Array (Tabel)
bagian 2

Tim Pengajar KU1071

Sem. 1 2009-2010

Tujuan Perkuliahan

- Mahasiswa dapat menggunakan notasi pendefinisian dan pengacuan array dengan benar
- Mahasiswa memahami proses pencarian nilai ekstrim dan pengurutan elemen yang dapat dilakukan terhadap array
- Mahasiswa dapat membuat program dengan menggunakan array

Harga Ekstrim Tabel

- Pencarian nilai ekstrim (minimum/maksimum) dalam tabel merupakan proses yang cukup penting
  - Mencari mahasiswa dengan nilai tertinggi
  - Mengeliminasi nilai tertinggi dan terendah dari suatu percobaan
- Persoalan:
  - Diketahui sebuah tabel bilangan integer T[1..N] yang telah diisi
  - Tuliskanlah Program Max, yang menghasilkan harga maksimum dari elemen tabel: $\forall i \in [1..N] \quad T_i \leq Max$.

  Contoh 1: N = 8, T berisi : { 1, -3, 5, 8, -12, 90, 3, 5}

  Output : Maximum adalah 90

  Contoh 2: N = 11, T berisi : { -11, 3, 45, 8,3,45,-6,7,8,9,1}

  Output : Maksimum adalah 45

Pencarian Nilai Maksimum
Versi mengembalikan nilai maksimum

procedure MAX1 (input T : TabInt, input N : integer, output MAX : integer)
{ Pencarian harga Maksimum: }
{ I.S. Tabel tidak kosong, karena jika kosong maka maks tidak terdefinisi, N >=0 }
{ F.S. Menghasilkan harga Maksimum MAX dari Tabel T[1..N] secara sekuensial mulai dari indeks 1..N}

Kamus Lokal :
i : integer {indeks untuk pencarian}

ALGORITMA

```
MAX ← T1  {inisialisasi, T1 diasumsikan merupakan nilai maksimum}
i ← 2     {pembandingan nilai maksimum dimulai dari elemen ke-2}
while i ≤ N do
        if (MAX < Ti) then
                MAX ← Ti
        i ← i + 1
{ i > N : semua elemen sudah selesai diperiksa }
```

Nilai yang dihasilkan adalah nilai maksimum, indeks tempat nilai maksimum tidak diketahui

Elemen pertama tabel diproses secara khusus

Pencarian Nilai Maksimum

Versi mengembalikan indeks maksimum

```
procedure MAX2 (input T : TabInt, input N : integer, output IMax : integer)
{ Pencarian indeks dengan harga Maksimum}
{ I.S. Tabel tidak kosong, karena jika kosong maka maks tidak terdefinisi,
N >0 }
{ F.S. Menghasilkan indeks IMax terkecil, dengan harga  T_IMax  dalam
Tabel T [1..N] adalah maksimum }
```

Kamus Lokal :

```
i : integer
```

ALGORITMA

```
IMax ← 1
i ← 2
while i ≤ N  do
        if (T_IMax < T_i ) then
                IMAX ← i
        i ← i + 1
{ i > N : semua elemen sudah selesai diperiksa }
```

Tidak menghasilkan nilai maksimum melainkan indeks dimana nilai maksimum berada

Elemen pertama tabel diproses secara khusus

Pencarian Nilai Maksimum
Versi maksimum dari bil. positif v.1

procedure MAXPOS(input T:TabInt, input N:integer, output MAX:integer)
{ Pencarian harga Maksimum di antara bilangan positif}
{ I.S. Tabel mungkin kosong: N=0, semua elemen tabel T bernilai positif }
{ F.S. Max berisi nilai elemen tabel yang maksimum. Jika kosong, MAX=-9999 }
{ Menghasilkan harga Maksimum (MAX) Tabel T [1..N] secara sekuensial mulai dari T1}

Kamus Lokal :

i : integer

ALGORITMA

MAX ← -9999 { inisialisasi dengan nilai yang pasti dapat digantikan!
dalam hal ini nilai minimum representasi integer }
i traversal [1.. N]
if (MAX < T_i) then
MAX ← T_i
{ i = ??, semua elemen sudah selesai diperiksa }

Semua elemen tabel diperiksa dengan cara yang sama. Oleh sebab itu, nilai MAX harus diinisialisasi dengan nilai yang sudah pasti akan digantikan oleh nilai yang ada di dalam tabel

Pengulangan ini tidak aman untuk seluruh kasus. Carilah letak permasalahannya.

Pencarian Nilai Maksimum
Versi maksimum dari bil. positif v.2

procedure MAXPOS(input T:TabInt, input N:integer, output MAX:integer)
{ Pencarian harga Maksimum di antara bilangan positif}
{ I.S. Tabel mungkin kosong: N=0, semua elemen tabel T bernilai positif }
{ F.S. Max berisi nilai elemen tabel yang maksimum. Jika kosong, MAX=-9999 }
{ Menghasilkan harga Maksimum (MAX) Tabel T [1..N] secara sekuensial mulai dari T_1}

Kamus Lokal :

```
i : integer
```

ALGORITMA

```
MAX ← -9999 { inisialisasi dengan nilai yang pasti dapat digantikan!
                dalam hal ini nilai minimum representasi integer }
i ← 1
while (i ≤ N) do
    if (MAX < Ti) then
        MAX ← Ti
    i ← i + 1
{ i = N+1, semua elemen sudah selesai diperiksa }
```

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, Maximum 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 :

  $T_1 \leq T_2 \leq T_3 \ldots\ldots \leq T_N$

- Kamus Umum:

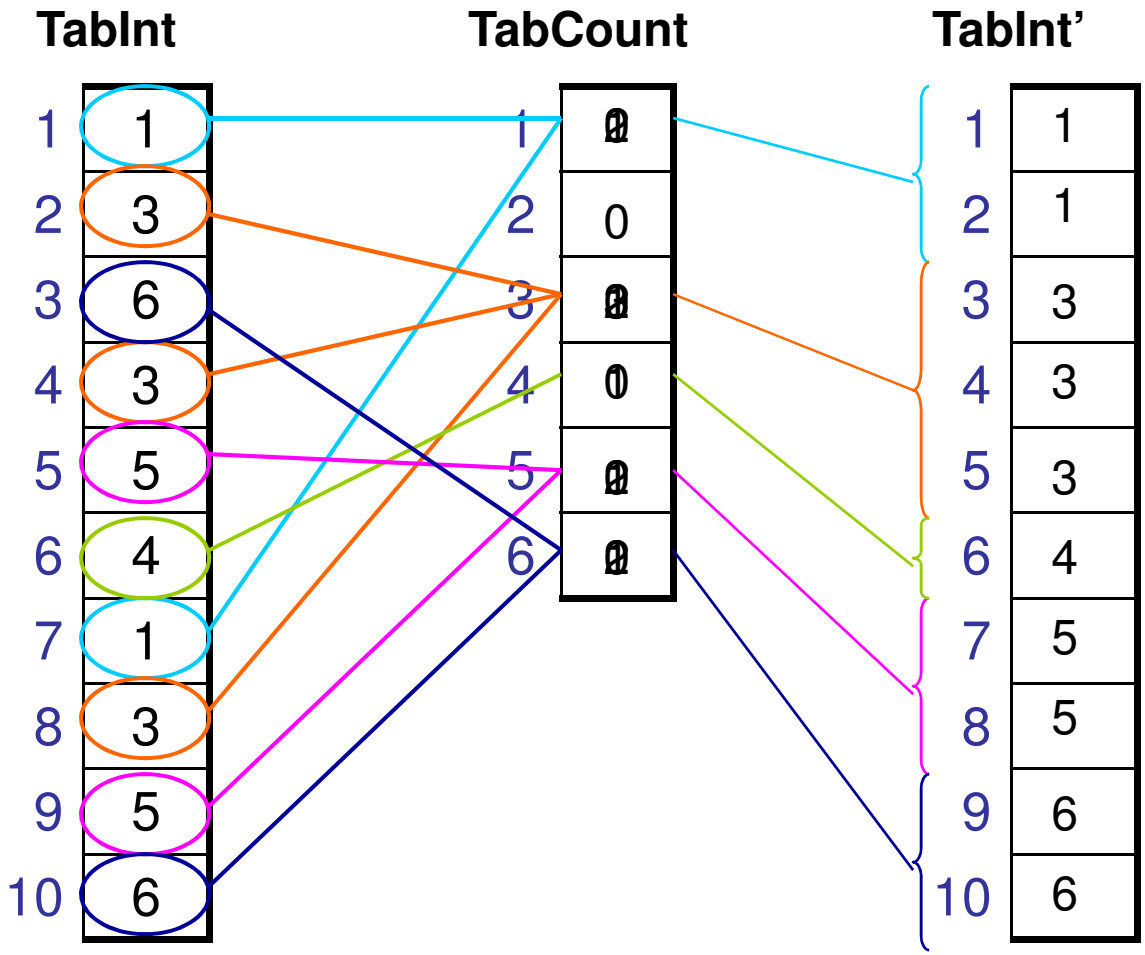
KAMUS
  constant NMax : integer = 100
  type TabInt : array [1..NMax] of integer
  N : integer {indeks efektif maksimum tabel yang terdefinisi, N ≤ Nmax}
{ jika diperlukan sebuah tabel, maka akan dibuat deklarasi sebagai berikut }
  T : TabInt { tabel integer }

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

Counting Sort
Ilustrasi

1. Elemen Tabel TabCount diinisialisasi 0
2. Telusuri TabInt, sambil mengupdate elemen TabCount → TabCount berisi jumlah kemunculan elemen pada TabInt
3. Telusuri TabCount, untuk mengisi TabInt sesuai isi TabCount → TabInt terurut



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 dan Maximum nilai dalam T, harus diketahui}
TabCount : array [ValMin..ValMax] of integer [0..NMax]
i, j : integer { indeks untuk traversal tabel }
K : integer { jumlah elemen T yang sudah diisi pada pembentukan kembali }

ALGORITMA

```
{ Inisialisasi TabCount }
i traversal [ValMin..ValMax]
        TabCount_i ← 0
{ Counting }
i traversal [1..N]
        TabCount_Ti ← TabCount_Ti + 1
{ Pengisian kembali : T_1 ≤ T_2 ... ≤ T_N }
K ← 0
i traversal [ValMin..ValMax]
        if (TabCount_i ≠ 0) then
                j traversal [1..TabCount_i]
                        K ← K + 1
                        T_K ← i
```

Elemen Tabel TabCount diinisialisasi 0

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

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

Selection Sort
(Pengurutan berdasarkan Seleksi)

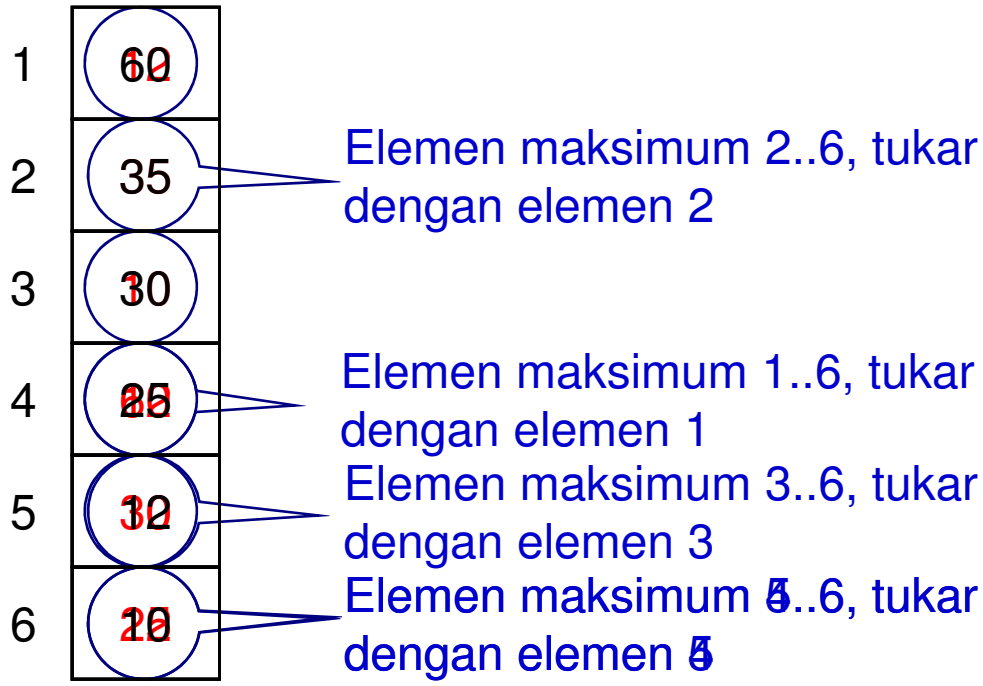
Contoh : maksimum suksesif

- 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**

  $T_1 \geq T_2 \geq T_3 \ldots\ldots \geq T_N$
- Proses dilakukan sebanyak N-1 tahapan (disebut "pass")

Selection Sort
Ilustrasi

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



Selection Sort Algoritma

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

Kamus Lokal :

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

Algoritma

```
Pass traversal  [1..N-1]
    { Tentukan Maximum [Pass..N] }
    IMax ← Pass
    i traversal [Pass+1..N]
            if (T_IMax < T_i ) then
                    IMax ← i
    { T_IMax adalah maximum T[Pass..N] }
    {Tukar T_IMax dengan T_Pass }
    Temp ← T_IMax
    T_IMax ← T_Pass
    T_Pass ← Temp
    { T[1..Pass] terurut: T1≥T2≥T3 .. ≥T_Pass}
{Seluruh tabel terurut, T1 ≥ T2 ≥ T3 ..... ≥ TN }
```

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

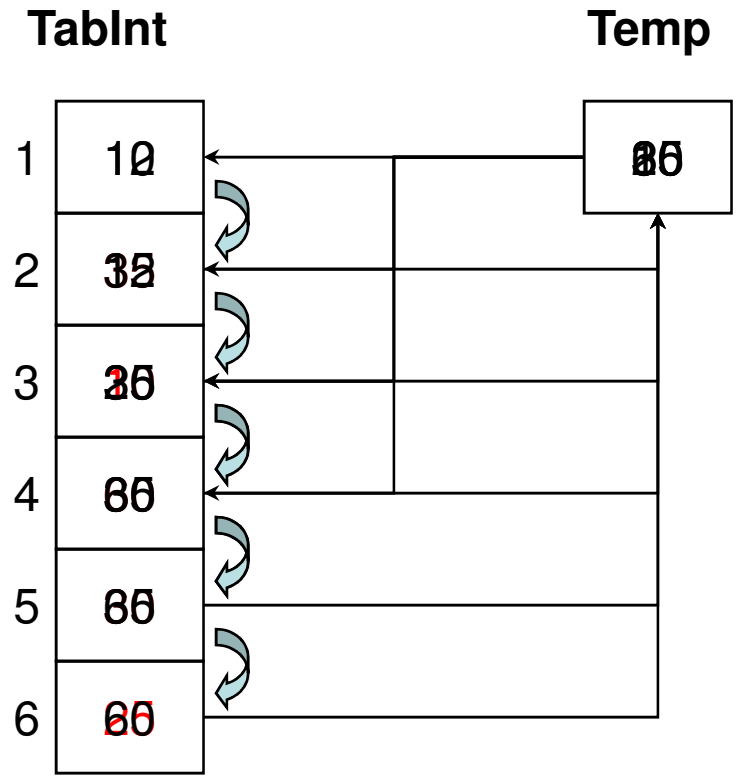
Tukarkan elemen pada indeks maksimum dengan elemen terujung dari bagian tabel yang belum terurut

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..NMax]
    - Ambil elemen T_{Pass}, sisipkan ke dalam T[1..Pass-1] dengan tetap menjaga keterurutan → dengan cara menggeser elemen-elemen, hingga ditemukan tempat yang cocok untuk elemen T_{Pass} tersebut

Insertion Sort
Ilustrasi

- Elemen 1 dianggap sudah terurut
- Proses diulang untuk elemen 2..NMax
- 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



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

```
{ T1 adalah terurut}
Pass traversal  [2..N]
        Temp ← TPass   {Simpan harga TPass 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)}
        depend on T, i, Temp
                Temp ≥ Ti:  Ti+1 ←  Temp      {Menemukan tempat yg tepat}
                Temp < Ti : Ti+1 ← Ti
                            Ti ←  Temp {sbg elemen pertama }
                {T[1..Pass] terurut membesar: T1≤ T2≤ .. ≤ TPass}
  {Seluruh tabel terurut, karena Pass = N : T1≤ T2 ≤ T3... ≤ TN}
```

Latihan

1. Diberikan definisi berikut ini:

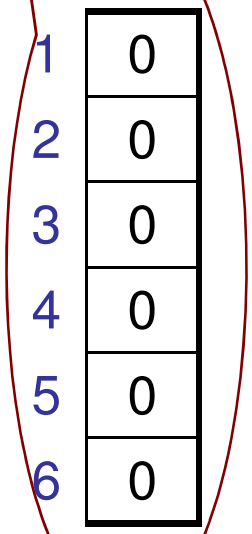
   type TMahasiswa : <NIM: string, Nama: string, Nilai: integer [0..100]>

   type TabMhs: array [1..Nmax] of TMahasiswa

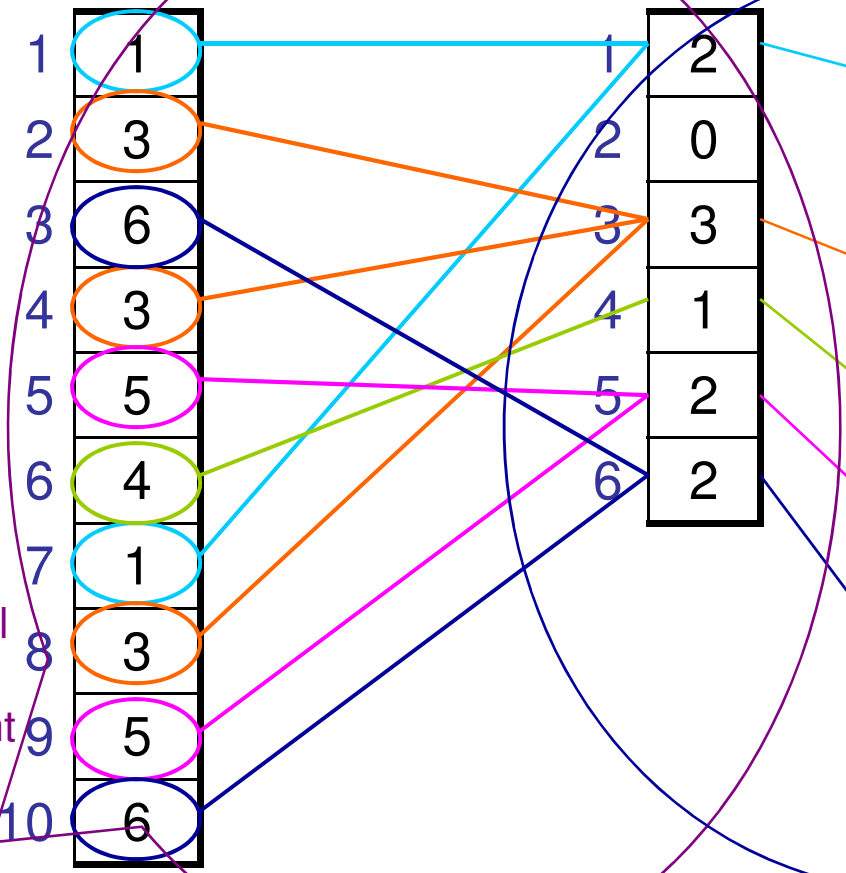
   Tuliskan prosedur UrutTabMhs(input/output TMhs: TabMhs, input N : integer) yang mengurutkan elemen TMhs berdasarkan atribut Nilai dengan urutan mengecil. Pengurutan dilakukan dengan pendekatan seleksi.
2. Bisakah persoalan no. 1 diselesaikan dengan menggunakan pendekatan pencacah (*counting sort*)? Jelaskan jawaban anda.
3. Tuliskan prosedur InputTerurut(input/output T: TabInt, input/output N: integer, X: integer), yang memasukkan X ke dalam T. T harus selalu dalam kondisi terurut (sebelum dan sesudah pemasukan elemen X). N menyimpan indeks efektif maksimum T

Counting Sort Ilustrasi

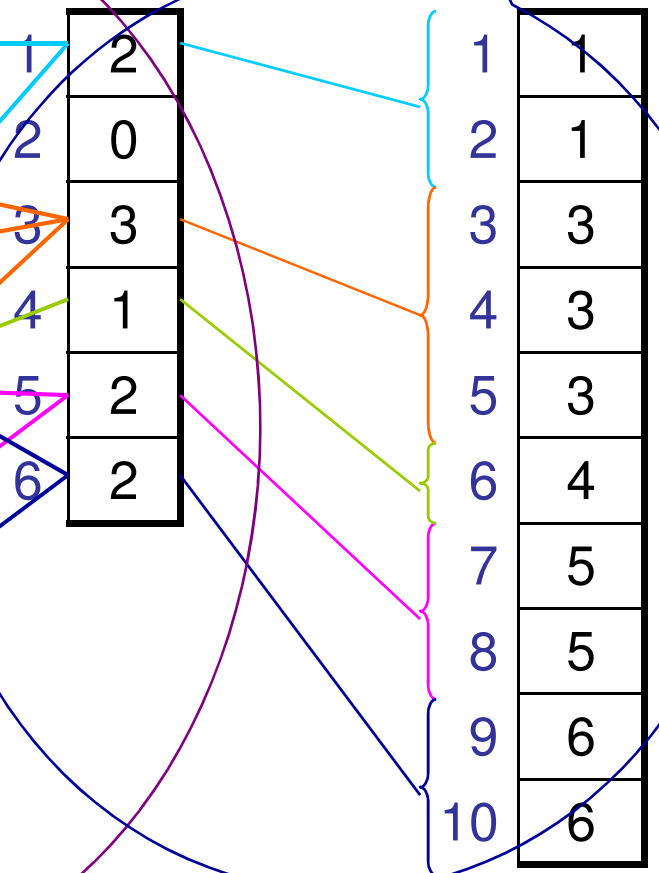
1. Elemen Tabel TabCount diinisialisasi 0



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

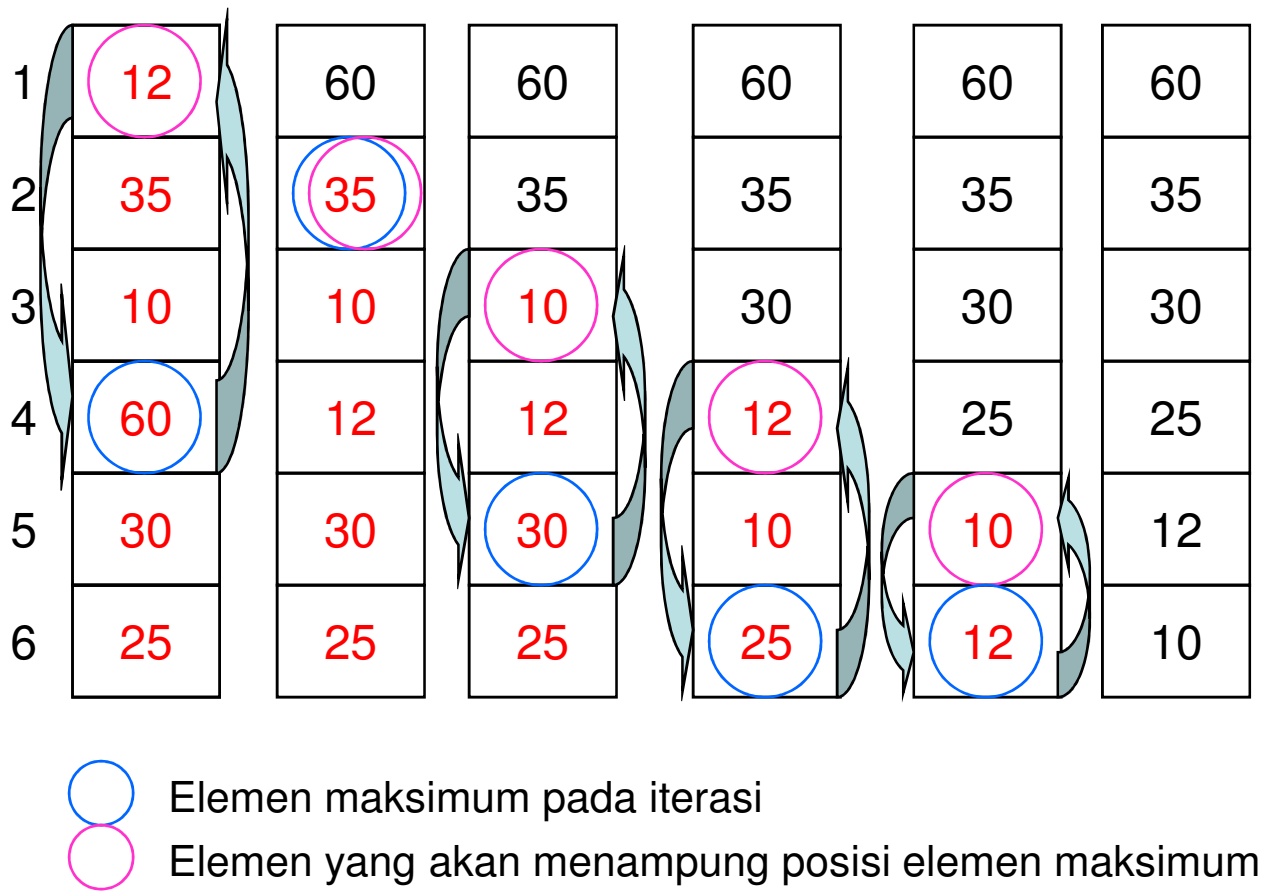


3. Telusuri TabCount untuk mengisi TabInt sesuai isi TabCount → TabInt terurut



Selection Sort
Ilustrasi

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



Insertion Sort Ilustrasi

- Elemen 1 dianggap sudah terurut
- Proses diulang untuk elemen 2..NMax
- 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