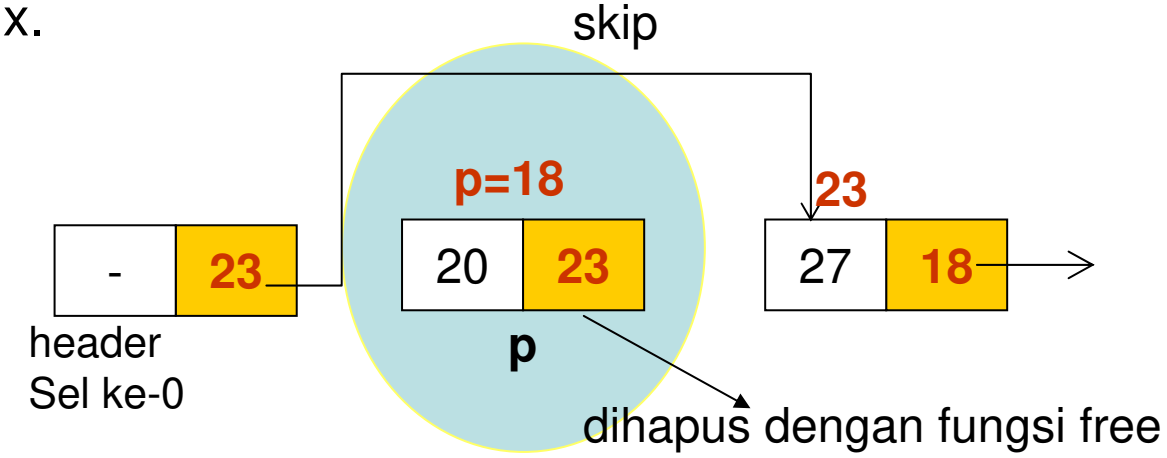
p dideklarasikan sebagai pointer yang menunjuk ke sel yang ingin dihapus

struct CELL header, *p;

(2)Substitusikan address header ke pointer x

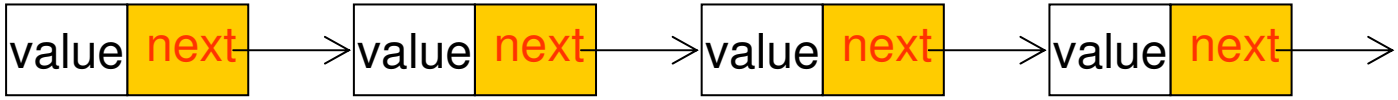
x=&header

(3)Selanjutnya pergunakan cara yang sama dengan cara menghapus sel yang berada di tengah-tengah sebuah list. Yaitu menghapus sel yang terletak sesudah sel yang ditunjuk oleh pointer x.



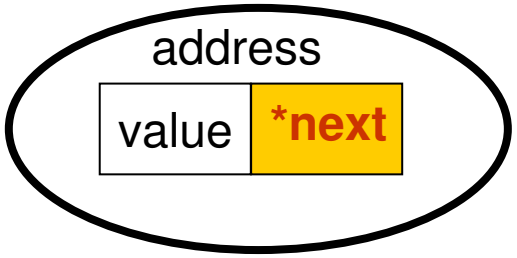
Rangkuman

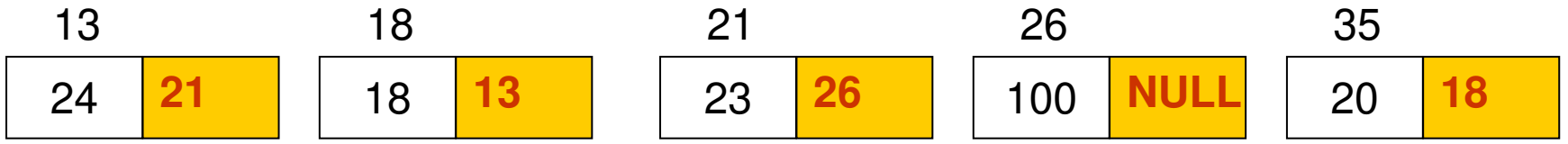
- Linked list terdiri dari komponen-komponen berikut
  - value
  - pointer ke sel berikutnya
- Data yang tersimpan (value) pada linked-list diakses dengan cara sekuensial. Untuk melihat value dari suatu sel yang berada di tengah list, kita harus mengikuti alur list, mulai dari sel pertama dan seterusnya sampai bertemu dengan sel yang value-nya ingin ditampilkan.
- Penambahan sel, penghapusan sel dapat dilakukan dengan cara mengubah pointer
- Ada dua cara operasi penambahan dan penghapusan sel yang terdepan. Yang lebih mudah adalah dengan memakai header bukan sebagai pointer, melainkan sebagai sel ke-0



Latihan

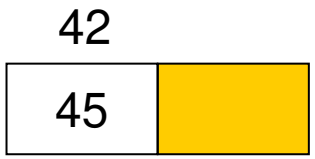
(1) Pada saat header = 35, gambarkan linked list yang dibuat dari sel-sel dibawah.





(2) Gambarkan kondisi linked-list saat sel dengan value 24 dihapus

(3) Gambarkan kondisi linked-list saat sel berikut ditambahkan Sesudah sel dengan value 23

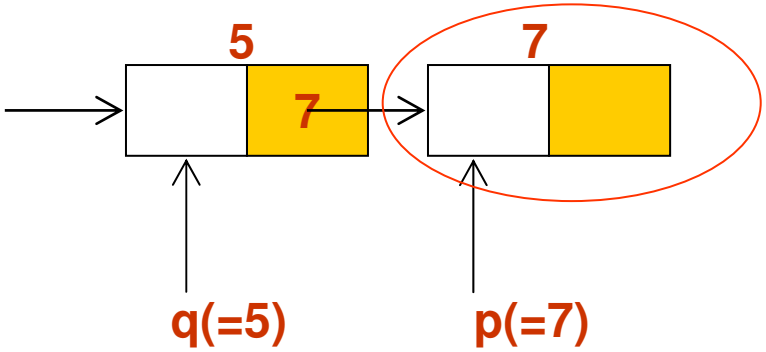


Operasi pada Linked List

1. Menambahkan data pada linked list
    1. Menambahkan sel baru yang terletak setelah sebuah sel tertentu
    2. Menambahkan sel baru pada posisi terdepan sebuah linked list
2. Menghapus data pada linked list
    1. Menghapus sebuah sel yang terletak setelah sebuah sel tertentu
    2. Menghapus sel pada posisi terdepan sebuah linked list
3. Syarat-syarat pada perbatasan

Syarat-syarat pada perbatasan

- Hal-hal yang harus diperhatikan
  - Bagaimana menghandle sel terdepan dan terbelakang
  - Proses saat list kosong
- Program linkedlist.c (Angka diurutkan, nilainya semakin tinggi)
  - Sel yang jadi pusat operasi (pointer p)
  - Pointer yang menunjuk ke address sel sebelumnya (pointer q)



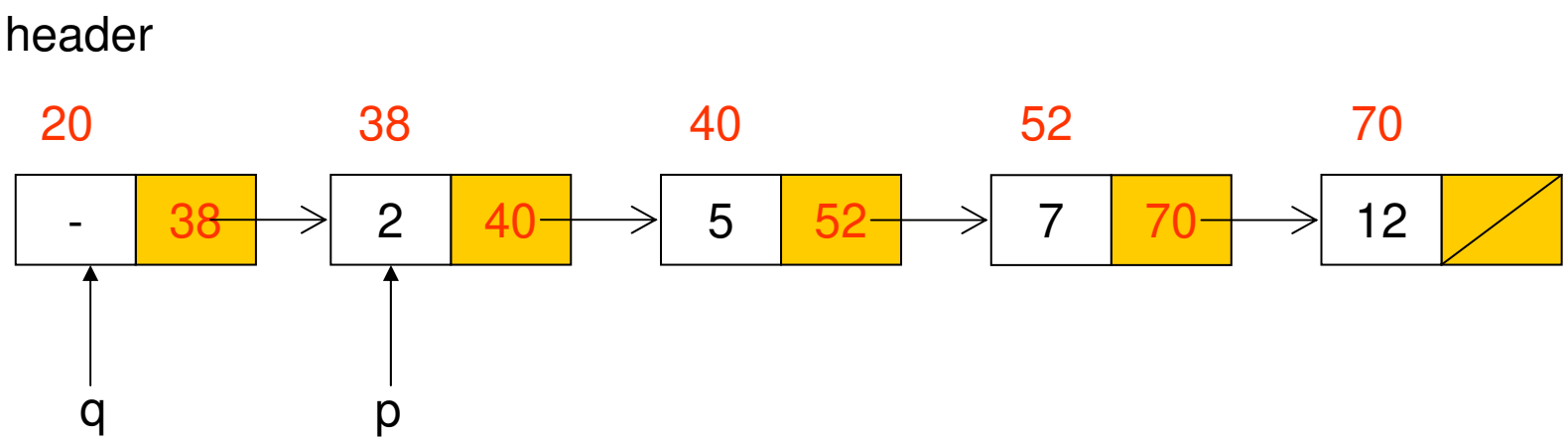
Contoh program Linked List

- Downloadlah program linkedlist.c yang disediakan di situs kuliah http://asnugroho.net/lecture/ds.html
- Pelajari isinya dan selesaikan fungsi cell_delete

Fungsi insert

```
struct CELL {
        struct CELL *next;
        int         value;
};
typedef struct CELL cell;
cell header;
cell_insert(int a)
{
        cell *p, *q, *new_cell;

/* mencari posisi dimana sel harus dimasukkan */
p = header.next;
q = &header;
while (p != NULL  && a > p-> value) {
        q = p;
        p  = p->next;
}
```

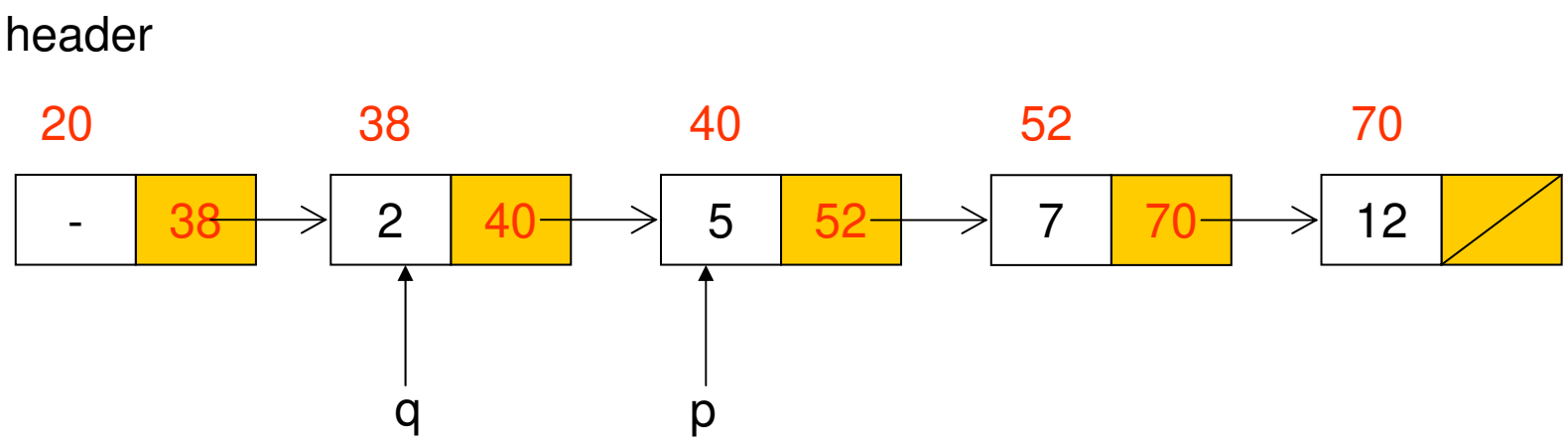
Akses ke tiap sel satu persatu



p=header.next = 38
q=&header = 20

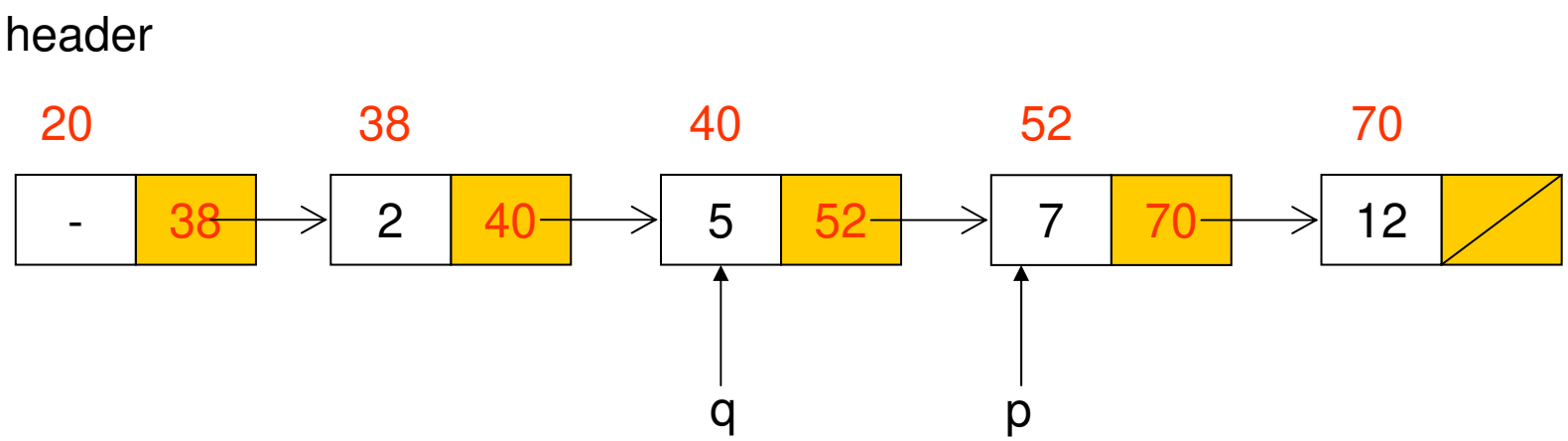
```
while (p != NULL) {
        q = p;
        p  = p->next;
}
```

Akses ke tiap sel satu persatu



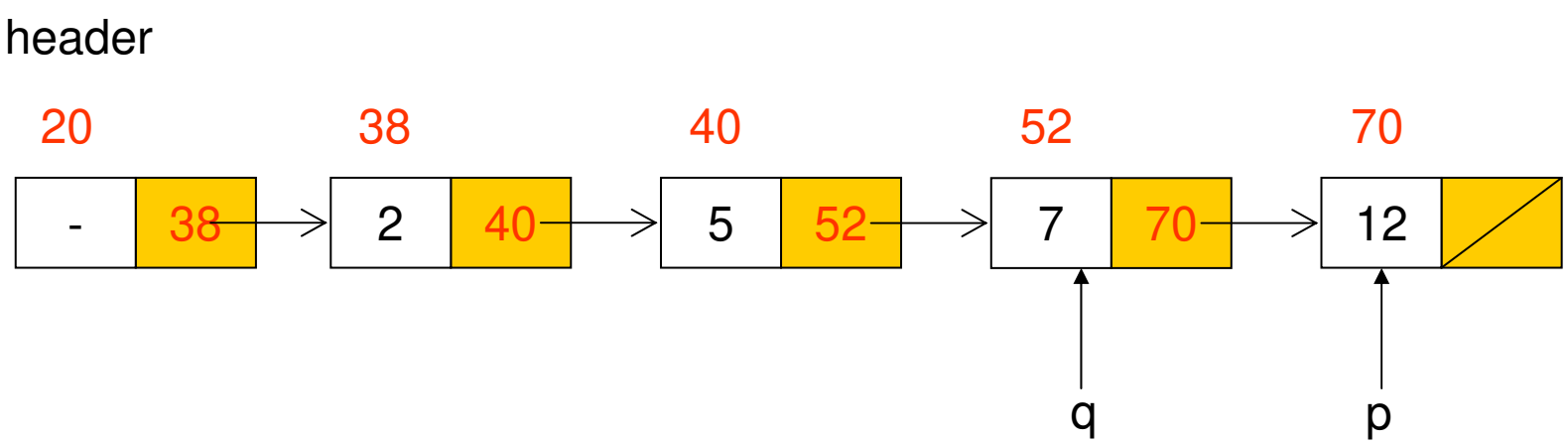

```
while (p != NULL ) {
        q = p;  q = 38
        p  = p->next;  p = 40
}
```

Akses ke tiap sel satu persatu



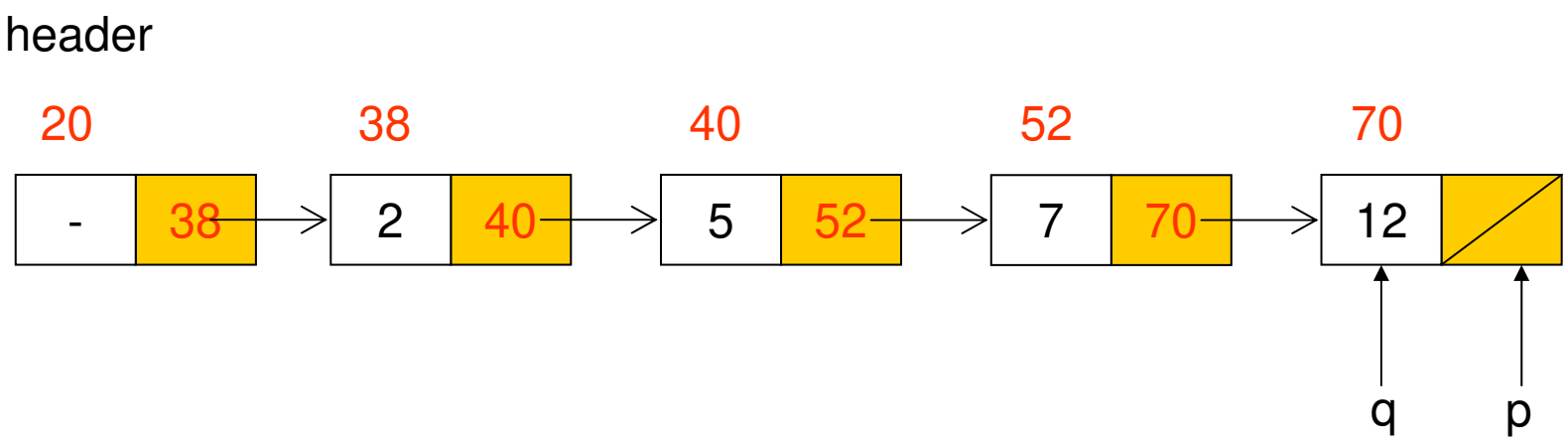

```
while (p != NULL) {
        q = p;  q = 40
        p  = p->next;  p = 52
}
```

Akses ke tiap sel satu persatu



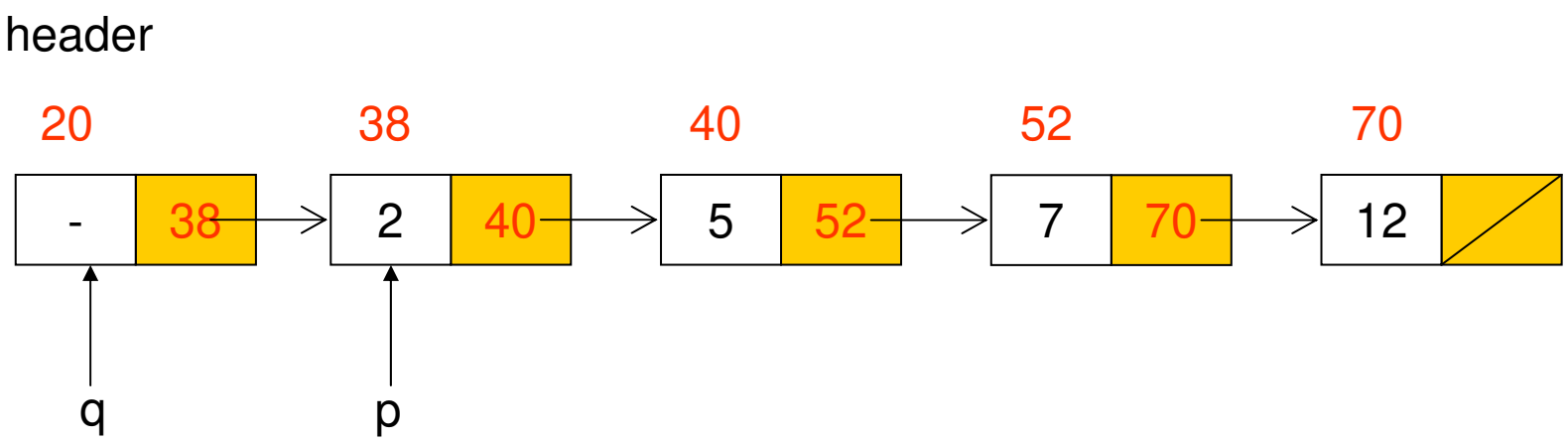

```
while (p != NULL) {
        q = p;  q = 52
        p  = p->next;  p = 70
}
```

Akses ke tiap sel satu persatu




```
while (p != NULL) {
        q = p;  q = 70
        p  = p->next;  p = NULL
}
```

Ingin memasukkan 4(a=4)

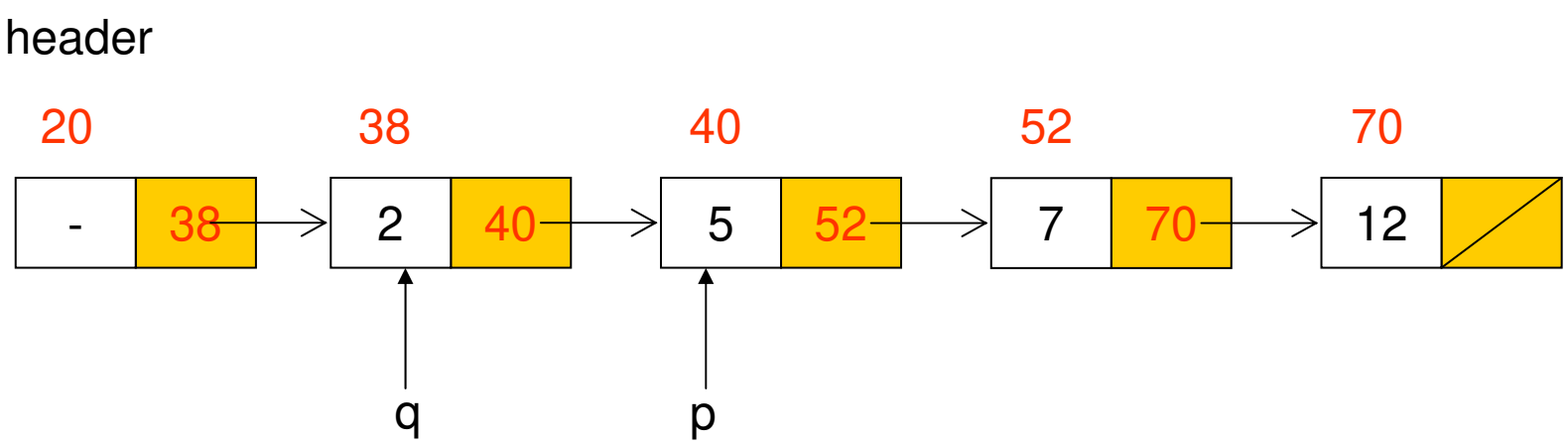


p=header.next = 38
q=&header = 20

p->value=2

```
while ((p != NULL ) && (p-> value < a)) {
        q = p;
        p  = p->next;
}
```

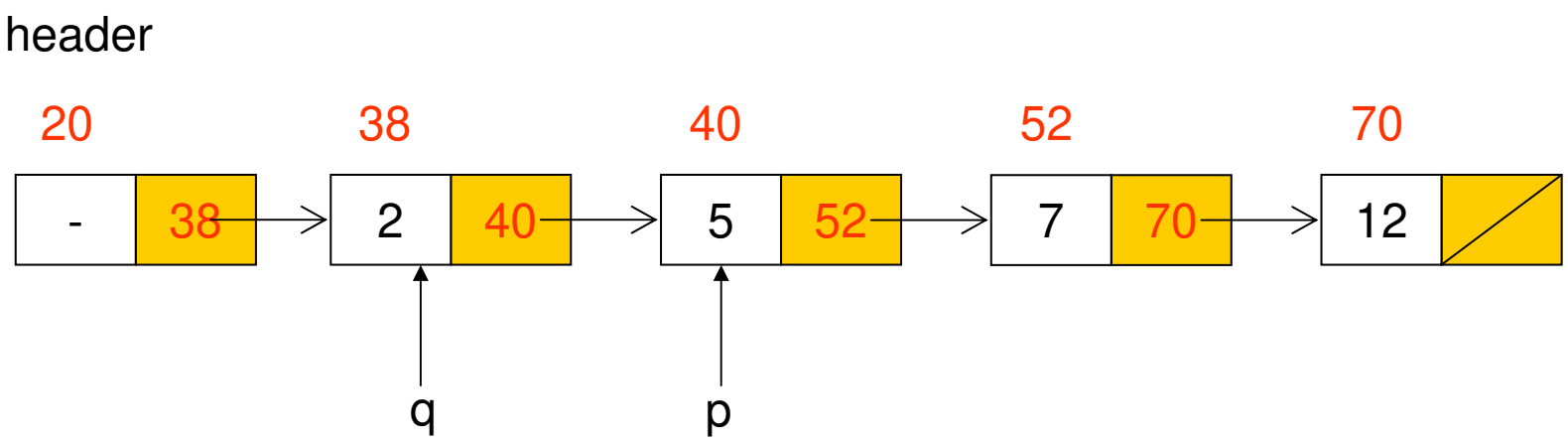
Ingin memasukkan 4(a=4)



p=header.next = 38
q=&header = 20

```
while ((p != NULL ) && (p-> value < a)) {
        q = p;  q = 38
        p  = p->next;  p = 40
}
```

Ingin memasukkan 4(a=4)



p=40
q=38

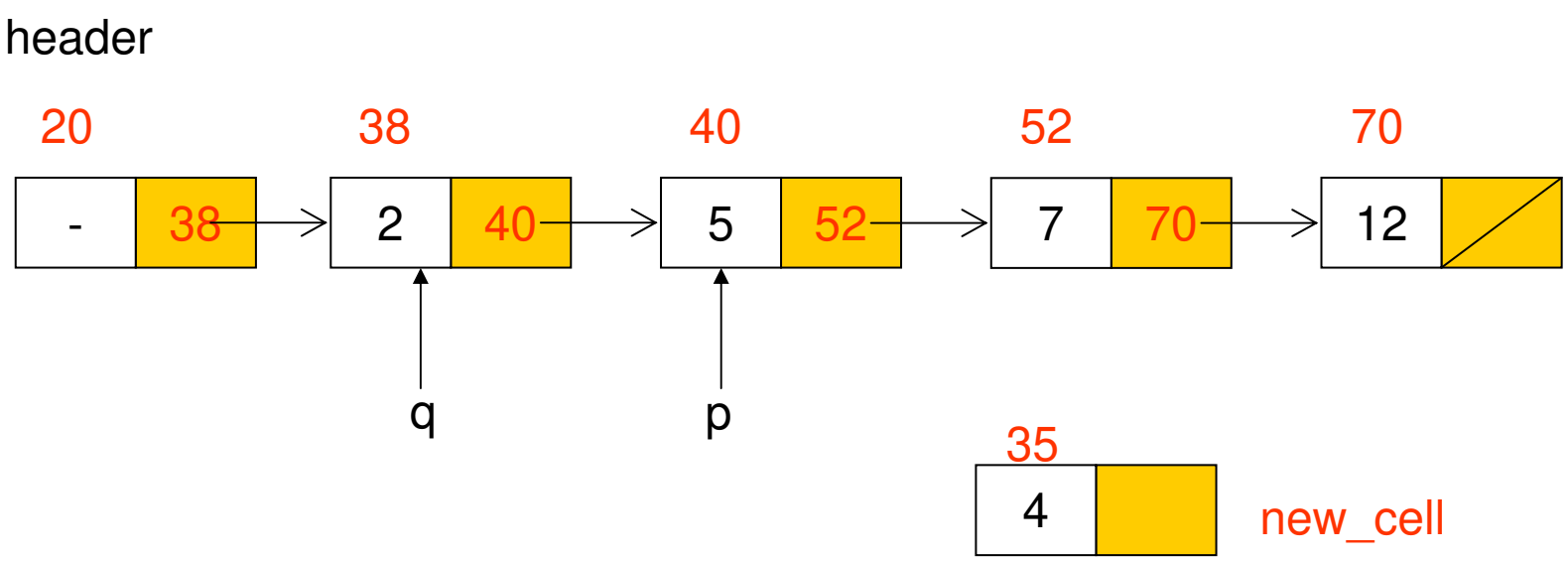
p->value=5

```
while ((p != NULL ) && (p-> value < a)) {
        q = p;
        p  = p->next;
}
    /* Masukkan sel baru */
```

Keluar dari loop

Masukkan 4 sebelum p(value 5)

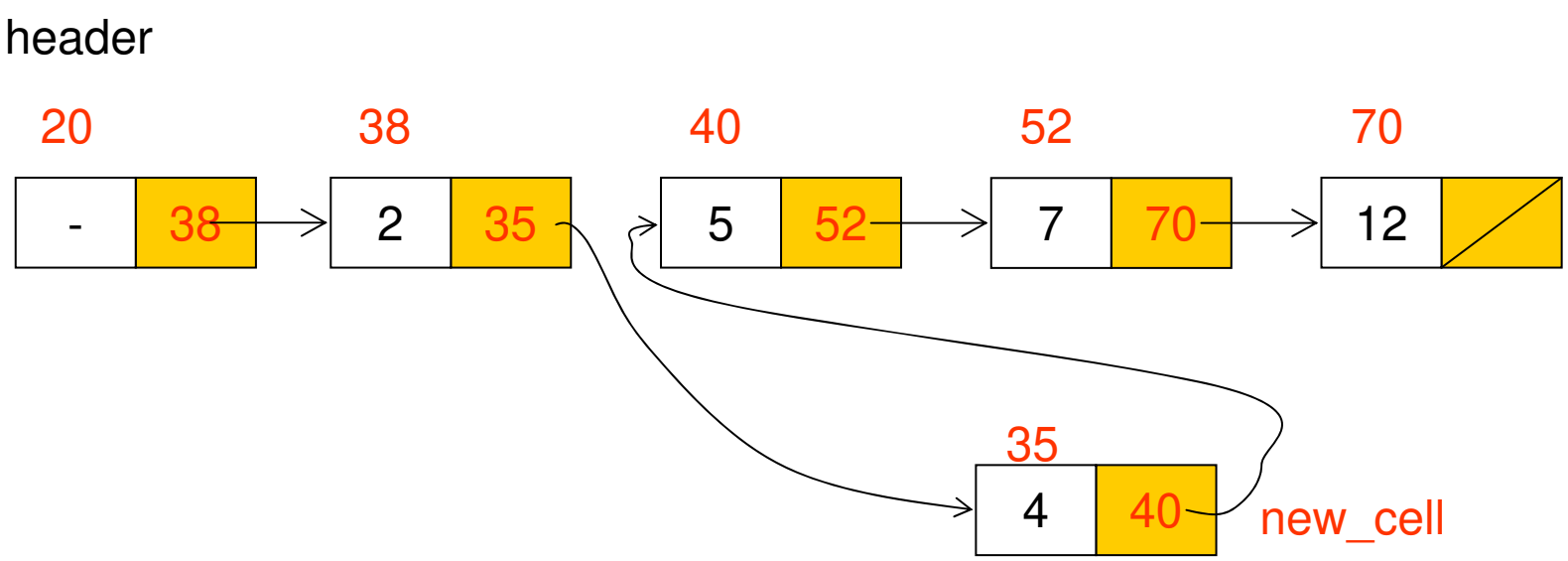
Ingin memasukkan 4(a=4)




```
/* Menambahkan sel baru  */
If ((new_cell=(cell*)malloc(sizeof(cell)))==NULL)
            fatal_error("memory tidak cukup");
new_cell->next=p;    ←di sini new_cell->next diisi dengan 40
q->next=new_cell;    ←di sini q->next diisi dengan 35
new_cell->value=a;
}
```

Memasukkan new_cell (value 4) sebelum p(value 5) = memasukkan sel baru sesudah q. Untuk itulah dalam proses ini diperlukan 2 pointer p dan q !

Ingin memasukkan 4(a=4)




```
/* Menambahkan sel baru  */
If ((new_cell=(cell*)malloc(sizeof(cell)))==NULL)
            fatal_error("memory tidak cukup");
new_cell->next=p;    ←di sini new_cell->next diisi dengan 40
q->next=new_cell;    ←di sini q->next diisi dengan 35
new_cell->value=a;
}
```

Memasukkan new_cell (value 4) sebelum p(value 5) = memasukkan sel baru sesudah q. Untuk itulah dalam proses ini diperlukan 2 pointer p dan q !

Latihan

- Selesaikan fungsi cell_delete pada program linkedlist.c, untuk menghapus data yang telah ditambahkan pada sebuah list
- Jelaskan secara singkat program yang dibuat (komentar)

Hint

Untuk menyelesaikan cell_delete buatlah program dengan urutan sbb.

① Periksalah, apakah list itu kosong atau tidak. Jika tidak kosong, lanjutkan ke step ②. Jika list itu kosong, tampilkan error message, dan kembalilah ke fungsi main

② Akseslah sel pada list itu satu persatu dengan pointer p dan q untuk mencari data yang akan dihapus (yaitu a)

③ Jika sampai akhir ternyata sel yang dihapus tidak ditemukan, tampilkan “Sel yang dihapus tidak ditemukan”, dan kembali ke fungsi main

④ Jika sel yang ingin dihapus ditemukan, hapuslah sel itu dan kembali ke fungsi main

```
void  cell_delete(int a)
{
  cell *p, *q;

  p=header.next;       q=&header;
```

① Periksalah, apakah list itu kosong atau tidak. Jika tidak kosong, lanjutkan ke step ②. Jika list itu kosong, tampilkan error message, dan kembalilah ke fungsi main

② Akseslah sel pada list itu satu persatu dengan pointer p dan q untuk mencari data yang akan dihapus (yaitu a)

③ Jika sampai akhir ternyata sel yang dihapus tidak ditemukan, tampilkan “Sel yang dihapus tidak ditemukan”, dan kembali ke fungsi main

④ Jika sel yang ingin dihapus ditemukan, hapuslah sel itu dan kembali ke fungsi main

```
}
```

PR

- Rancanglah bagaimana membuat program stack, tetapi tidak memakai array, melainkan memakai linked-list（Dengan kata lain: Selesaikan fungsi push, pop dan stack_print pada linkedliststack.c）
- Hint：

  Penambahan dan penghapusan elemen pada stack harus bisa dijalankan pada O(1). Karena itu penambahan dan penghapusan data harus dilakukan pada sel yang terdepan

Beberapa Jenis Struktur Data

1. Array
    1. Linear List
    2. Stack
    3. Queue

2. List
    1. Linked List
    2. Circular List
    3. Bidirectional List
    4. Multi list structure

3. Tree Structure

Beberapa Jenis Struktur Data

1. Array
    1. Linear List
    2. Stack
    3. Queue

2. List
    1. Linked List
    2. Circular List
    3. Bidirectional List
    4. Multi list structure

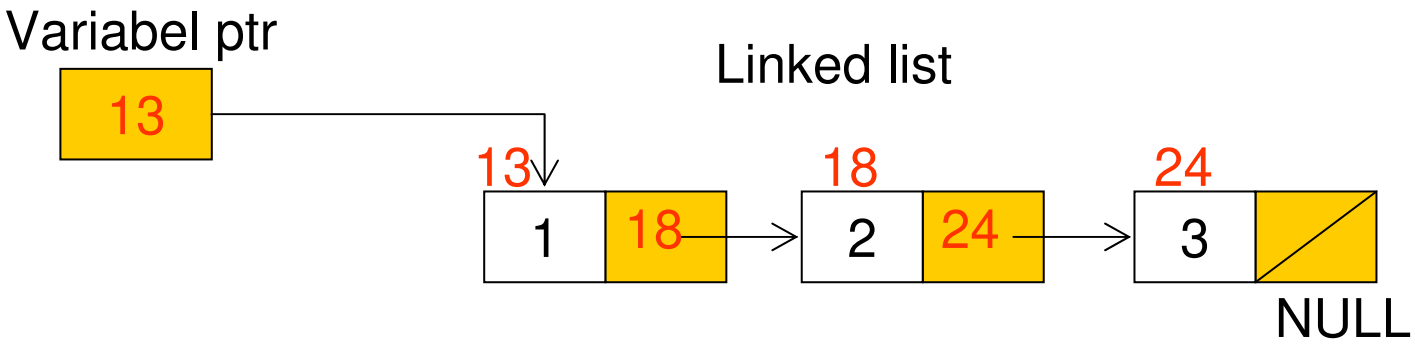
3. Tree Structure

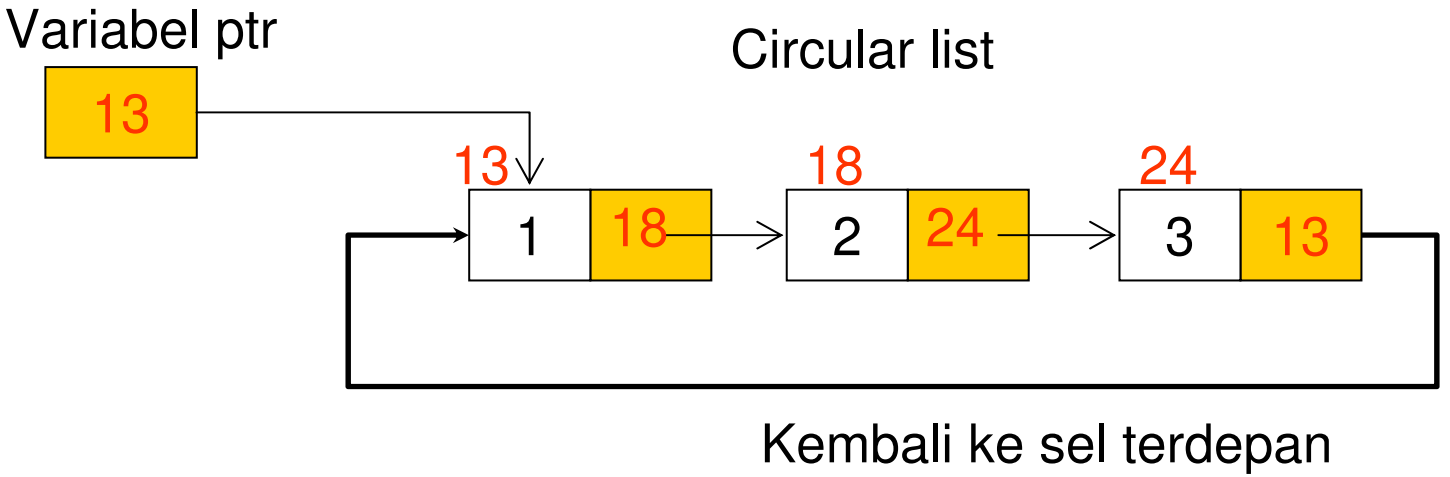
Circular List

Apakah Circular List itu ?

- Semua sel dalam list disambungkan dengan pointer, dan sel yang terakhir disambungkan dengan sel pertama
- Struktur data berbentuk cincin
- Tidak ada sel pertama maupun sel terakhir dalam list, karena setiap sel disambungkan oleh pointer sehingga berbentuk cincin.

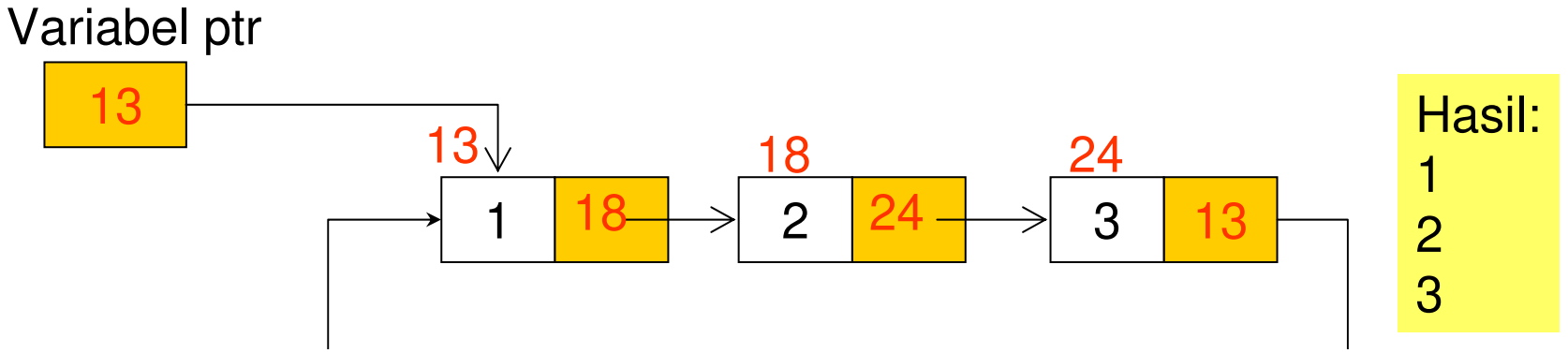
Linked List vs Circular List





Cara akses tiap sel pada Circular List

```
struct CELL *ptr, *p;
if(ptr != NULL) {
    p = ptr;
    do {
        printf("%d\n",p->value)
        /* tampilkan value pada sel yang ditunjuk oleh p */
        p = p->next;
    } while (p != ptr);
    // Setelah semua bagian dalam loop dieksekusi,
    // evaluasilah apakah p!= ptr
}
```



Cara ini tidak benar !

```
struct CELL *ptr, *p;
if(ptr != NULL) {
    p = ptr;
     while (p != ptr) {
            printf("%d\n",p->value)
            p = p->next;
    }
}
```

Cara ini tidak benar. Bagian dalam loop tidak akan dijalankan karena sebelum masuk loop syarat p!=ptr tidak akan pernah terpenuhi

Circular List memakai head

```
struct CELL *ptr, *p;
if(ptr != NULL) {
    p = ptr;
    do {
        printf("%d\n",p->value
        p = p->next;
    } while (p != ptr);
}
```

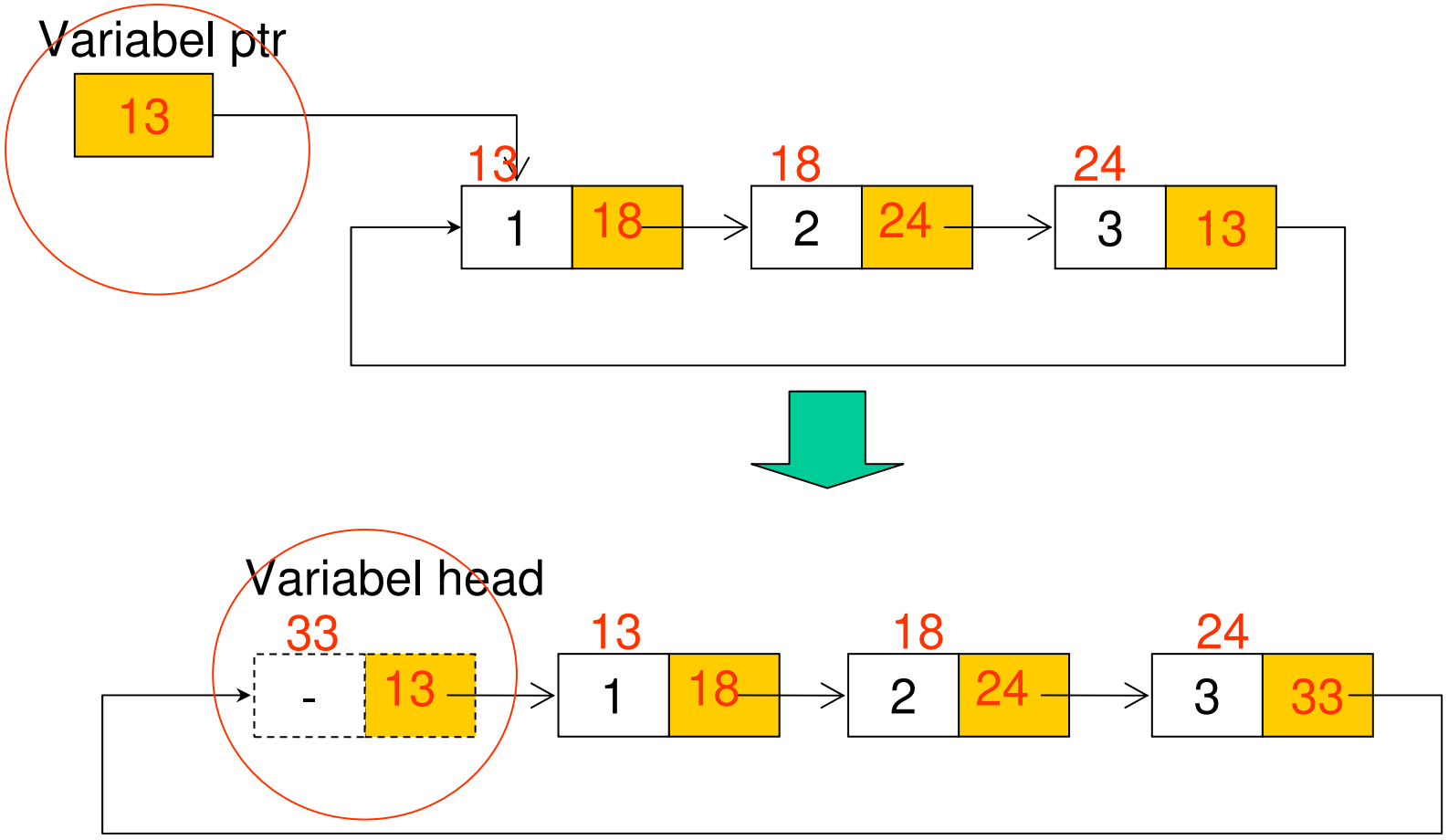
Saat ptr==NULL
berarti list kosong

Dua syarat ini
mungkinkah disatukan ?

Saat p==ptr
Berarti sudah sampai di akhir list

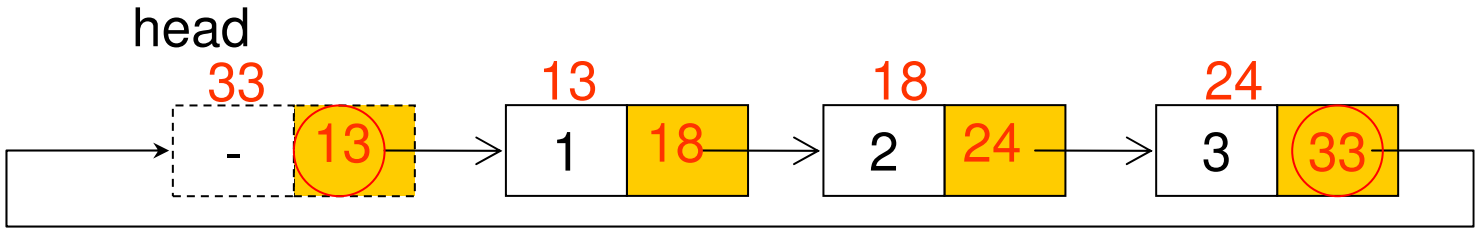
memakai head

Circular List memakai head

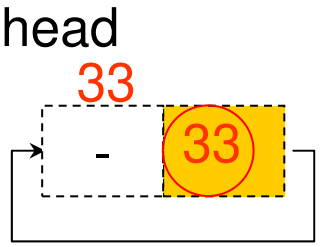


Circular List memakai head

struct CELL head;



Circular List dengan 3 sel

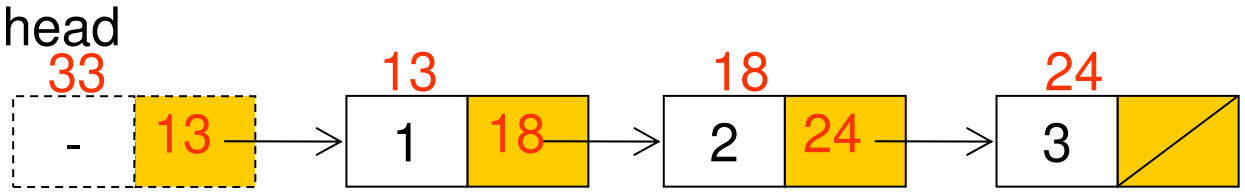


Circular List dalam kondisi kosong

head.next== & head

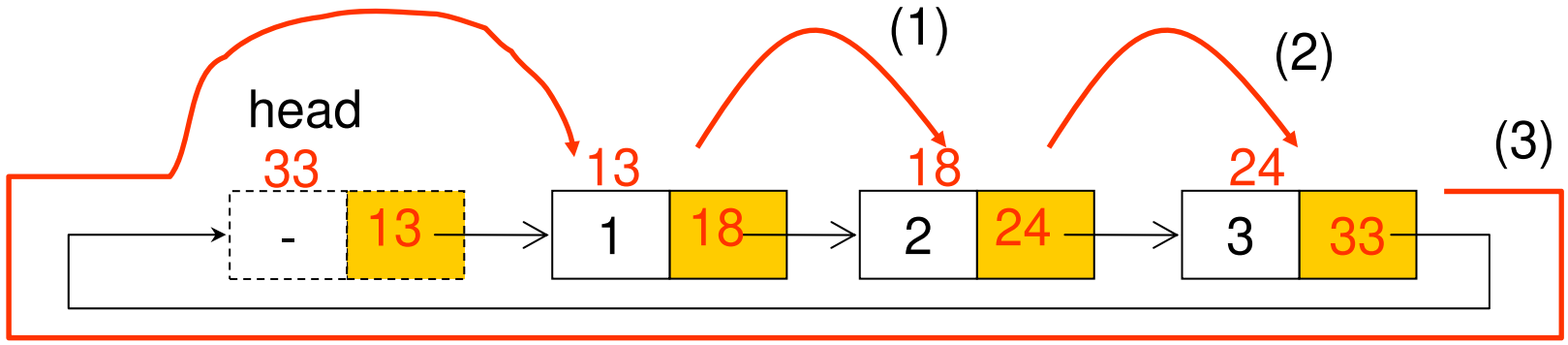
（pointer next dari head menunjuk ke head itu sendiri)

Catatan: bandingkan Circular List di atas dengan linked list berikut



Circular List memakai head

```
struct CELL head, *p;
for(p = head.next; p!= &head; p = p->next)  // head tidak di-lompati (skip)
{
    perlu ditambahkan bagian untuk melewati  “head” agar sel tsb.
    tidak diproses
}
```



Beberapa Jenis Struktur Data

1. Array
    1. Linear List
    2. Stack
    3. Queue

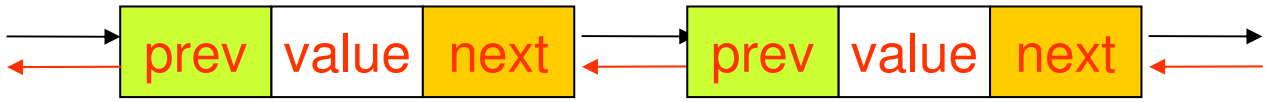
2. List
    1. Linked List
    2. Circular List
    3. Bidirectional List
    4. Multi list structure

3. Tree Structure

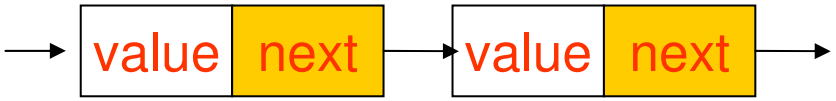
Bidirectional List

Apakah bidirectional list itu ?

- Semua sel yang terdapat pada list disambungkan dengan pointer, sedangkan tiap sel memiliki TIGA komponen : value, pointer ke sel sebelumnya dan pointer ke sel berikutnya
- Dengan memiliki dua buah pointer ini, maka bidirectional list dapat diakses dengan DUA arah : ke arah depan dan ke belakang

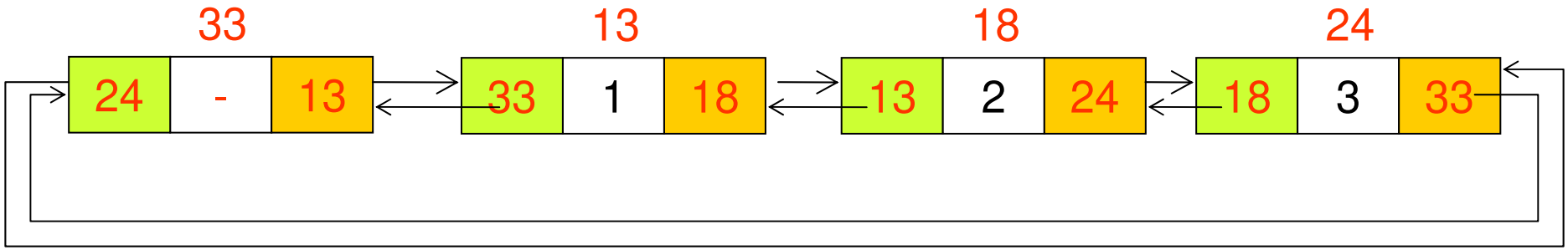


Bidirectional List

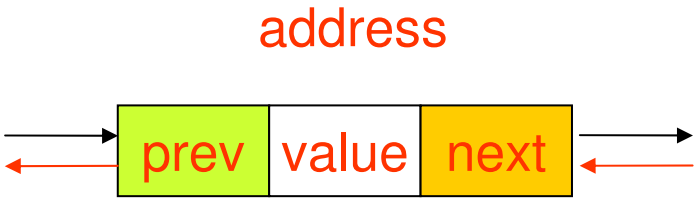


Linked List

Bidirectional List




```
struct CELL {
        struct CELL *prev;
        struct CELL *next;
        MYDATA value;
};
```



Misalnya int, char, float, long, double, dsb

Kelebihan dan kelemahan

Kelebihan

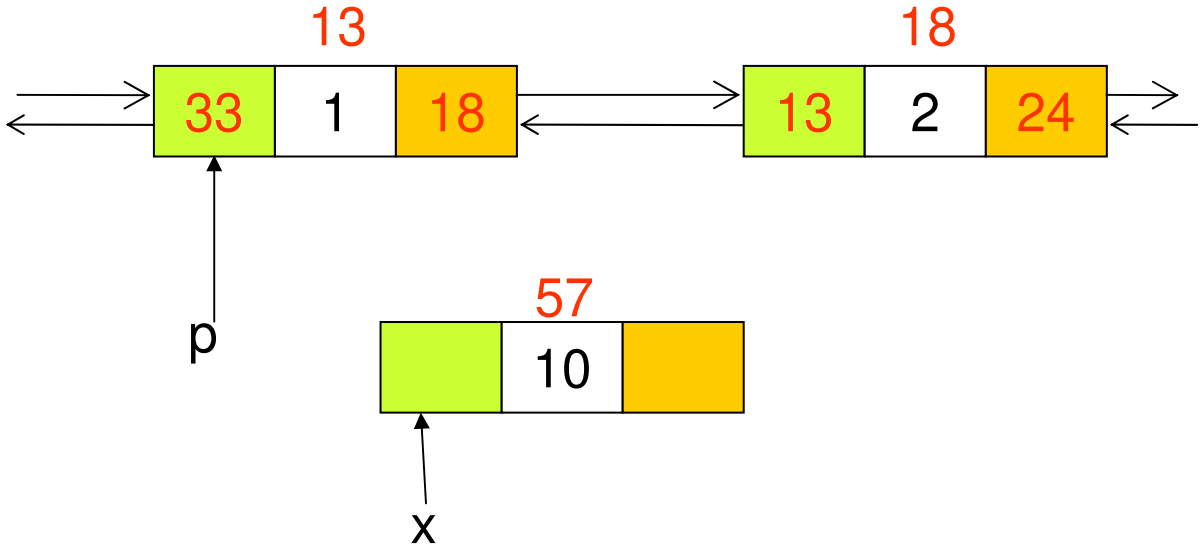
1. List bisa diakses dengan dua arah : ke depan maupun ke belakang
2. Penambahan dan penghapusan data menjadi mudah, karena pada tipe ini kita dapat menambahkan sel baru sesudah maupun sebelum sebuah sel. Demikian juga dalam hal menghapus sel.

Kelemahan

1. Sebuah data memiliki dua buah pointer, sehingga memerlukan space yang lebih besar

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

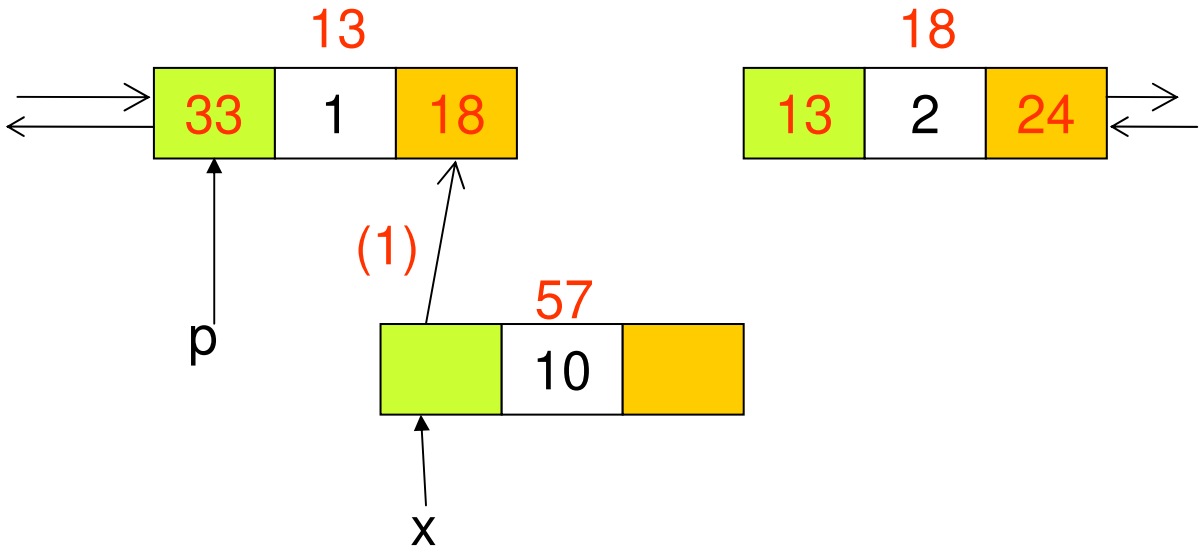



```
x->prev=p;              (1)
x->next=p->next;        (2)
p->next->prev=x;        (3)
p->next=x;              (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

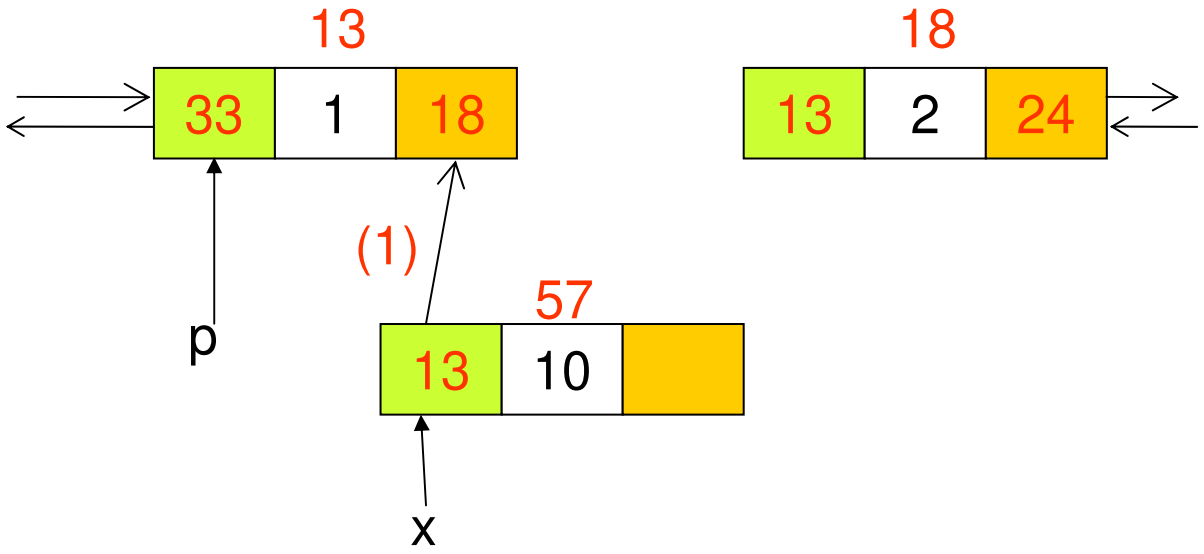



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p



x–>prev=p; (1)

x–>next=p–>next; (2)

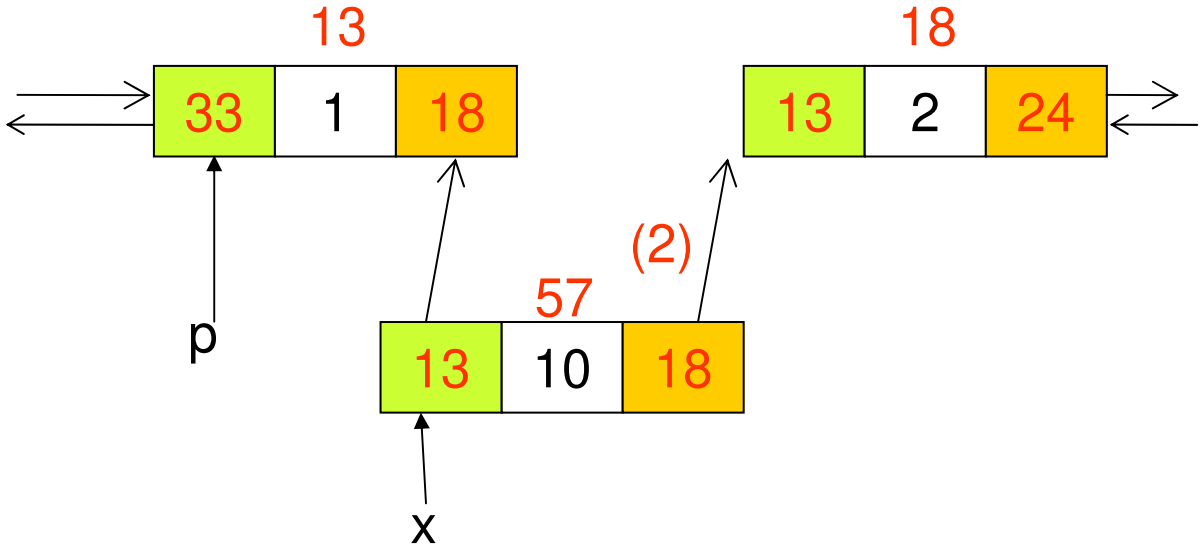
p–>next–>prev=x; (3)

p–>next=x; (4)

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

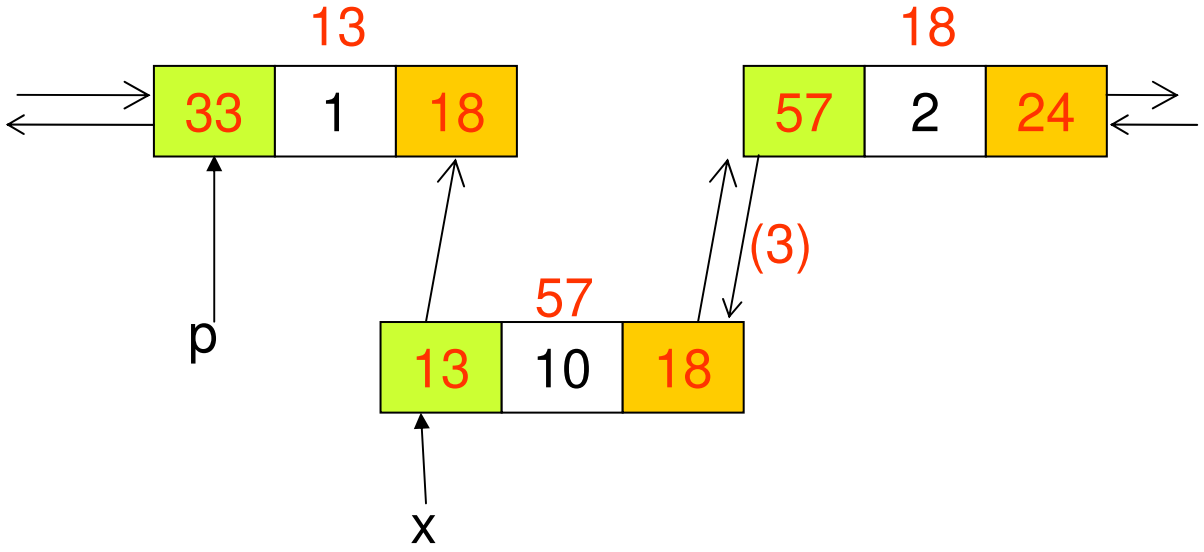



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

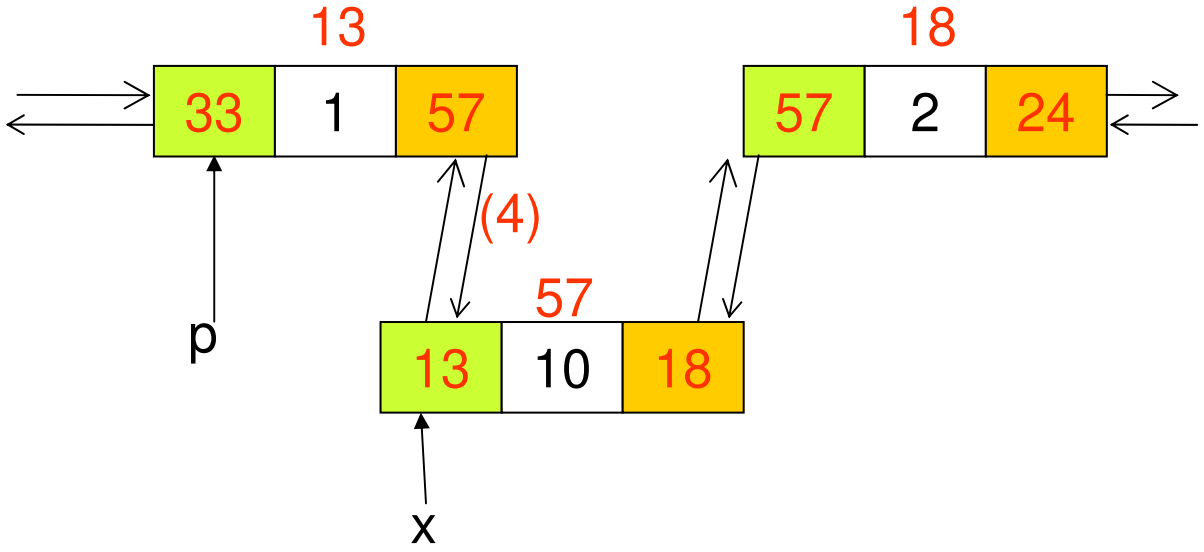



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p



x–>prev=p; (1)

x–>next=p–>next; (2)

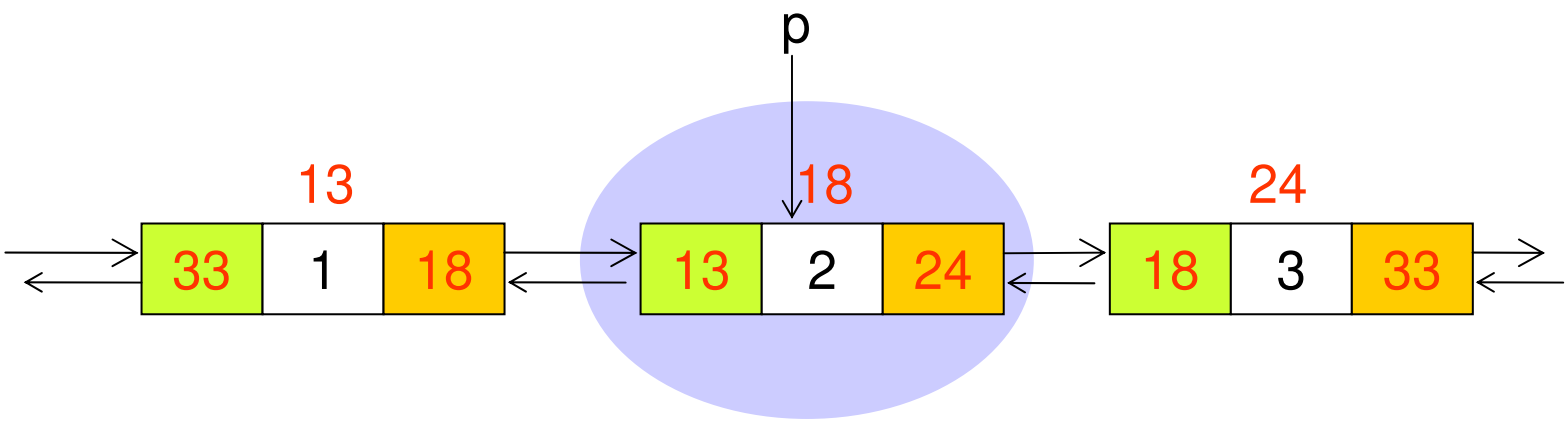
p–>next–>prev=x; (3)

p–>next=x; (4)

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p

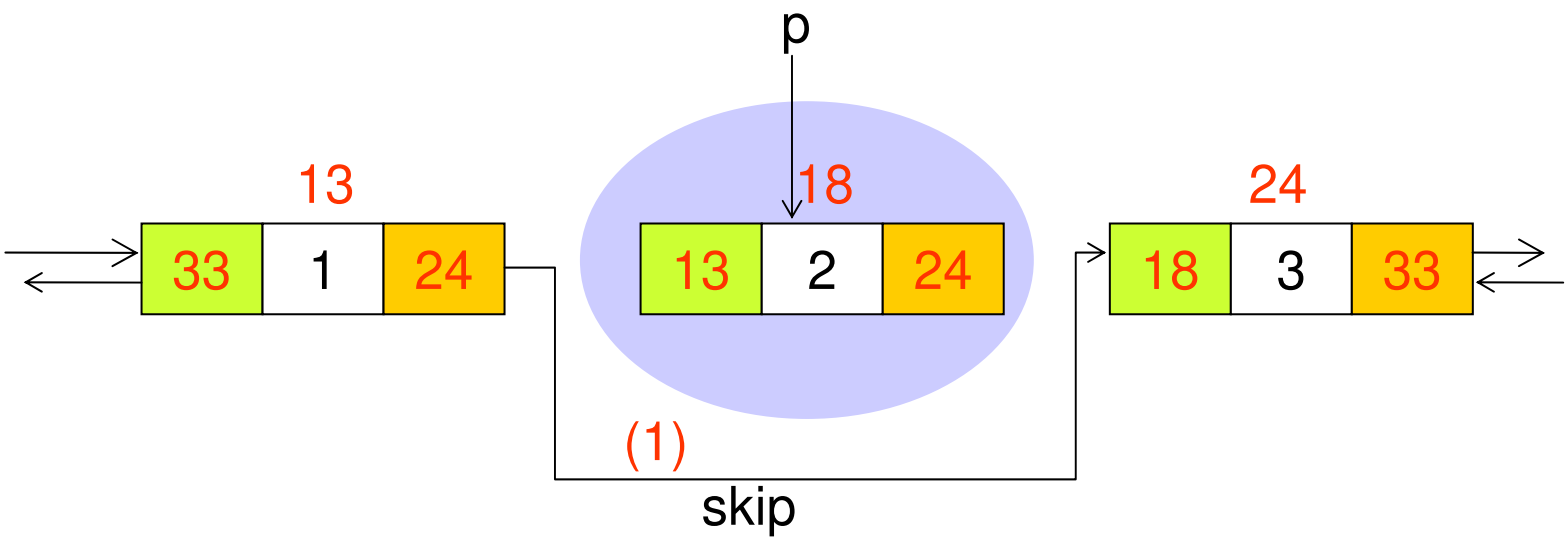



```
p->prev->next=p->next;        (1)
p->next->prev=p->prev;        (2)
free(p);
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p

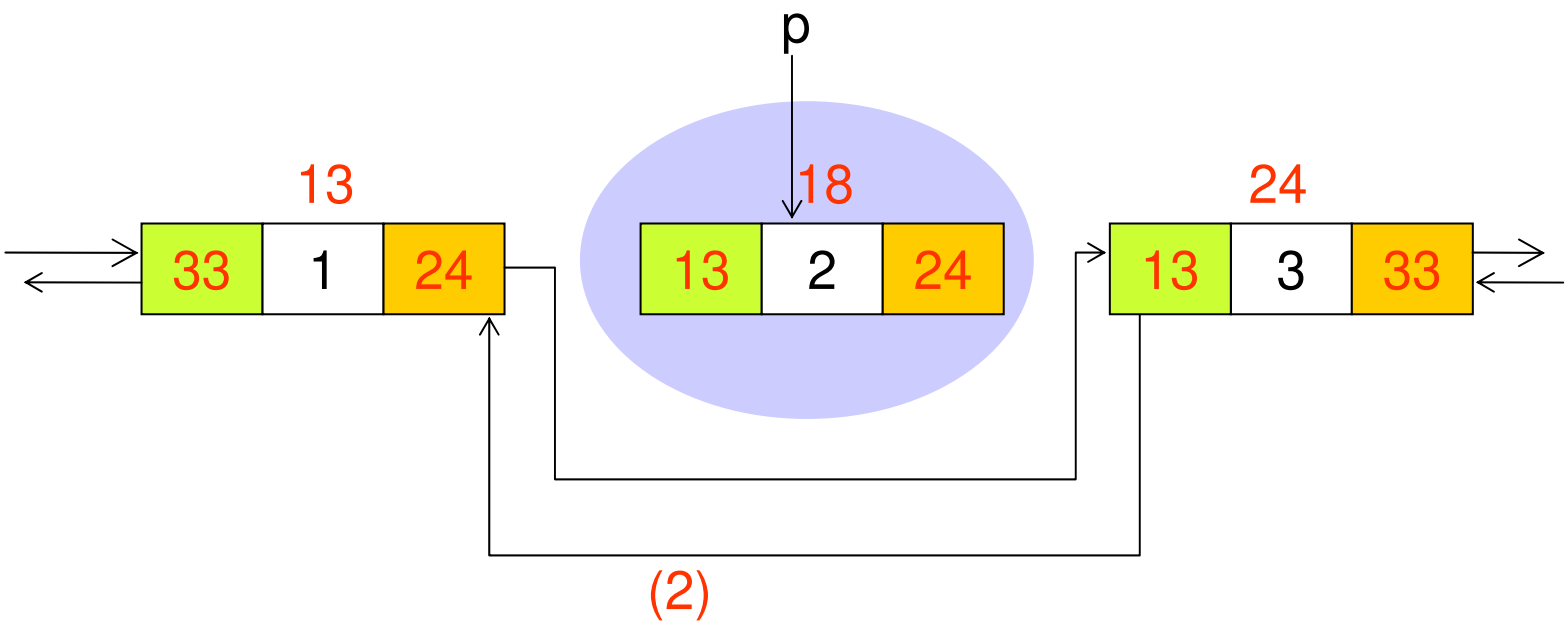



```
p->prev->next=p->next;      (1)
p->next->prev=p->prev;      (2)
free(p);
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p

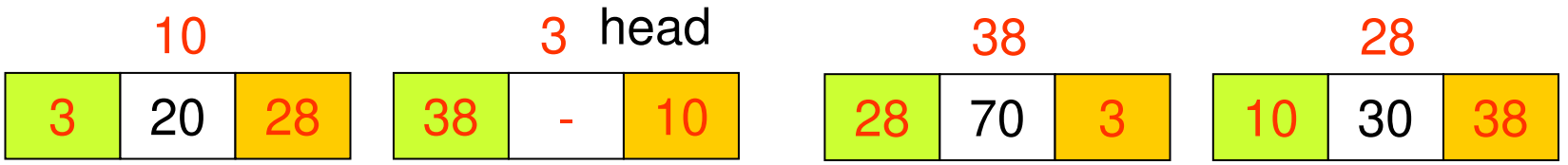



```
p->prev->next=p->next;      (1)
p->next->prev=p->prev;      (2)
free(p);
```

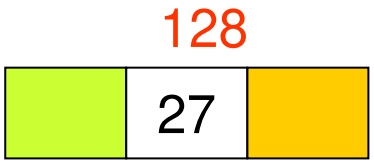
Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Latihan Bidirectional List

1. Gambarkan bidirectional list berikut



2. Tambahkan sel berikut setelah sel yang berisi value “20”



3. Gambarkan kondisi bidirectional list, setelah sel yang berisi value “70” dihapus. Kerjakan soal ini terhadap list yang diperoleh dari no.2 di atas.

Doubly Linked List

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Beberapa Jenis Struktur Data

1. Array
    1. Linear List
    2. Stack
    3. Queue

2. List
    1. Linked List
    2. Circular List
    3. Doubly Linked List
    4. Multi list structure

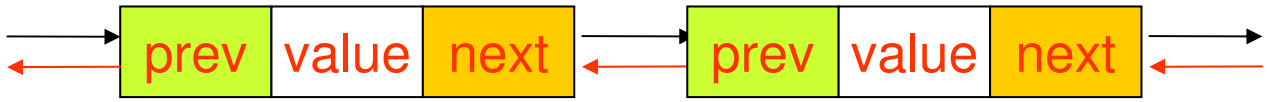
3. Tree Structure

Outline

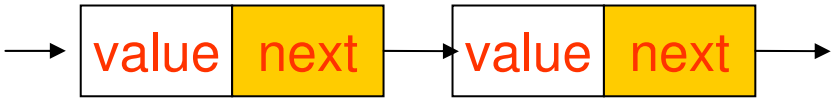
- Review doubly-linked list
- Latihan

Apakah doubly-linked list itu ?

- Semua sel yang terdapat pada list disambungkan dengan pointer, sedangkan tiap sel memiliki TIGA komponen : value, pointer ke sel sebelumnya dan pointer ke sel berikutnya
- Dengan memiliki dua buah pointer ini, maka doubly-linked list dapat diakses dengan DUA arah : ke arah depan dan ke belakang

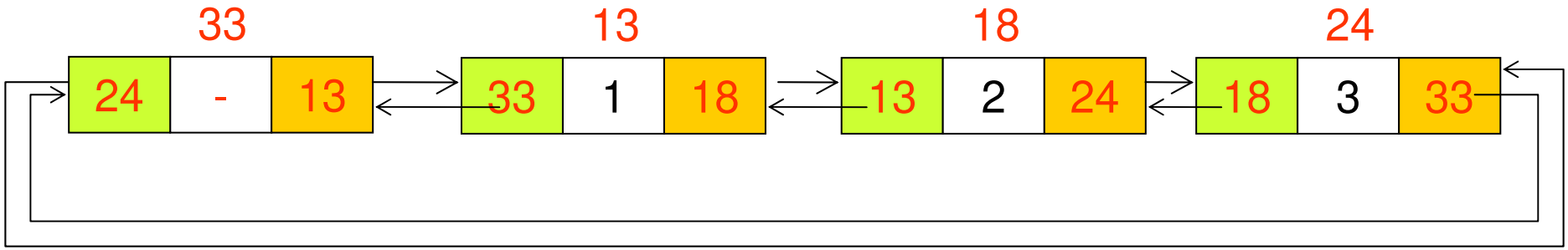


Bidirectional List

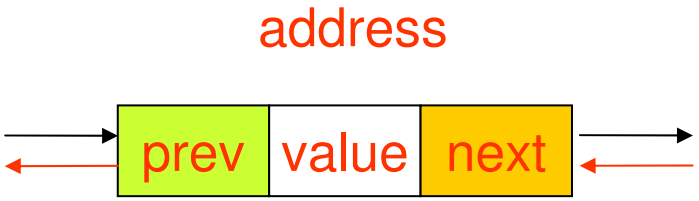


Linked List

Doubly-linked List




```
struct CELL {
        struct CELL *prev;
        struct CELL *next;
        MYDATA value;
};
```



Misalnya int, char, float, long, double, dsb

Kelebihan dan kelemahan

Kelebihan

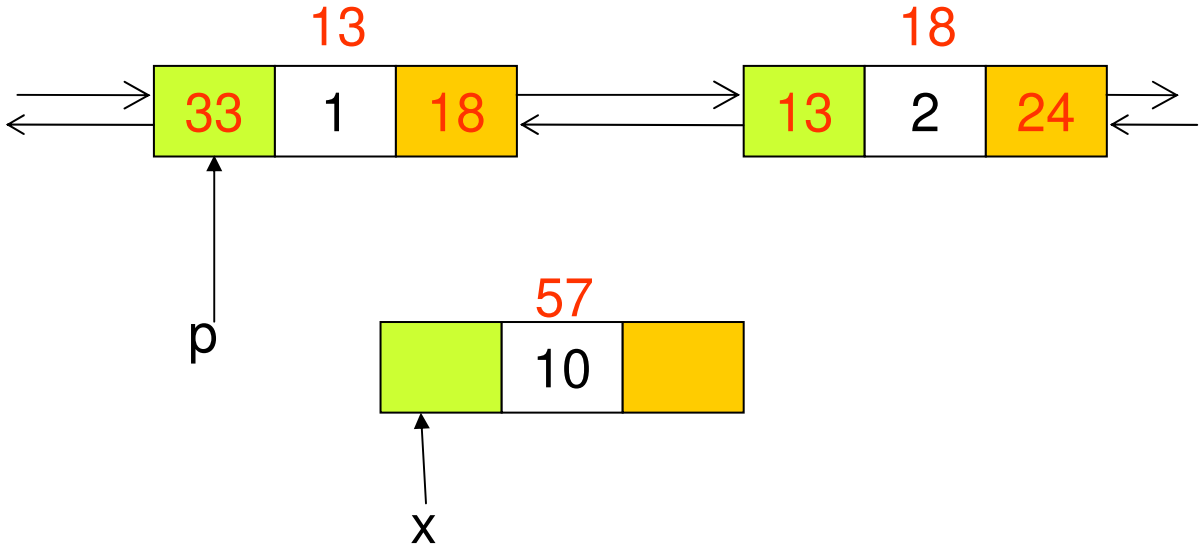
1. List bisa diakses dengan dua arah : ke depan maupun ke belakang
2. Penambahan dan penghapusan data menjadi mudah, karena pada tipe ini kita dapat menambahkan sel baru sesudah maupun sebelum sebuah sel. Demikian juga dalam hal menghapus sel.

Kelemahan

1. Sebuah data memiliki dua buah pointer, sehingga memerlukan space yang lebih besar

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

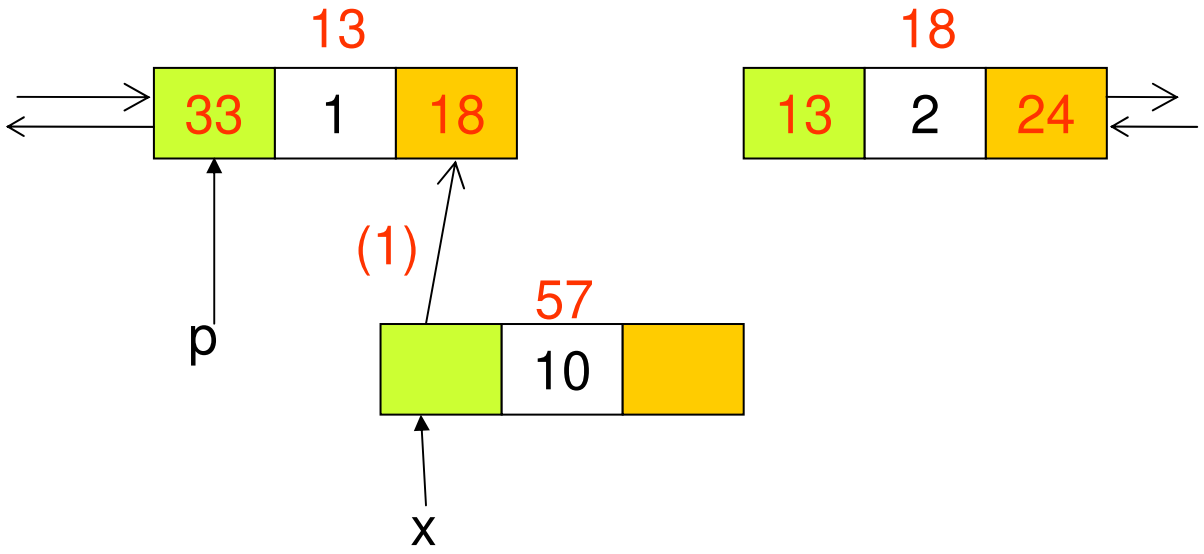



```
x->prev=p;              (1)
x->next=p->next;        (2)
p->next->prev=x;        (3)
p->next=x;              (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

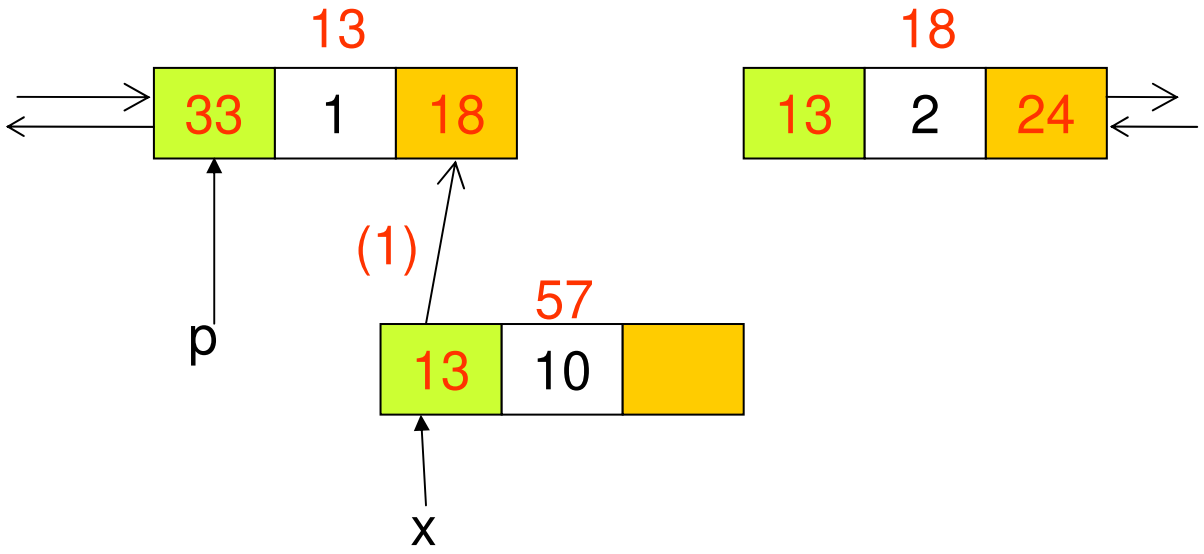



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p



x–>prev=p; (1)

x–>next=p–>next; (2)

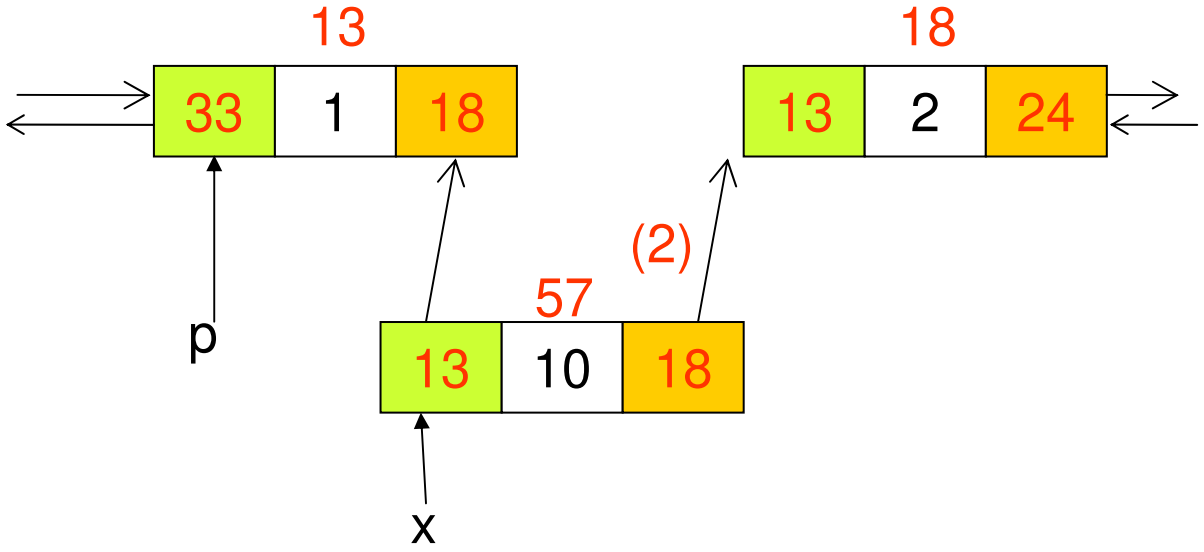
p–>next–>prev=x; (3)

p–>next=x; (4)

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

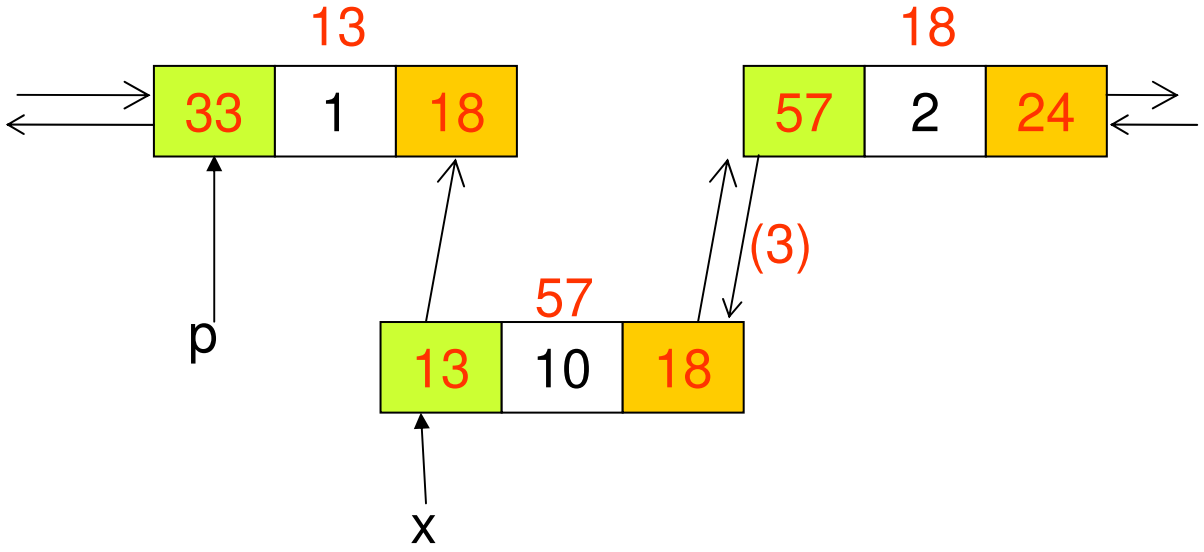



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p

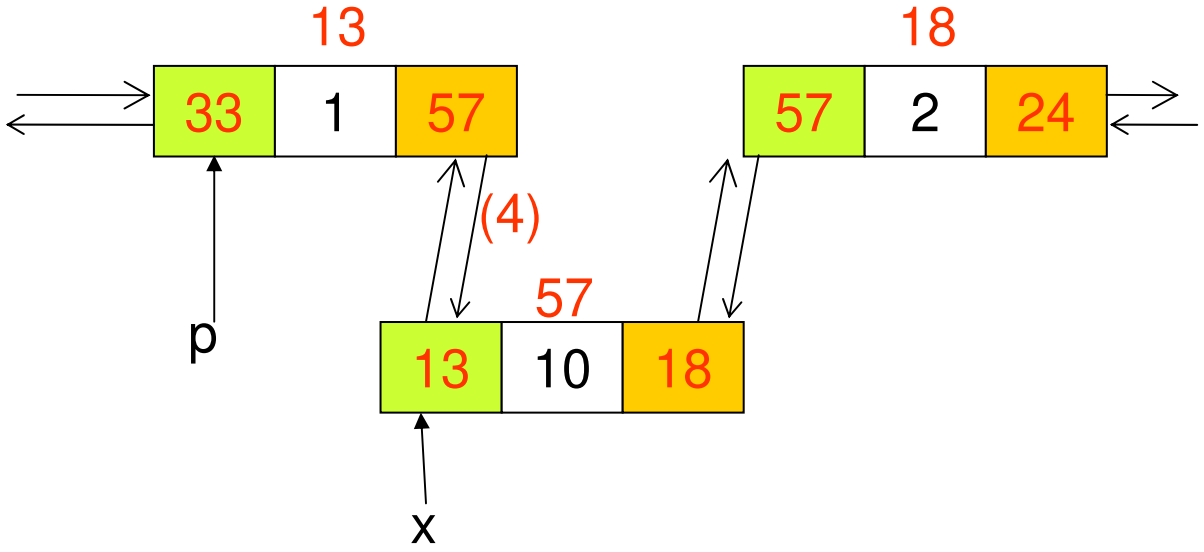



```
x->prev=p;          (1)
x->next=p->next;    (2)
p->next->prev=x;    (3)
p->next=x;          (4)
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menambahkan sel baru

Tambahkan sel baru yang ditunjuk oleh pointer x, sesudah suatu sel yang ditunjuk oleh pointer p



x–>prev=p; (1)

x–>next=p–>next; (2)

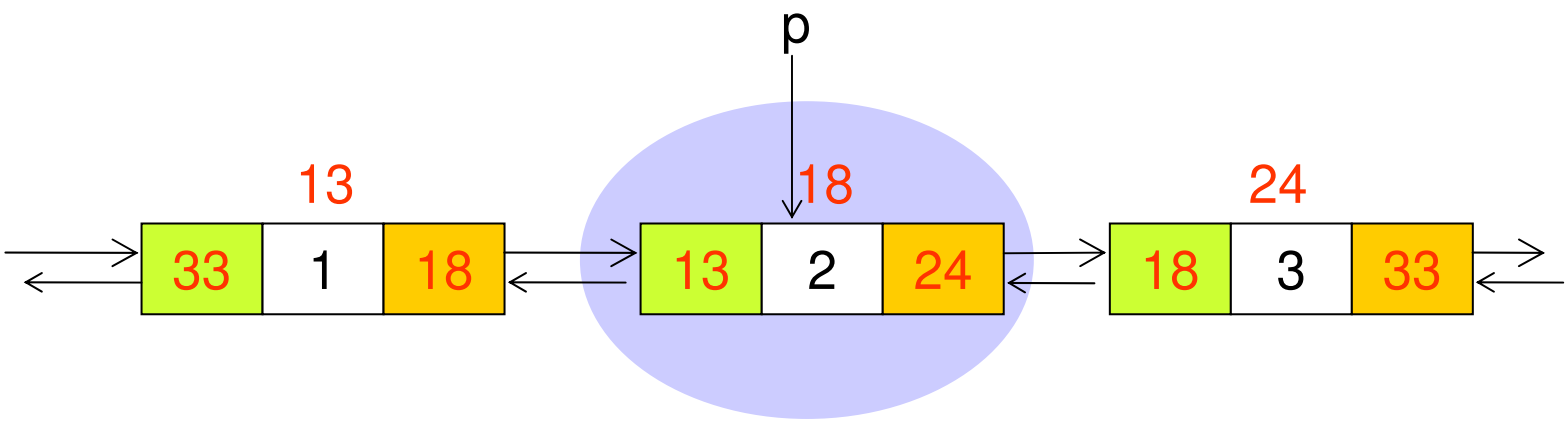
p–>next–>prev=x; (3)

p–>next=x; (4)

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p

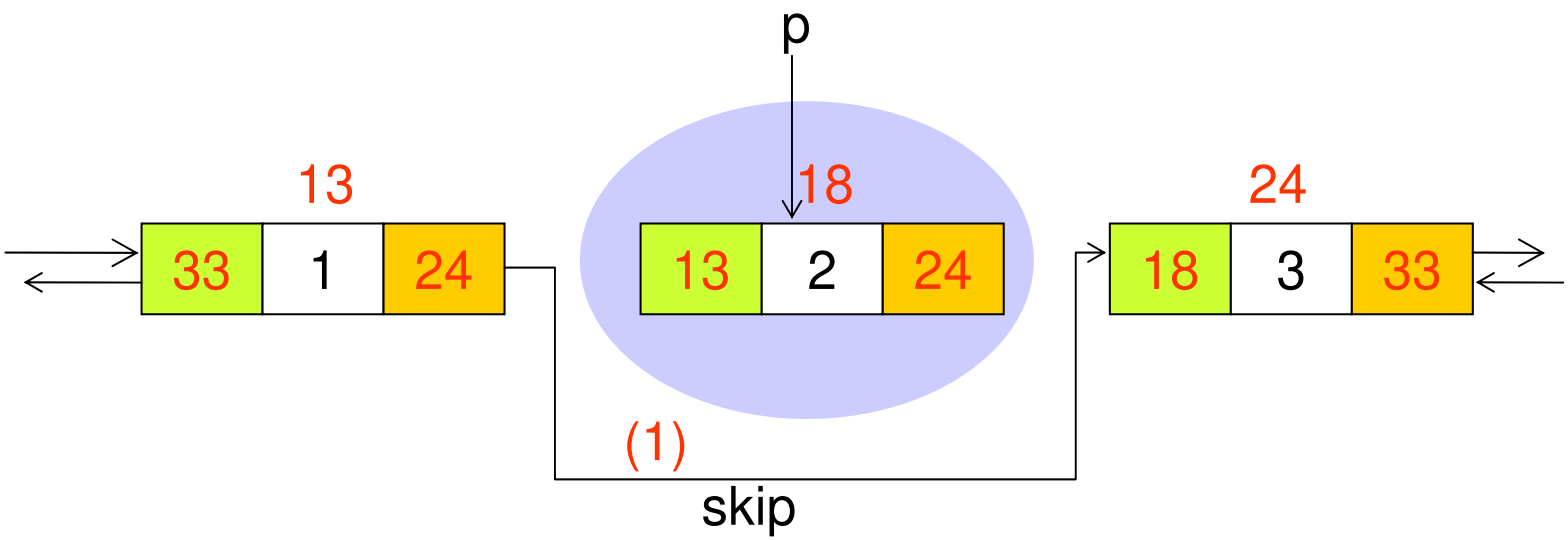



```
p->prev->next=p->next;        (1)
p->next->prev=p->prev;        (2)
free(p);
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p

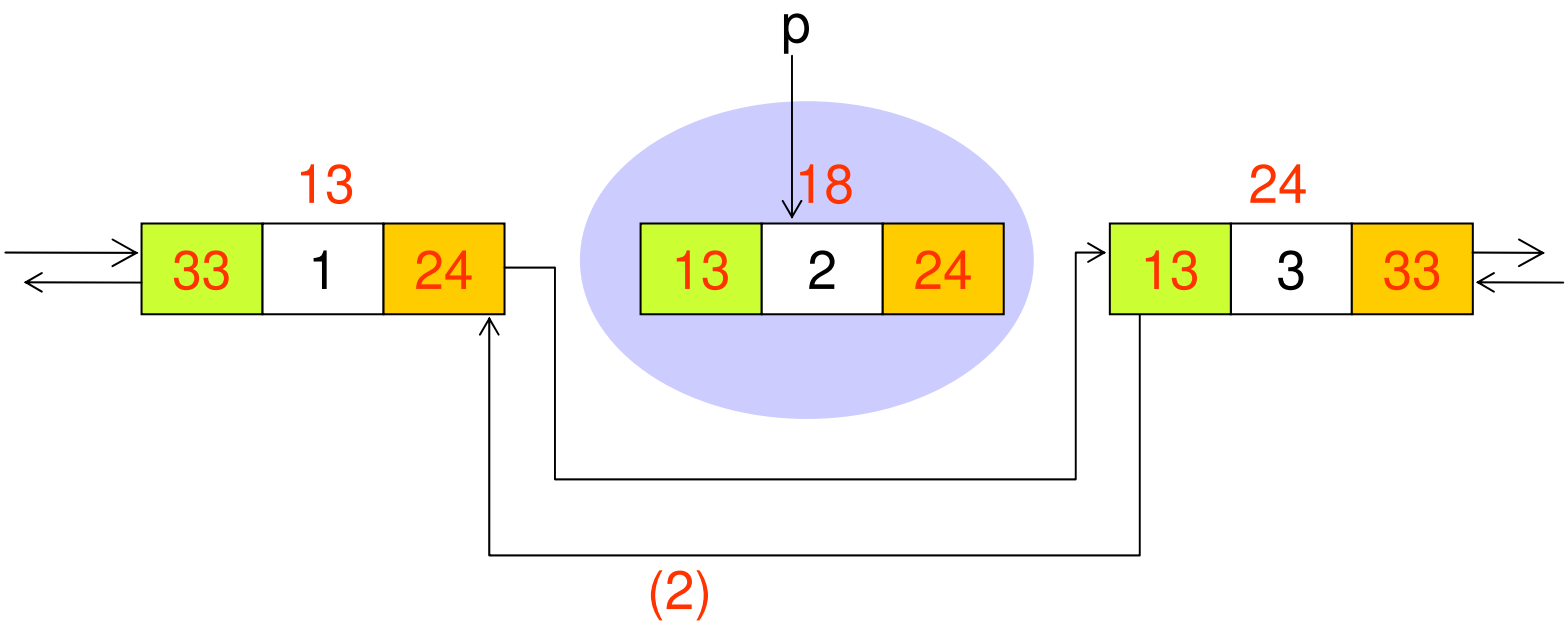



```
p->prev->next=p->next;      (1)
p->next->prev=p->prev;      (2)
free(p);
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

Cara menghapus sebuah sel

Hapuslah sel yang ditunjuk pointer p



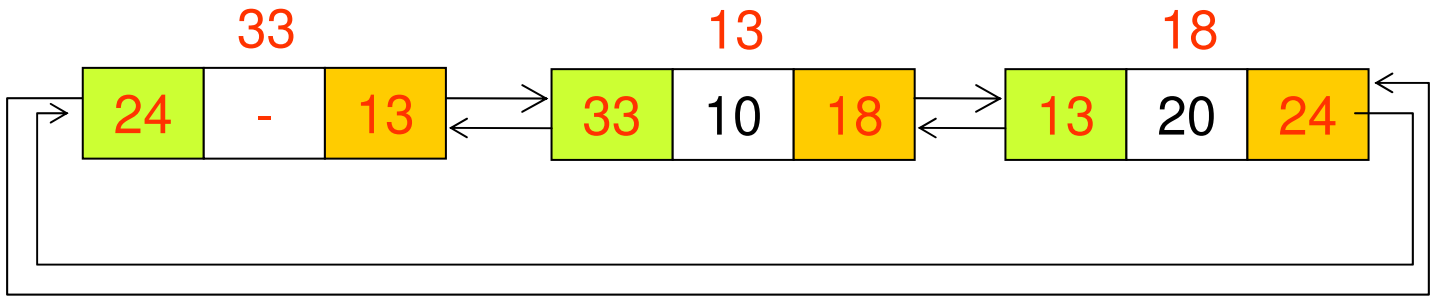

```
p->prev->next=p->next;      (1)
p->next->prev=p->prev;      (2)
free(p);
```

Yang diubah bukan hanya pointer next
tetapi juga pointer prev

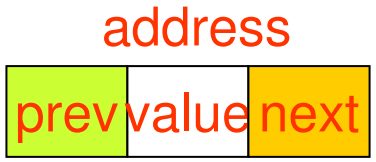
Latihan pemrograman

1. Downloadlah doublyll.c dari http://asnugroho.net/lecture/ds.html
2. Selesaikan fungsi cell_insert()
3. Fungsi cell_insert() adalah menambahkan databaru pada sebuah doubly linked list, dimana data harus dalam kondisi terurutkan. Dimulai dari data dengan nilai kecil, dan semakin ke belakang semakin besar.

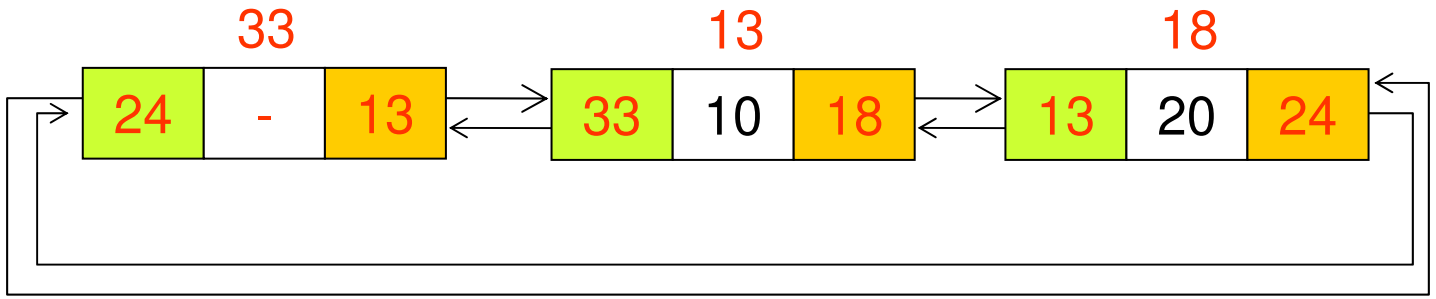
Doubly-linked List

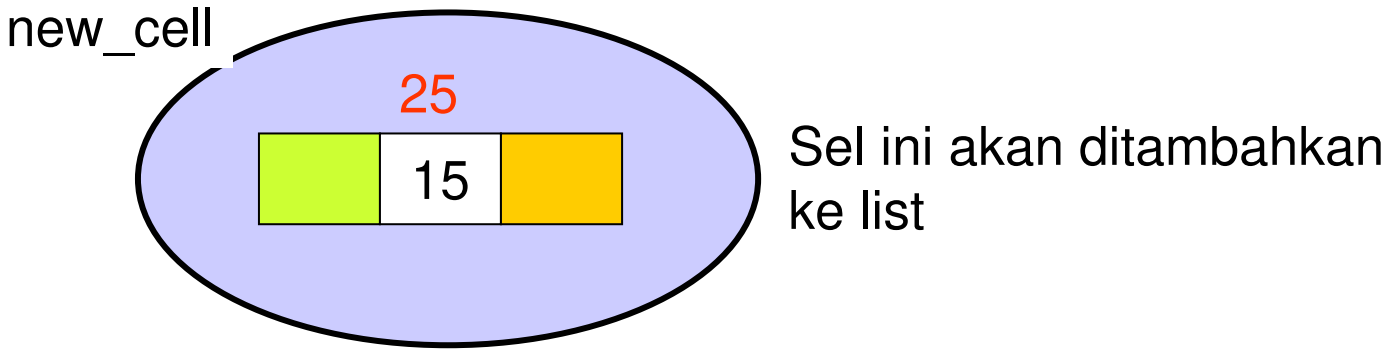



```
struct CELL {
        struct CELL *prev;
        struct CELL *next;
        int  value;
};
```

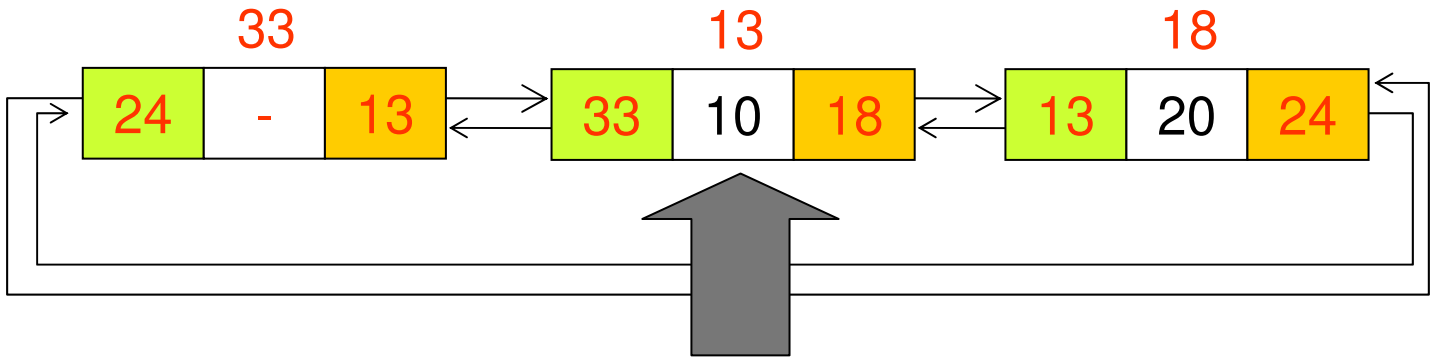


Doubly-linked List





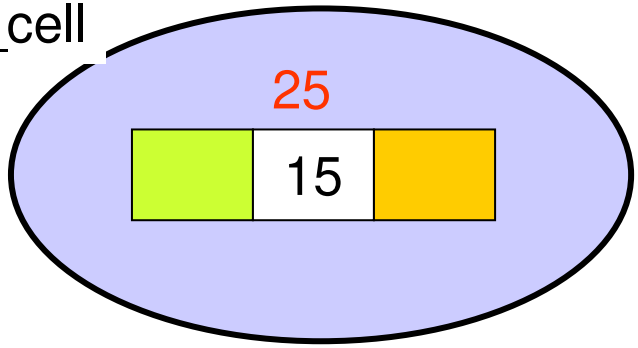
Doubly-linked List



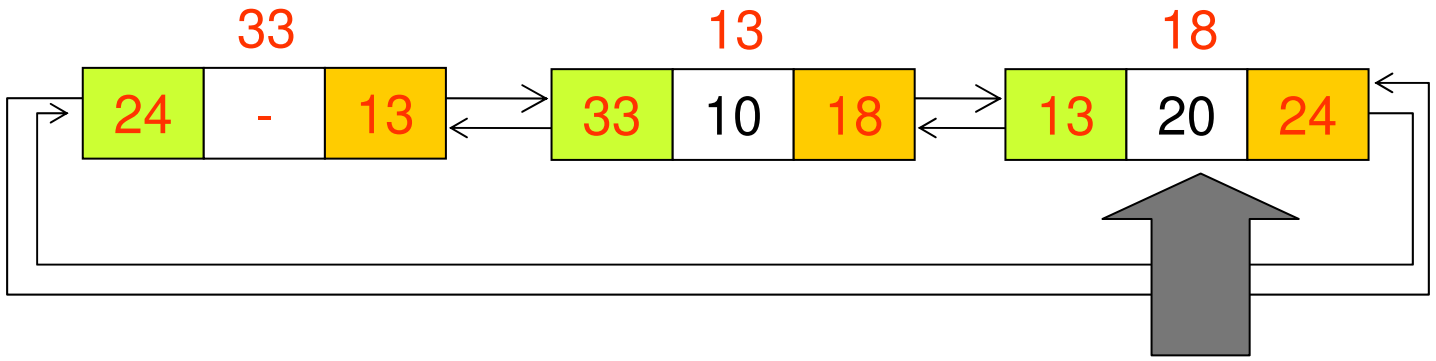
p=13

p->value=10 → 10 < 15 →pindah ke sel berikutnya

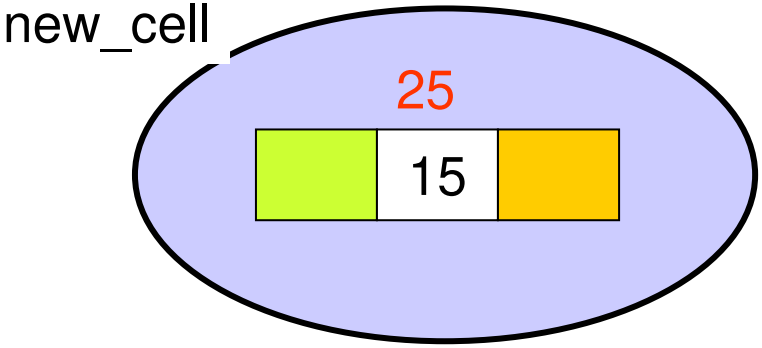
new_cell



Doubly-linked List

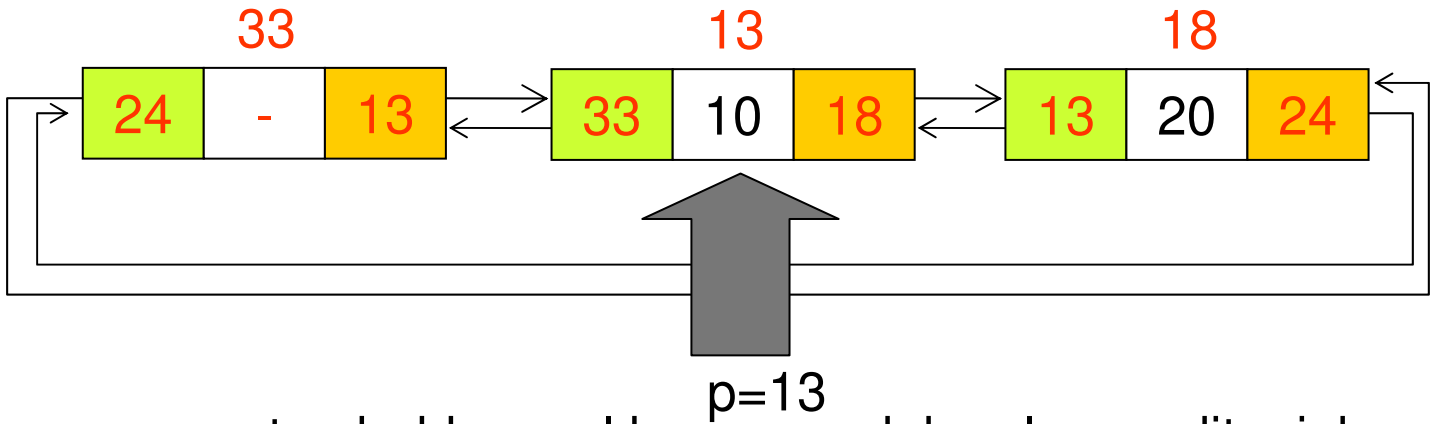


p=18
p->value=20 → 20 > 15

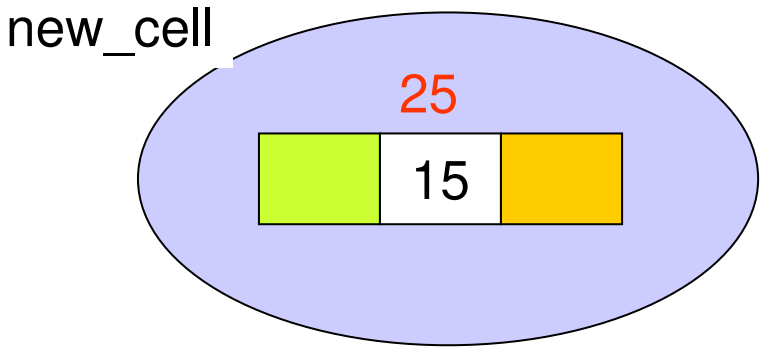


pindah ke sel sebelumnya

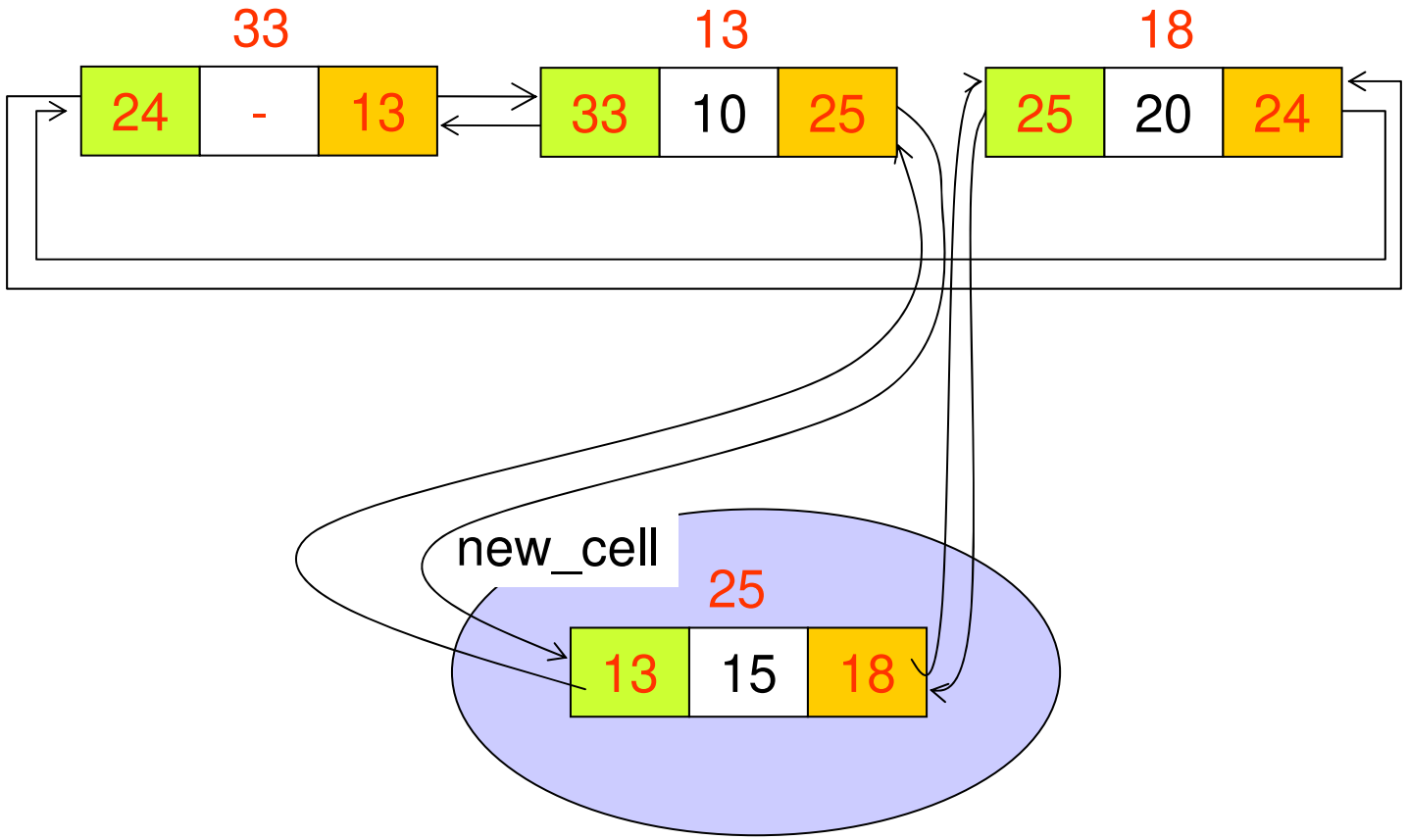
Doubly-linked List



tambahkan sel baru sesudah sel yang ditunjuk



Doubly-linked List



Doubly-linked List

```
void cell_insert(int a)
{
        cell *p, *new_cell;
```

① Mencari posisi dimana new_cell harus dimasukan

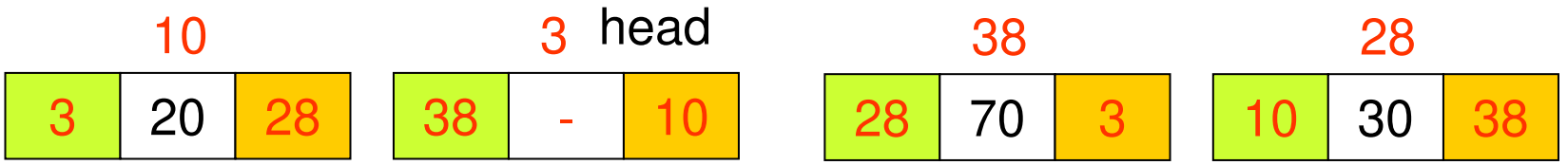
② Alokasi memory untuk new_cell memakai malloc

③ Tambahkan new_cell pada posisi yang ditentukan

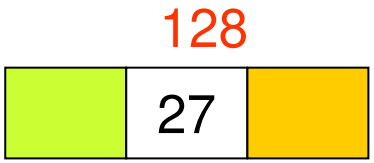
```
}
```

Latihan pemrograman

1. Gambarkan bidirectional list berikut



2. Tambahkan sel berikut setelah sel yang berisi value “20”



3. Gambarkan kondisi bidirectional list, setelah sel yang berisi value “70” dihapus. Kerjakan soal ini terhadap list yang diperoleh dari no.2 di atas.

Tree Structure

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Beberapa Jenis Struktur Data

1. Array
    1. Linear List
    2. Stack
    3. Queue

2. List
    1. Linked List
    2. Circular List
    3. Doubly Linked List
    4. Multi list structure

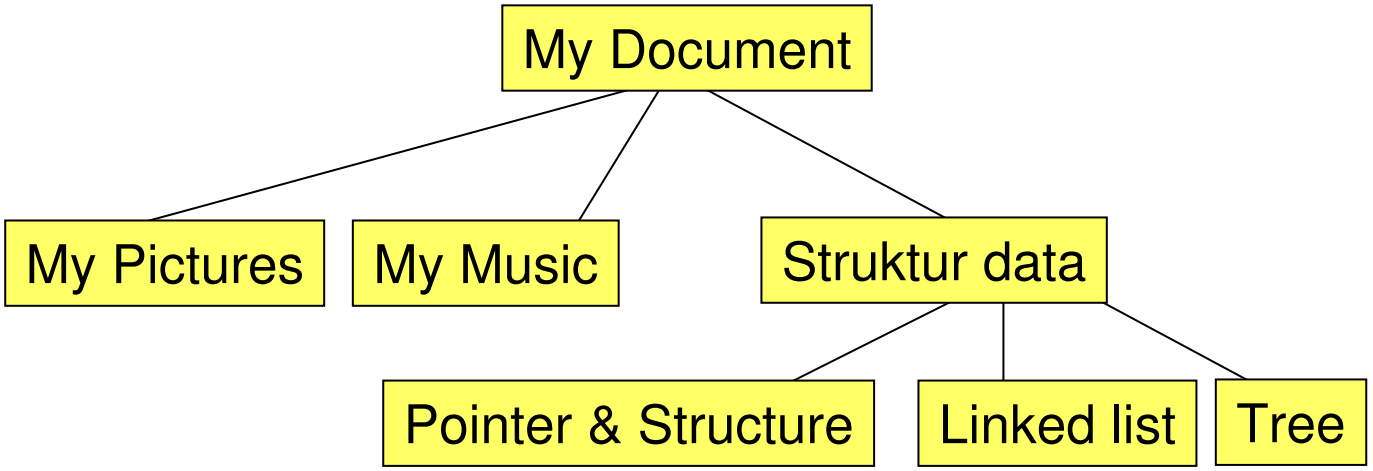
3. Tree Structure

Outline

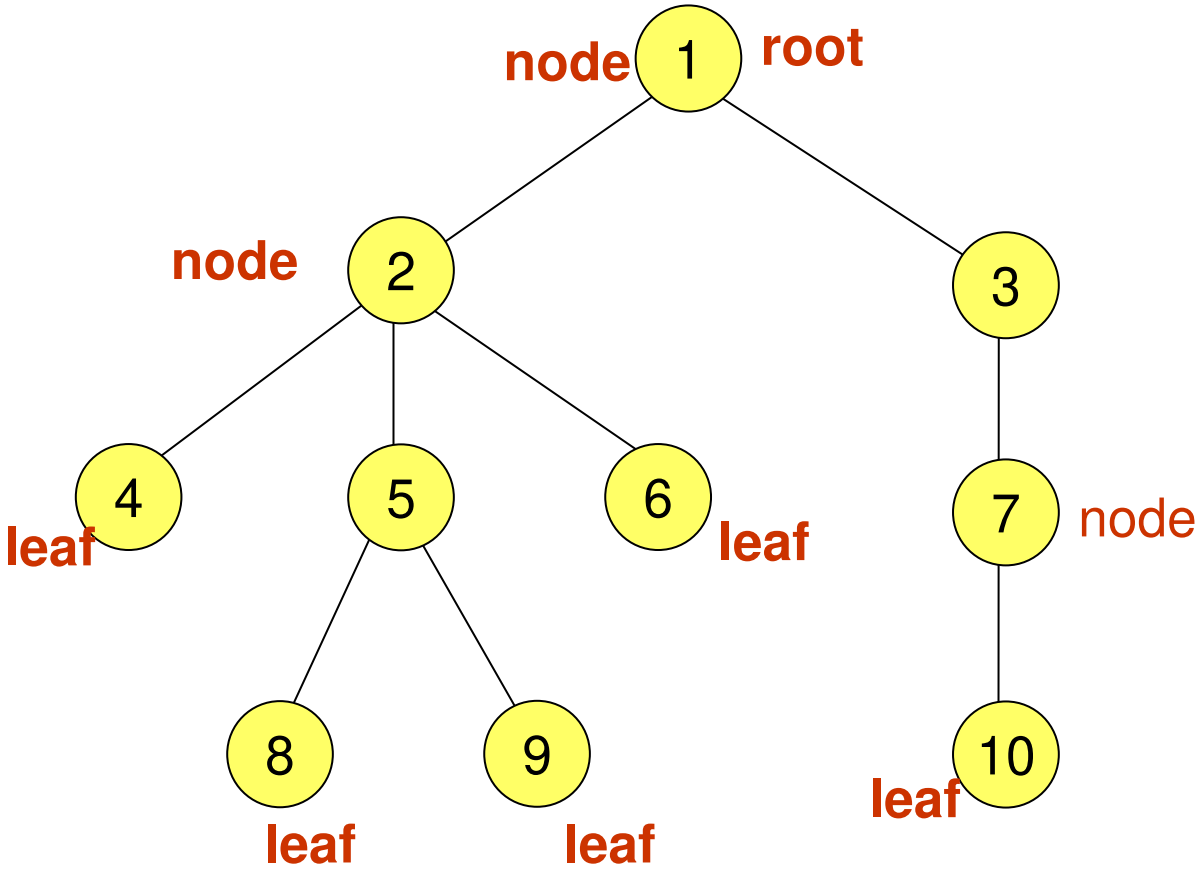
1. Apakah Tree Structure itu ?
2. Binary Tree & implementasinya
3. Tree Traversal
4. Implementasi tree (selain binary tree)

Apakah Tree Structure itu ?

- Struktur data yang menunjukkan hubungan bertingkat (memiliki hierarkhi)
- Contoh: direktori/folder pada windows atau linux

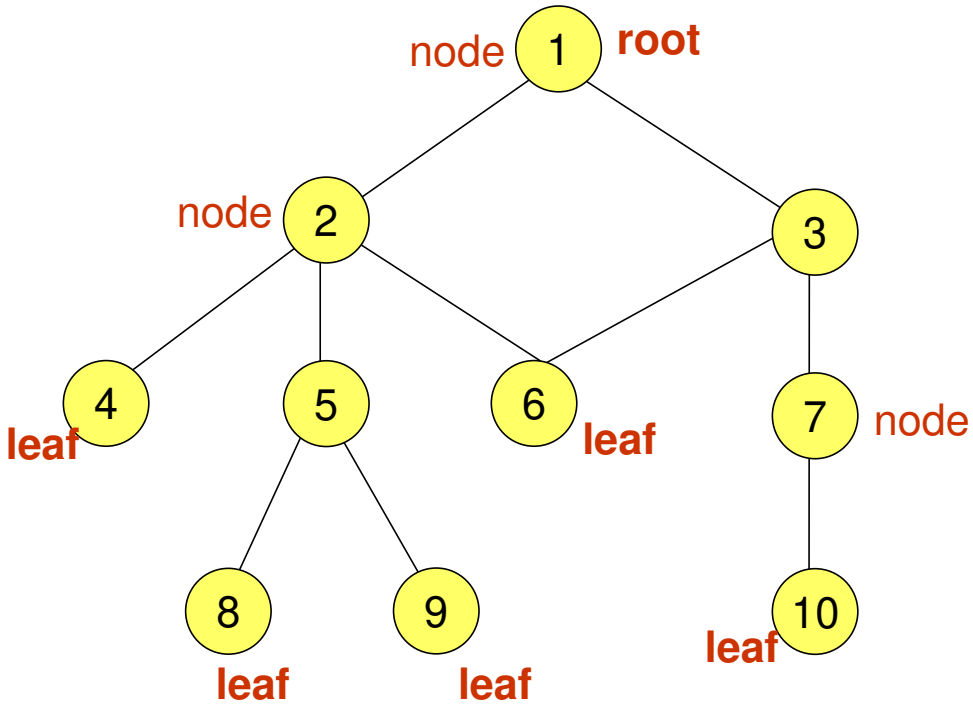


Nama komponen pada Tree



Hubungan antar komponen

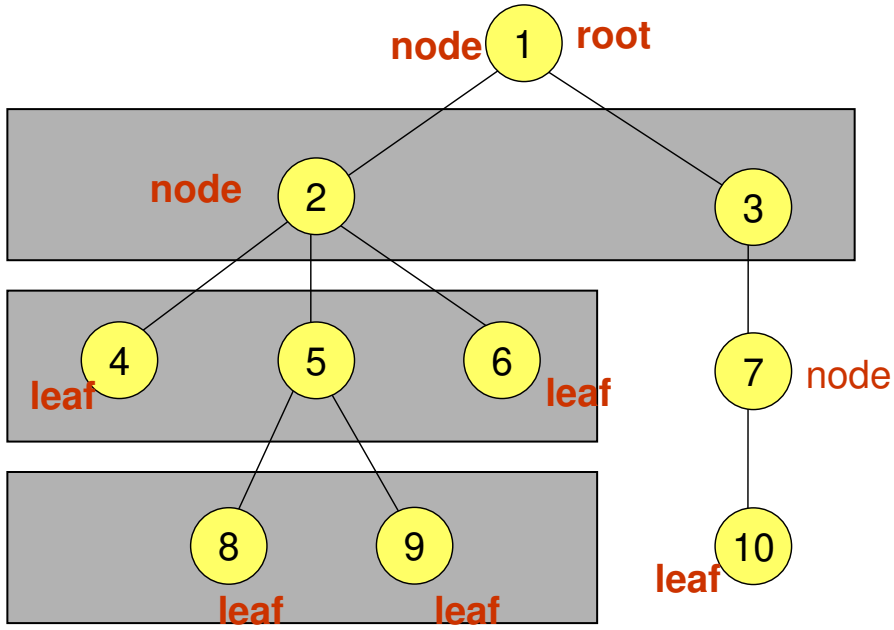
- Hubungan antar elemen: parent-child, father-son, mother-daughter
- Nama node: nama(angka) yang dipakai untuk membedakan sebuah node dengan node yang lain. Dalam kuliah ini adalah angka yang tertulis dalam lingkaran.
- Label: nilai yang diingat oleh sebuah node
- Tree vs Graph
  - Tree: setiap node kecuali root hanya memiliki sebuah parent
  - Graph: dapat memiliki lebih dari sebuah parent



Contoh graph

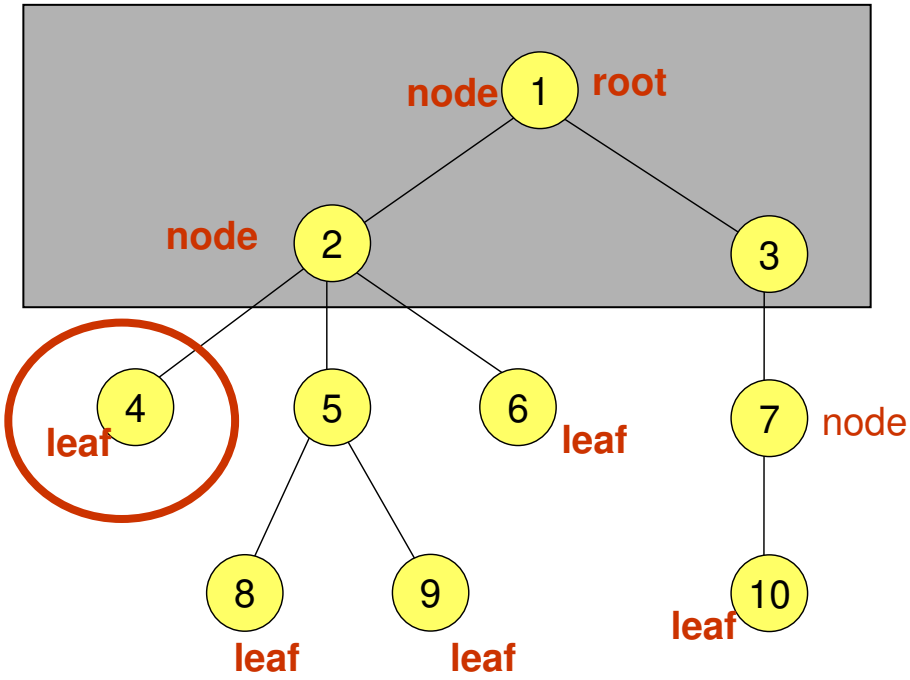
Hubungan antar komponen

- sibling : node-node yang memiliki parent yang sama
- Ancestor dari node x : node yang ditemukan, ketika menyusuri tree ke atas dari node x
- Descendant dari node x : node yang ditemukan ketika menyusuri tree ke bawah dari node x



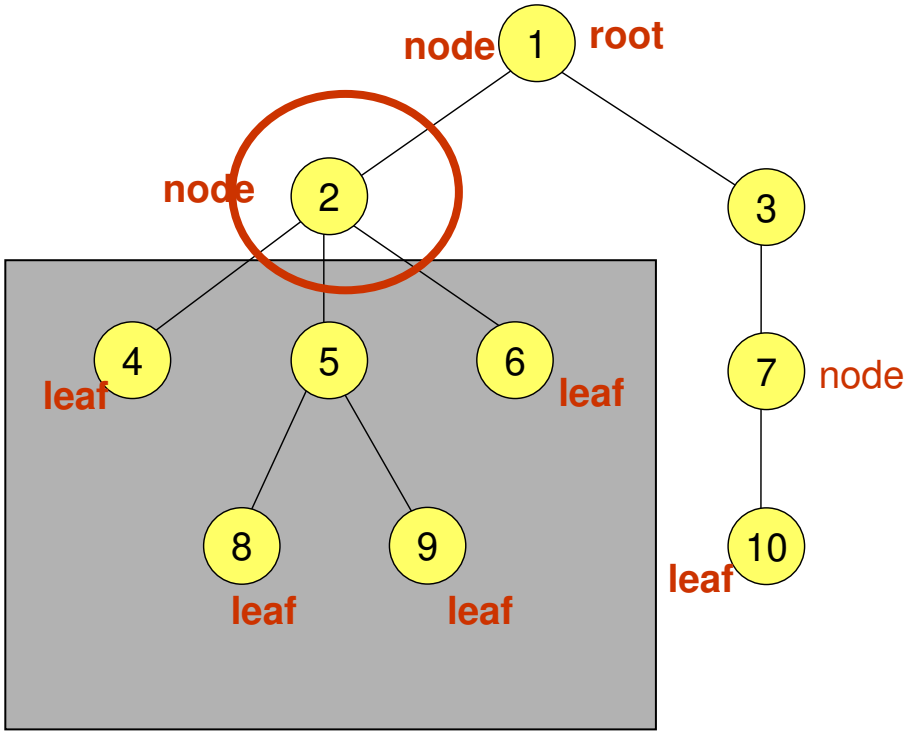
Hubungan antar komponen

- sibling : node-node yang memiliki parent yang sama
- Ancestor dari node x : node yang ditemukan, ketika menyusuri tree ke atas dari node x
- Descendant dari node x : node yang ditemukan ketika menyusuri tree ke bawah dari node x

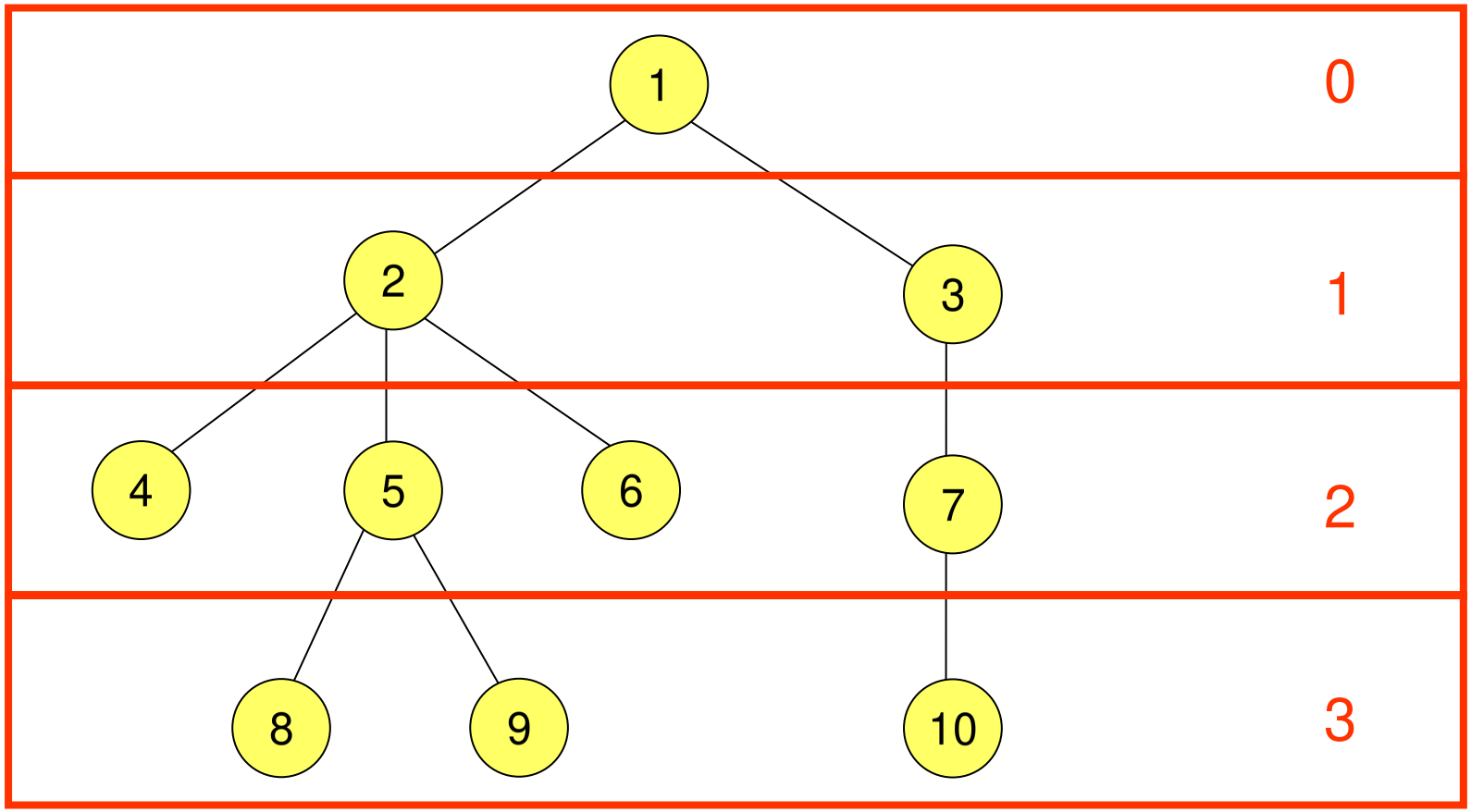


Hubungan antar komponen

- sibling : node-node yang memiliki parent yang sama
- Ancestor dari node x : node yang ditemukan, ketika menyusuri tree ke atas dari node x
- Descendant dari node x : node yang ditemukan ketika menyusuri tree ke bawah dari node x



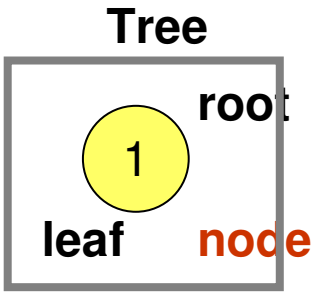
Level



Definisi TREE

Sebuah tree didefinisikan sebagai struktur y ang dibentuk secara recursive oleh kedua rule berikut:

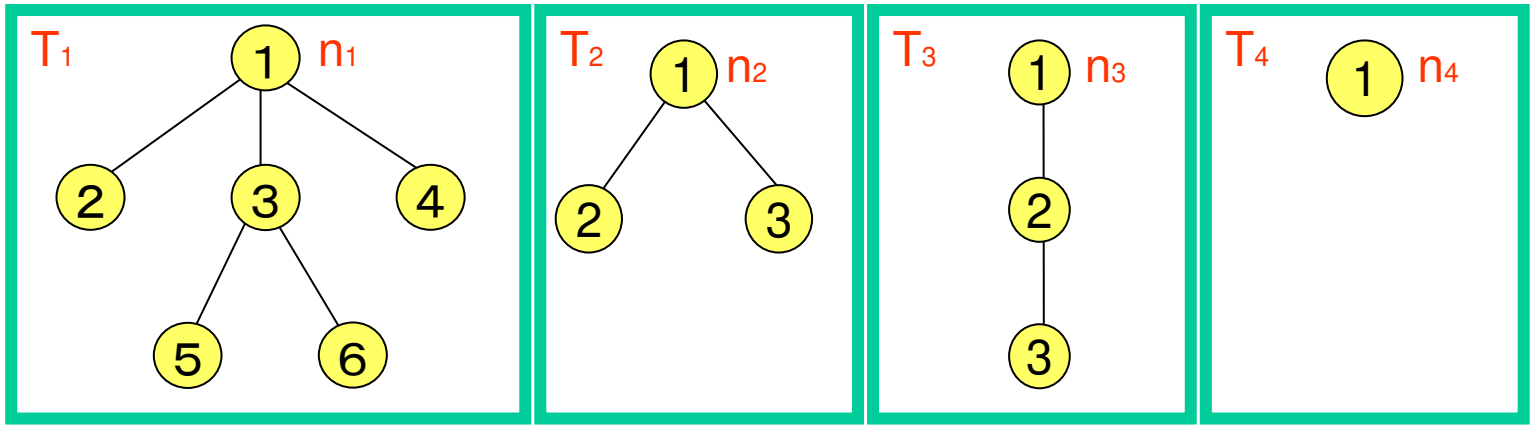
1. Sebuah node adalah sebuah tree. Node satu-satunya pada tree ini berfungsi sebagai root maupun leaf.
2. Dari k buah tree $T_1 \sim T_k$, dan masing-masing memiliki root $n_1 \sim n_k$.Jika node n adalah parent dari node$n_1 \sim n_k$, akan diperoleh sebuah tree baru T yang memiliki root n. Dalam kondisi ini, tree $T_1 \sim T_k$menjadi sub-tree dari tree T. Root dari sub-tree $n_1 \sim n_k$adalah child dari node n .



Definisi TREE

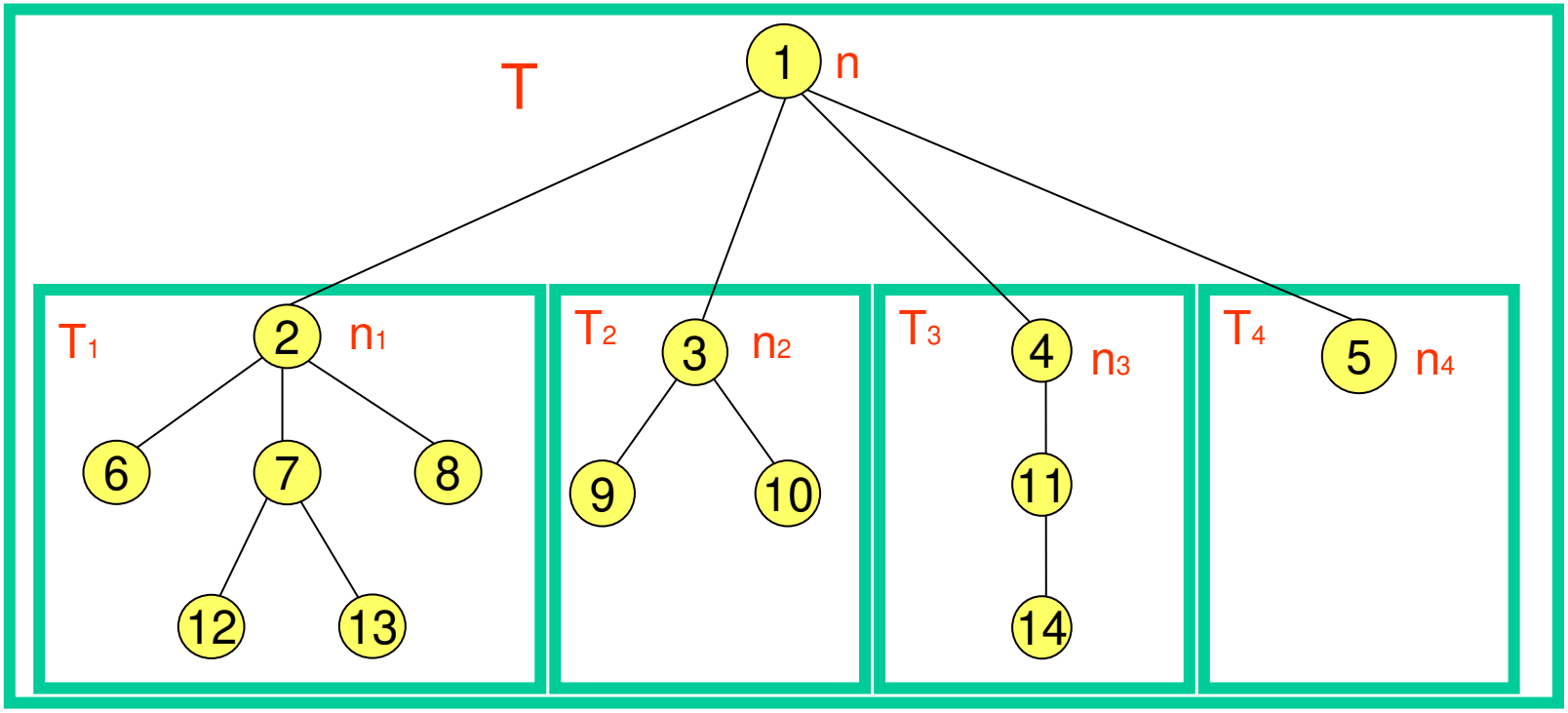
Sebuah tree didefinisikan sebagai struktur y ang dibentuk secara recursive oleh kedua rule berikut:

1. Sebuah node adalah sebuah tree. Node satu-satunya pada tree ini berfungsi sebagai root maupun leaf.
2. Dari k buah tree $T_1 \sim T_k$, dan masing-masing memiliki root $n_1 \sim n_k$.Jika node n adalah parent dari node$n_1 \sim n_k$, akan diperoleh sebuah tree baru T yang memiliki root n. Dalam kondisi ini, tree $T_1 \sim T_k$menjadi sub-tree dari tree T. Root dari sub-tree $n_1 \sim n_k$adalah child dari node n .



Definisi TREE

2. Dari k buah tree $T_1 \sim T_k$, dan masing-masing memiliki root $n_1 \sim n_k$.Jika node n adalah parent dari node$n_1 \sim n_k$, akan diperoleh sebuah tree baru T yang memiliki root n. Dalam kondisi ini, tree $T_1 \sim T_k$menjadi sub-tree dari tree T. Root dari sub-tree $n_1 \sim n_k$adalah child dari node n .



Ordered vs Unordered tree

Ordered tree

- Antar sibling terdapat urutan “usia”
- Node yang paling kiri berusia paling tua (sulung), sedangkan node yang paling kanan berusia paling muda (bungsu)
- Posisi node diatur atas urutan tertentu

Unordered tree

- Antar sibling tidak terdapat urutan tertentu

Outline

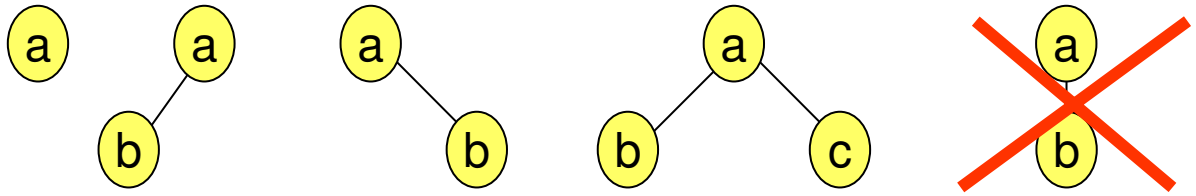
1. Apakah Tree Structure itu ?
2. Binary Tree & implementasinya
3. Tree Traversal
4. Implementasi tree (selain binary tree)

Definisi

Definisi Binary Tree

1. Sebuah tree yang kosong juga merupakan sebuah binary tree
2. Binary tree harus memenuhi salah satu syarat berikut
   - Tidak memiliki anak
   - Memiliki subtree di sebelah kiri (left subtree)
   - Memiliki subtree di sebelah kanan (right subtree)
   - Memiliki baik left subtree maupun right subtree

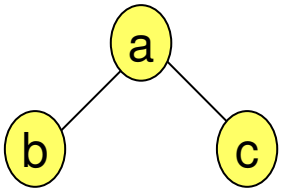
HATI-HATI DALAM MENGGAMBAR BINARY TREE

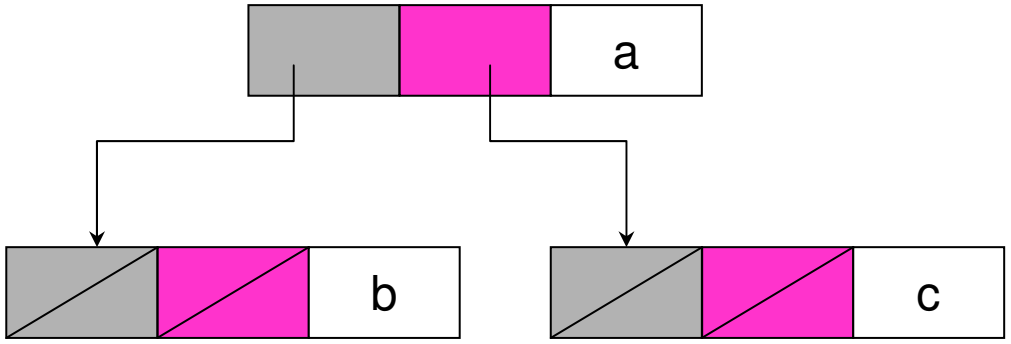


Subtree (child) yang dimiliki bukan kiri maupun kanan

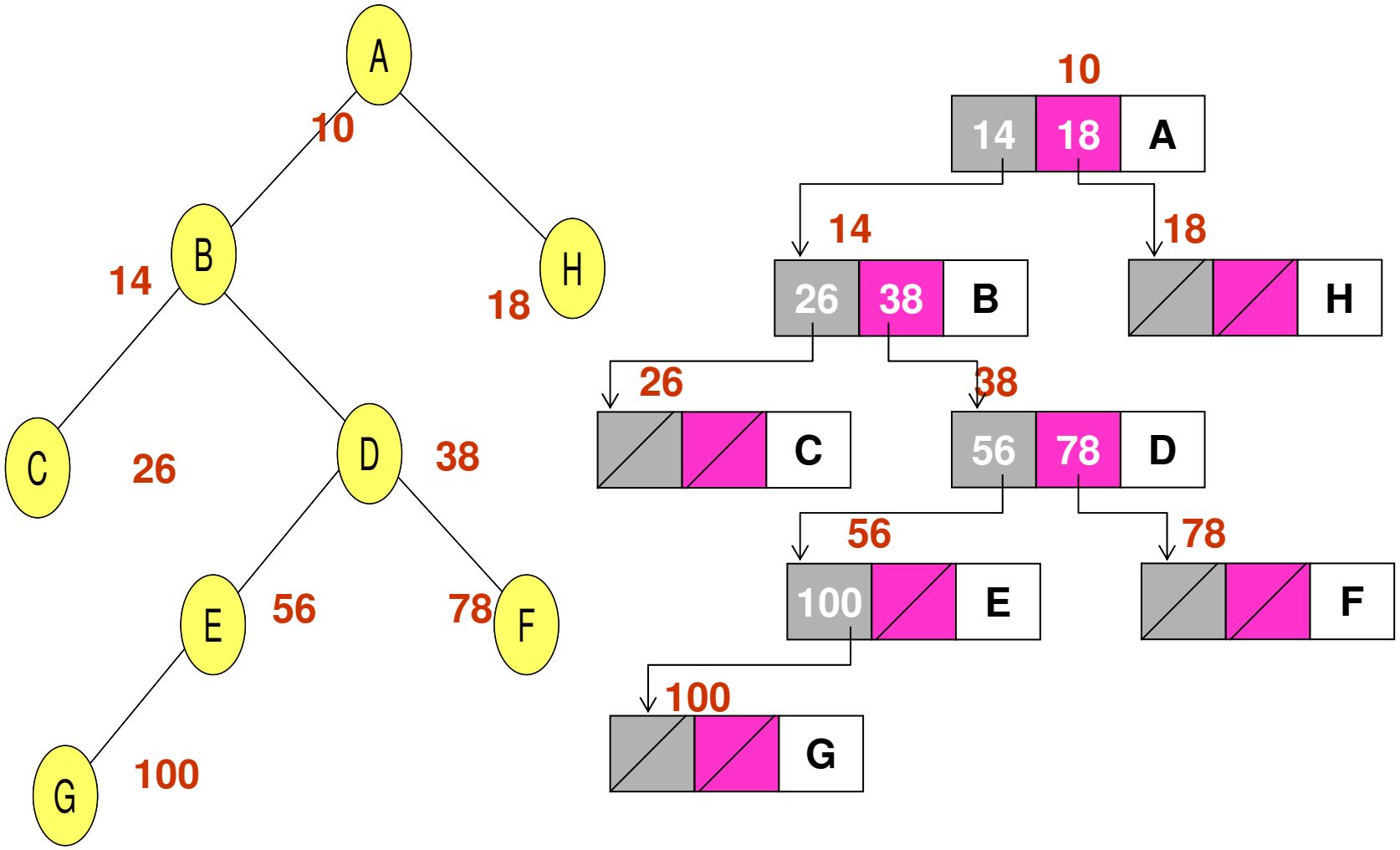
Implementasi Binary Tree

```
struct node  {
        struct node  *left;
        struct node  *right;
        mydata       label;
}
```



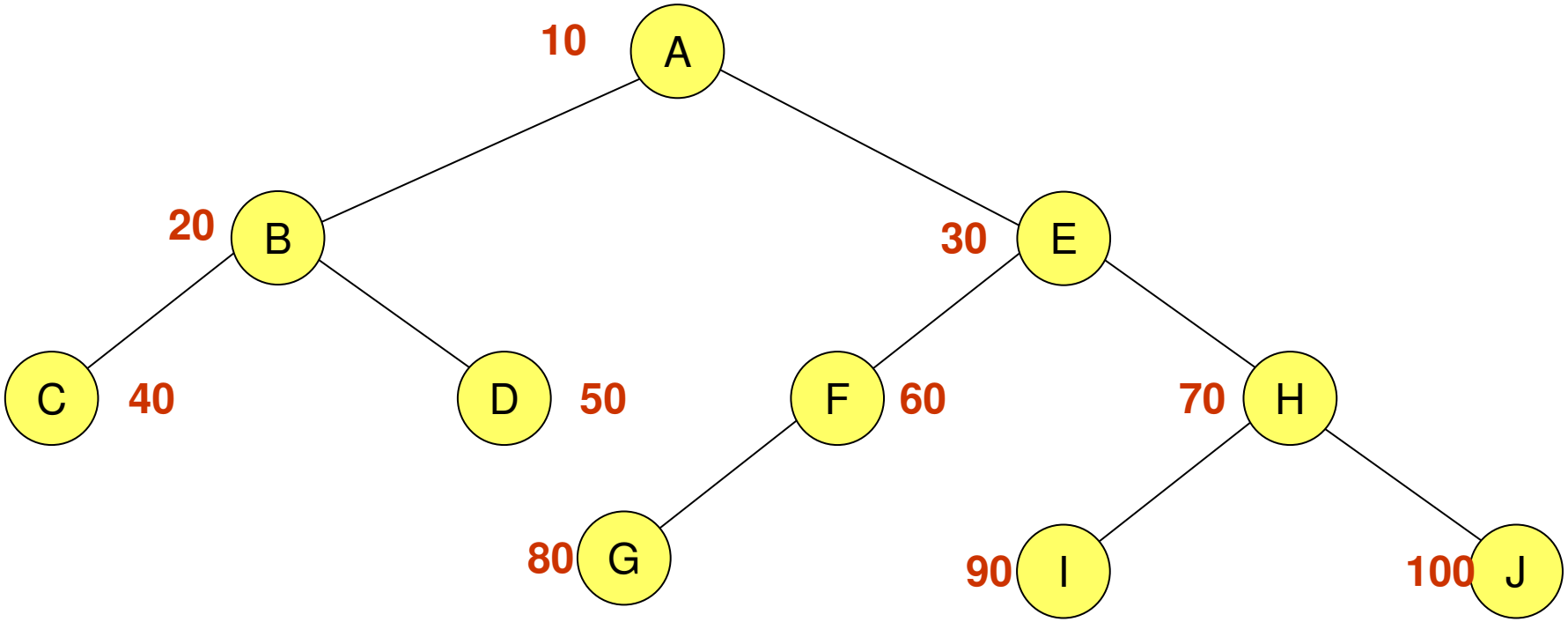


Contoh


A
10
B
14
H
18
C
26
D
38
E
56
78
F
G
100
10
14
18
A
14
26
38
B
18
H
26
38
C
56
78
D
56
78
100
E
F
100
G

Latihan 1

- Gambarkan implementasi dari binary tree berikut



Outline

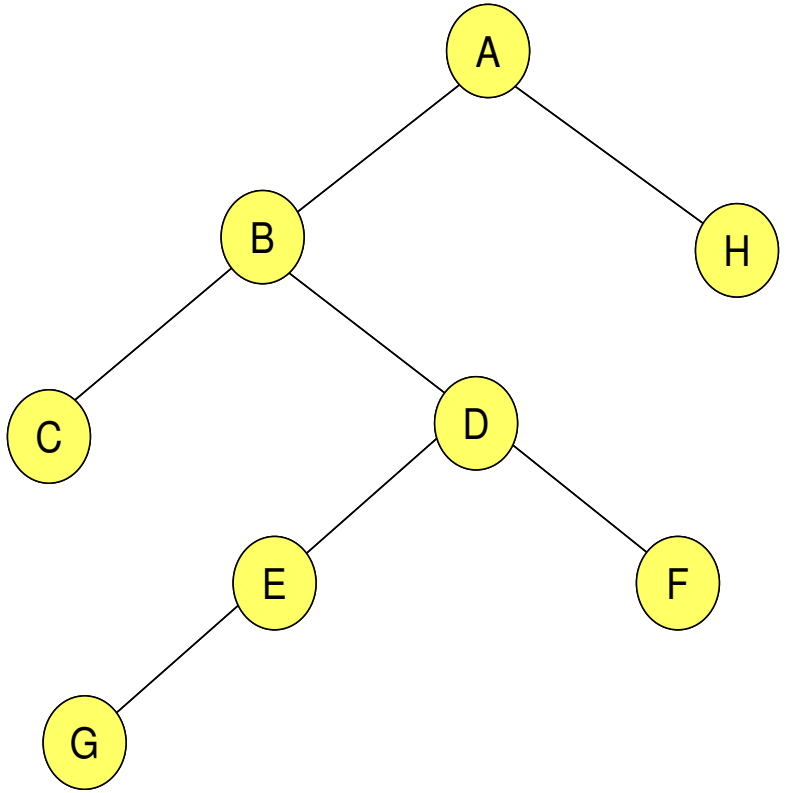
1. Apakah Tree Structure itu ?
2. Binary Tree & implementasinya
3. Tree Traversal
4. Implementasi tree (selain binary tree)

Definisi Tree Traversal

- Teknik menyusuri tiap node dalam sebuah tree secara sistematis, sehingga semua node dapat dan hanya satu kali saja dikunjungi
- Ada tiga cara traversal
  - preorder
  - inorder
  - postorder
- Untuk tree yang kosong, traversal tidak perlu dilakukan

Preorder

1. Visit the root
2. Traverse the left subtree
3. Traverse the right subtree



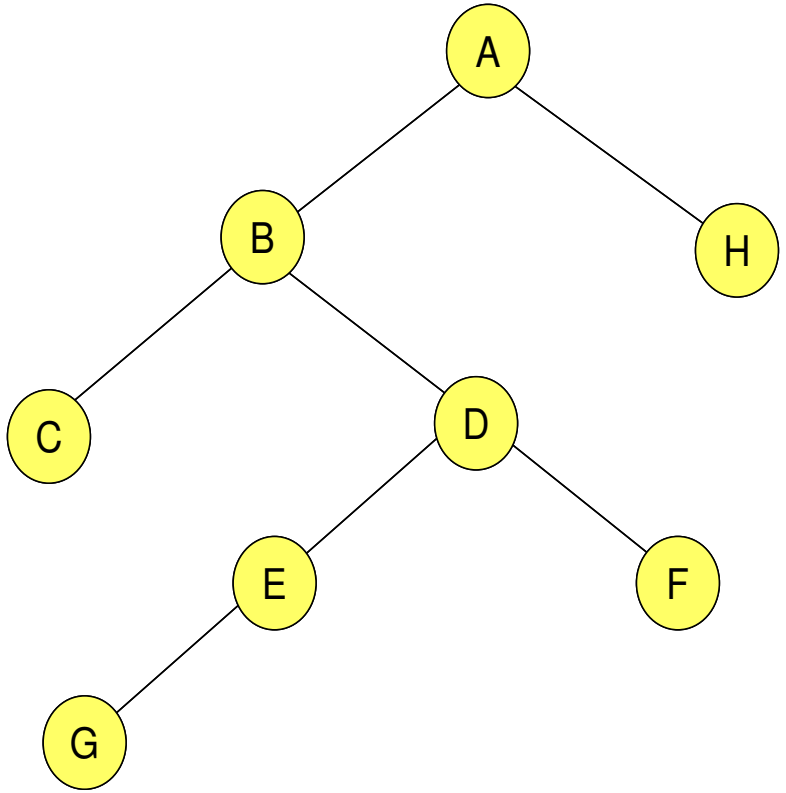
A→B→C→D→E→G→F→H

Implementasi dalam bahasa C

```
struct node {
    struct node *left;
    struct node *right;
    char        label;
}

void preorder(struct node *p)
{
    if (p==NULL)  return;       jika empty-tree, tidak perlu lakukan apa-apa
    printf("visit %c ", p->label);  tampilkan label node yang dikunjungi
    preorder(p->left);          traverse the left subtree
    preorder(p->right);         traverse the right subtree
}
```

Inorder

1. Traverse the left subtree
2. Visit the root
3. Traverse the right subtree



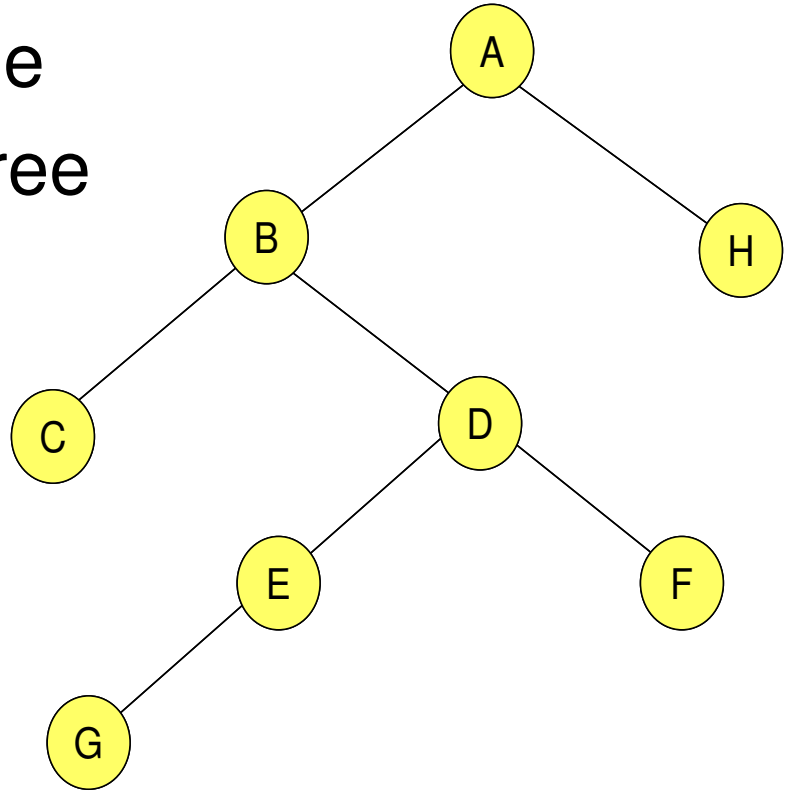
C→B→G→E→D→F→A→H

Implementasi dalam bahasa C

```
struct node {
    struct node *left;
    struct node *right;
    char        label;
}

void inorder(struct node *p)
{
    if (p==NULL)  return;       jika empty-tree, tidak perlu lakukan apa-apa
    inorder(p->left);           traverse the left subtree
    printf("visit %c ", p->label);  tampilkan label node yang dikunjungi
    inorder(p->right);          traverse the right subtree
}
```

Postorder

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root



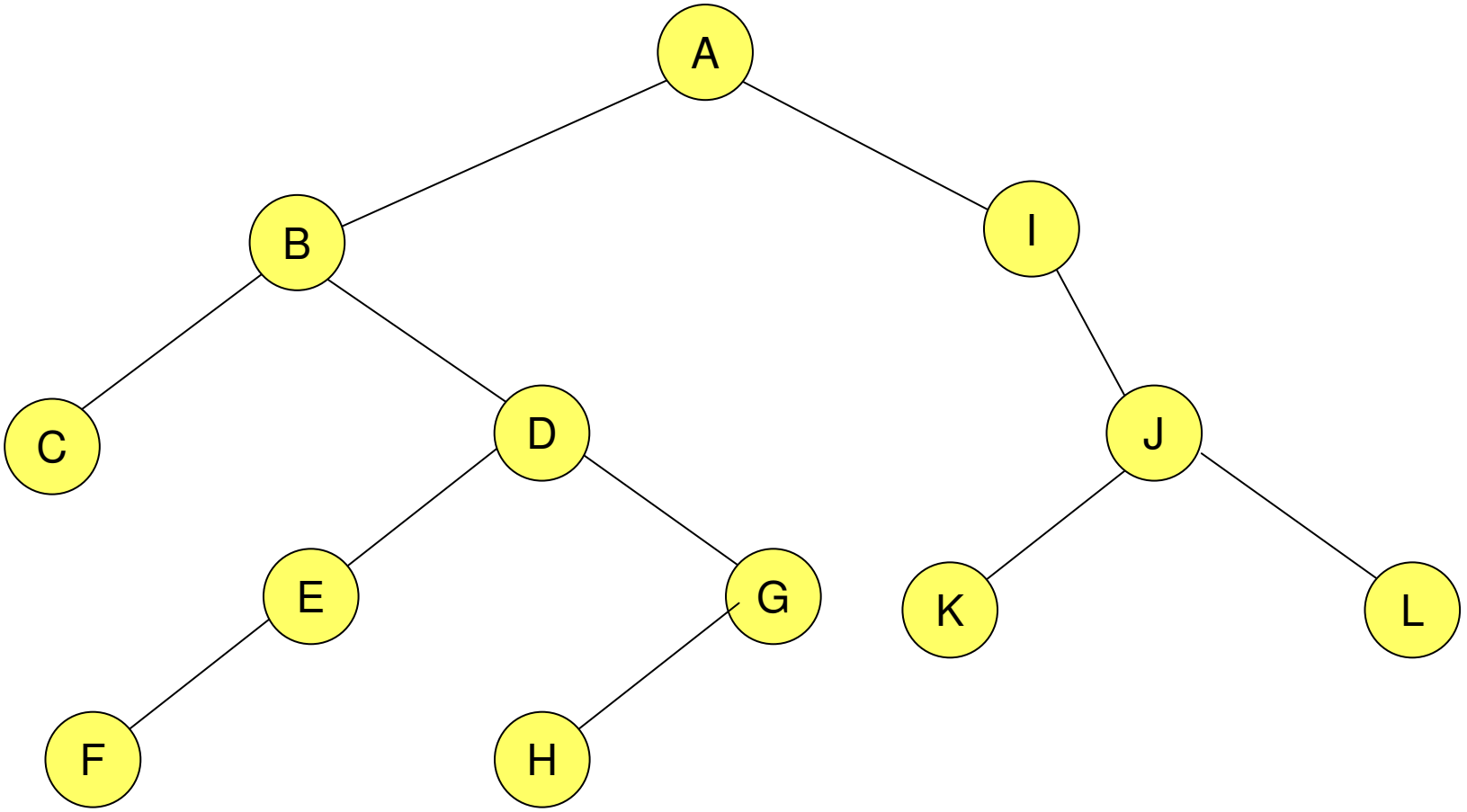
C→G→E→F→D→B→H→A

Implementasi dalam bahasa C

```
struct node {
    struct node *left;
    struct node *right;
    char        label;
}

void postorder(struct node *p)
{
    if (p==NULL)  return;       jika empty-tree, tidak perlu lakukan apa-apa
    postorder(p->left);           traverse the left subtree
    postorder(p->right);          traverse the right subtree
    printf("visit %c ", p->label);  tampilkan label node yang dikunjungi
}
```

Latihan-2

Tuliskan hasil preorder, inorder dan postorder traversal untuk binary tree berikut.



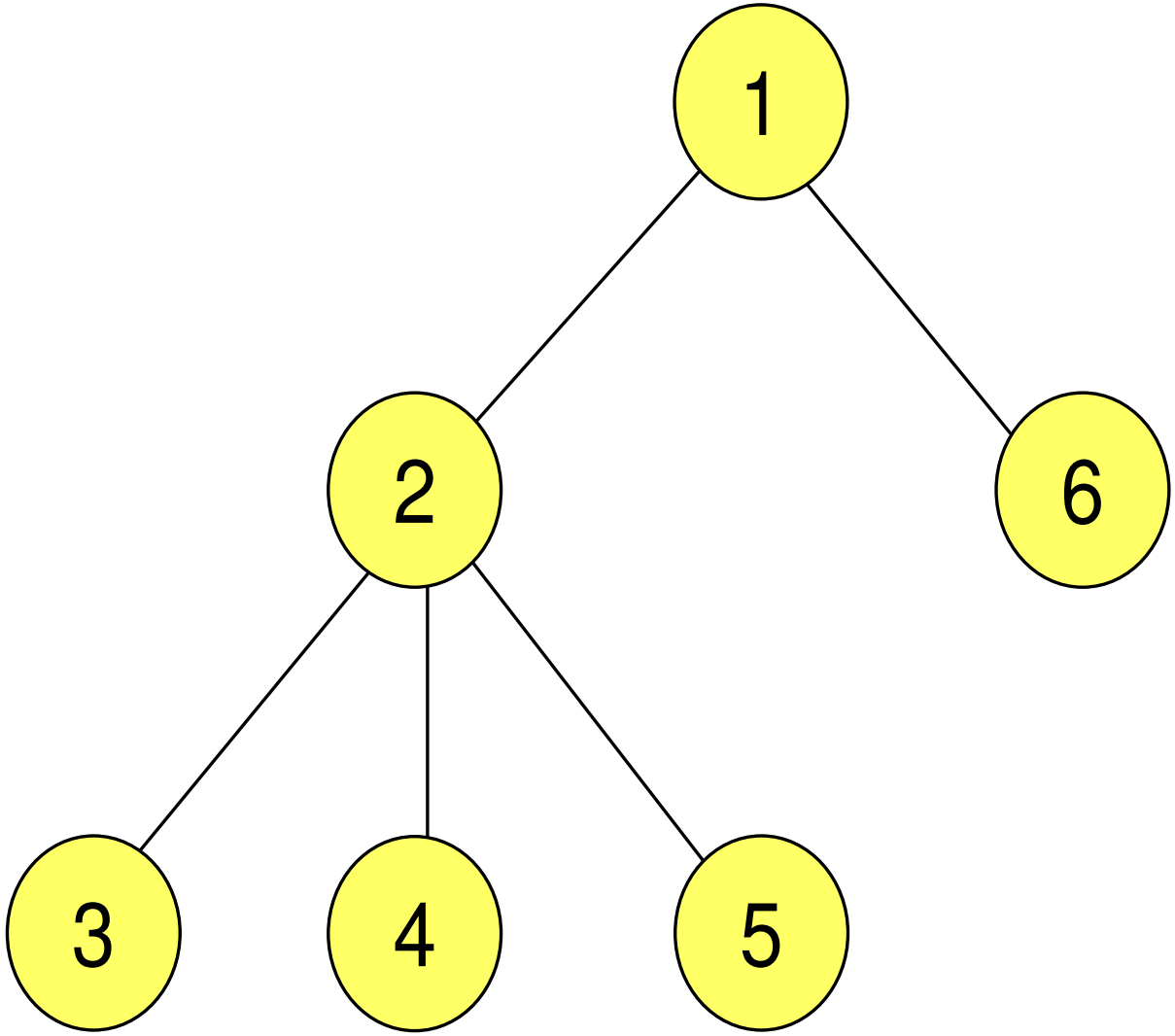
Outline

1. Apakah Tree Structure itu ?
2. Binary Tree & implementasinya
3. Tree Traversal
4. Implementasi tree (selain binary tree)

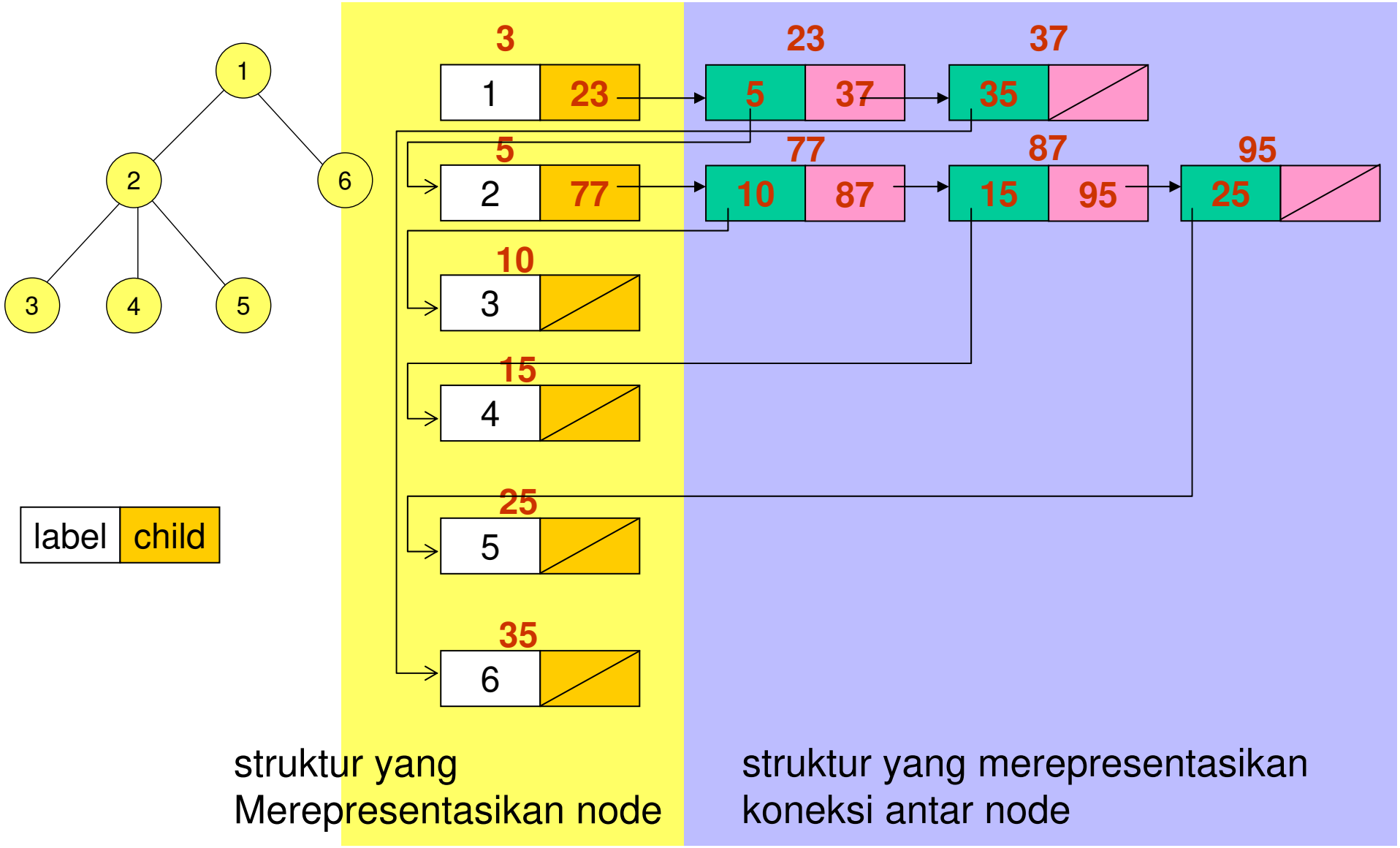
Teknik implementasi tree

- Binary tree hanya memiliki dua anak: kiri dan kanan. Karena itu implementasinya hanya memerlukan dua buah pointer untuk masing-masing subtree.
- Untuk implementasi tree yang memiliki sebarang anak, dapat dilakukan dengan dua cara
  - memakai linked-list
  - memakai binary tree

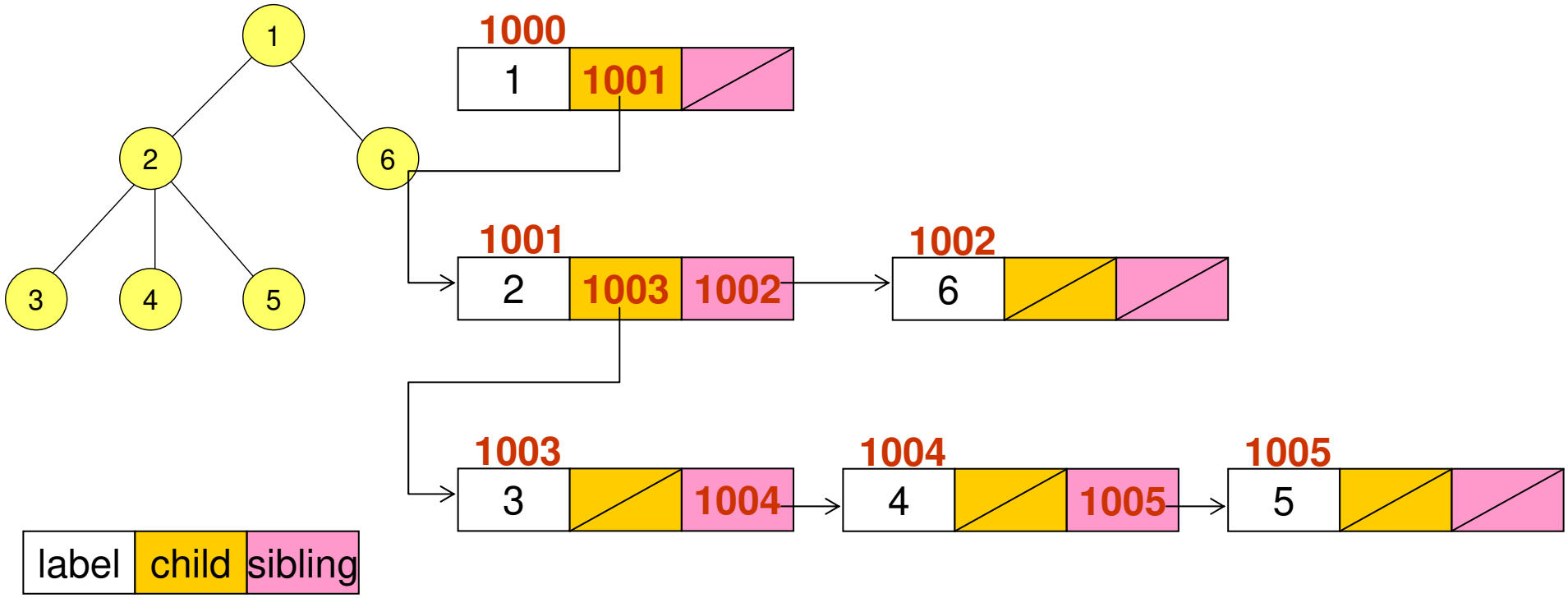
Contoh



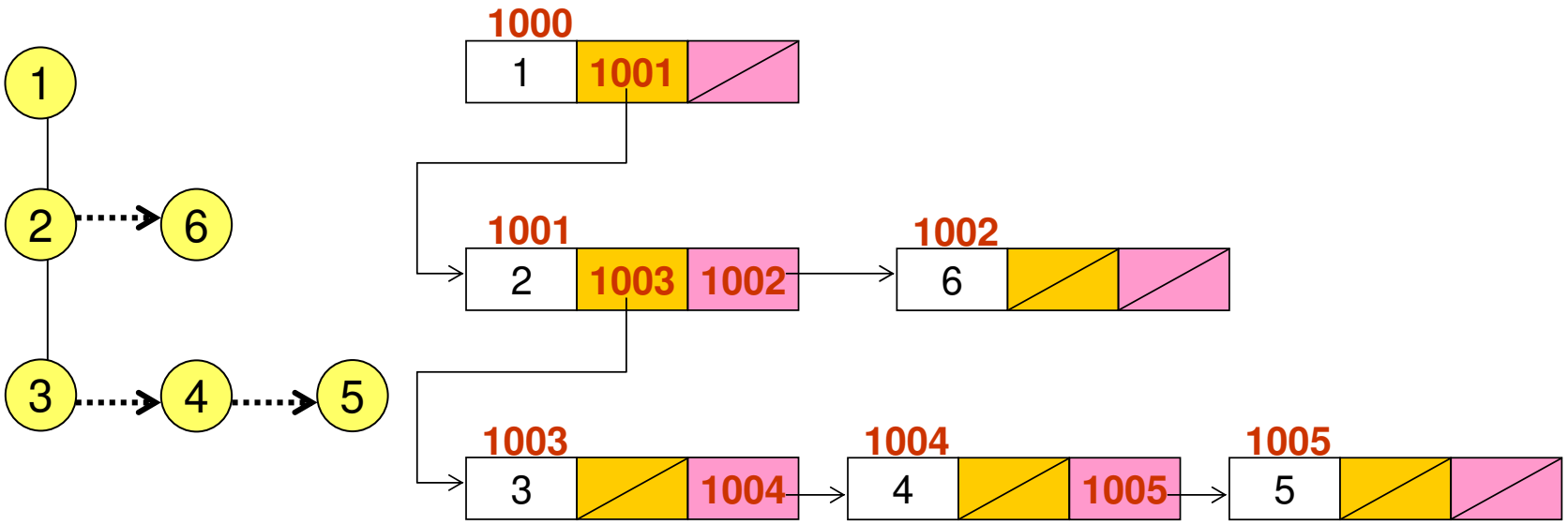
Implementasi memakai linked-list



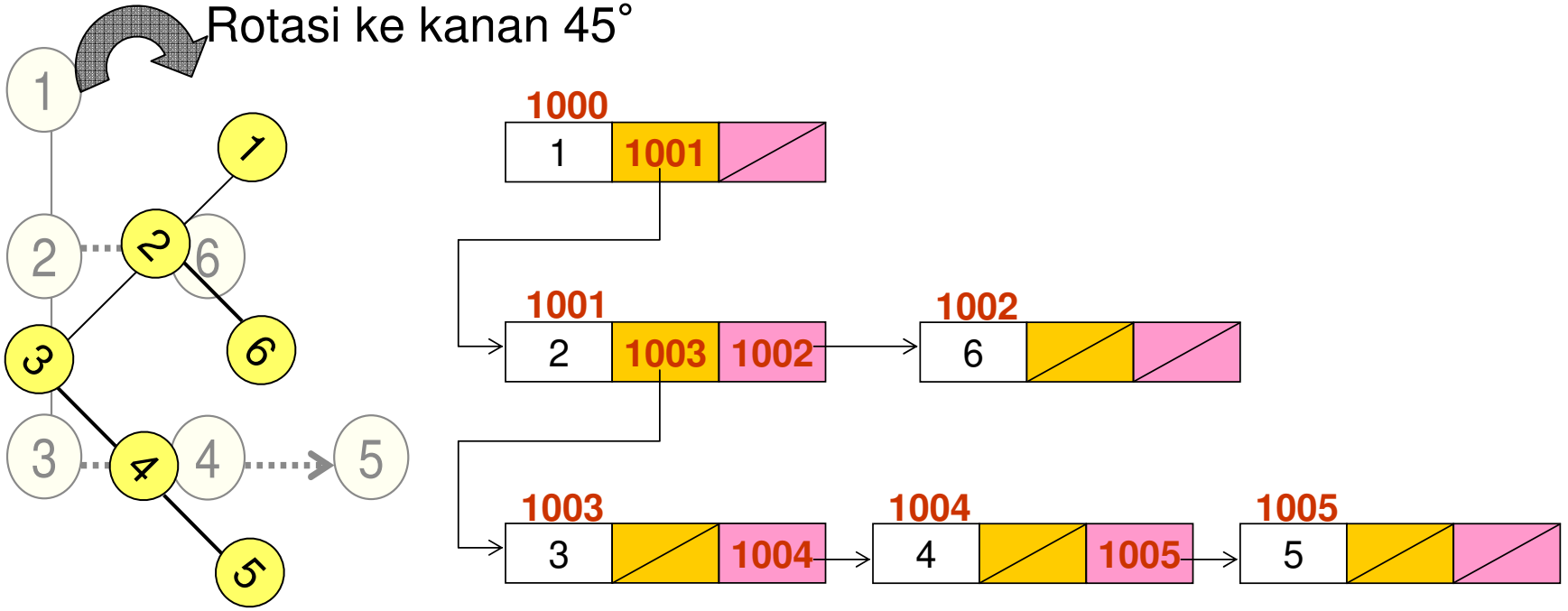
Implementasi memakai binary-tree



Implementasi memakai binary-tree



Implementasi memakai binary-tree

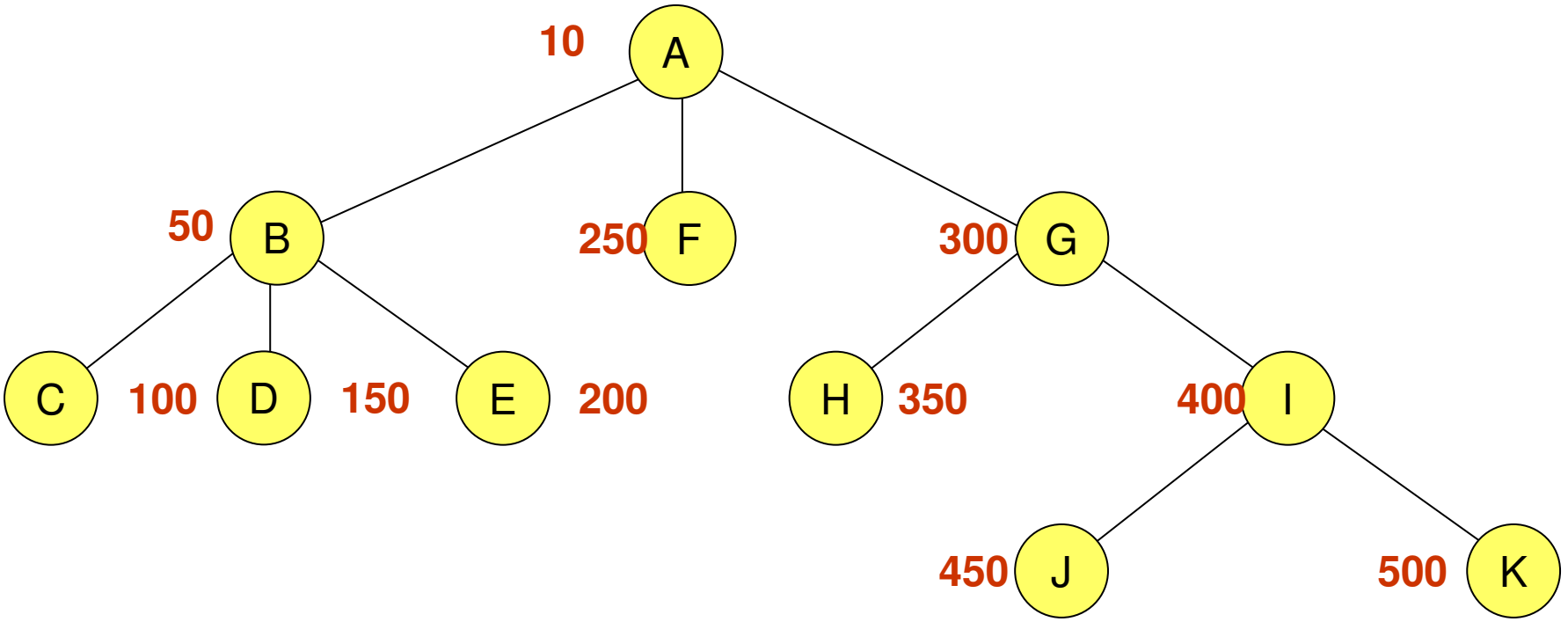


Latihan-3

Jelaskan implementasi tree berikut

a) memakai linked-list

b) memakai binary tree



Linear & Binary Search

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

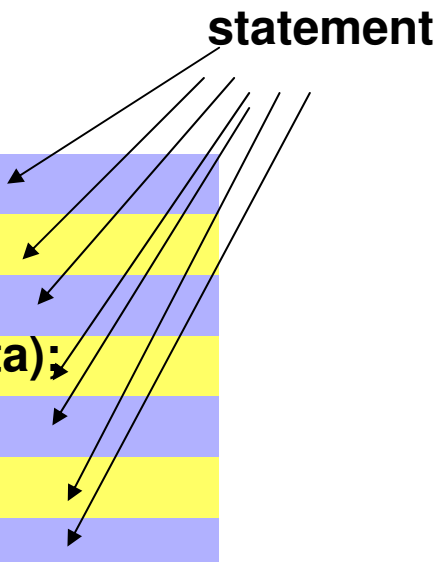
Agenda

- Linear Search
- Binary Search

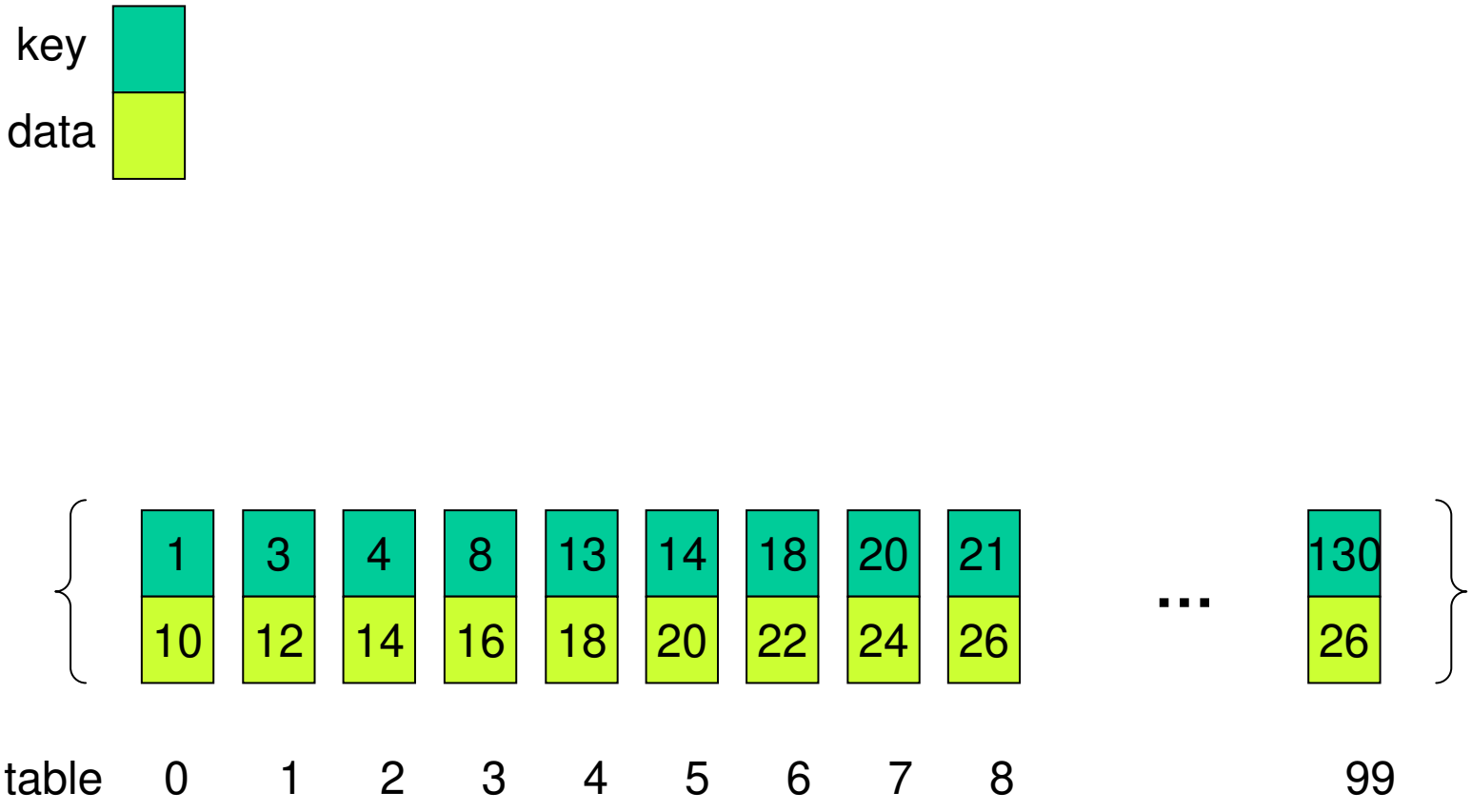
3 operasi:
- Penambahan data
- Pencarian data
- Penghapusan data

Linear Search

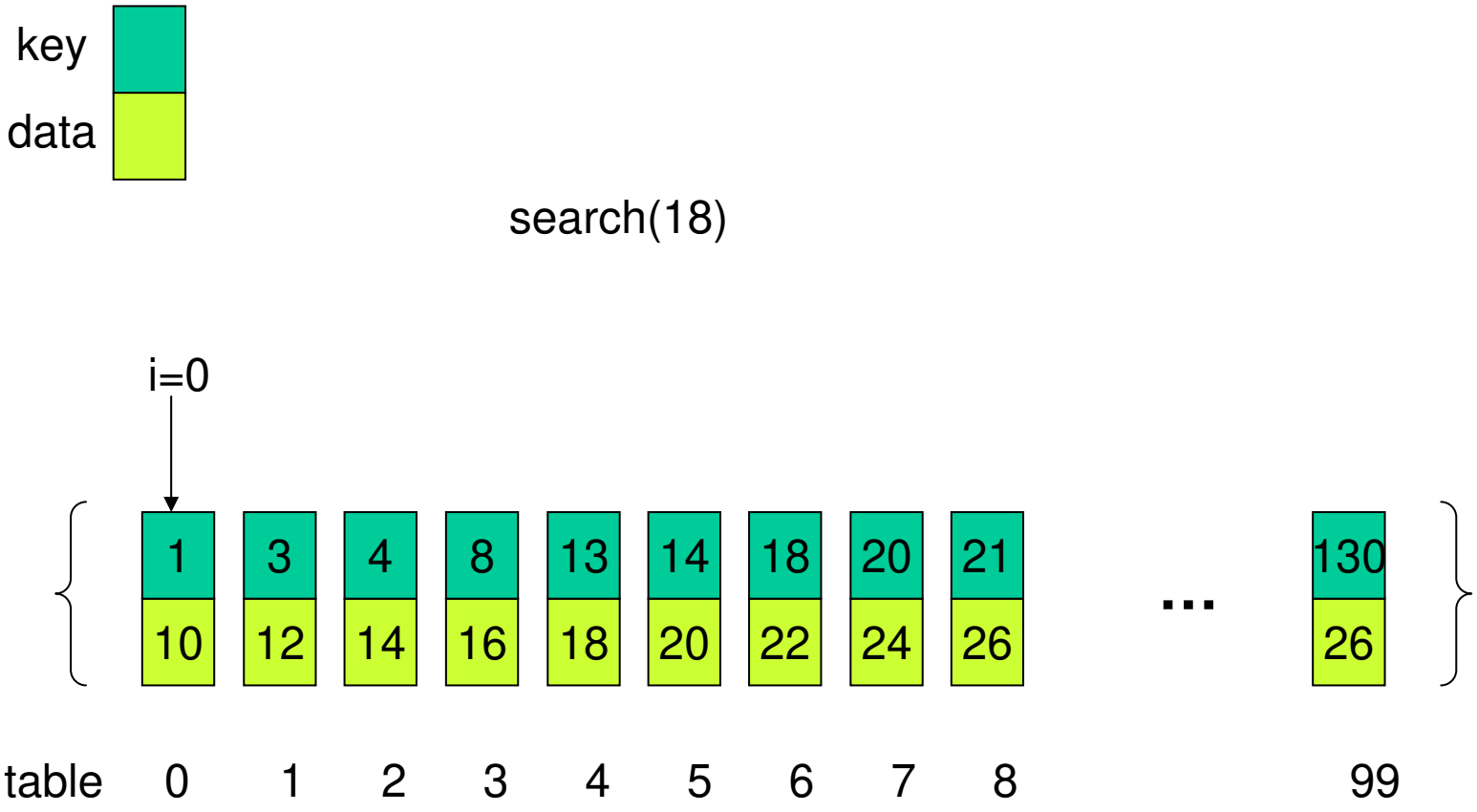
```
struct {
        int key;
        int data;
} table[100];
int  n;

int search(int key)
{
        int i;
(1)     i=0;
(2)     while (i < n){
(3)             if(table[i].key==key)
(4)                     return(table[i].data);
(5)             i++;
        }
(6)     return -1;
}
```



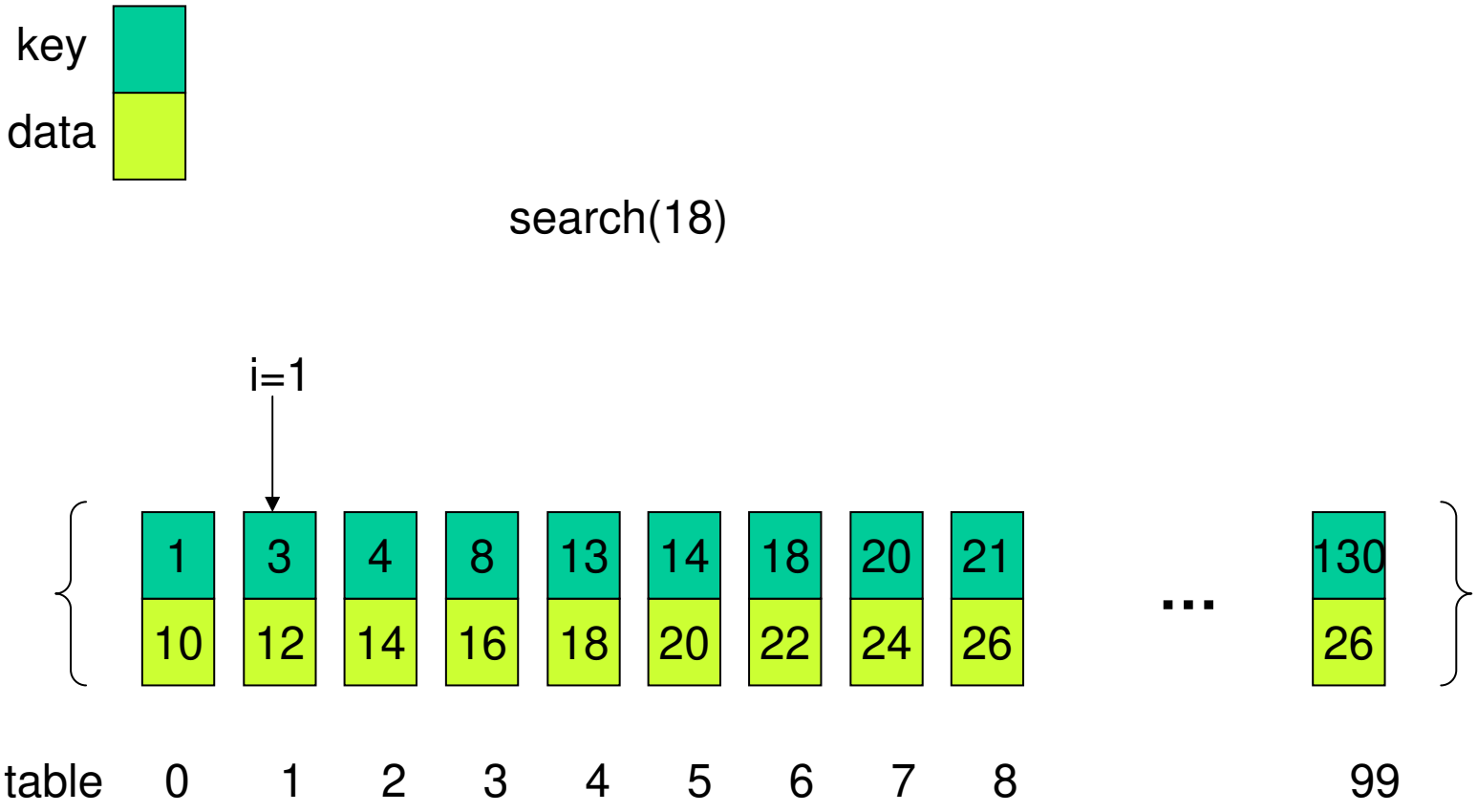
Cara kerja Linear Search



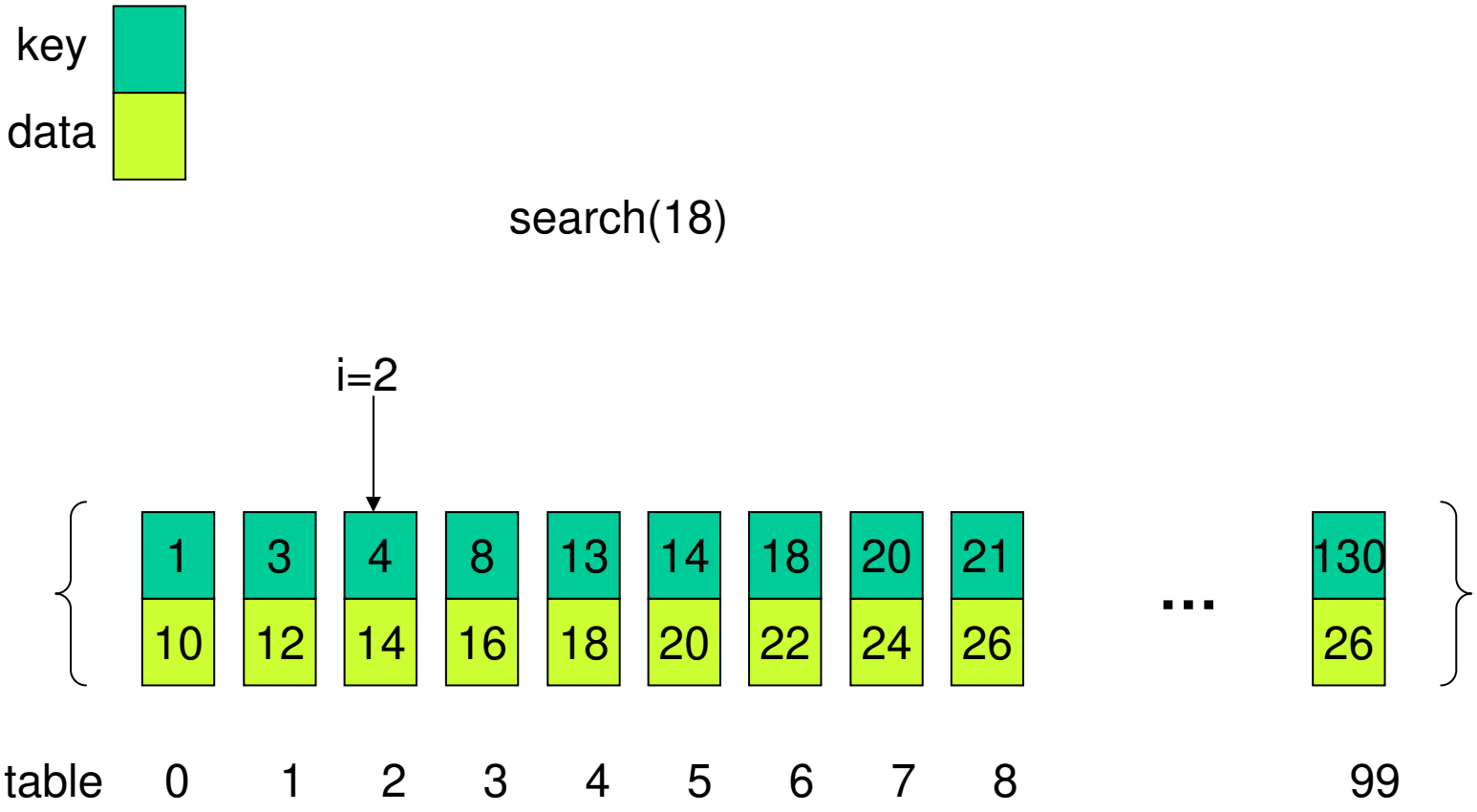
Cara kerja Linear Search



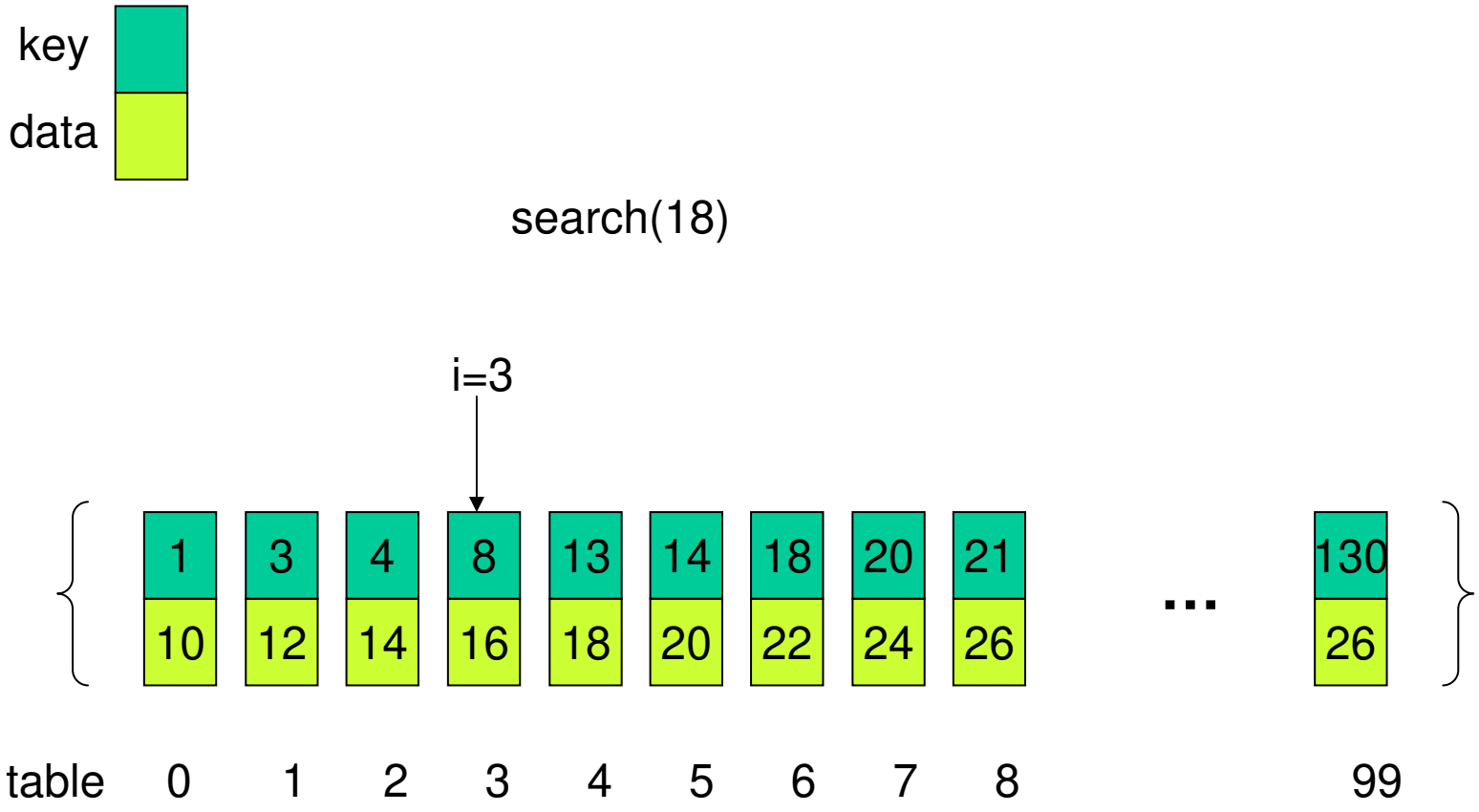
Cara kerja Linear Search



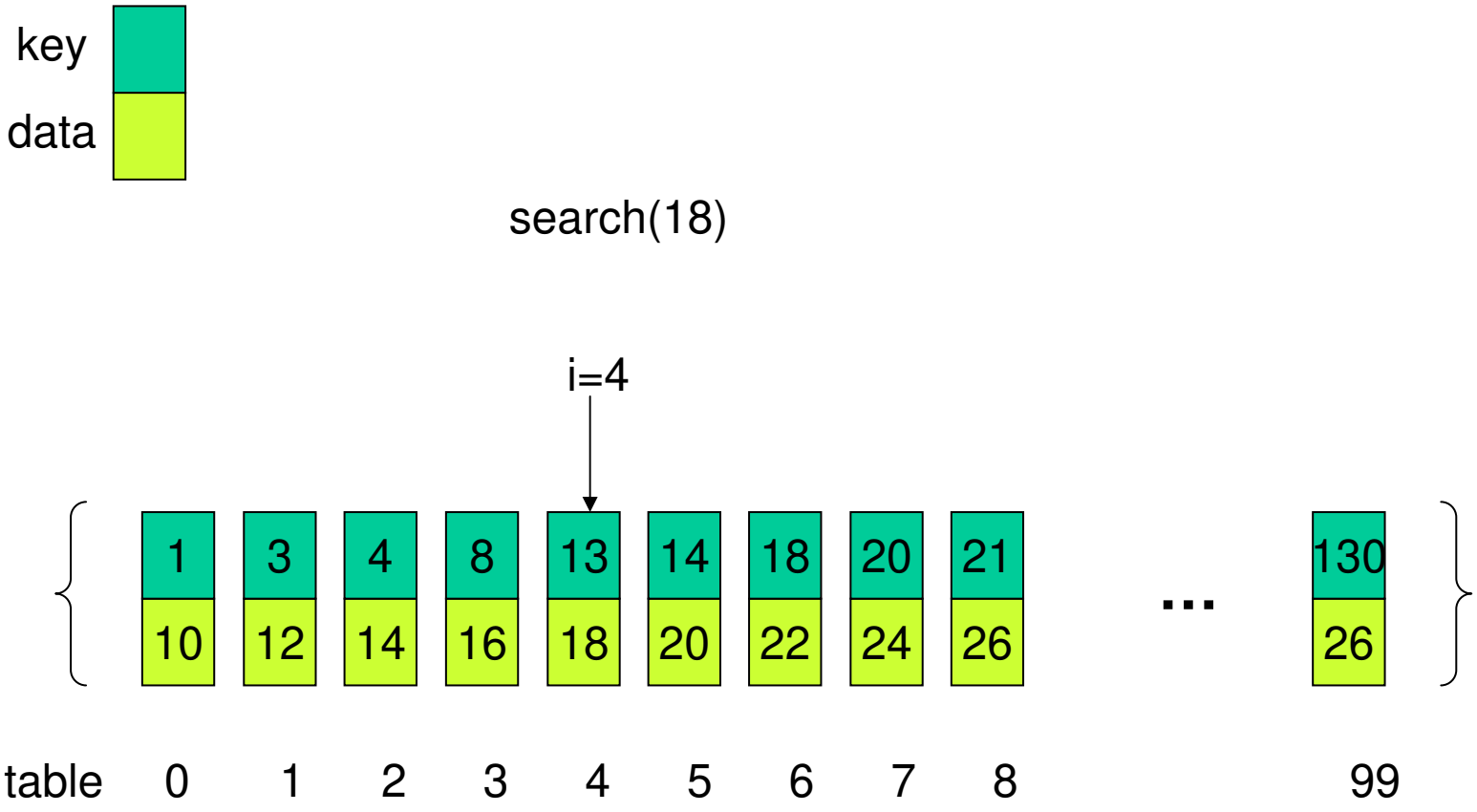
Cara kerja Linear Search



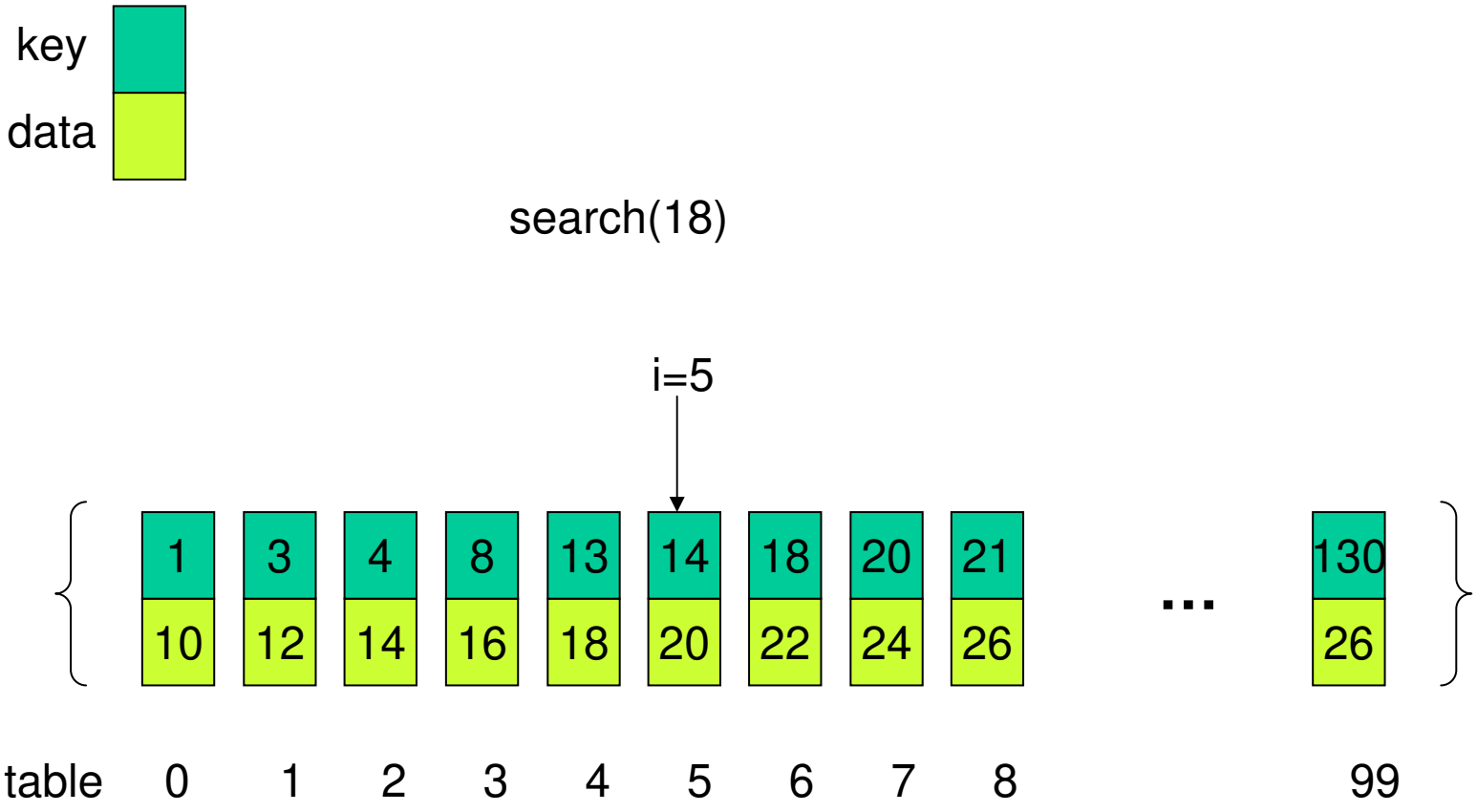
Cara kerja Linear Search



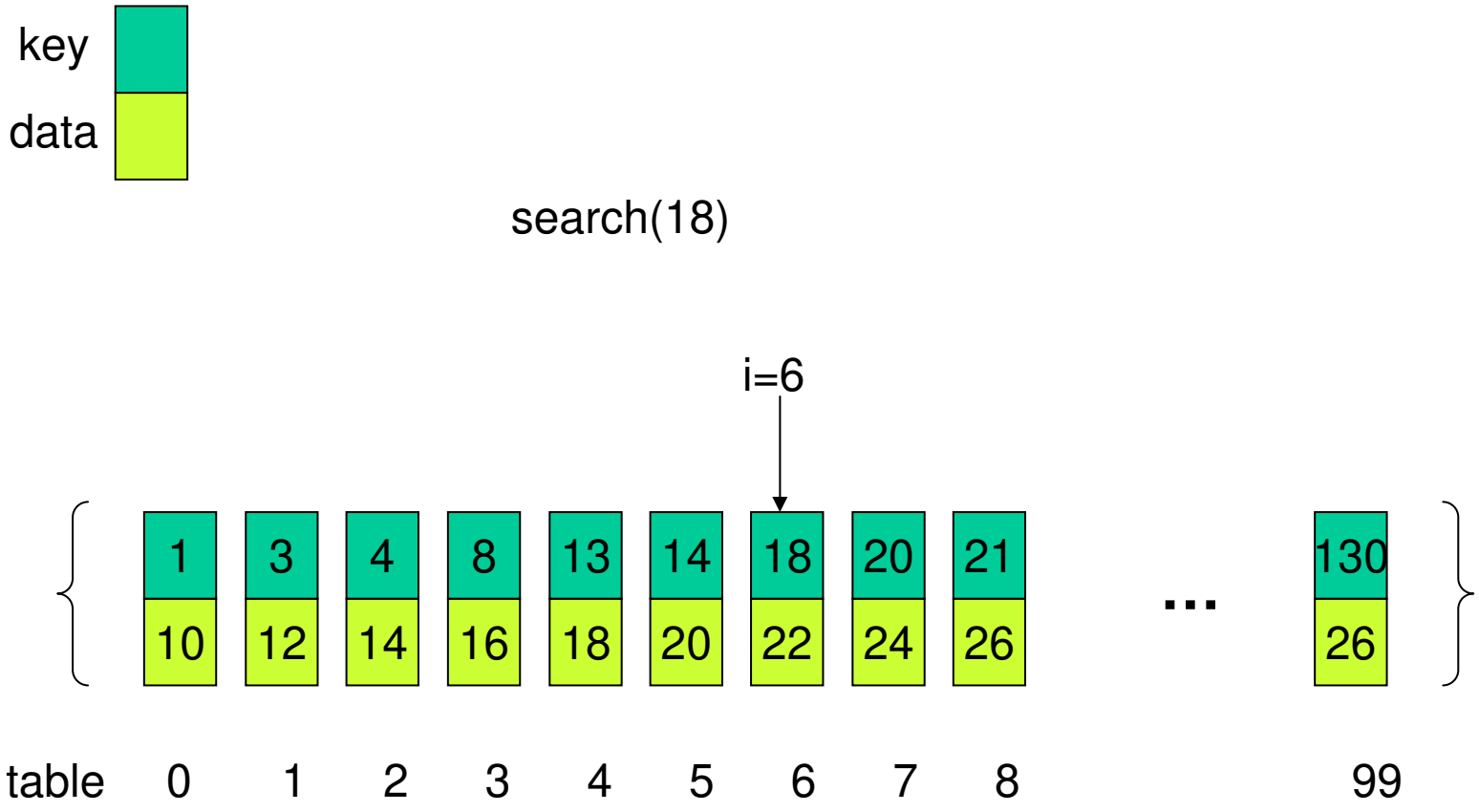
Cara kerja Linear Search



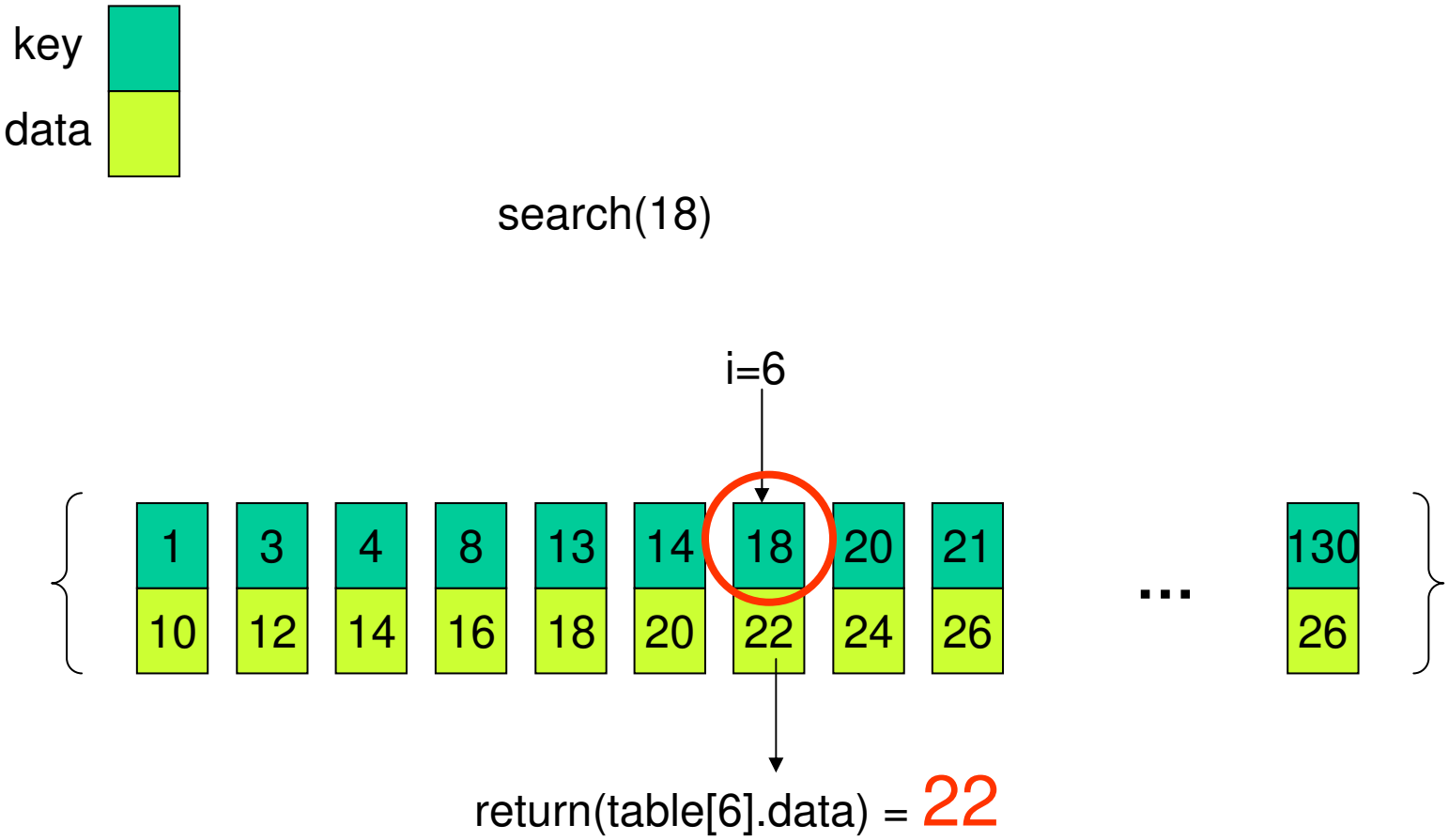
Cara kerja Linear Search



Cara kerja Linear Search



Cara kerja Linear Search



Complexity & banyaknya eksekusi untuk n data

Statement	Banyaknya eksekusi	complexity
(1)	1	O(1)
(2)	n/2	O(n)
(3)	n/2	O(n)
(4)	1	O(1)
(5)	n/2	O(n)
(6)	1	O(1)

Computational complexity:

$$O(1)+O(n)+O(n)+O(1)+O(n)+O(1)=O(\max(1,n,n,1,n,1))=O(n)$$

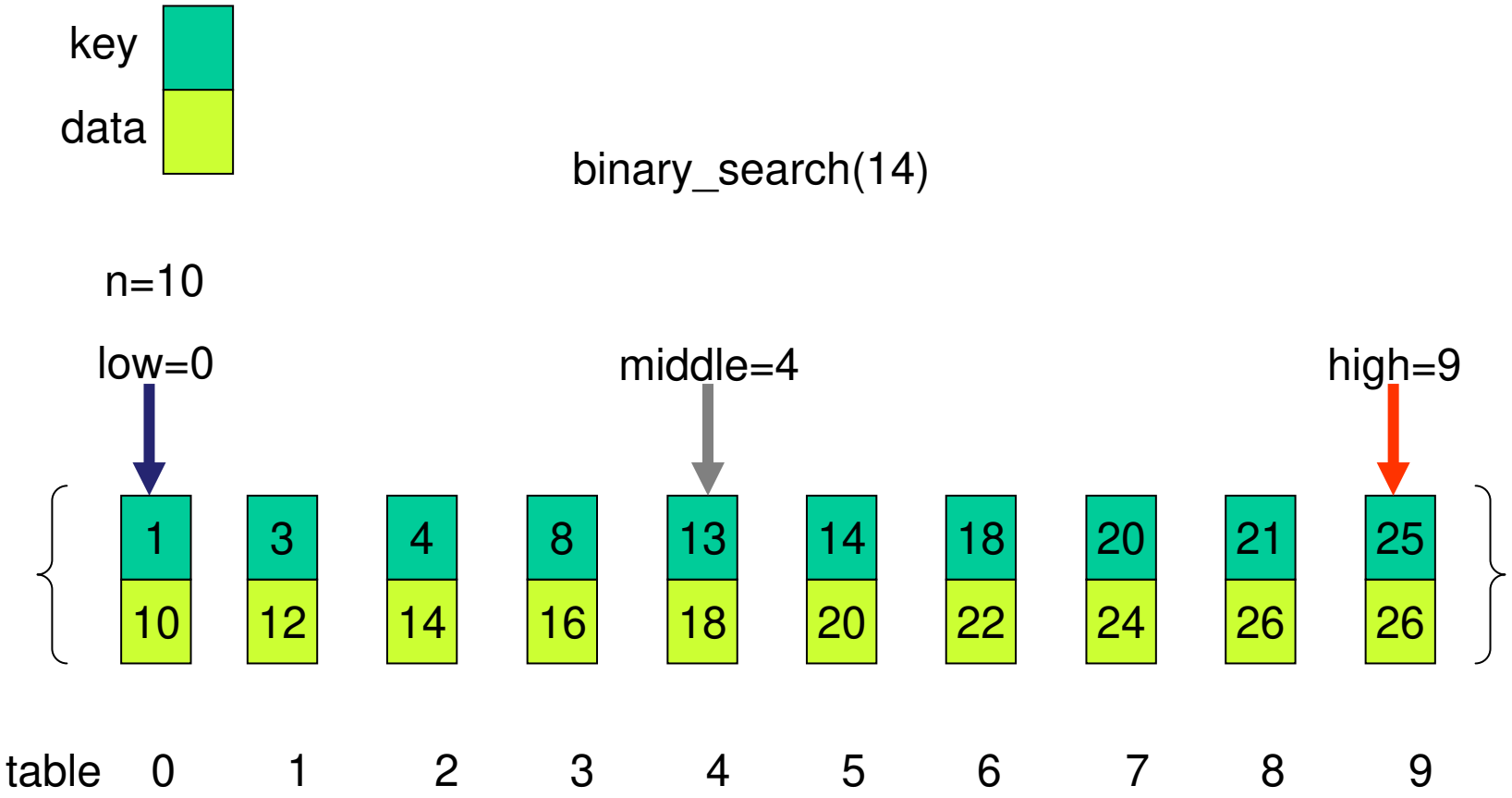
Diskusi

- Hitunglah computational complexity penambahan data
- Hitunglah computational complexity penghapusan data

Binary Search

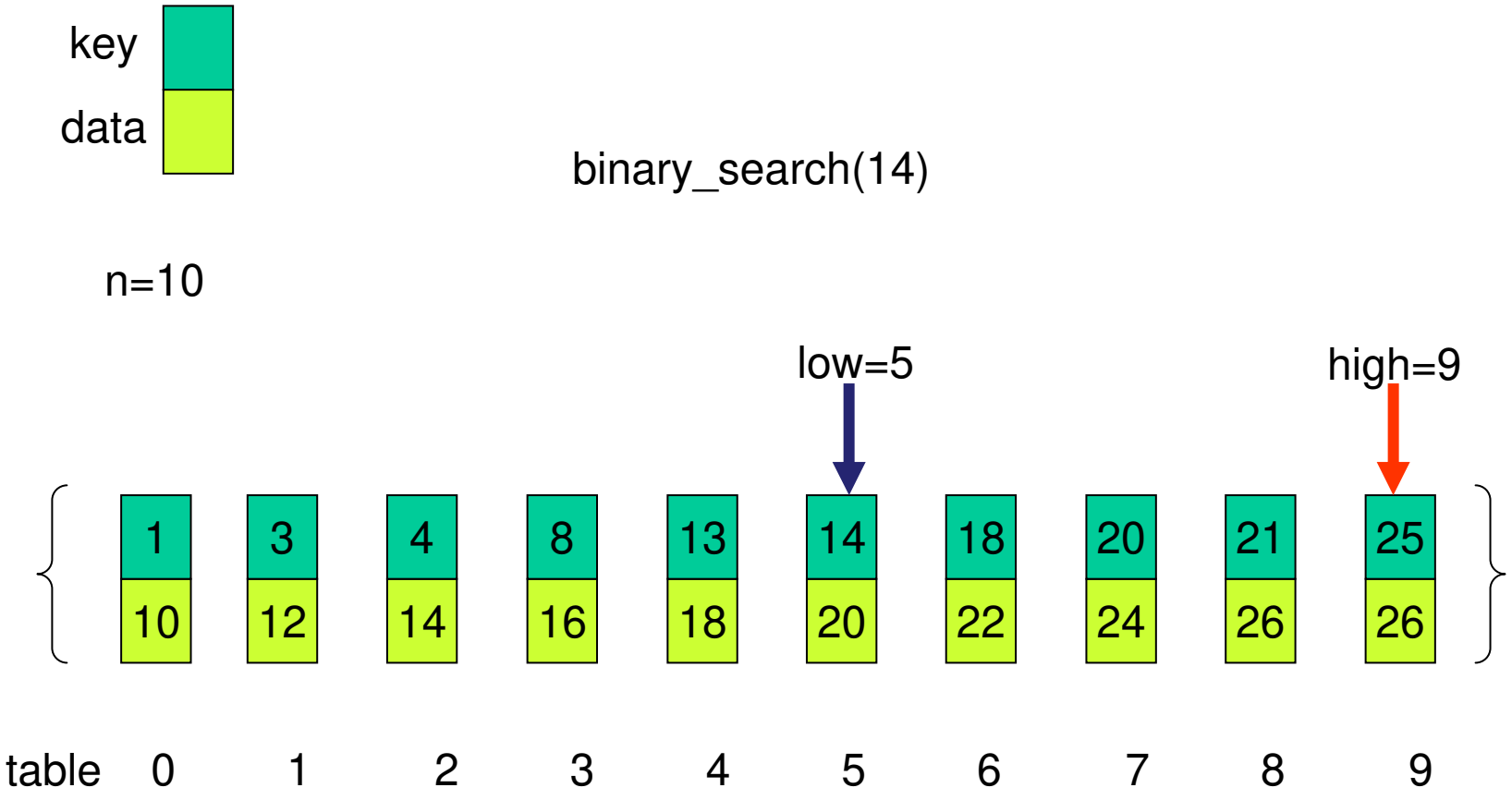
```
int binary_search(int key)
{
        int low, high, middle;
(1)     low  = 0;
(2)     high = n-1;
(3)     while (low <= high){
(4)             middle = (low+high) / 2;
(5)             if (key == table[middle].key)
(6)                     return(table[middle].data)
(7)             else if(key <  table[middle].key)
(8)                     high = middle – 1;
(9)             else
(10)                    low = middle + 1;
                        }
(11)    return -1;
}
```

Cara kerja Binary Search



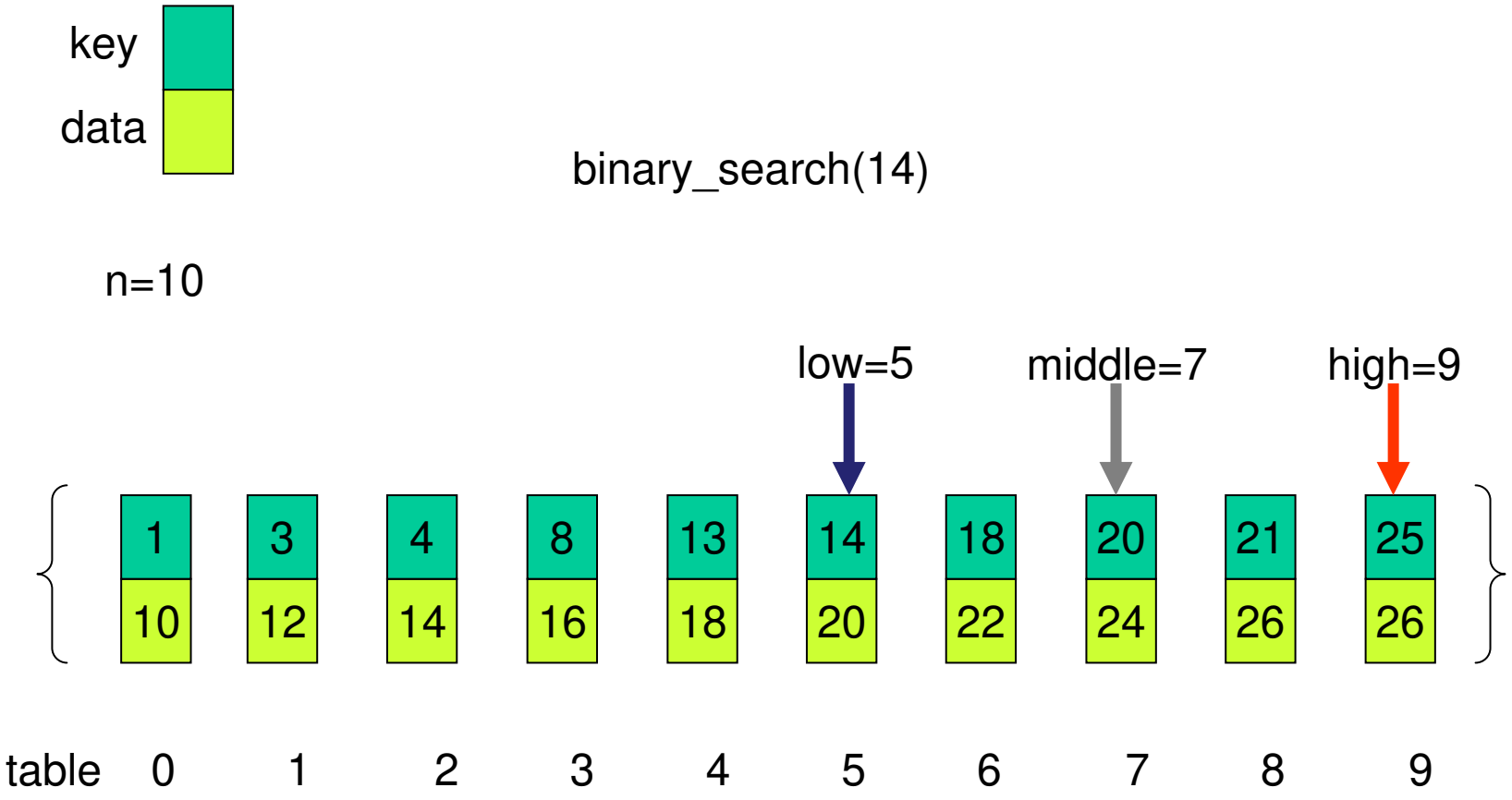
Syarat Binary Search: Data harus sudah terurut (sorted)

Cara kerja Binary Search



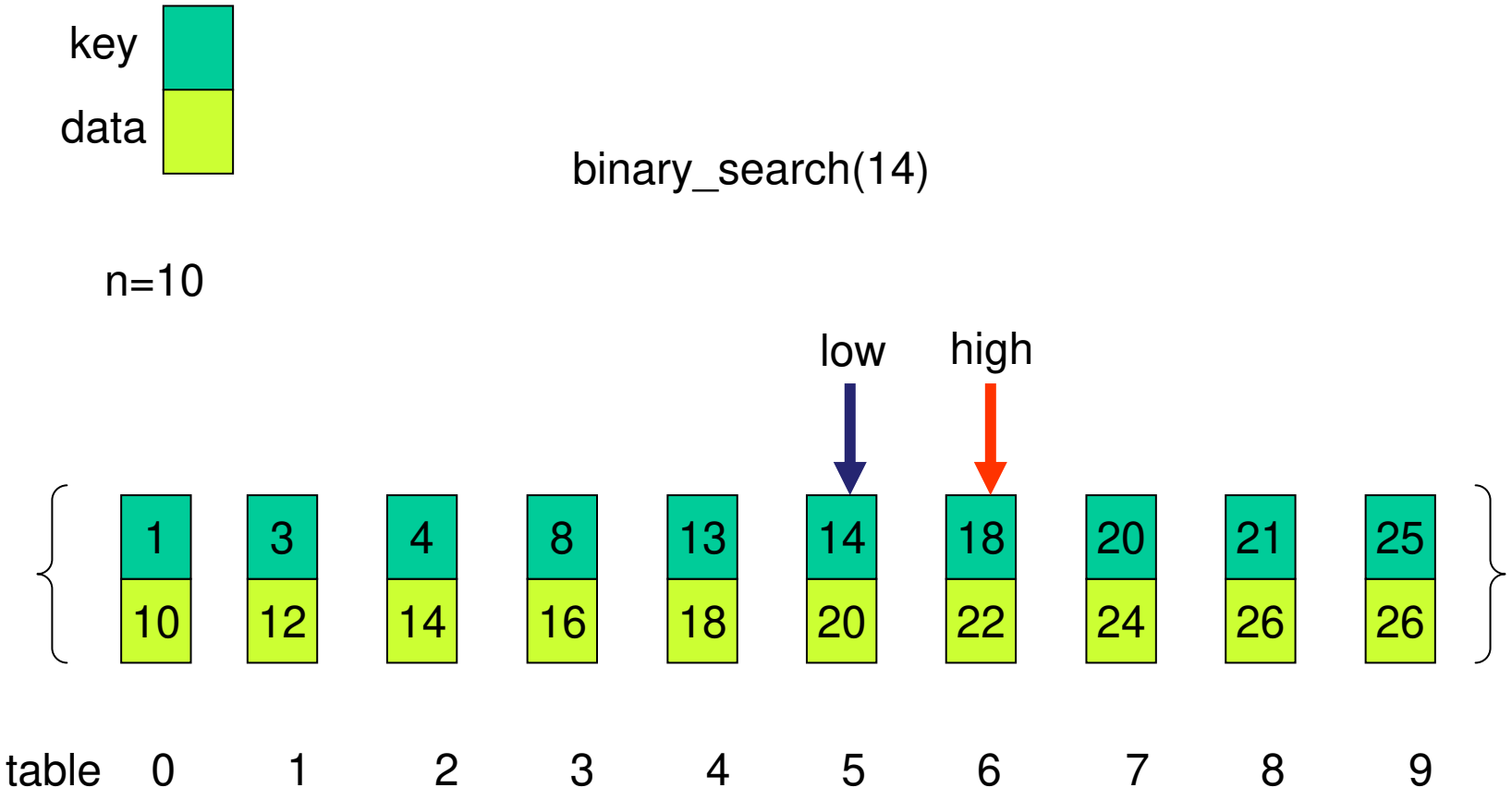
Syarat Binary Search: Data harus sudah terurut (sorted)

Cara kerja Binary Search



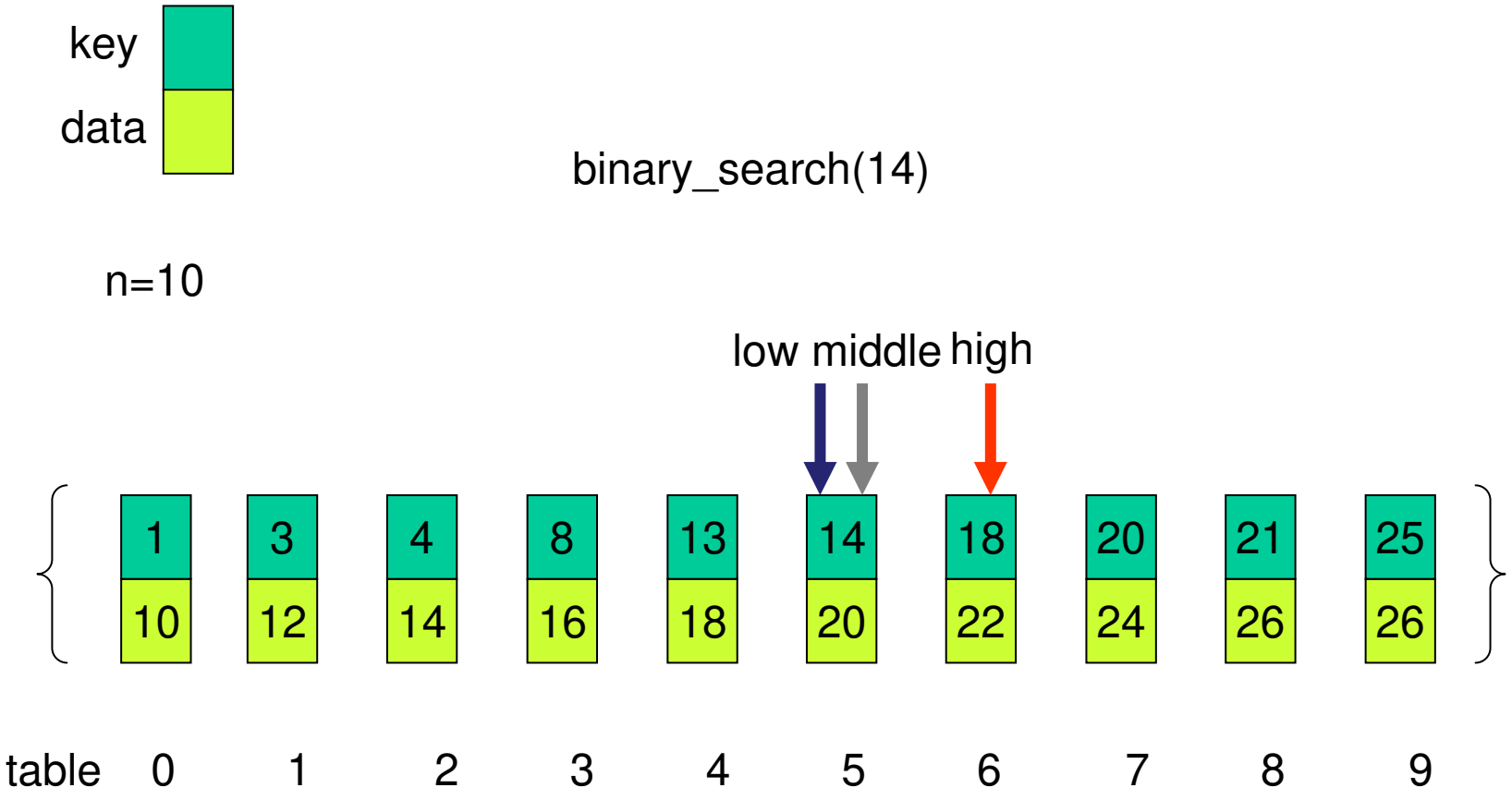
Syarat Binary Search: Data harus sudah terurut (sorted)

Cara kerja Binary Search



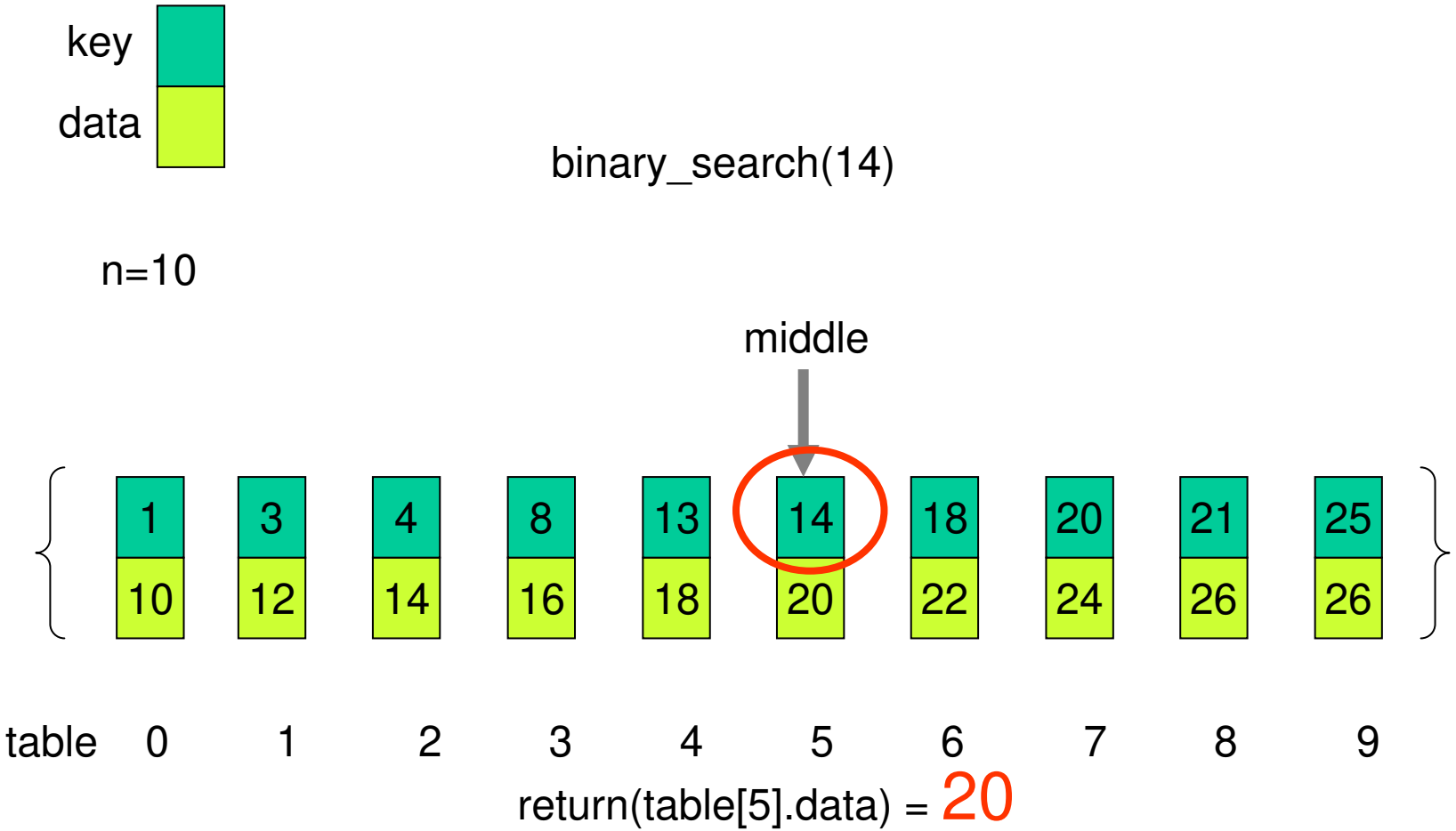
Syarat Binary Search: Data harus sudah terurut (sorted)

Cara kerja Binary Search



Syarat Binary Search: Data harus sudah terurut (sorted)

Cara kerja Binary Search



Syarat Binary Search: Data harus sudah terurut (sorted)

Computational Complexity

Statement	Complexity
(1)～(2)	O(1)
(3) ～(10)	O(log n)
(11)	O(1)

Loop ke-1→area pencarian berkurang menjadi 1/2
Loop ke-2→area pencarian berkurang menjadi 1/4
Loop ke-3→area pencarian berkurang menjadi 1/8
Loop ke-k→area pencarian berkurang menjadi 2^{-k}

pada saat itu banyaknya data adalah $\frac{n}{2^k}$

Berarti bila suatu saat area pencarian menjadi 1, maka pada saat itu berapakah nilai k ?

$$O(1)+O(\log n)+O(1)=O(\max(1,\log n,1))=O(\log n)$$

Computational Complexity

Loop ke-1→area pencarian berkurang menjadi 1/2
Loop ke-2→area pencarian berkurang menjadi 1/4
Loop ke-3→area pencarian berkurang menjadi 1/8
Loop ke-k→area pencarian berkurang menjadi 2^{-k}

pada saat itu banyaknya data adalah $\frac{n}{2^k}$

Bila suatu saat banyaknya data di area pencarian menjadi 1, maka pada saat itu berapakah nilai k ?

$$\frac{n}{2^k} = 1$$

$$2^k = n$$

$$k = \log n$$

Computational Complexity

Statement	Complexity
(1)～(2)	O(1)
(3) ～(10)	O(log n)
(11)	O(1)

Berarti bila suatu saat area pencarian menjadi 1, maka pada saat itu berapakah nilai k ?
Jawab: log(n)

Computational complexity pencarian pada Binary Search :

$$O(1)+O(\log n)+O(1)=O(\max(1,\log n,1))=O(\log n)$$

Penambahan Data pada Linear Search

Penambahan Data pada Linear Search

Menggeser data dengan nilai lebih besar ke belakang

Penambahan Data pada Linear Search

Complexity Penambahan & Pencarian Data

	Penambahan Data (untuk tiap 1 data)	Pencarian Data (untuk tiap 1 data)
Linear Search	$O(1)$	$O(n)$
Binary Search	$O(n)$	$O(\log n)$
Hash	$O(1)$	$O(1)$

Bagaimana dengan penghapusan data ?

Binary Search Tree

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Apakah Binary Search Tree itu ?

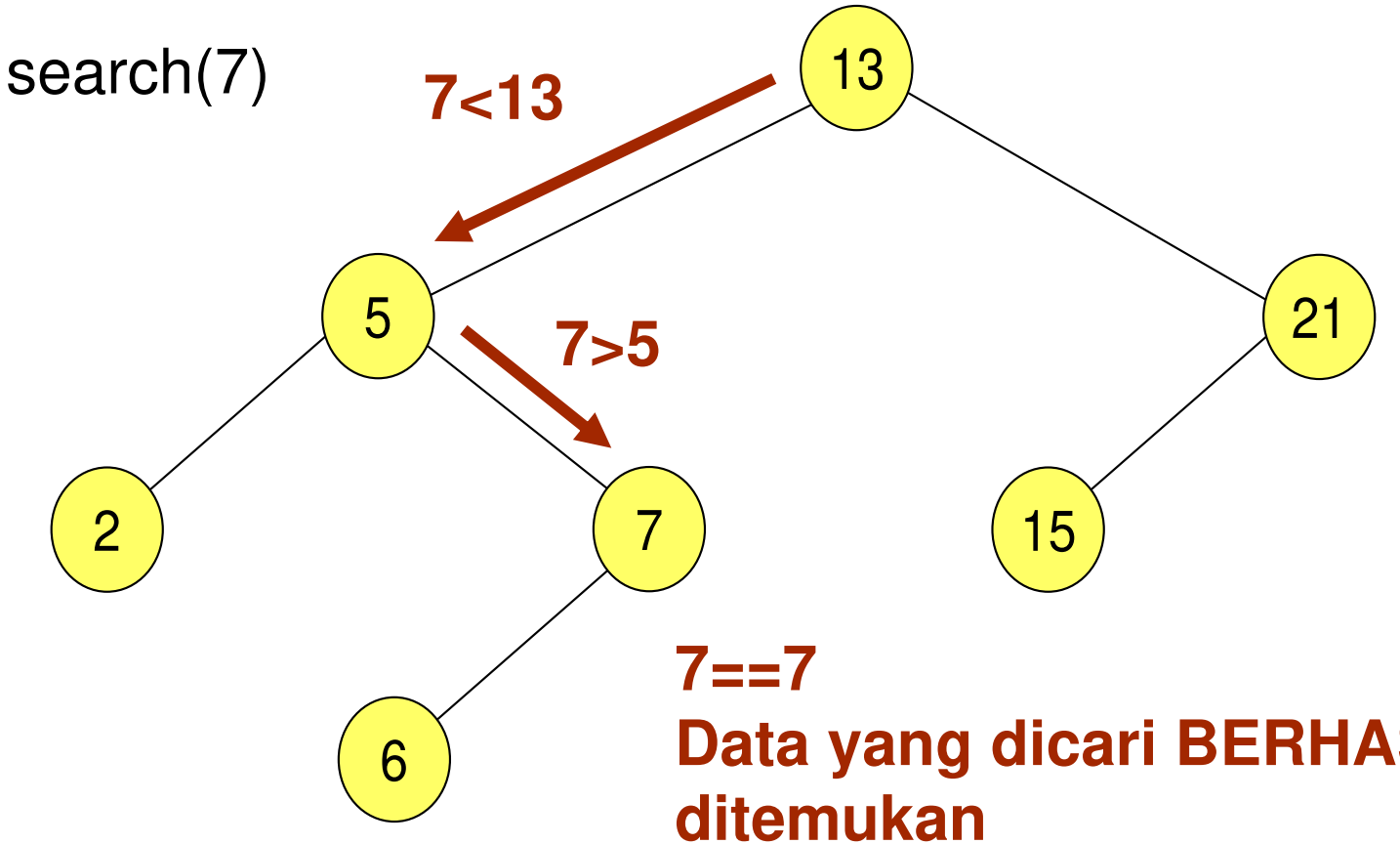
- Pemakaian tree structure dalam proses pencarian (search)
- Contoh: Binary Search Tree, AVL Tree, 2-3 Tree, B Tree, B* tree, dsb
- Binary Tree:

  Pada sebuah node x,

  elemen yang berada di LEFT sub-tree selalu lebih KECIL daripada x

  elemen yang berada di RIGHT sub-tree selalu lebih BESAR daripada x
- Binary Search Tree: proses pencarian (SEARCHING) berbasis binary tree

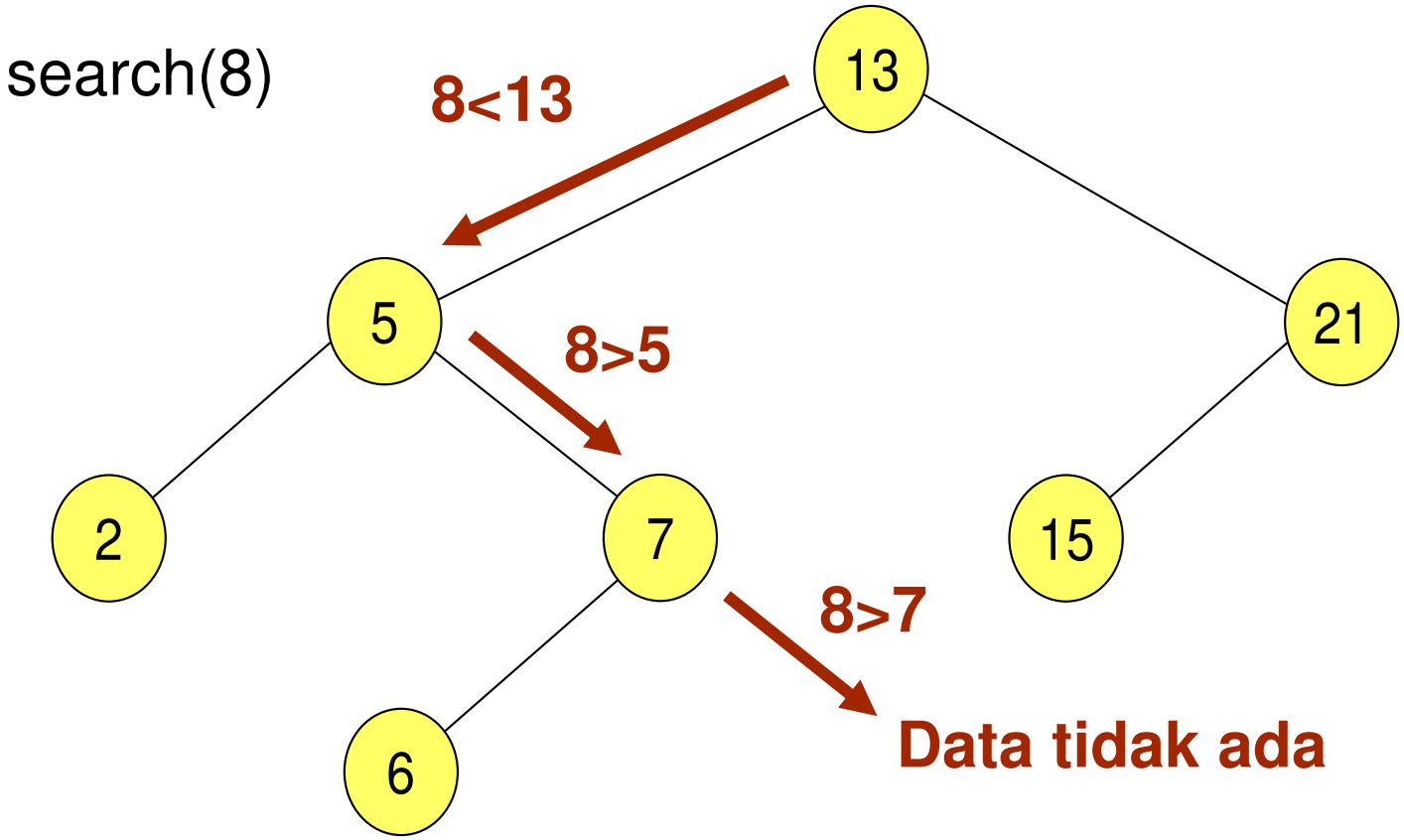
Proses Pencarian (contoh SUKSES)



7==7
Data yang dicari BERHASIL ditemukan

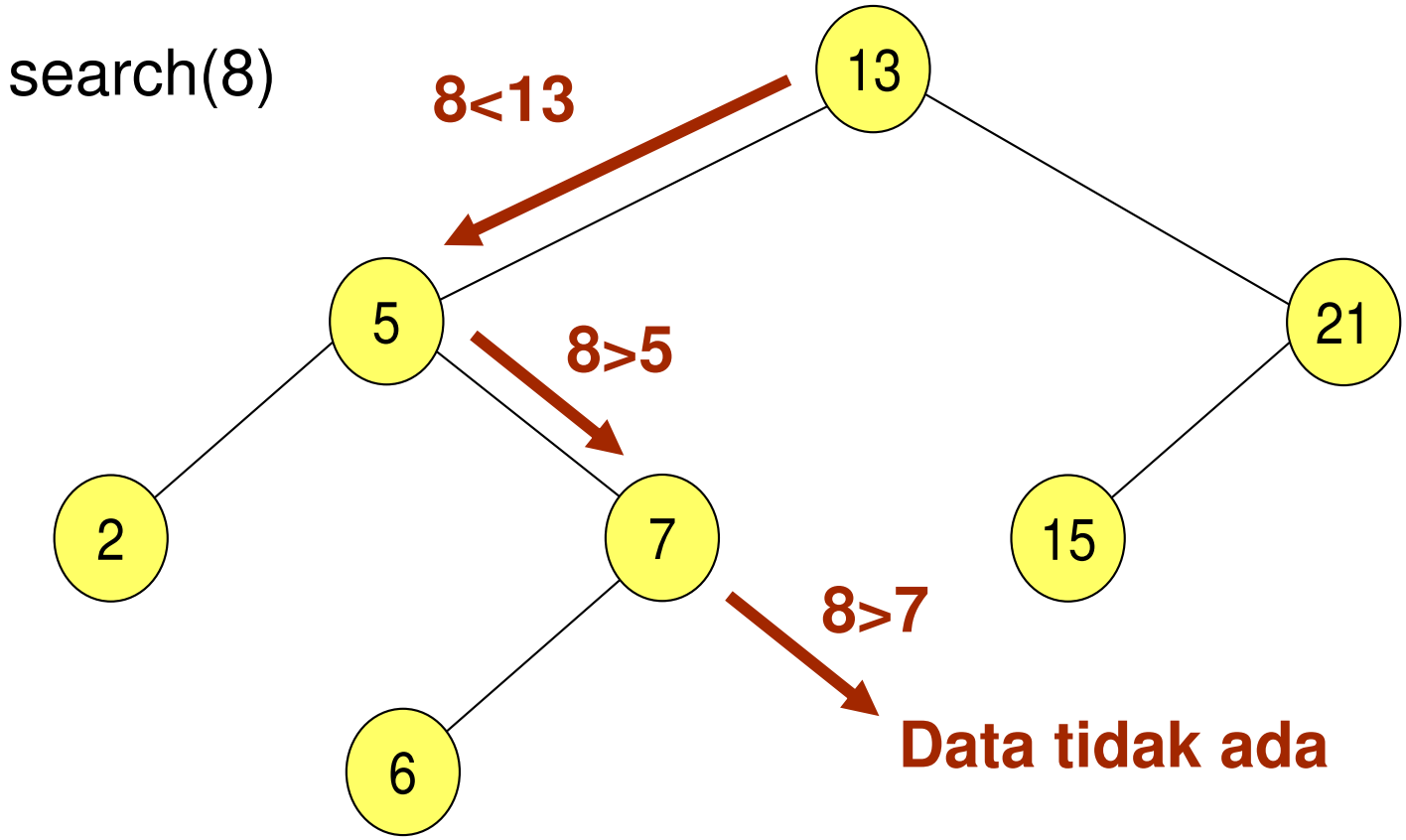
13→5 →7

Proses Pencarian (contoh GAGAL)



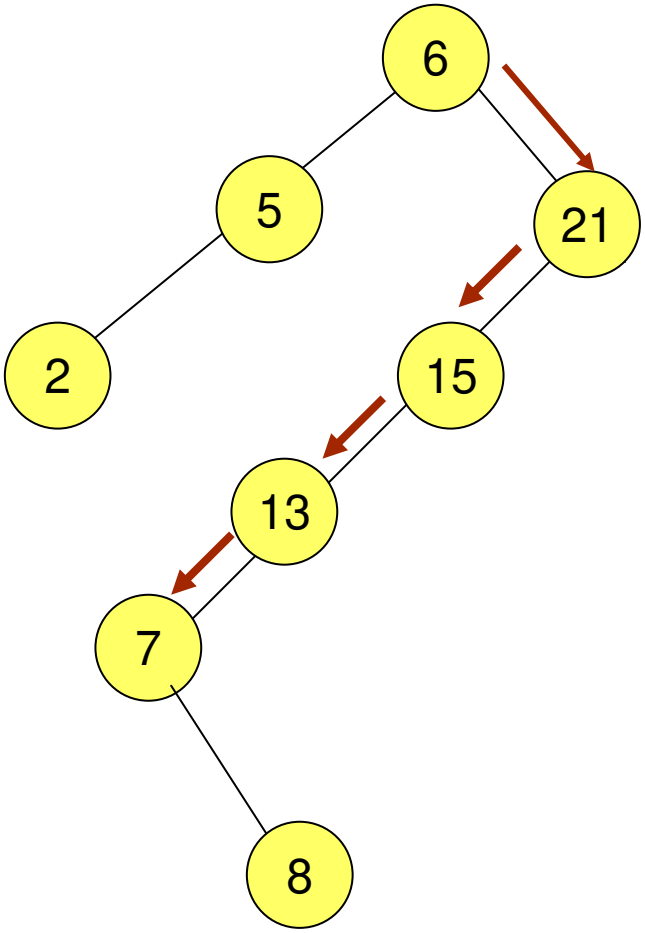
Data yang dicari GAGAL ditemukan

Proses Pencarian (contoh GAGAL)



Data yang dicari GAGAL ditemukan

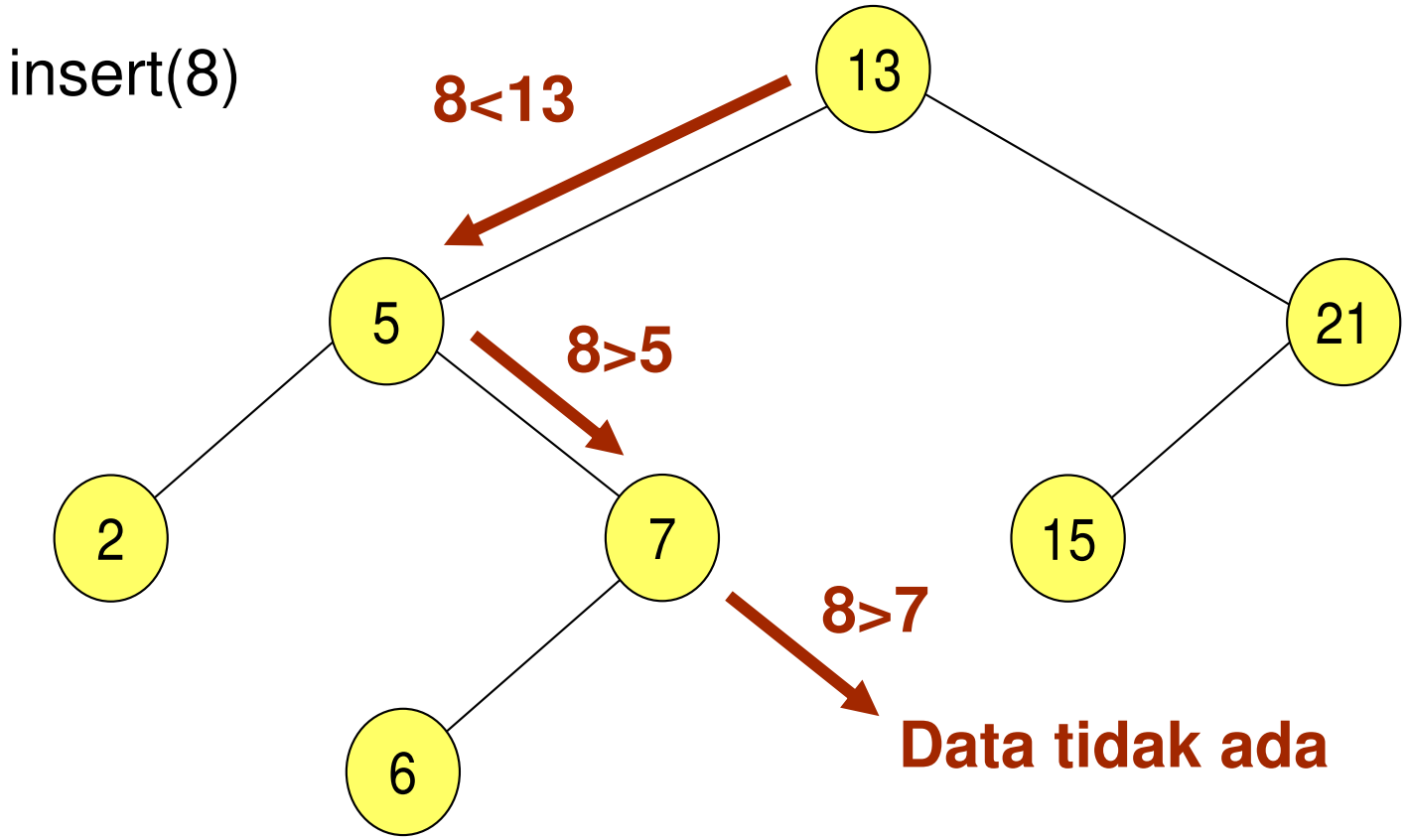
Proses Pencarian



search(7)

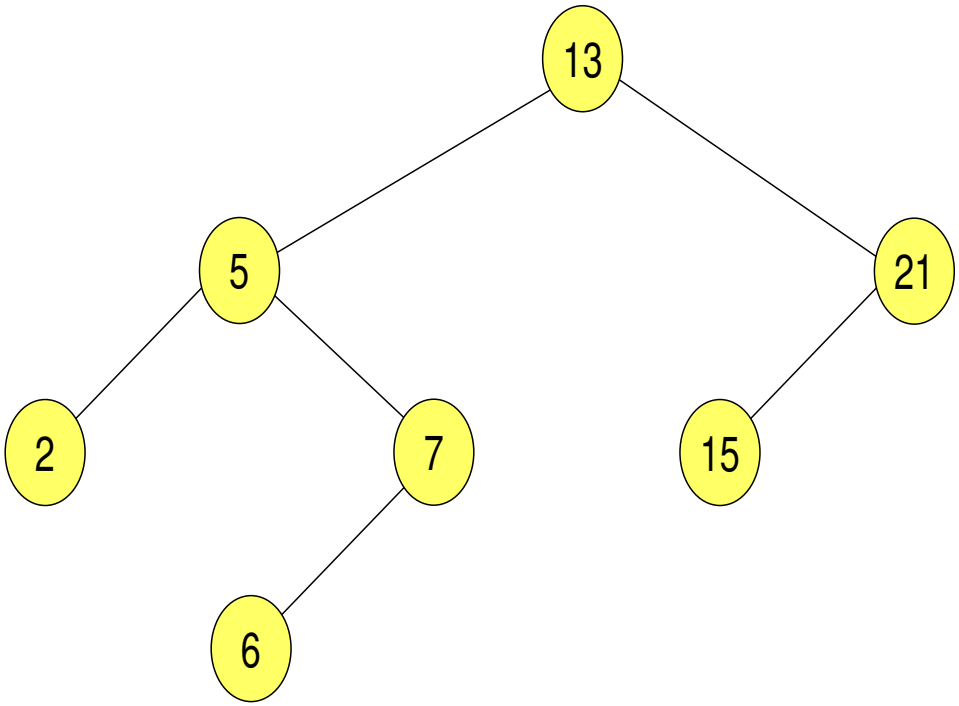
6→21 →15→13 →7

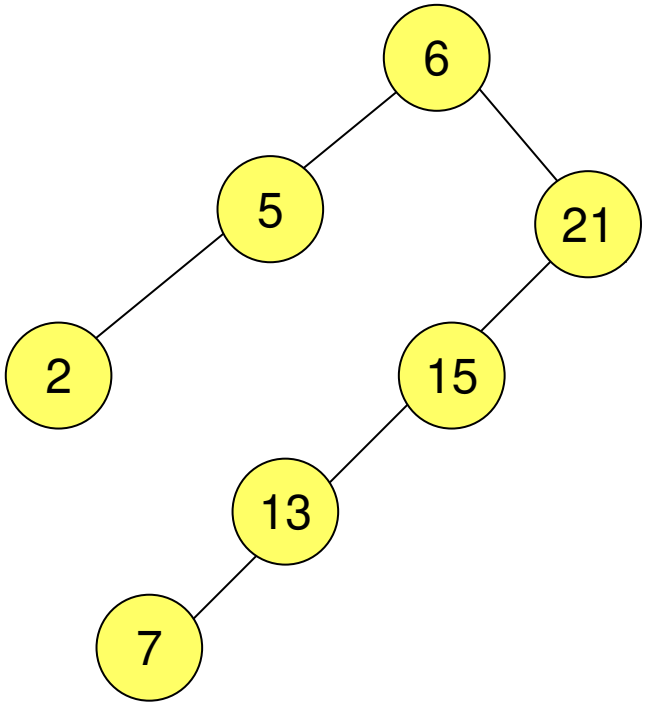
Proses Penambahan (INSERTION)



Data yang dicari GAGAL ditemukan = disitulah data ditambahkan (insertion)

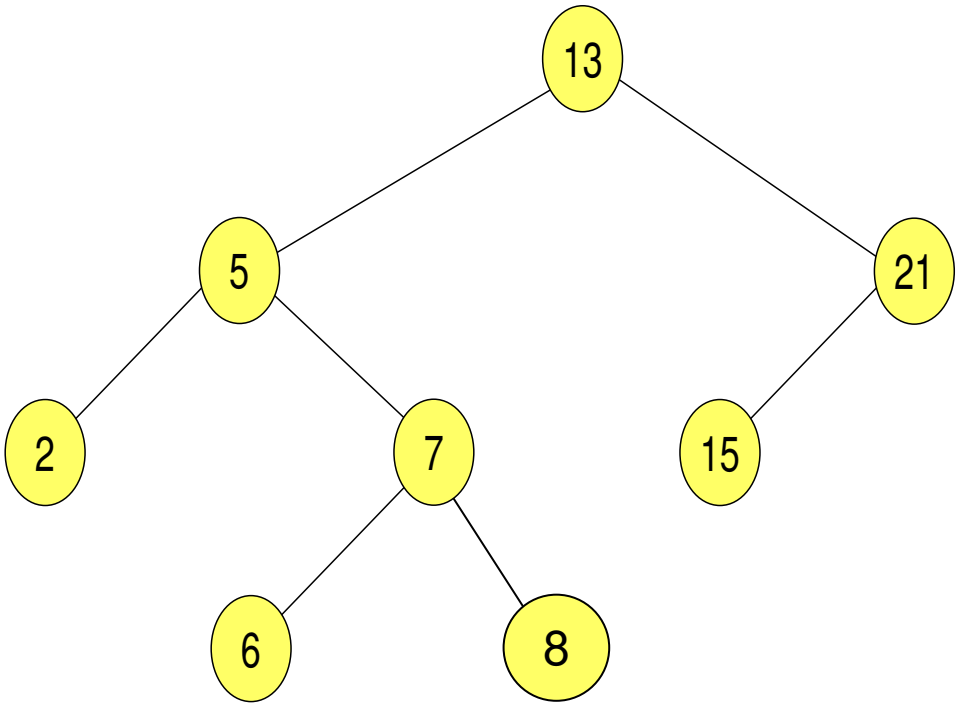
Penambahan Data pada Binary Search Tree



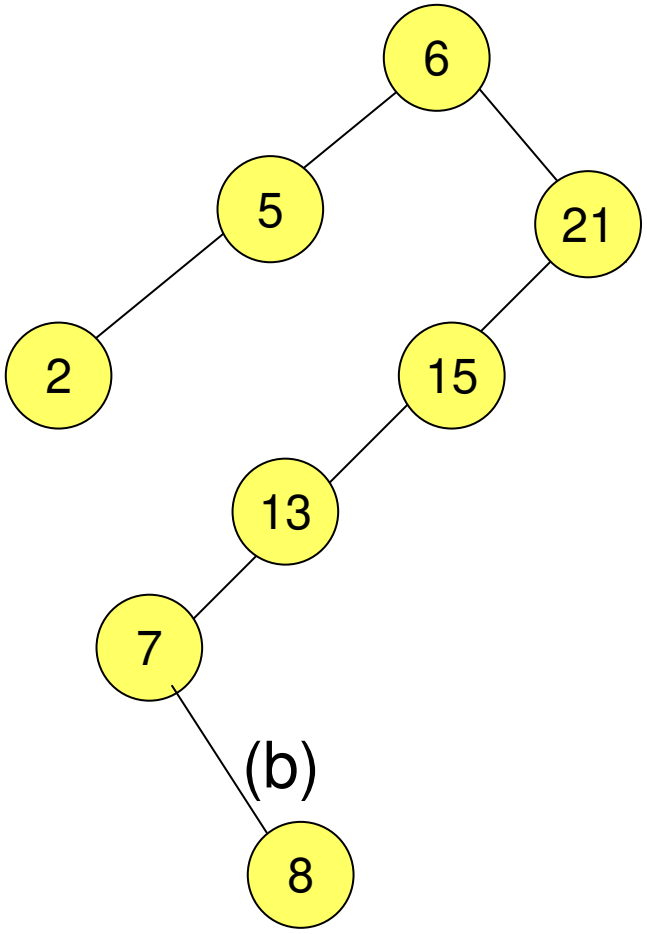


Tambahkan node: 8

Penambahan Data pada Binary Search Tree



(a)



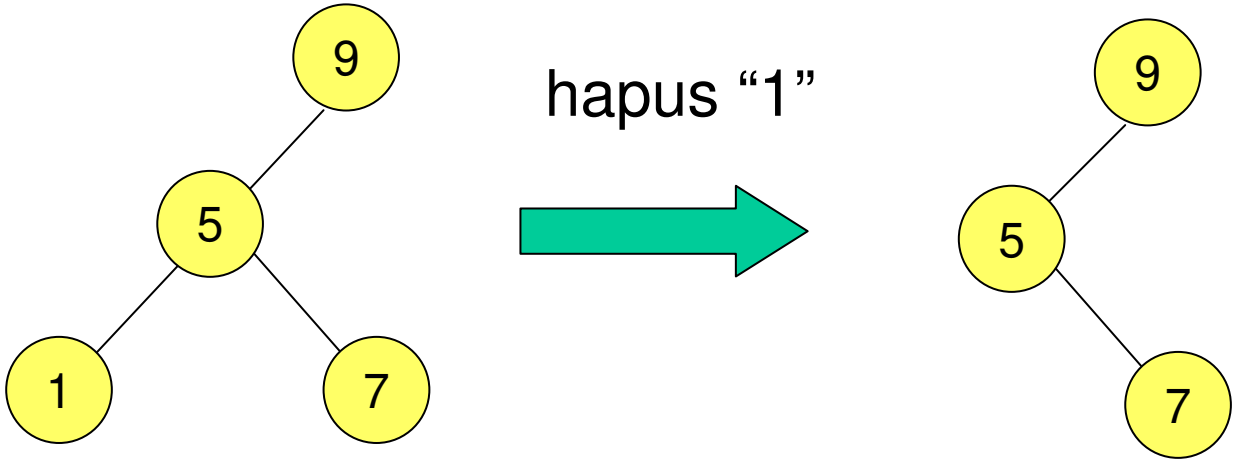
(b)

Perhatikan:

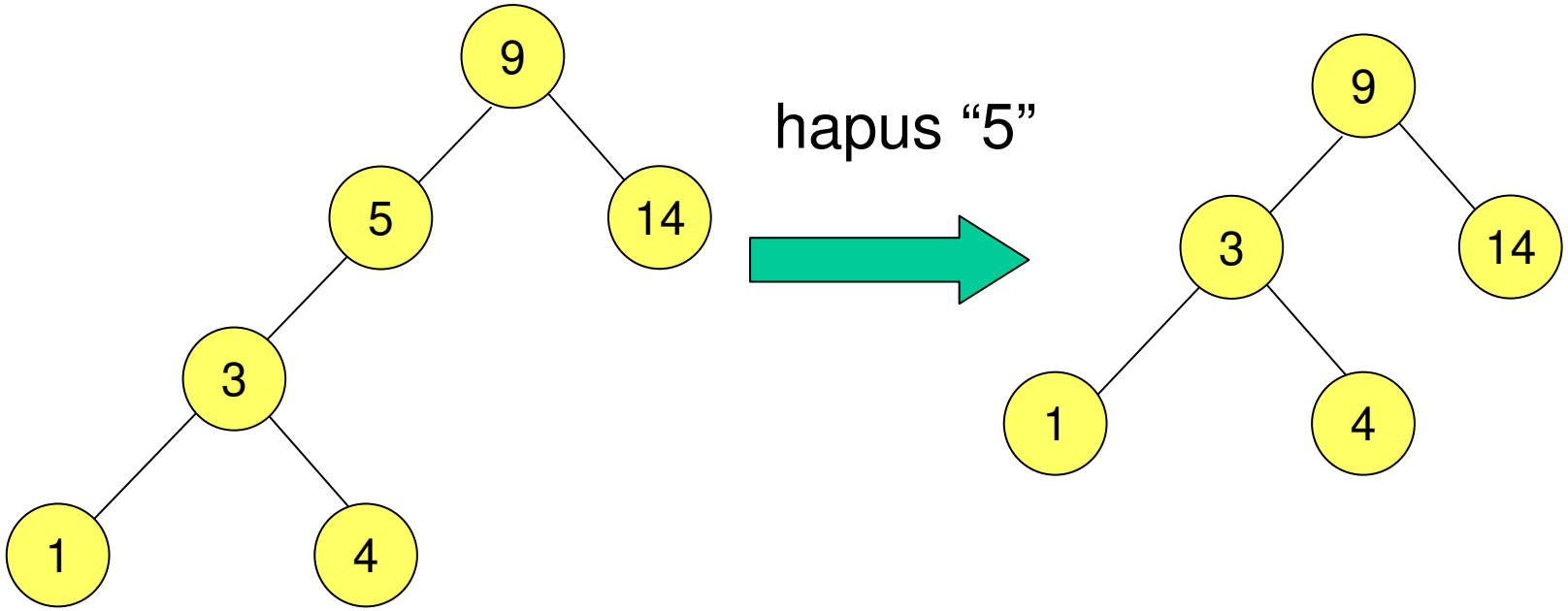
(b) lebih bercabang/panjang daripada (a), akibatnya proses pencarian pada (b) lebih lama daripada (a)

Penghapusan Data pada Binary Search Tree

- Proses penghapusan data pada binary search tree lebih rumit daripada proses searching maupun proses insertion
- Tahapan:
    1. Carilah node yang akan dihapus
    2. Apabila node tersebut leaf (tidak mempunyai anak), hapuslah node tersebut
    3. Bila node tersebut memiliki 1 anak, setelah node dihapus, anaknya menggantikan posisi orangtuanya (node yang dihapus)



Penghapusan Data pada Binary Search Tree



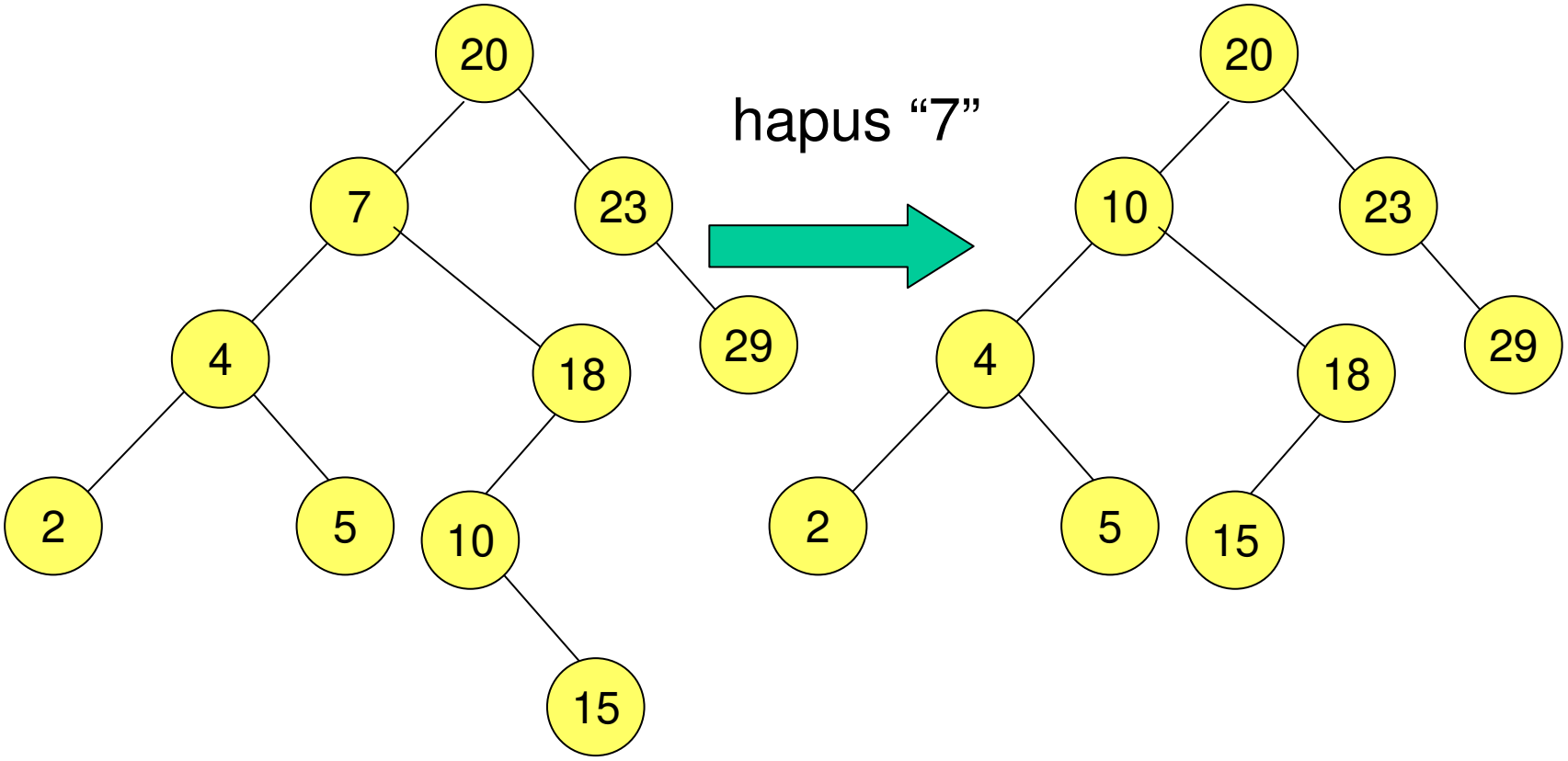
Penghapusan Data pada Binary Search Tree

- Proses penghapusan data pada binary search tree lebih rumit daripada proses searching maupun proses insertion
- Tahapan:
    1. Carilah node yang akan dihapus
    2. Apabila node tersebut leaf (tidak mempunyai anak), hapuslah node tersebut
    3. Bila node tersebut memiliki 1 anak, setelah node dihapus, anaknya menggantikan posisi orangtuanya (node yang dihapus)
    4. Bila node tersebut memiliki 2 anak, setelah node dihapus, gantikan node tersebut dengan

        elemen terkecil dari right sub-tree ATAU

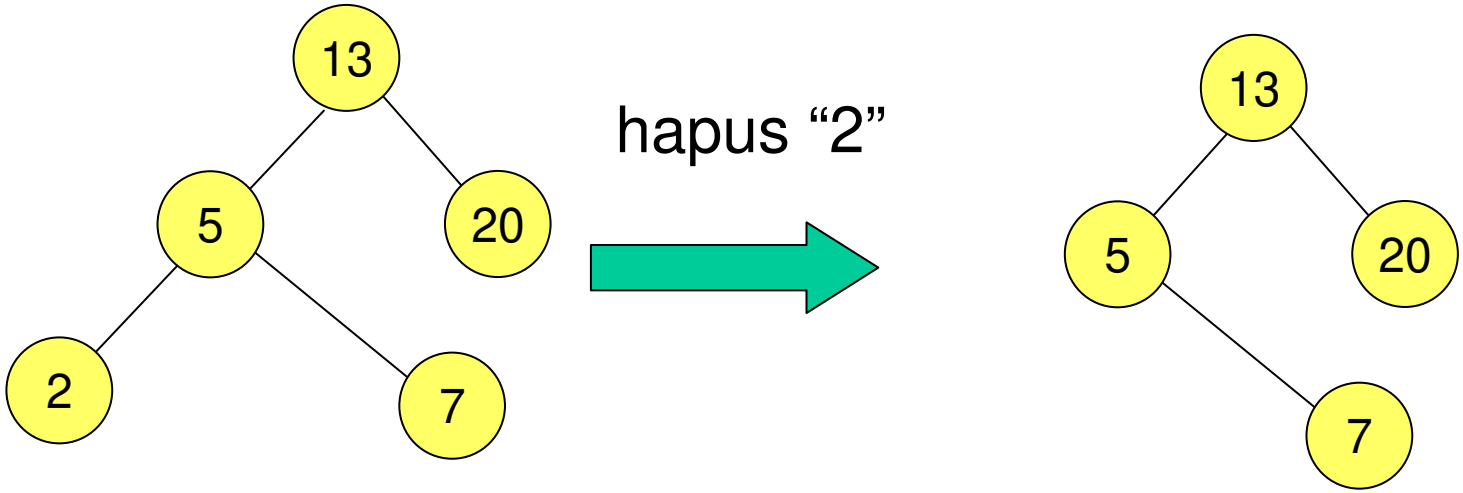
        elemen terbesar dari left sub-tree

Penghapusan Data pada Binary Search Tree



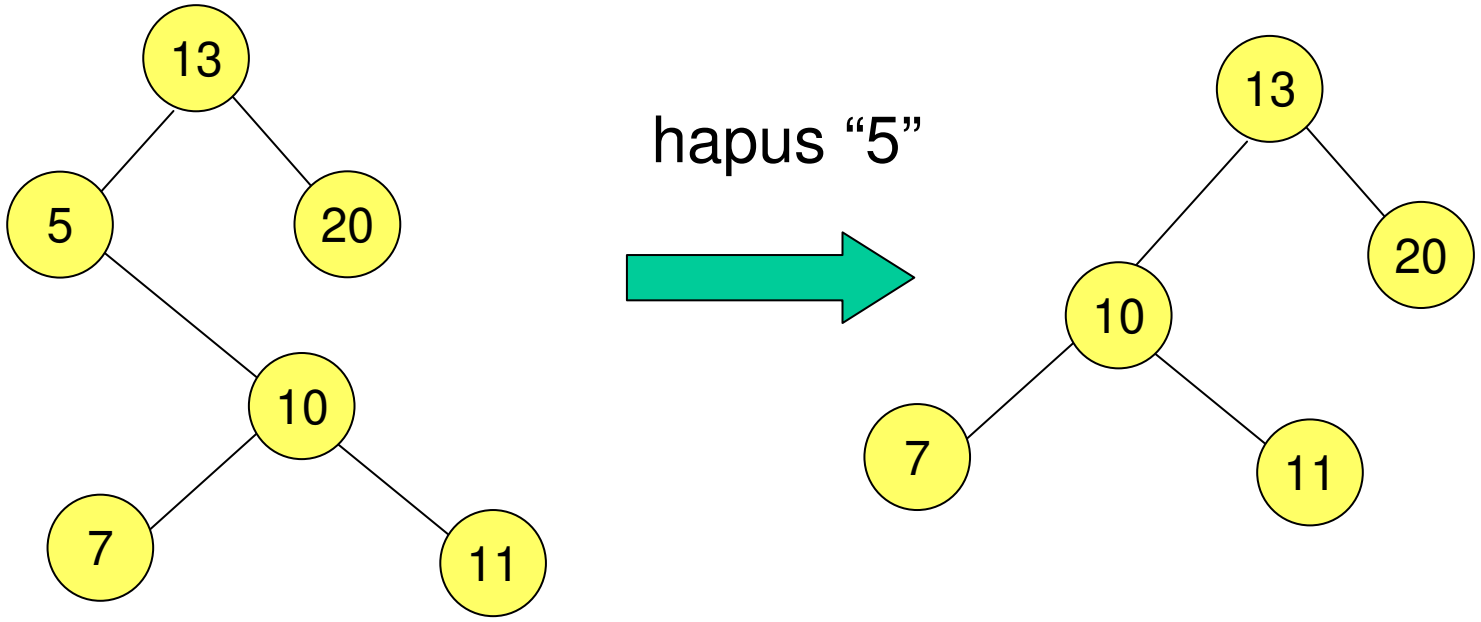
Menghapus node dengan nilai terkecil

node yang dihapus leaf (tidak mempunyai anak)



Menghapus node dengan nilai terkecil

node yang mempunyai 1 anak



Menghapus node dengan nilai terkecil

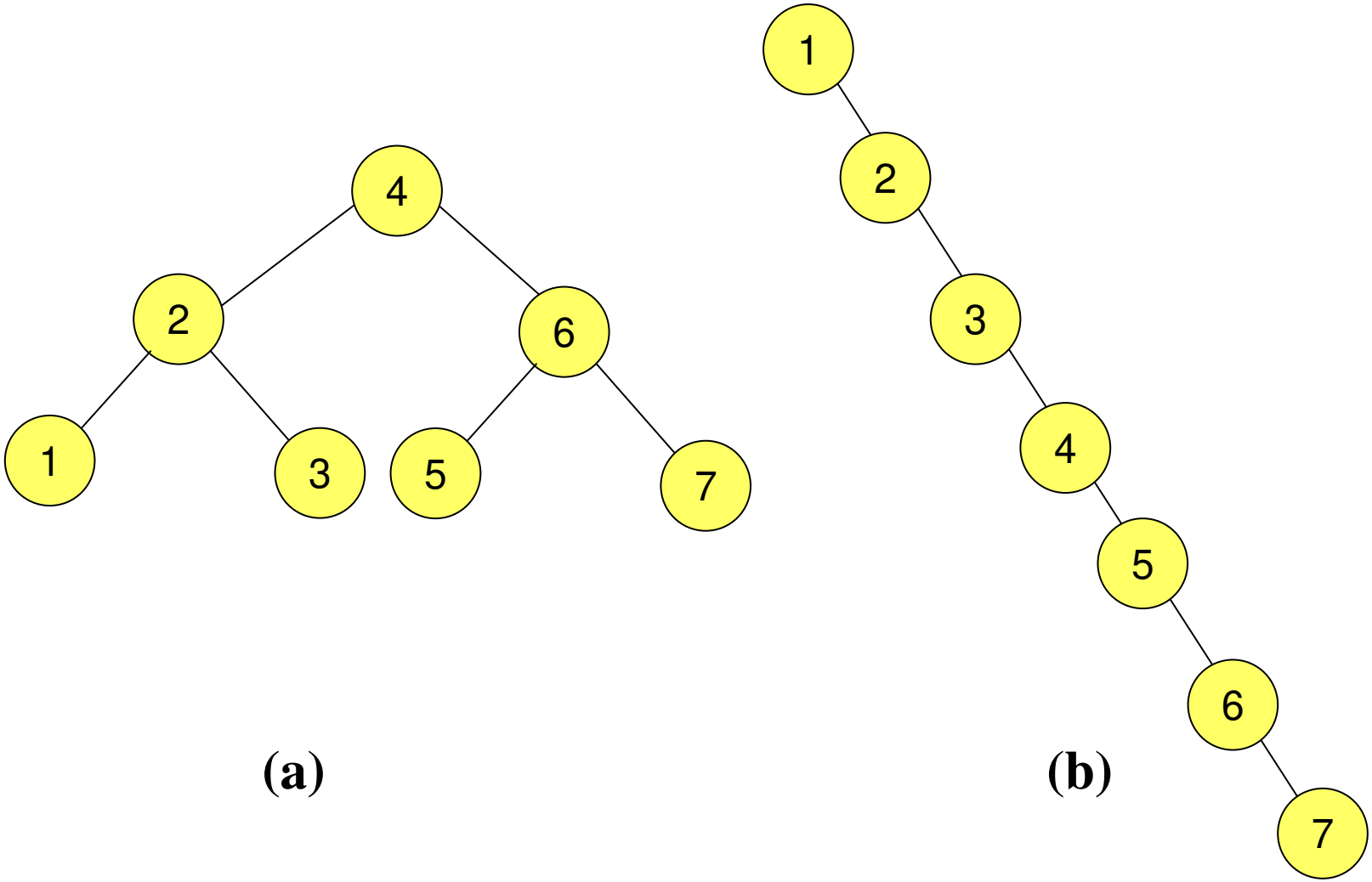
Bagaimana kalau node yang akan dihapus tersebut adalah yang memiliki nilai terkecil DAN mempunyai 2 anak ?

Karakteristik Binary Search Tree

- Complete binary tree : Binary tree yang untuk semua leaf memiliki jarak (panjang/tinggi) sama terhadap root
- Banyaknya node pada complete binary tree adalah $2^n - 1$
- Complete binary tree dalam arti yang lebih luas, adalah jika selisih jarak terpanjang (root ke leaf) dan jarak terpendek kurang dari atau sama dengan 1
- Panjang dari root ke leaf, dengan banyak node n adalah

$$\log_2 n + 1$$

- Computational complexity proses searching pada complete binary search tree: $O(\log n)$


4
2
6
1
3
5
7
1
2
3
4
5
6
7
(a)
(b)

Karakteristik Binary Search Tree

- Complete binary tree : Binary tree yang untuk semua leaf memiliki jarak (panjang/tinggi) sama terhadap root
- Banyaknya node pada complete binary tree adalah $2^n - 1$
- Complete binary tree dalam arti yang lebih luas, adalah jika selisih jarak terpanjang (root ke leaf) dan jarak terpendek kurang dari atau sama dengan 1 (contoh: gambar (a))
- Panjang dari root ke leaf, dengan banyak node n adalah

$$\log_2 n + 1$$

- Computational complexity proses searching pada complete binary search tree: $O(\log n)$
- Kondisi terburuk: gambar (b) dengan proses searching O(n)

Sorting Algorithms

Dr. Anto Satriyo Nugroho, M.Eng

Email: asnugroho@gmail.com
Web: http://asnugroho.net/lecture/ds.html

Beberapa Algoritma Sorting

1. **Bubble Sort**
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort

Bubble Sort: pseudocode

```
BUBBLESORT(A)
1     for i←1 to length[A]
2         do for j←length[A] downto i+1
3                do if A[j] < A[j-1]
4                       then exchange A[j] ↔ A[j-1]
```

Contoh Algoritma: BUBBLE SORT

banyaknya data: n

Data diurutkan/disorting dari yang bernilai besar

Proses

step 1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

step 2 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-2. Jika nilai kiri<kanan, tukarkan kedua data itu.

step n-1 : Periksalah nilai dua elemen mulai dari urutan ke-n sampai urutan ke-n-1. Jika nilai kiri<kanan, tukarkan kedua data itu.

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	8	10

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	5	10	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	4	10	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	7	10	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	5	8

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	4	8	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	7	8	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	4	5

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4

Bubble Sort: tahap demi tahap

Awal	7	4	5	8	10
Step-1	10	7	4	5	8
Step-2	10	8	7	4	5
Step-3	10	8	7	5	4
Step-4	10	8	7	5	4

Beberapa Algoritma Sorting

1. Bubble Sort
2. **Selection Sort**
3. Insertion Sort
4. Merge Sort
5. Quick Sort

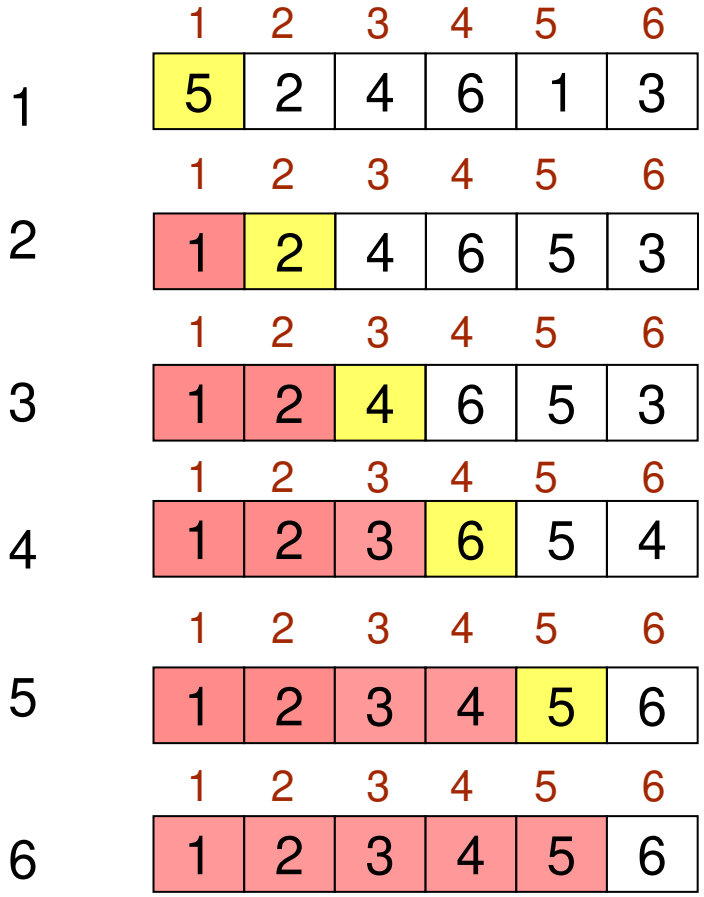
Selection Sort: Pseudocode

```
SELECTIONSORT(A)
1  for i← 1 to length[A]-1
2       min = i;
3       do for j ← i+1 to length[A]
4            do if  A[j] < A[min]
5                 min = j;
6       exchange A[min] ↔ A[i]
```

Prinsip kerja:

Dari elemen sebanyak n,
Carilah elemen terkecil dari array A, dan swap-lah elemen terkecil tersebut dengan elemen pertama (A[1]).
Carilah elemen terkecil kedua dari array A, dan swap-lah elemen tersebut dengan elemen kedua (A[2])
Ulangi sampai n-1 elemen pertama dari array A

Selection Sort: contoh



Carilah elemen terkecil & tukar dengan “5”

1 fixed. Carilah elemen terkecil & tukar dengan “2”

1,2 fixed. Carilah elemen terkecil & tukar dengan “4”

1,2,3 fixed. Carilah elemen terkecil & tukar dengan “6”

1,2,3,4 fixed. Carilah elemen terkecil & tukar dengan “5”

1,2,3,4,5 fixed, otomatis elemen terakhir sudah pada posisi yang benar

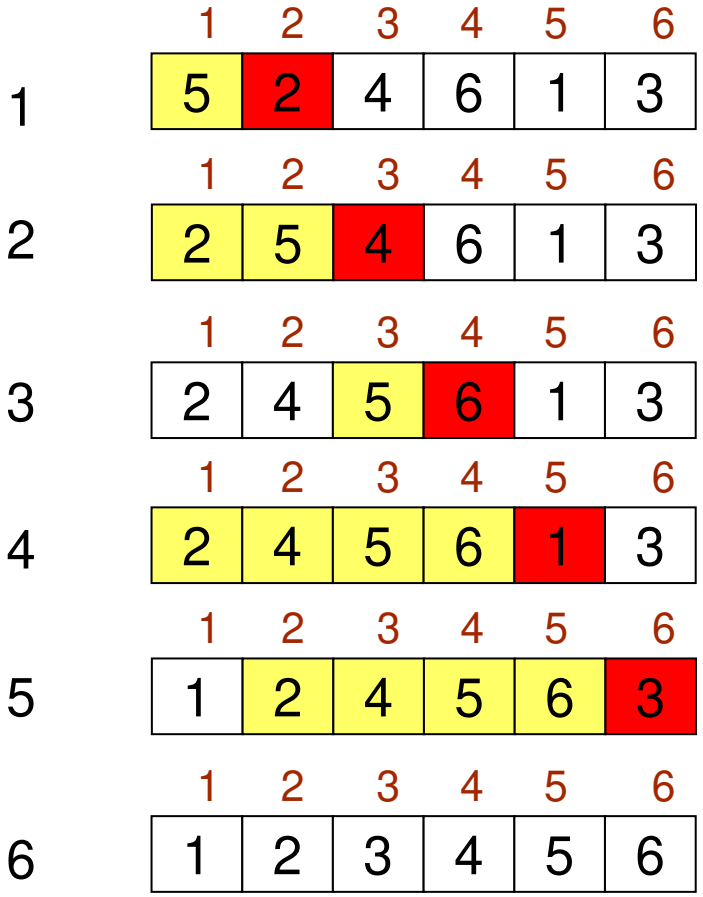
Beberapa Algoritma Sorting

1. Bubble Sort
2. Selection Sort
3. **Insertion Sort**
4. Merge Sort
5. Quick Sort

Insertion Sort: pseudocode

```
INSERTION-SORT(A)
1       for j←2 to length[A]
2           do key←A[j]
3                   Insert A[j] ke sekuens yang sudah disorting A[1…j-1]
4                   i← j-1
5                   while i>0  and A[i] > key
6                               do  A[i+1] ←A[i]
7                                       i ← i -1
8                   A[i+1] ←key
```

Insertion Sort: contoh



Quiz

Diketahui deretan data sbb.

80 84 100 24 79 85 91 65 17 3 1 21

1. Urutkan data tsb. memakai Selection Sort, agar elemen terkecil berada paling depan (urutan pertama), semakin ke belakang semakin besar
2. Urutkan data tsb. memakai Selection Sort, agar elemen terbesar berada paling depan (urutan pertama), semakin ke belakang semakin kecil
3. Urutkan data tsb. memakai Insertion Sort, agar elemen terkecil berada paling depan (urutan pertama), semakin ke belakang semakin besar
4. Urutkan data tsb. memakai Insertion Sort, agar elemen terbesar berada paling depan (urutan pertama), semakin ke belakang semakin kecil

Beberapa Algoritma Sorting

1. Bubble Sort
2. **Selection Sort**
3. Insertion Sort
4. Merge Sort
5. Quick Sort

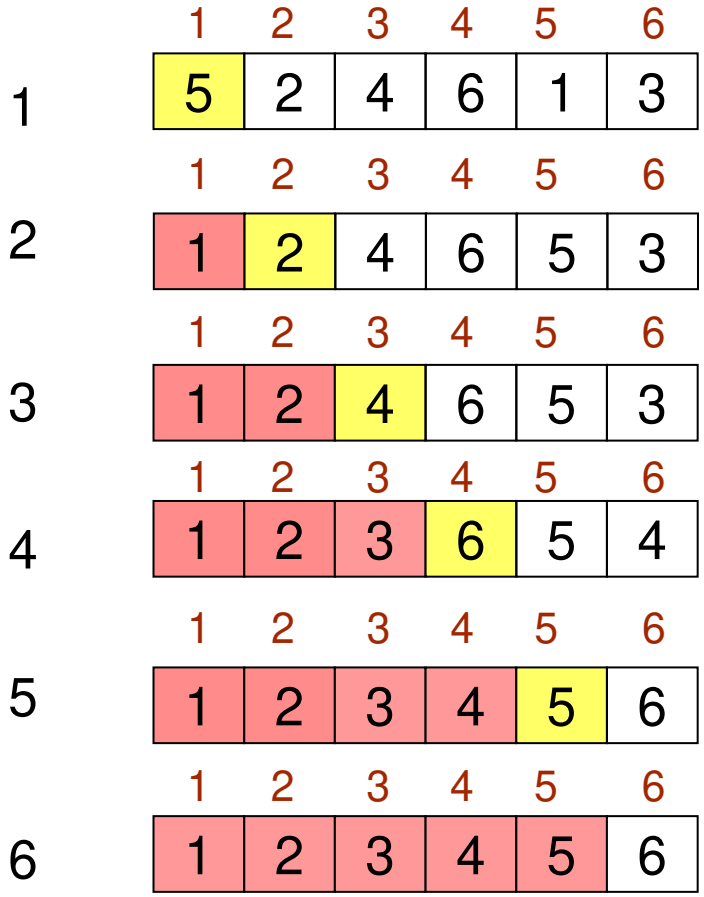
Selection Sort: Pseudocode

```
SELECTIONSORT(A)
1  for i← 1 to length[A]-1
2       min = i;
3       do for j ← i+1 to length[A]
4            do if  A[j] < A[min]
5                 min = j;
6       exchange A[min] ↔ A[i]
```

Prinsip kerja:

Dari elemen sebanyak n,
Carilah elemen terkecil dari array A, dan swap-lah elemen terkecil tersebut dengan elemen pertama (A[1]).
Carilah elemen terkecil kedua dari array A, dan swap-lah elemen tersebut dengan elemen kedua (A[2])
Ulangi sampai n-1 elemen pertama dari array A

Selection Sort: contoh



Carilah elemen terkecil & tukar dengan “5”

1 fixed. Carilah elemen terkecil & tukar dengan “2”

1,2 fixed. Carilah elemen terkecil & tukar dengan “4”

1,2,3 fixed. Carilah elemen terkecil & tukar dengan “6”

1,2,3,4 fixed. Carilah elemen terkecil & tukar dengan “5”

1,2,3,4,5 fixed, otomatis elemen terakhir sudah pada posisi yang benar

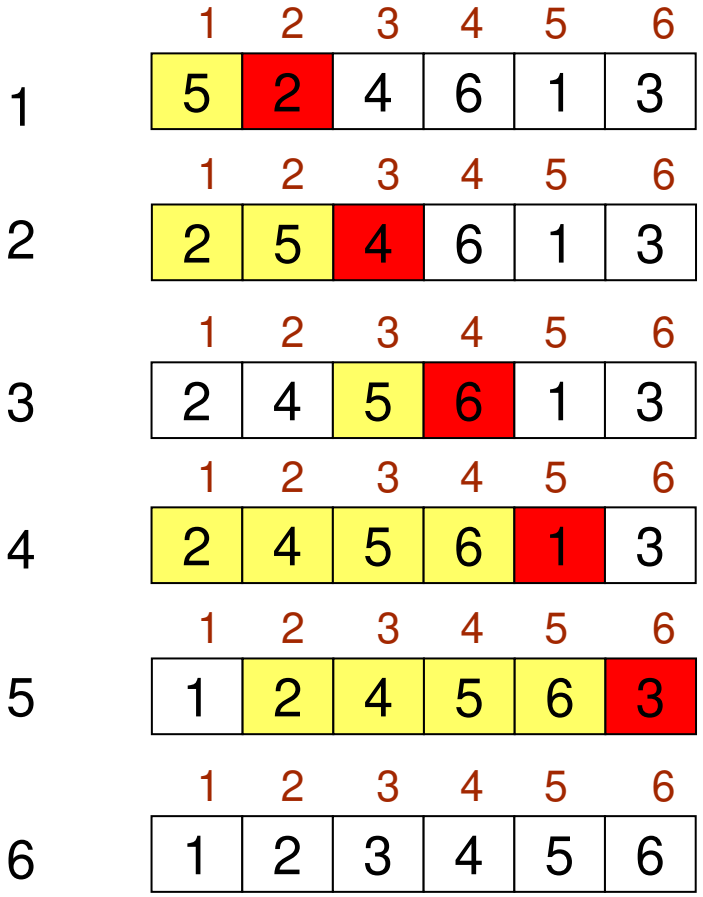
Beberapa Algoritma Sorting

1. Bubble Sort
2. Selection Sort
3. **Insertion Sort**
4. Merge Sort
5. Quick Sort

Insertion Sort: pseudocode

```
INSERTION-SORT(A)
1       for j←2 to length[A]
2           do key←A[j]
3                   Insert A[j] ke sekuens yang sudah disorting A[1…j-1]
4                   i← j-1
5                   while i>0  and A[i] > key
6                               do  A[i+1] ←A[i]
7                                       i ← i -1
8                   A[i+1] ←key
```

Insertion Sort: contoh



Latihan

Downloadlah program sorting “mysort.c” dan selesaikan fungsi insertion_sort(A[],datanum) & selection_sort (A[],datanum).

Beberapa Algoritma Sorting

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. **Merge Sort**
5. Quick Sort

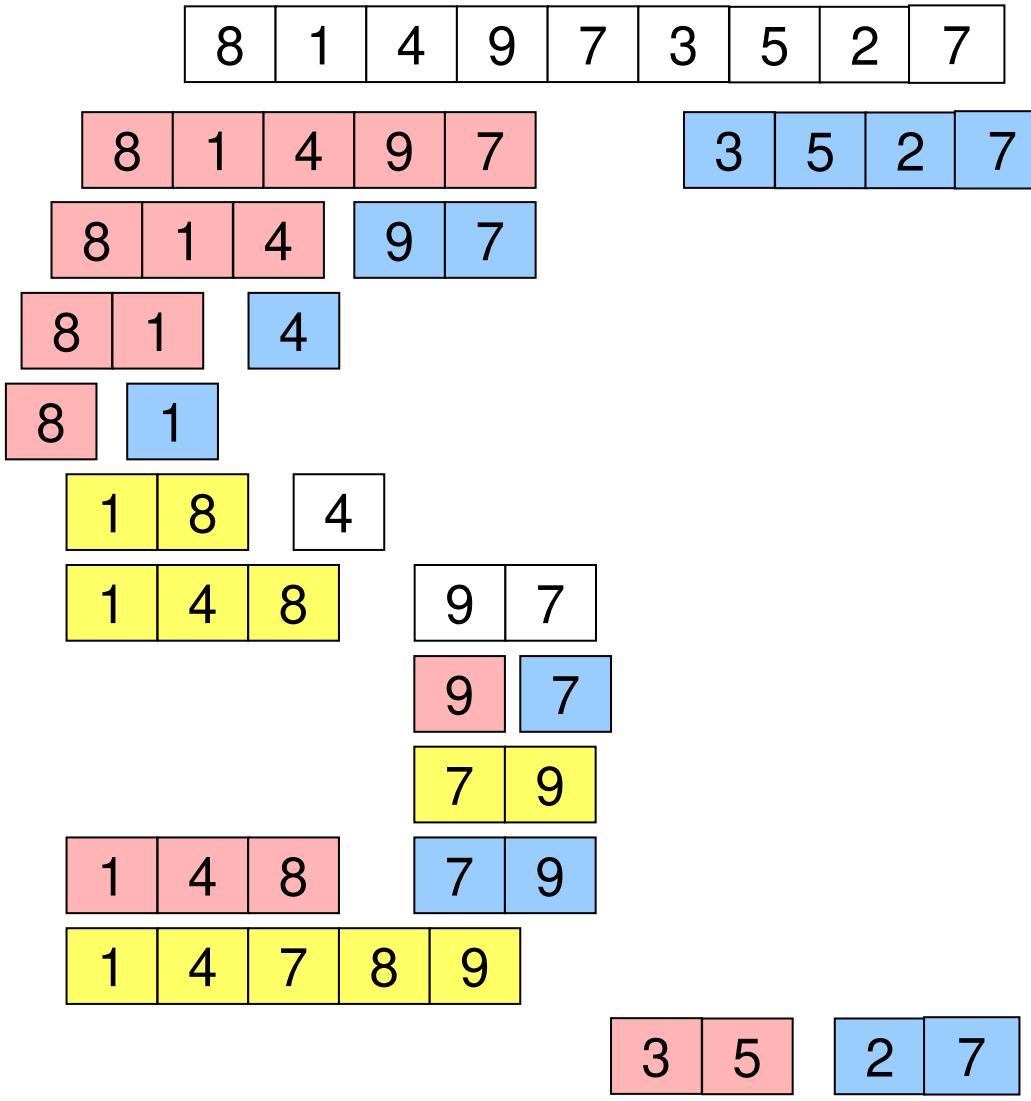
Merge Sort: pseudocode

```
MERGE (A, p,q,r)
1       n1←q-p+1
2       n2←r-q
3       create arrays L[1..n1+1]  and R[1..n2+1]
4       for  i← 1 to n1
5           do L[i] ← A[p+i-1]
6        for  j ← 1 to n2
7           do  R[j] ←A[q+j ]
8       L[n1+1] ←∞
9       R[n2+1] ←∞
10      i ←1
11      j ←1
12      for k ← p to r
13          do if L[i] ≤ R[j]
14              then A[k] ← L[i]
15                   i ← i+1
16              else A[k] ← R[j]
17                   j ← j+1
```

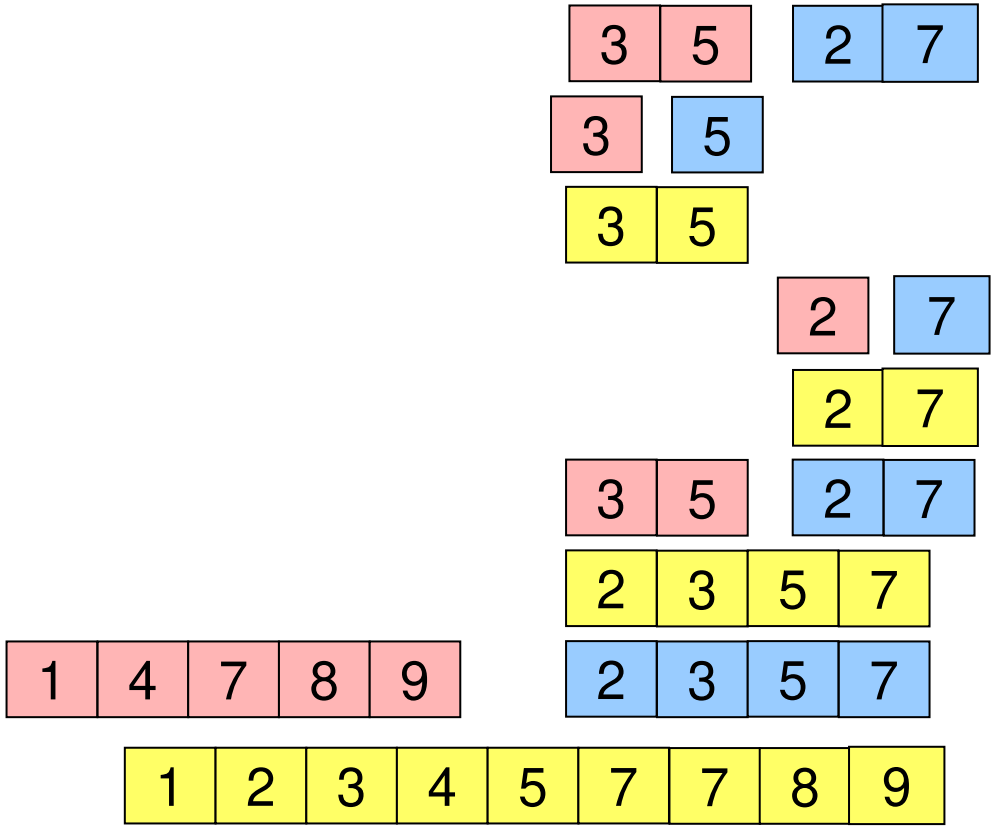
Merge Sort: pseudocode

```
MERGE-SORT (A, p, r)
1     If p<r
2         then q← (p+r)/2
3              MERGE-SORT (A, p, q)
4              MERGE-SORT (A, q+1, r)
5              MERGE(A,p,q,r)
```

Merge Sort: contoh



Merge Sort: contoh



Beberapa Algoritma Sorting

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. **Quick Sort**

Prinsip Kerja Quick Sort

- Divide
  - Partisilah array *A[p…r]* ke dalam dua buah subarray *A[p…q-1]* dan *A[q+1…r]* sedemikian hingga
    - tiap elemen pada *A[p…q-1]* senantiasa lebih kecil atau sama dengan *A[q]* DAN
    - tiap elemen pada *A[q+1…r]* senantiasa sama atau lebih besar dari *A[q]*
  - Hitunglah *q*
- Conquer
  - Urutkan (sorting-lah) *A[p…q-1]* dan *A[q+1…r]* secara rekursif
- Combine
  - Kedua subarray telah diurutkan pada posisi masing-masing, sehingga tidak diperlukan upaya khusus untuk mengkombinasikan mereka. *A[p…r]* telah ter-sorting

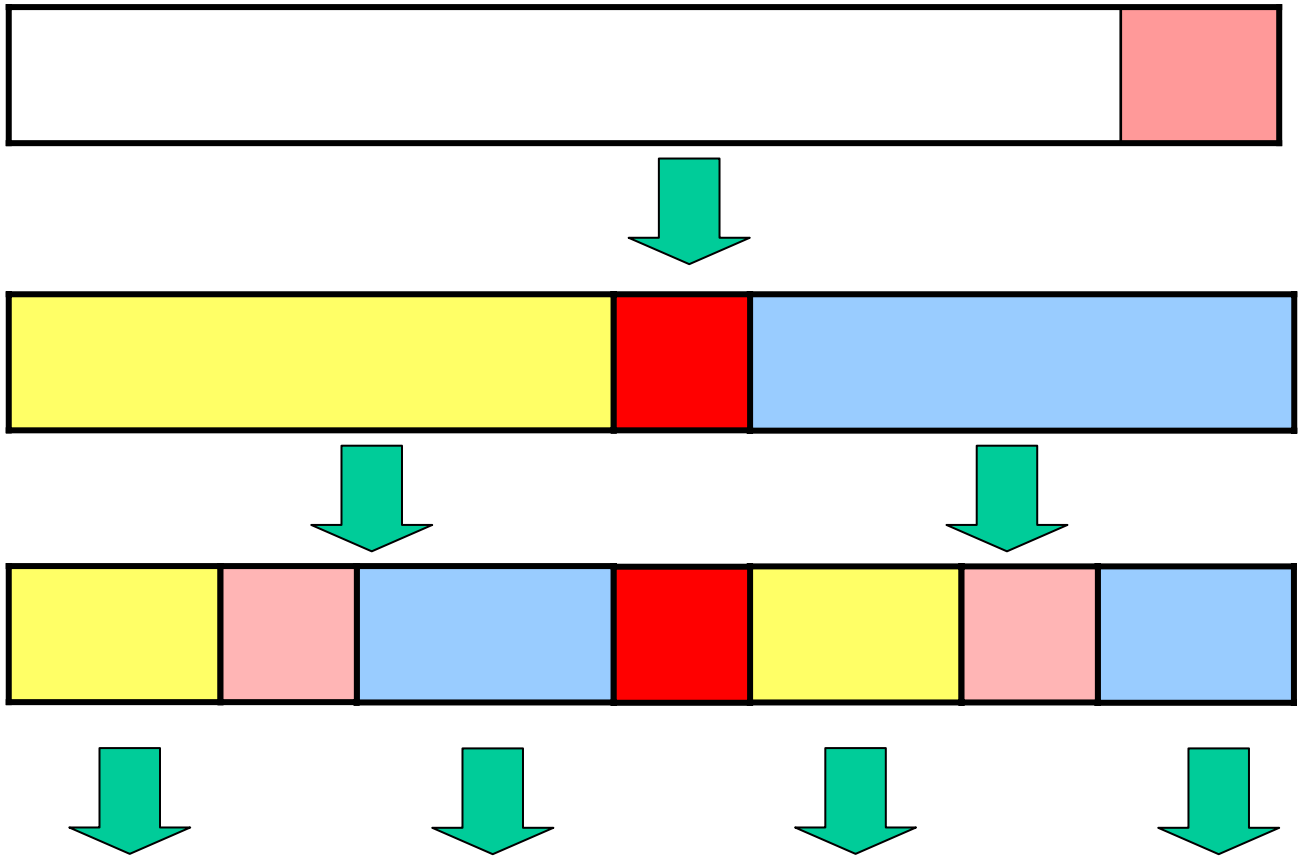
Quick Sort: pseudocode

```
QUICKSORT (A, p,r)
1     If p<r
2        then q←PARTITION (A,p,r)
3             QUICKSORT(A,p,q-1)
4             QUICKSORT(A,q+1,r)
```

```
PARTITION(A, p,r)
1    x←A[r]
2    i←p-1
3    for j←p to r-1
4   do if A[j] ≤ x
5       then i←i+1
6            exchange A[i] ↔A[j]
7    exchange A[i+1] ↔A[r]
8    return i+1
```

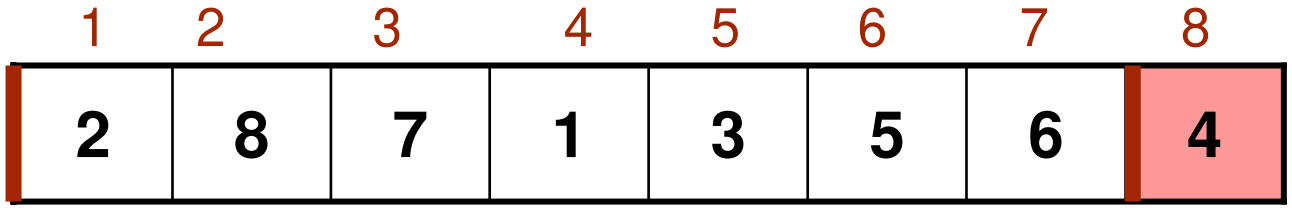
1. Tetapkan data paling kanan sebagai pivot
2. Pindahkan semua data yang lebih kecil daripada pivot ke sayap kiri
3. Pindahkan semua data yang lebih besar daripada pivot ke sayap kanan
4. Pivot urutan ke berapa ?

Cara Kerja Quick Sort



Quick Sort: Contoh

(a)



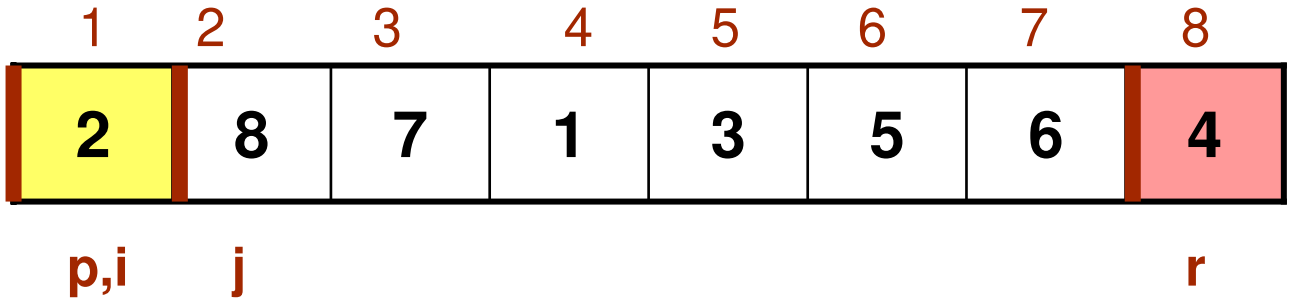
i=0
p=j=1
r=8

i=1
j=1
p=1
r=8

```
PARTITION(A, p,r)
1    x←A[r]          x=4
2    i←p-1           i=0
3    for j←p to r-1  j=1
4        do if A[j] ≤ x      2 ≤ 4 ? YES
5                then i←i+1  i=1
6                     exchange A[i] ↔A[j]  exchange 2 & 2
7    exchange A[i+1] ↔A[r]
8    return i+1
```

Quick Sort: Contoh

(b)



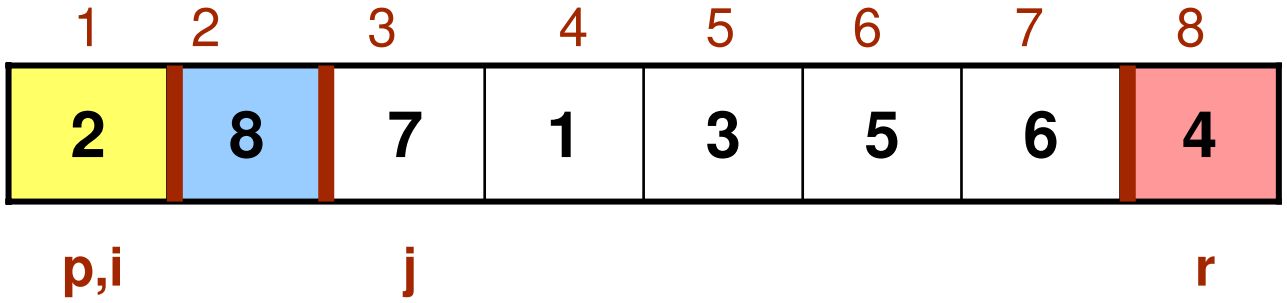
i=1
j=1
p=1
r=8

i=1
j=2
p=1
r=8

```
PARTITION(A, p,r)
1     x←A[r]
2     i←p-1
3     for j←p to r-1        j=2
4         do if A[j] ≤ x        8 ≤ 4 ? NO
5                 then i←i+1
6                      exchange A[i] ↔A[j]
7     exchange A[i+1] ↔A[r]
8     return i+1
```

Quick Sort: Contoh

(c)



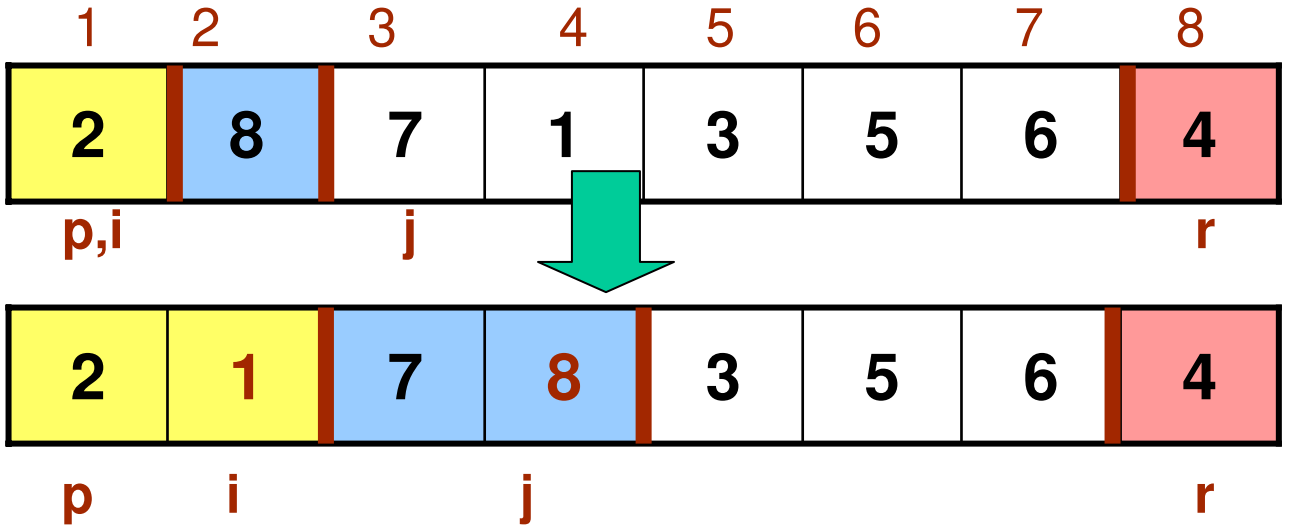
i=1
j=2
p=1
r=8

i=1
j=3
p=1
r=8

```
PARTITION(A, p,r)
1    x←A[r]
2    i←p-1
3    for j←p to r-1        j=3
4        do if A[j] ≤ x        7 ≤ 4 ? NO
5                then i←i+1
6                     exchange A[i] ↔A[j]
7    exchange A[i+1] ↔A[r]
8    return i+1
```

Quick Sort: Contoh

(d)



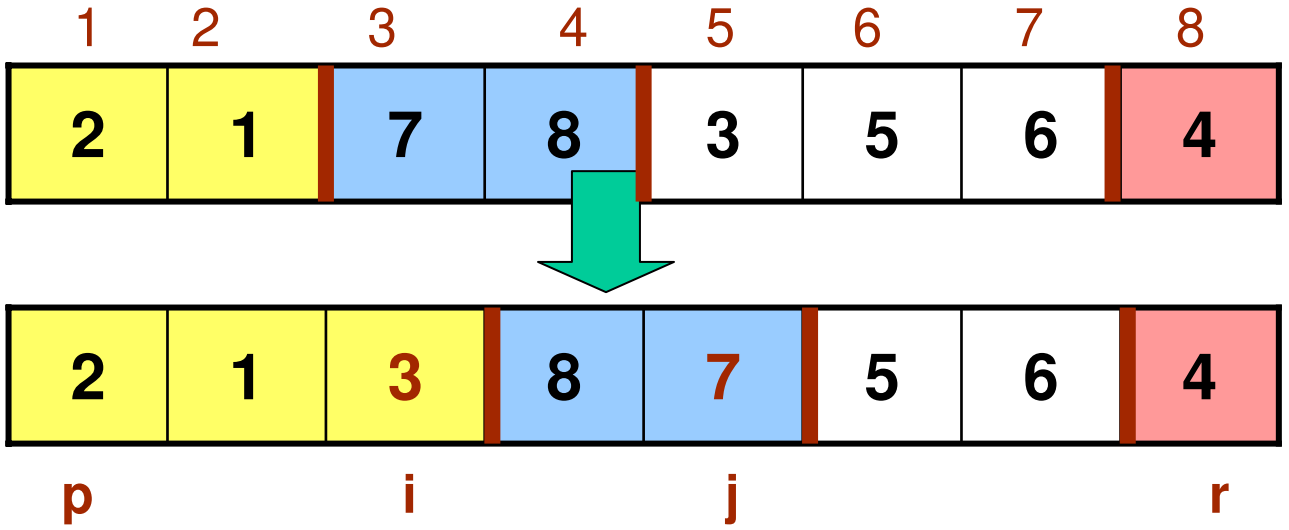
i=1
j=3
p=1
r=8

i=2
j=4
p=1
r=8

```
PARTITION(A, p,r)
1    x←A[r]
2    i←p-1
3    for j←p to r-1        j=4
4        do if A[j] ≤ x        1 ≤ 4 ? YES
5                then i←i+1        i=2
6                     exchange A[i]↔A[j]    exchange 8 & 1
7    exchange A[i+1]↔A[r]
8    return i+1
```

Quick Sort: Contoh

(e)



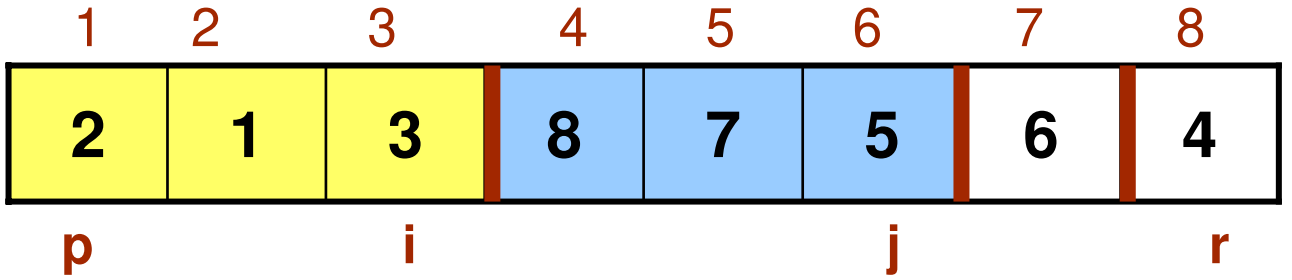
i=2
j=4
p=1
r=8

i=3
j=5
p=1
r=8

```
PARTITION(A, p,r)
1     x←A[r]
2     i←p-1
3     for j←p to r-1        j=5
4         do if A[j] ≤ x        3 ≤ 4 ? YES
5                 then i←i+1        i=3
6                      exchange A[i] ↔A[j]   exchange 7 & 3
7     exchange A[i+1] ↔A[r]
8     return i+1
```

Quick Sort: Contoh

(f)



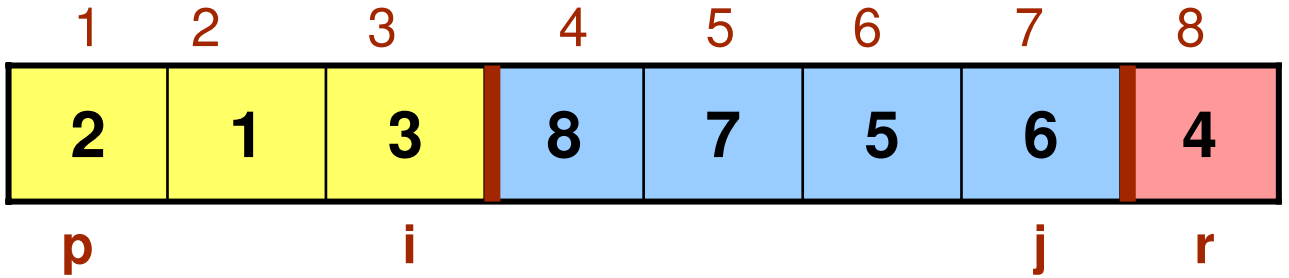
i=3
j=5
p=1
r=8

```
PARTITION(A, p,r)
1    x←A[r]
2    i←p-1
3    for j←p to r-1        j=6
4        do if A[j] ≤ x        5 ≤ 4 ? NO
5                then i←i+1
6                     exchange A[i] ↔A[j]
7    exchange A[i+1] ↔A[r]
8    return i+1
```

i=3
j=6
p=1
r=8

Quick Sort: Contoh

(g)

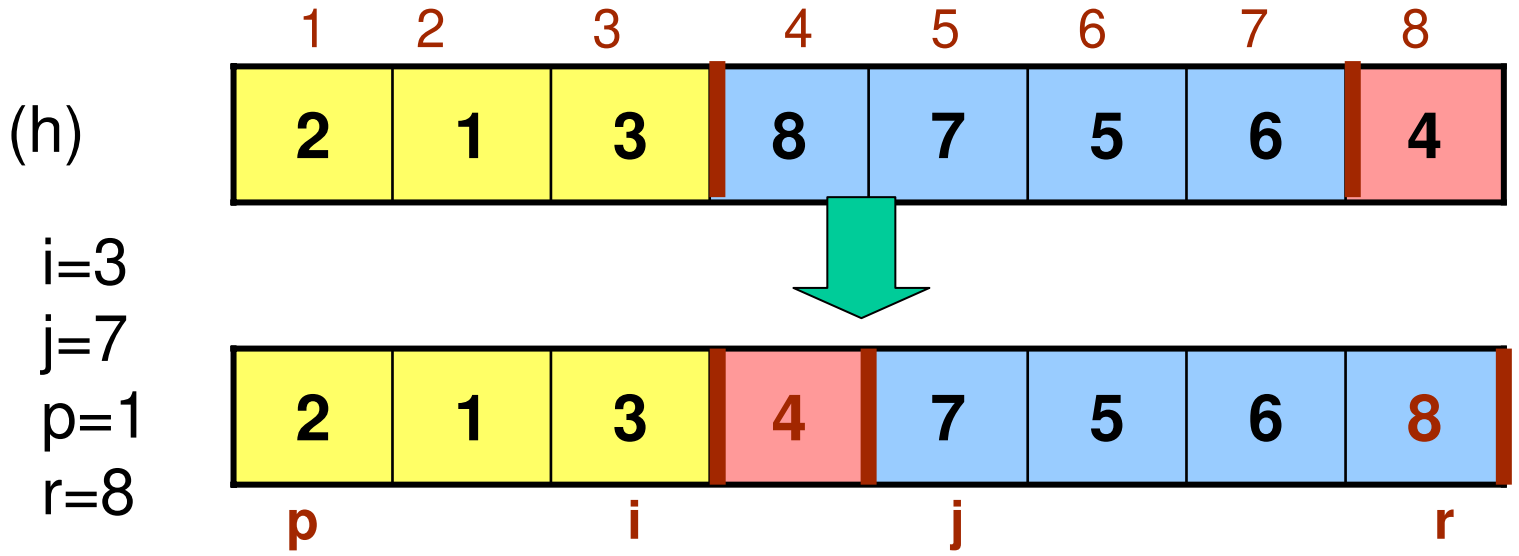


i=3
j=6
p=1
r=8

```
PARTITION(A, p,r)
1    x←A[r]
2    i←p-1
3    for j←p to r-1        j=7
4        do if A[j] ≤ x        6 ≤ 4 ? NO
5                then i←i+1
6                     exchange A[i] ↔A[j]
7    exchange A[i+1] ↔A[r]
8    return i+1
```

i=3
j=7
p=1
r=8

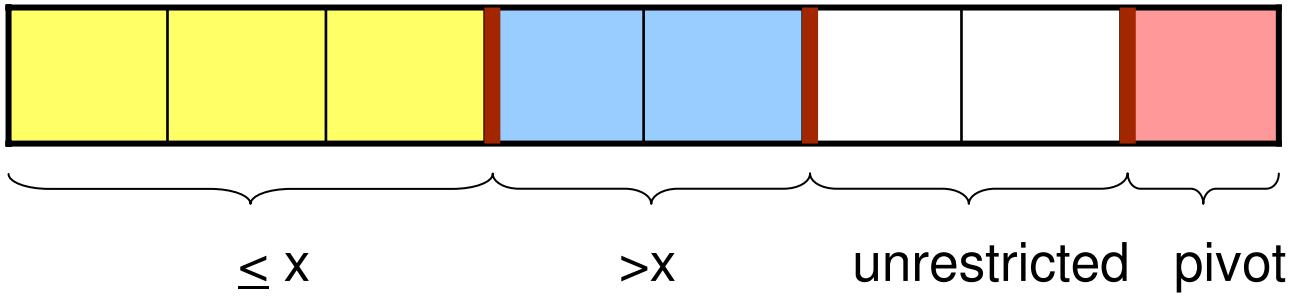
Quick Sort: Contoh




```
PARTITION(A, p,r)
1     x←A[r]
2     i←p-1
3     for j←p to r-1
4         do if A[j] ≤ x
5                   then i←i+1
6                        exchange A[i] ↔A[j]
7     exchange A[i+1] ↔A[r]   exchange 4 & 8
8     return i+1
```

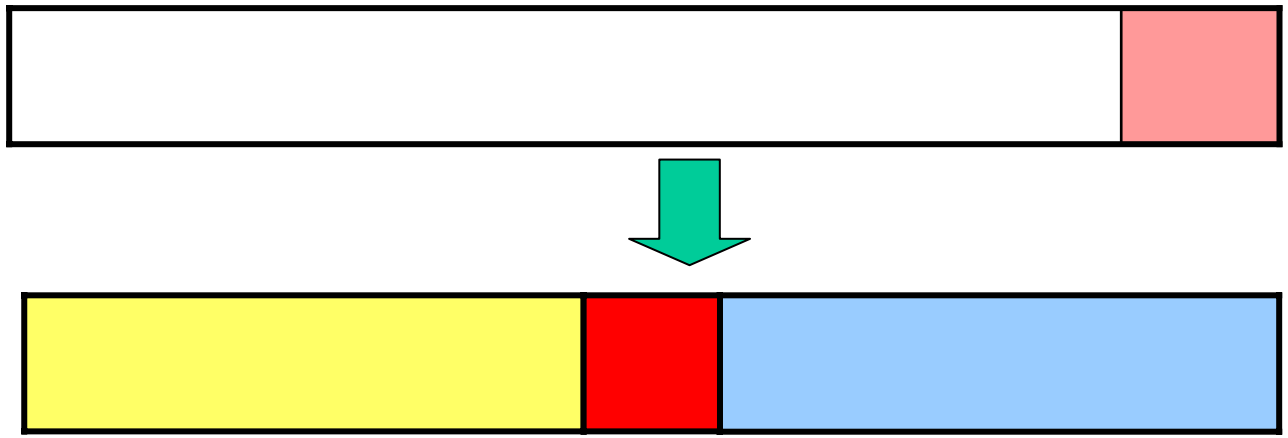
return 4 (q=4)

4 region dalam procedure PARTITION

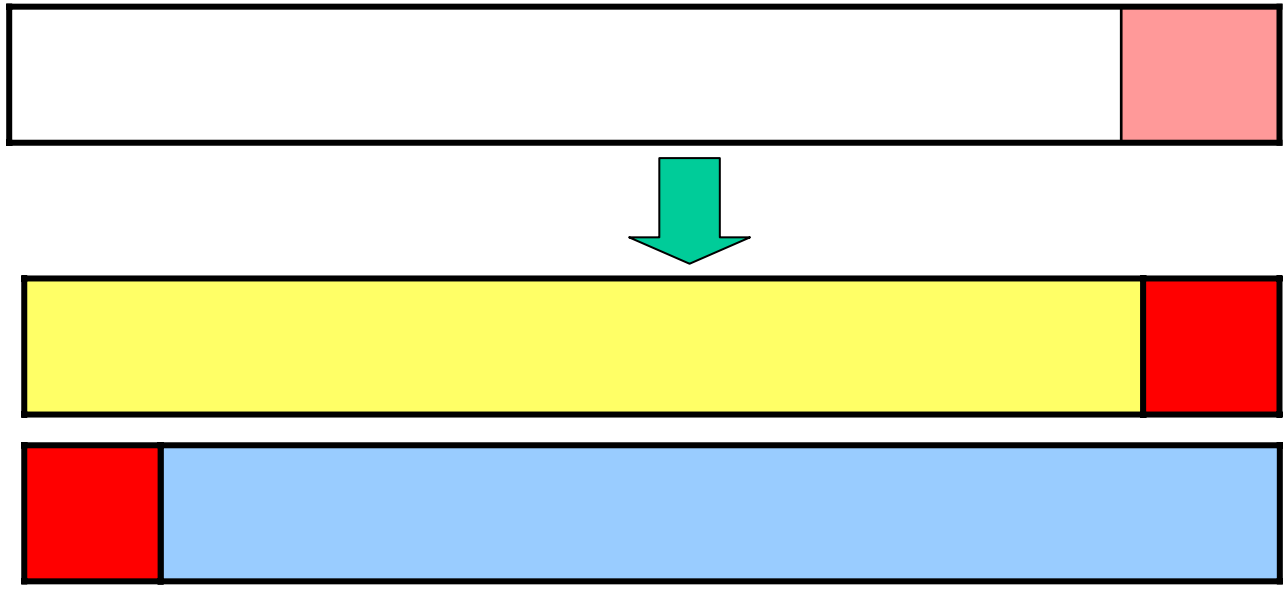


Best Case & Worst Case

- Best Case Partition



- Worst Case Partition



atau

Quiz

Diketahui deretan data sbb.

80 84 100 24 79 85 91 65 17 3 1 21

1. Urutkan data tsb. memakai Merge sort, agar elemen terkecil berada paling depan (urutan pertama), semakin ke belakang semakin besar
2. Urutkan data tsb. memakai Merge Sort, agar elemen terbesar berada paling depan (urutan pertama), semakin ke belakang semakin kecil
3. Urutkan data tsb. memakai Quick Sort, agar elemen terkecil berada paling depan (urutan pertama), semakin ke belakang semakin besar
4. Urutkan data tsb. memakai Quick Sort, agar elemen terbesar berada paling depan (urutan pertama), semakin ke belakang semakin kecil

Randomized Quicksort

```
RANDOMIZED-QUICKSORT (A, p,r)
1     If p<r
2        then q←RANDOMIZED-PARTITION (A,p,r)
3             RANDOMIZED-QUICKSORT(A,p,q-1)
4             RANDOMIZED-QUICKSORT(A,q+1,r)
```

```
RANDOMIZED-PARTITION(A, p,r)
1    i←RANDOM(p,r)
2    exchange A[r] ↔A[i]
3    return PARTITION (A,p,r)
```

```
//    Quick Sort
//    Source: Algorithms and Data Structure in C,BohYoh Shibata, Soft Bank
      Publishing, p.178

#include   <stdio.h>

#define  swap(type,x,y)  do {type t=x; x=y; y=t;} while (0)
```

```
void quick(int a[], int left, int right)
{

  int i=left;
  int j=right;
  int pivot= a[(i+j)/2];

  do{
    while (a[i]<pivot) i++;
    while (a[j]>pivot) j--;
    if(i<=j) {
      swap(int, a[i], a[j]);
      i++;
      j--;
    }
  } while (i <= j);

  if(left<j)  quick(a,left,j);
  if(i<right) quick(a,i,right);
}
```

```
int main(void)
{
  int i;
  int x[4];
  int nx=sizeof(x)/sizeof(x[0]);

  printf("Masukkan %d buah bilangan bulat\n",nx);
  for(i=0;i<nx;i++) {
    printf("x[%d] : ",i);
    scanf("%d", &x[i]);

  }
  quick(x,0,nx-1);
  puts("Proses sorting selesai.");
  for(i=0;i<nx;i++)
    printf("x[%d]=%d\n",i,x[i]);

  return(0);

}
```

Tugas

- Modifikasilah program 04-04.c (kuliah ke-4), dengan mengganti bubble sort menjadi quick sort.

```
void quick(struct NILAI a[], int left, int right)
{

  int i=left;
  int j=right;
  float pivot= a[(i+j)/2].average;

  do{
    while (a[i].average<pivot) i++;
    while (a[j].average>pivot) j--;
    if(i<=j) {
      swap(struct NILAI, a[i], a[j]);
      i++;
      j--;
    }
  } while (i <= j);

  if(left<j)  quick(a,left,j);
  if(i<right) quick(a,i,right);
}
```