



Skema Standar (Bag. 3)

Sorting pada Array

Tim Pengajar

IF1210 Dasar Pemrograman

Sem. 2 2019/2020



Review

- Apa yang sudah dipelajari sejauh ini?
 - Pentingnya skema standar
 - Skema standar
 - Skema validasi
 - Skema pengulangan
 - Skema pemrosesan array (pengisian dan penulisan array, nilai ekstrim)
 - Skema pencarian pada array



SORTING PADA ARRAY



Pengurutan (*Sorting*)

- *Sorting* atau pengurutan data adalah proses yang sering harus dilakukan dalam pengolahan data
- Ada 2 macam teknik pengurutan:
 - **pengurutan internal**, terhadap data yang tersimpan di memori
 - **pengurutan eksternal**, terhadap data yang tersimpan di *secondary storage*
- Algoritma pengurutan internal yang utama antara lain: ***Counting Sort, Selection Sort, Insertion Sort, Bubble Sort***
- Performansi pengurutan data sangat menentukan performansi sistem, karena itu pemilihan metoda pengurutan yang cocok akan berperan dalam suatu aplikasi

Pengurutan (*Sorting*)

Definisi dan Kamus Umum



- **Definisi Persoalan:**

Diberikan sebuah Tabel integer $T [1..N]$ yang isinya sudah terdefinisi. Tuliskan sebuah algoritma yang mengurutkan elemen tabel sehingga terurut membesar :

$$T_1 \leq T_2 \leq T_3 \leq \dots \leq T_N$$

- **Kamus Umum:**

KAMUS

```
constant NMax : integer = 100
type TabInt : array [1..NMax] of integer
{ jika diperlukan sebuah tabel, maka akan dibuat deklarasi
  sebagai berikut: }
  N : integer    { indeks efektif maksimum tabel yang
                  terdefinisi,  $0 \leq N \leq NMax$  }
  T : TabInt      { tabel integer }
```



Counting Sort

(Pengurutan dengan Pencacah)

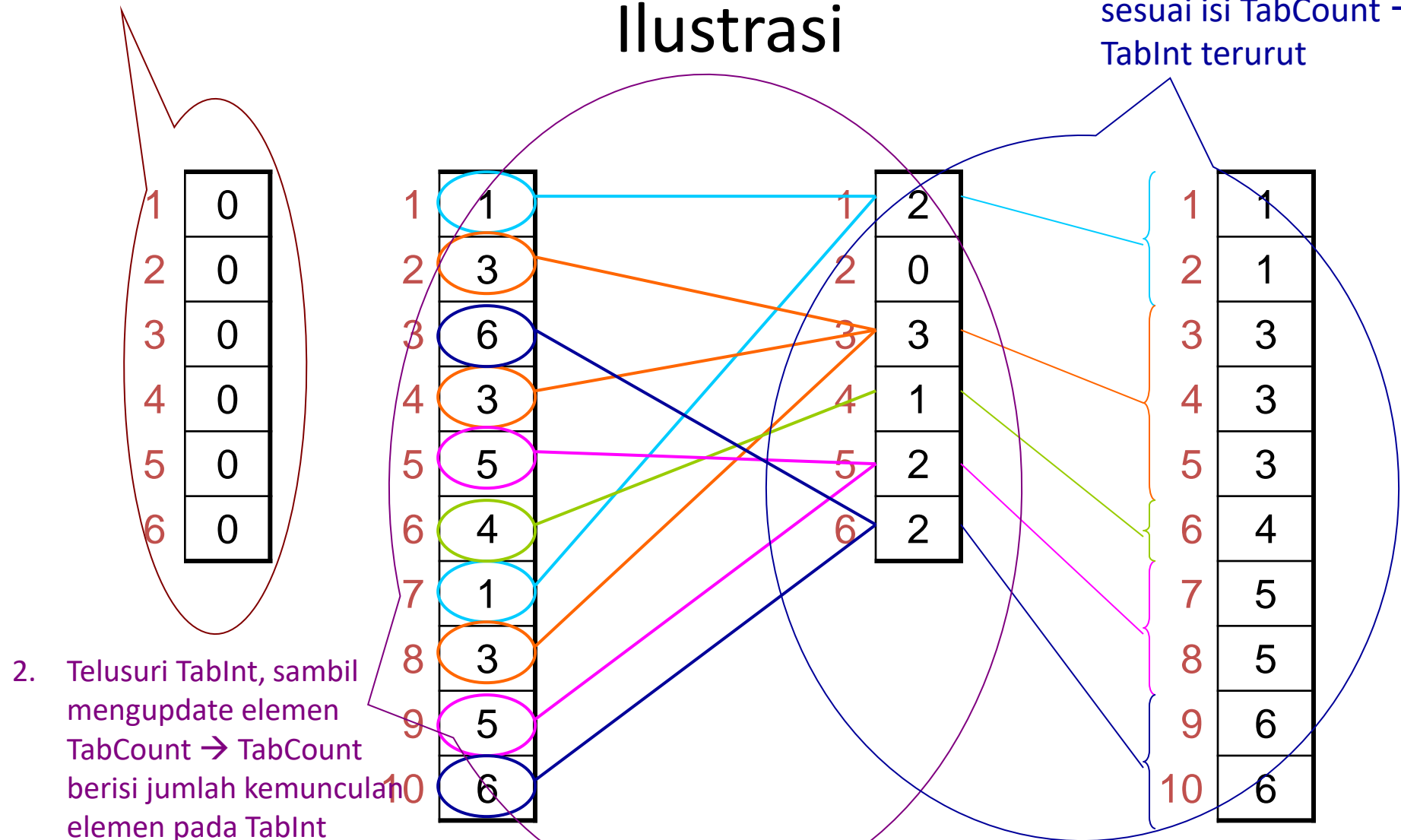
- Pengurutan dengan pencacahan adalah pengurutan yang paling sederhana
- Jika diketahui bahwa data yang akan diurut mempunyai daerah jelajah (*range*) tertentu, dan merupakan bilangan bulat, misalnya [Min..Max] maka cara paling sederhana untuk mengurut adalah :
 - Sediakan array TabCount_{Min..Max} yang elemennya diinisialisasi dengan nol, dan pada akhir proses TabCount_i berisi banyaknya data pada tabel asal yang bernilai i
 - Tabel dibentuk kembali dengan menuliskan kembali harga-harga yang ada berdasarkan isi dari TabCount

1. Elemen Tabel TabCount diinisialisasi 0

Counting Sort

Ilustrasi

3. Telusuri TabCount, untuk mengisi TabInt sesuai isi TabCount → TabInt terurut



2. Telusuri TabInt, sambil mengupdate elemen TabCount → TabCount berisi jumlah kemunculan elemen pada TabInt

Counting Sort (Algoritma)



procedure CountSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] dengan pencacahan }

Kamus Lokal

{ ValMin & ValMax: batas Minimum & Maximum nilai di T, hrs diketahui }
TabCount : array [ValMin..ValMax] of integer;
i, j : integer; { indeks untuk traversal tabel }
K : integer; { jml elemen T yg sudah diisi pada pembentukan kembali }

ALGORITMA

if (N > 1) then

 { Inisialisasi TabCount }

 i traversal [ValMin..ValMax]
 TabCount_i ← 0

 { Counting }

 i traversal [1..N]
 TabCount_{T[i]} ← TabCount_{T[i]} + 1

 { Pengisian kembali : $T_1 \leq T_2 \leq \dots \leq T_N$ }

 K ← 0

 i traversal [ValMin..ValMax]
 if (TabCount_i ≠ 0) then
 j traversal [1..TabCount_i]
 K ← K + 1
 T_K ← i

Counting Sort (Algoritma)



procedure CountSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] dengan pencacahan }

Kamus Lokal

{ ValMin & ValMax: batas Minimum & Maximum nilai di T, hrs diketahui }
TabCount : array [ValMin..ValMax] of integer;
i, j : integer; { indeks untuk traversal tabel }
K : integer; { jml elemen T yg sudah diisi pada pembentukan kembali }

ALGORITMA

if (N > 1) then

{ Inisialisasi TabCount }
i traversal [ValMin..ValMax]
 TabCount_i ← 0

Elemen Tabel TabCount
diinisialisasi 0

{ Counting }
i traversal [1..N]
 TabCount_{T[i]} ← TabCount_{T[i]} + 1

Telusuri TabInt, sambil
mengupdate elemen TabCount
→ TabCount berisi jumlah
kemunculan elemen pada
TabInt

{ Pengisian kembali : $T_1 \leq T_2 \leq \dots \leq T_N$ }
K ← 0
i traversal [ValMin..ValMax]
 if (TabCount_i ≠ 0) then
 j traversal [1..TabCount_i]
 K ← K + 1
 T_K ← i

Telusuri TabCount, untuk
mengisi TabInt sesuai isi
TabCount → TabInt terurut



Selection Sort

(Pengurutan berdasarkan Seleksi)

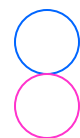
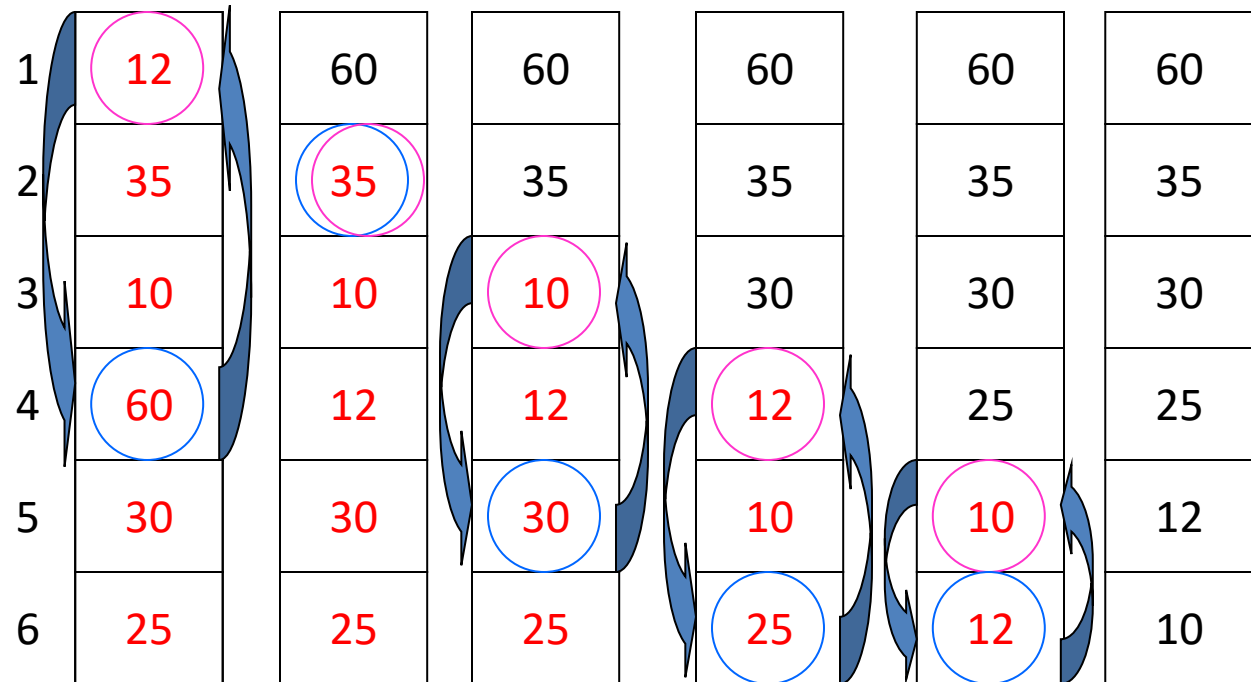
Contoh : maksimum suksesif

- Idenya adalah:
 - Cari indeks penampung nilai maksimum 'tabel'
 - Tukar elemen pada indeks maksimum dengan elemen ter'ujung'
 - elemen terujung "diisolasi", tidak disertakan pada proses berikutnya
 - proses diulang untuk sisa tabel
- Hasil proses: tabel terurut mengecil
$$T_1 \geq T_2 \geq T_3 \geq \dots \geq T_N$$
- Proses dilakukan sebanyak N-1 tahapan (disebut "pass")

Selection Sort

Ilustrasi

- Proses diulang untuk elemen $1..N-1$
- Pada iterasi ke- i :
 - Elemen $1..i-1$ sudah terurut
 - Cari indeks dgn nilai maksimum elemen $i..N$
 - Tukar elemen ke- i dengan elemen pada indeks dengan nilai maksimum



Elemen maksimum pada iterasi

Elemen yang akan menampung posisi elemen maksimum

Selection Sort (Algoritma)



procedure MAXSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] terurut mengecil dgn maksimum suksesif }

Kamus Lokal

i : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { memorisasi harga untuk penukaran }
IMax : integer { indeks, di mana T[Pass..N] bernilai maksimum }

Algoritma

```
if (N > 1) then
    Pass traversal [1..N-1]
        { Tentukan Maximum [Pass..N] }
        IMax ← Pass
        i traversal [Pass+1.. N]
            if (TIMax < Ti ) then
                IMax ← i
        { TIMax adalah maximum T[Pass..N] }
        {Tukar TIMax dengan TPass }
        Temp ← TIMax
        TIMax ← TPass
        TPass ← Temp
        { T1..Pass terurut: T1 ≥ T2 ≥ ... ≥ TPass }
{ Seluruh tabel terurut, T1 ≥ T2 ≥ ... ≥ TN }
```

Selection Sort (Algoritma)



procedure MAXSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] terurut mengecil dgn maksimum suksesif }

Kamus Lokal

i : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { memorisasi harga untuk penukaran }
IMax : integer { indeks, di mana T[Pass..N] bernilai maksimum }

Algoritma

if (N > 1) then

 Pass traversal [1..N-1]

 { Tentukan Maximum [Pass..N] }

 IMax \leftarrow Pass

 i traversal [Pass+1.. N]

if ($T_{IMax} < T_i$) then

 IMax \leftarrow i

 { T_{IMax} adalah maximum T[Pass..N] }

 { Tukar T_{IMax} dengan T_{Pass} }

 Temp $\leftarrow T_{IMax}$

$T_{IMax} \leftarrow T_{Pass}$

$T_{Pass} \leftarrow$ Temp

 { $T_{1..Pass}$ terurut: $T_1 \geq T_2 \geq \dots \geq T_{Pass}$ }

{ Seluruh tabel terurut, $T_1 \geq T_2 \geq \dots \geq T_N$ }

Cari indeks dgn nilai maksimum (di bagian tabel yang belum terurut)

Tukarkan elemen pada indeks maksimum dengan elemen terujung dari bagian tabel yang belum terurut



Insertion Sort

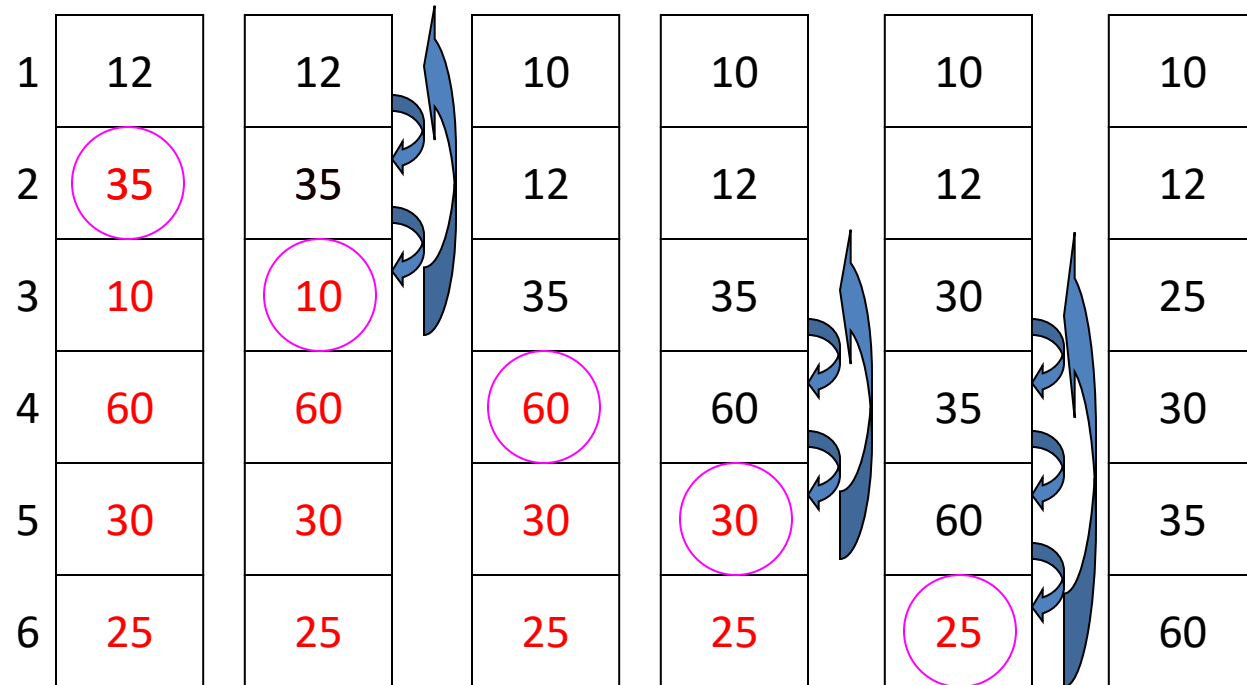
(Pengurutan dengan Penyisipan)


- **Idenya adalah:**
 - Mencari tempat yang "tepat" untuk setiap elemen tabel dengan cara menyisipkan elemen tersebut pada tempatnya di bagian tabel yang sudah terurut
 - Proses dilakukan sebanyak $N-1$ tahapan (disebut "pass").
 - Pada setiap Pass:
 - tabel "terdiri dari" dua bagian: yang sudah terurut yaitu $[1..Pass - 1]$ dan yang belum terurut yaitu $[Pass..N]$
 - Ambil elemen TPass, sisipkan ke dalam $T[1..Pass-1]$ dengan tetap menjaga keterurutan → dengan cara menggeser elemen-elemen, hingga ditemukan tempat yang cocok untuk elemen TPass tersebut

Insertion Sort

Ilustrasi

- Elemen 1 dianggap sudah terurut
- Proses diulang untuk elemen 2..N
- Pada iterasi ke-i:
 - Elemen 1..i-1 sudah terurut
 - Sisipkan elemen ke-i di antara elemen 1..i-1 dengan tetap menjaga keterurutan elemen
 - Dapat dicapai dengan cara menggeser elemen yang nilainya lebih besar



 Elemen yang akan disisipkan

Insertion Sort (Algoritma)



procedure InsertionSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] dengan insertion }

Kamus Lokal

i : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { penampung nilai sementara, untuk pergeseran }

ALGORITMA

```
if N > 1 then
  { T1 adalah terurut }
  Pass traversal [2..N]

  Temp ← TPass { Simpan harga T[Pass] sebelum pergeseran }
  { Sisipkan elemen ke Pass dalam T[1..Pass-1] sambil menggeser: }
  i ← Pass-1
  while (Temp < Ti) and (i > 1) do

    Ti+1 ← Ti { Geser }
    i ← i - 1 { Berikutnya }

  { Temp >= Ti (tempat yg tepat) or i = 1 (sisipkan sbg elmt pertama) }
  if (Temp >= Ti) then
    Ti+1 ← Temp { Menemukan tempat yg tepat }
  else

    Ti+1 ← Ti
    Ti ← Temp { sisipkan sbg elemen pertama }
  { T1..Pass terurut membesar: T1 ≤ T2 ≤ T3 ≤ ... ≤ TPass }

{ Seluruh tabel terurut, karena Pass = N: T1 ≤ T2 ≤ ... ≤ TN }
```


Insertion Sort (Algoritma)



procedure InsertionSORT (input/output T : TabInt, input N : integer)
{ mengurut tabel integer [1..N] dengan insertion }

Kamus Lokal

i : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { penampung nilai sementara, untuk pergeseran }

ALGORITMA

if N > 1 then
{ T₁ adalah terurut }
Pass traversal [2..N]

Search posisi yang tepat
untuk menyisipkan nilai

Temp \leftarrow T_{Pass} { Simpan harga T_{Pass} sebelum pergeseran }
{ Sisipkan elemen ke Pass dalam T[1..Pass-1] sambil menggeser: }
i \leftarrow Pass-1
while (Temp < T_i) and (i > 1) do

T_{i+1} \leftarrow T_i { Geser }
i \leftarrow i - 1 { Berikutnya }

{ Temp \geq T_i (tempat yg tepat) or i = 1 (sisipkan sbg elmt pertama) }
if (Temp \geq T_i) then
T_{i+1} \leftarrow Temp { Menemukan tempat yg tepat }
else

T_{i+1} \leftarrow T_i
T_i \leftarrow Temp { sisipkan sbg elemen pertama }
{ T_{1..Pass} terurut membesar: T₁ \leq T₂ \leq T₃ \leq ... \leq T_{Pass} }

{ Seluruh tabel terurut, karena Pass = N: T₁ \leq T₂ \leq ... \leq T_N }



Bubble Sort

- Idenya adalah gelembung air yang akan "mengapung" untuk tabel yang terurut membesar
- Elemen bernilai kecil akan "diapungkan" (ke indeks terkecil) melalui pertukaran
- Proses dilakukan sebanyak $N-1$ tahapan (1 tahap disebut sebagai 1 pass)

Bubble Sort

Ilustrasi



1	25
2	30
3	10
4	60
5	35
6	12

Initial State

1	10
2	25
3	30
4	12
5	60
6	35

pass = 1

1	10
2	12
3	25
4	30
5	35
6	60

pass = 2

1	10
2	12
3	25
4	30
5	35
6	60

pass = 3

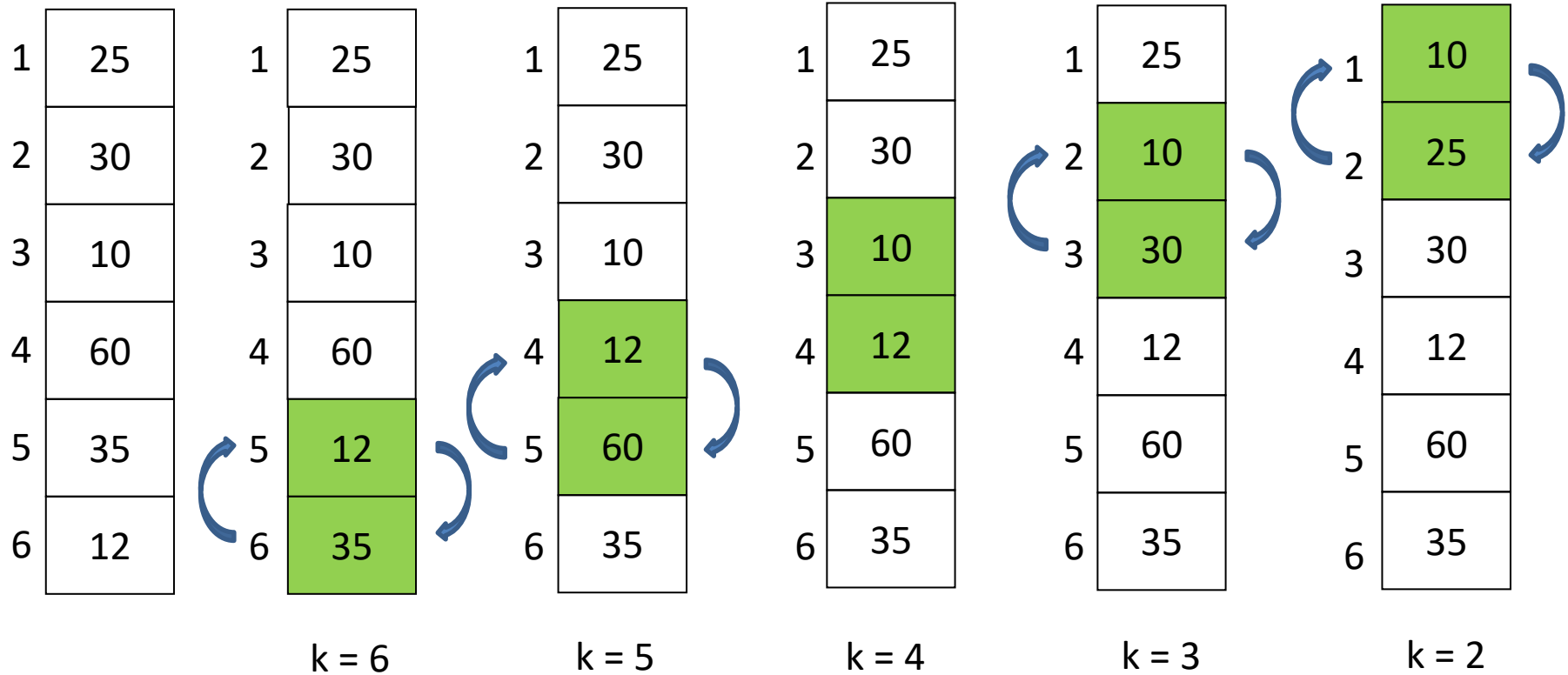
1	10
2	12
3	25
4	30
5	35
6	60

pass = 4

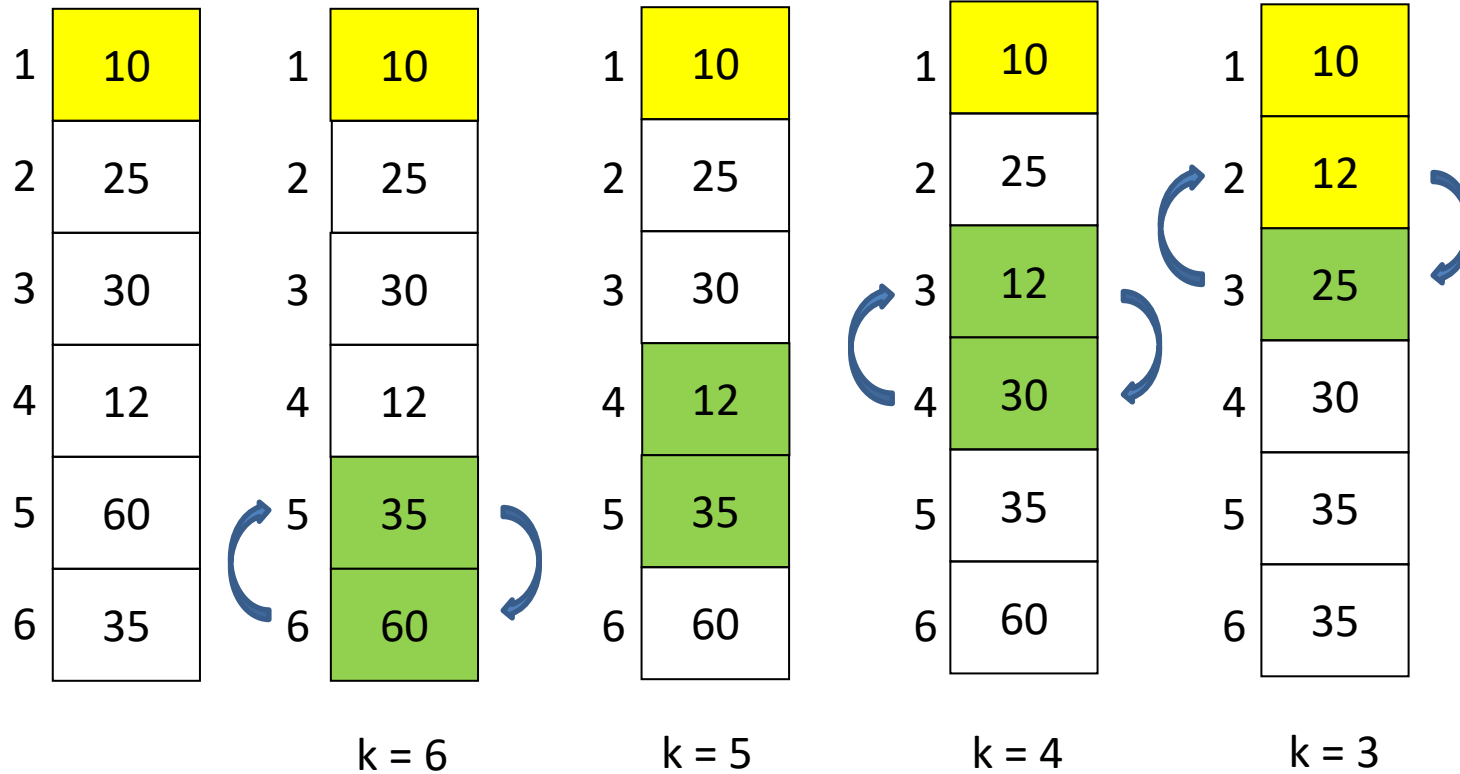
1	10
2	12
3	25
4	30
5	35
6	60

pass = 5

Pass=1



Pass=2



dst...

Bubble Sort (Algoritma) versi asli



procedure BubbleSort (input/output T : TabInt, input N : integer)
{ Mengurutkan tabel integer [1..N] dengan bubble sort }

KAMUS LOKAL

i, K : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { Memorisasi untuk pertukaran harga }

ALGORITMA

```
if N > 1 then
    Pass traversal [1..N-1]
    K traversal [N..Pass+1]
        if ( $T_K < T_{K-1}$ ) then
            Temp  $\leftarrow T_K$ 
             $T_K \leftarrow T_{K-1}$ 
             $T_{K-1} \leftarrow Temp$ 
    { T[1..Pass] terurut:  $T_1 \leq T_2 \leq T_3 \leq \dots \leq T_{Pass}$  }
    { Seluruh tabel terurut, karena Pass = N:  $T_1 \leq T_2 \leq T_3 \leq \dots \leq T_N$  }
```



Bubble Sort (versi optimum)

- Versi asli biasanya mudah diingat karena prinsipnya yang alamiah
- Proses dapat dihentikan jika tidak terjadi pertukaran
 - Manfaatkan variable boolean
- Selanjutnya versi asli tidak digunakan → yang digunakan adalah versi optimum

Bubble Sort (versi optimum) - Algoritma



procedure BubleSortPlus (input/output T : TabInt, input N : integer)
{ Mengurut tabel integer [1..N] dengan bubble sort }
{ Pengurutan dihentikan jika tak ada pertukaran lagi }

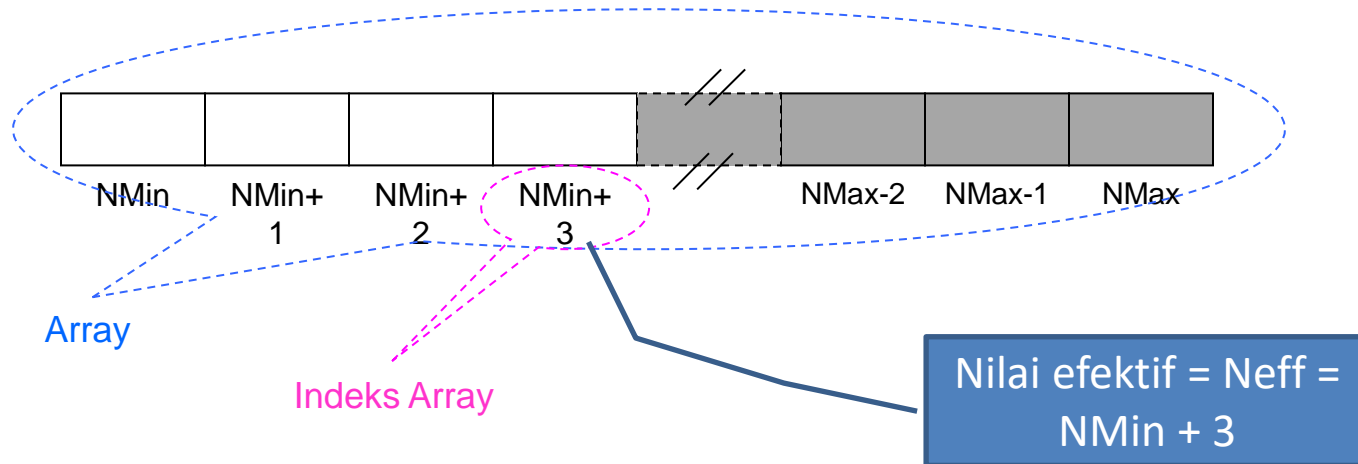
KAMUS LOKAL

i : integer { indeks untuk traversal tabel }
Pass : integer { tahapan pengurutan }
Temp : integer { memorisasi untuk pertukaran harga }
Tukar : boolean { true jika dalam satu pass ada pertukaran }

ALGORITMA

```
if N > 1 then
    Pass ← 1
    Tukar ← true      { masih harus ada pertukaran }
    while (Pass ≤ N-1) and (Tukar) do
        Tukar ← false
        K traversal [N..Pass+1]
            if (TK < TK-1) then
                Temp ← TK
                TK ← TK-1
                TK-1 ← Temp
                Tukar ← true
        { T[1..Pass] terurut: T1 ≤ T2 ≤ T3 ≤ ... ≤ TPass }
        { Tukar = true jika ada pertukaran }
        Pass ← Pass + 1      { ke pass yang berikutnya }
    { Seluruh tabel terurut, karena Pass = N: T1 ≤ T2 ≤ T3 ≤ ... ≤ TN }
```


Array yang Terisi Sebagian (1)



- Pada pembahasan sebelumnya, kita mendefinisikan array yang hanya terisi “sebagian” yaitu hanya terisi/terdefinisi dari elemen ke- $NMin$ s.d. ke- N_{eff} (lihat contoh di atas)
 - Elemen $NMin+4$ s.d. $NMax$ dianggap tidak terdefinisi (oleh karena itu, tidak boleh diakses)



Array yang Terisi Sebagian (2)

- Sejauh ini, kita mendeklarasikan array dan nilai efektif secara terpisah, spt contoh berikut:

```
{ KAMUS }  
constant NMax : integer = 100  
type TabInt : array [1..NMax] of integer  
{ jika diperlukan sebuah tabel, maka akan dibuat deklarasi  
  sebagai berikut: }  
  N : integer      { indeks efektif maksimum tabel yang  
                    terdefinisi,  $0 \leq N \leq NMax$  }  
  T : TabInt       { tabel integer }
```

- Akibatnya selalu harus didefinisikan 2 buah variabel (N, T) untuk mendefinisikan sebuah array



Array yang Terisi Sebagian (3)

- Bagaimana jika kedua elemen dikumpulkan dalam sebuah type:

```
{ KAMUS }  
constant NMax : integer = 100  
type TabInt : < Tab : array [1..NMax] of integer;  
                Neff : integer { indeks efektif tabel yang  
                                terdefinisi,  $0 \leq N \leq \text{NMax}$  } >  
  
{ jika diperlukan sebuah tabel, maka akan dibuat deklarasi  
  sebagai berikut: }  
  
T1 : TabInt    { Cara akses elemen:  
                  T1.Tabi untuk akses elemen ke-i dari T1.Tab  
                  T1.Neff untuk akses nilai efektif - Neff }
```



Array dengan Elemen Type Bentukan

- Elemen array dapat bertipe dasar maupun bentukan
 - Contoh: array integer, real, Point, dll.
- Contoh deklarasi array of Point:

```
{ KAMUS }
constant NMax : integer = 100
type Point : < x : integer; y : integer > { Titik di bidang kartesius }
type TabPoint : < Tab : array [1..NMax] of Point;
                  Neff : integer { indeks efektif tabel yang
                                terdefinisi,  $0 \leq N \leq NMax$  } >

{ Contoh deklarasi: }
TP : TabPoint
    { Cara akses elemen:
      TP.Tabi.x untuk akses bagian x dari elemen ke-i dari TP.Tab
      TP.Tabi.y untuk akses bagian y dari elemen ke-i dari TP.Tab
      TP.Neff untuk akses nilai efektif - Neff }
```



Latihan Skema Sorting (1a)

- Diberikan definisi kamus berikut ini:

```
type TMahasiswa : < NIM : string,  
                    Nama : string,  
                    Nilai : integer { [0..100] }  
                    >  
type TabMhs : < TM : array [1..NMax] of TMahasiswa,  
              Neff : integer  
              >
```

- Buatlah prosedur **UrutTabMhs** yang digunakan untuk mengurutkan elemen TMhs secara terurut mengecil pada atribut Nilai. Pengurutan dilakukan dengan pendekatan seleksi.

```
procedure UrutTabMhs (input/output TMhs : TabMhs)
```



Latihan Skema Sorting (1b)

- Bisakah persoalan 1a diselesaikan dengan menggunakan pendekatan pencacah (*counting sort*)?
- Jelaskan jawaban anda.



Latihan Skema Sorting (2a)

Menggunakan definisi pada slide 27, buatlah procedure di bawah ini:

```
procedure InputTerurut (input/output T : TabInt,  
                        input X : integer)
```

yang digunakan untuk memasukkan elemen X ke dalam T.

T harus selalu dalam kondisi terurut membesar (sebelum dan sesudah pemasukan elemen X).



SELAMAT BELAJAR