



PROGRAM STUDI

S1 SISTEM KOMPUTER

UNIVERSITAS DIPONEGORO

Algoritma dan Pemrograman Sorting dan Searching

Oky Dwi Nurhayati, ST, MT
email: okyd@undip.ac.id

SORTING

- ▶ *Sorting* = pengurutan
- ▶ *Sorted* = terurut menurut kaidah/aturan tertentu
- ▶ Data pada umumnya disajikan dalam bentuk *sorted*.
- ▶ Contoh:
 - Nama di buku telpon
 - Kata-kata dalam kamus
 - File-file di dalam sebuah directory
 - Indeks sebuah buku
 - Data mutasi rekening tabungan
 - CD di toko musik

SORTING

- ▶ Beberapa algoritma untuk melakukan sorting:
 - Bubble sort
 - Selection sort
 - Insertion sort
 - Shell sort
 - Merge sort
 - Quick sort
- ▶ Untuk masing-masing algoritma:
 - Ide dasar
 - Contoh eksekusi
 - Algoritma
 - Analisa *running time*/kompleksitas

SORTING

unitan ke

I	A (I)	1	2	3	4	5	6	7	8	9
1	42	11	11	11	11	11	11	11	11	11
2	23	23	23	23	23	23	23	23	23	23
3	74	74	74	36	36	36	36	36	36	36
4	11	42	42	42	42	42	42	42	42	42
5	65	65	65	65	65	58	58	58	58	58
6	58	58	58	58	58	65	65	65	65	65
7	94	94	94	94	94	94	94	74	74	74
8	36	36	36	74	74	74	74	94	87	87
9	99	99	99	99	99	99	99	99	99	94
10	87	87	87	87	87	87	87	87	94	99

Gambar 1. Metode Seleksi

SORTING

Metode Bubble

Algoritma beroperasi sebagai berikut;

- elemen pertama dibandingkan dengan elemen kedua. Bila elemen kedua $<$, kedua elemen tersebut ditukar.
- elemen kedua dan ketiga dibandingkan, bila elemen ketiga $<$ kedua elemen ditukar
- proses terus berlangsung dengan elemen ketiga dan keempat, dan seterusnya. Sampai akhir deretan data tercapai.
- bila tak ada lagi yang ditukarkan, algoritma berhenti. Bila terjadi pertukaran selama berurutan, proses akan diulang. Sehingga akhirnya semua elemen tersusun, tidak ada pertukaran lagi, dan algoritma berhenti.

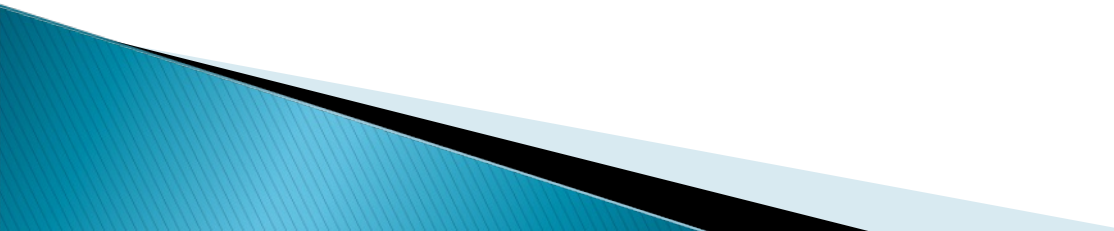
SORTING

urutan ke

I	A (I)	1	2	3	4	5	6	7	8	9
1	3 3	3	3	3	3	3	3	3	1
2	11 11	4	4	4	4	4	4	3
3	4 11	6	5	5	5	5	4
4	6 11	6	6	6	6	5
5	5 11	9	7	7	6
6	9 11	9	8	7
7	7 11	9	8
8	8 11	9
9	1 11
										↑
										hasil

Gambar 3. Metode Sisipan

Selection sort: Ide dasar

- ▶ Kondisi awal:
 - Unsorted list = data
 - Sorted list = kosong
 - ▶ Ambil yang terbaik (**select**) dari unsorted list, **tambahkan** di belakang sorted list.
 - ▶ Lakukan terus sampai unsorted list habis.
- 

Selection sort: Contoh

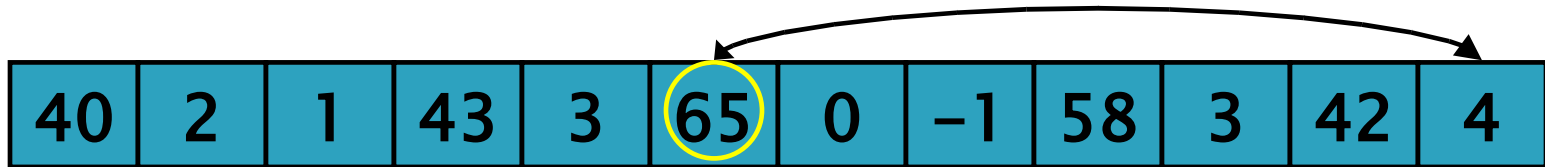


Diagram illustrating the first step of Selection Sort. The array contains 12 elements. The element 65 is circled in yellow, indicating it is the current element being compared. A curved arrow points from 65 to the element 4 at the end of the array, indicating that 4 is the minimum element found in the unsorted portion.

40	2	1	43	3	65	0	-1	58	3	42	4
----	---	---	----	---	----	---	----	----	---	----	---

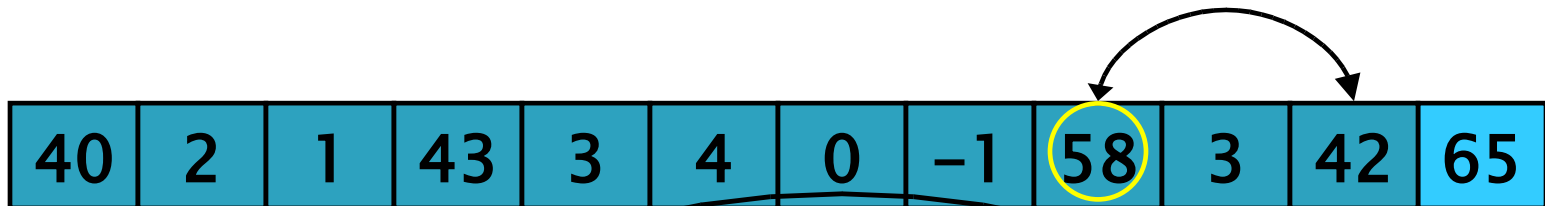


Diagram illustrating the second step of Selection Sort. The element 58 is circled in yellow, indicating it is the current element being compared. A curved arrow points from 58 to the element 42 at the end of the array, indicating that 42 is the minimum element found in the unsorted portion.

40	2	1	43	3	4	0	-1	58	3	42	65
----	---	---	----	---	---	---	----	----	---	----	----

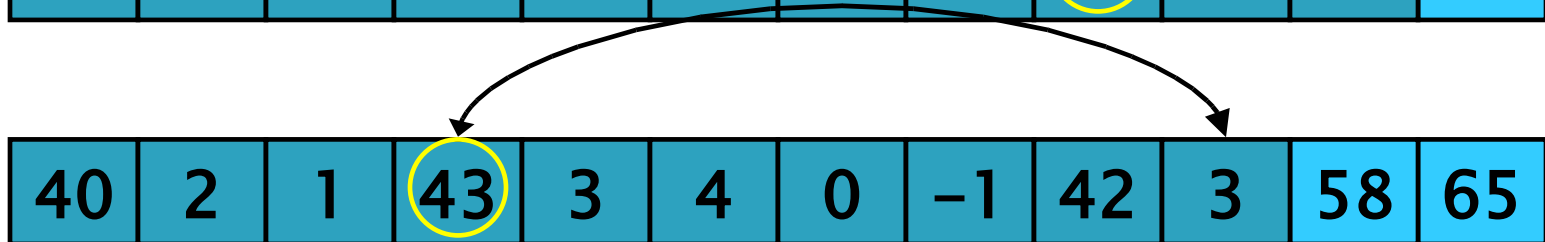


Diagram illustrating the third step of Selection Sort. The element 43 is circled in yellow, indicating it is the current element being compared. A curved arrow points from 43 to the element 3 at the end of the array, indicating that 3 is the minimum element found in the unsorted portion.

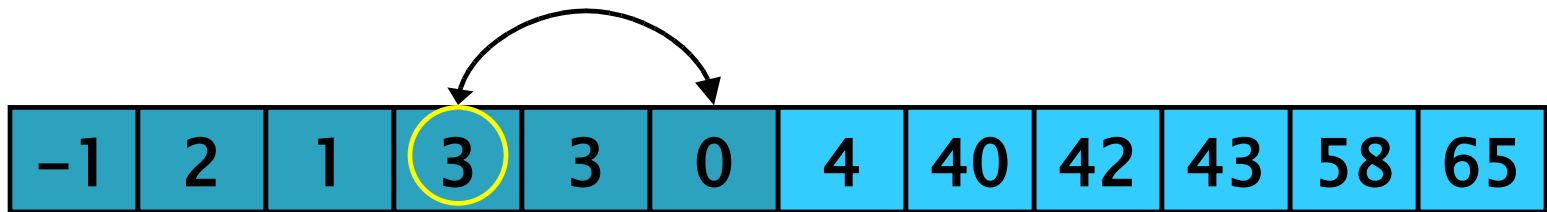
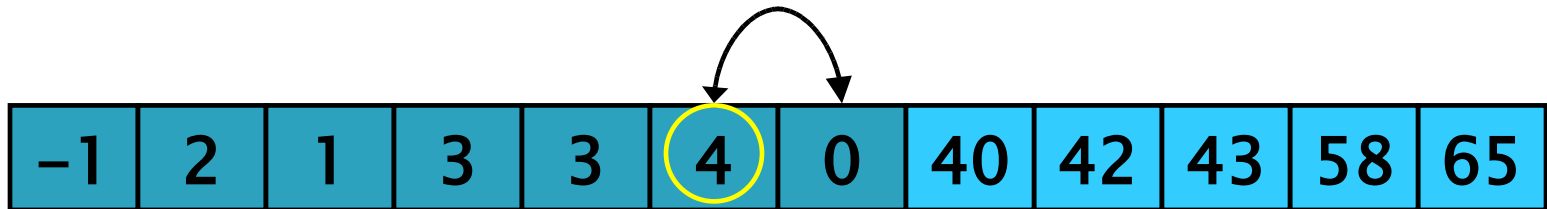
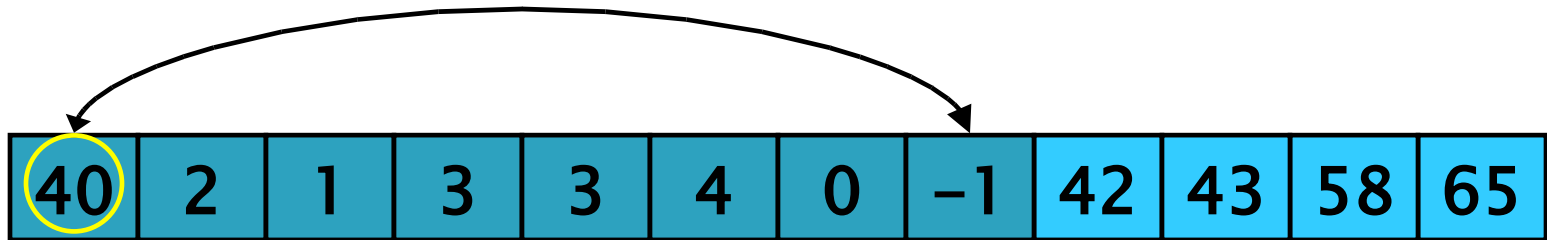
40	2	1	43	3	4	0	-1	42	3	58	65
----	---	---	----	---	---	---	----	----	---	----	----



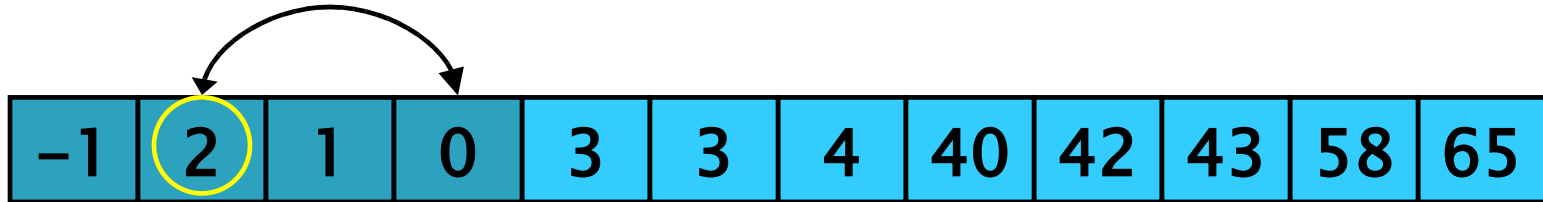
Diagram illustrating the fourth step of Selection Sort. The element 42 is circled in yellow, indicating it is the current element being compared. A curved arrow points from 42 to the element 43 at the end of the array, indicating that 43 is the minimum element found in the unsorted portion.

40	2	1	3	3	4	0	-1	42	43	58	65
----	---	---	---	---	---	---	----	----	----	----	----

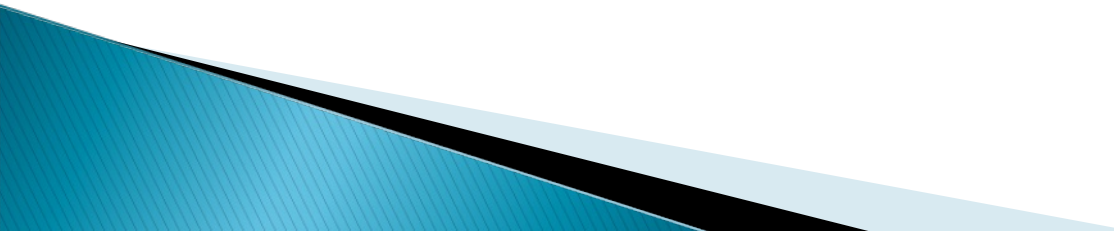
Selection sort: Contoh (lanj.)



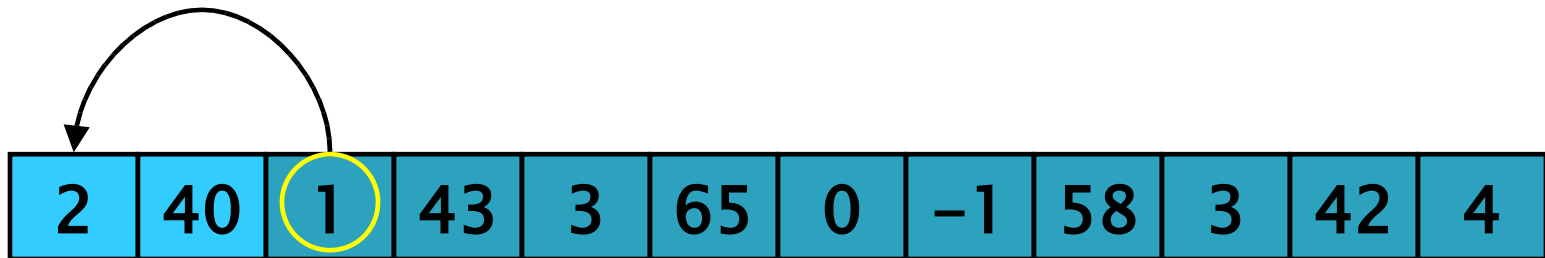
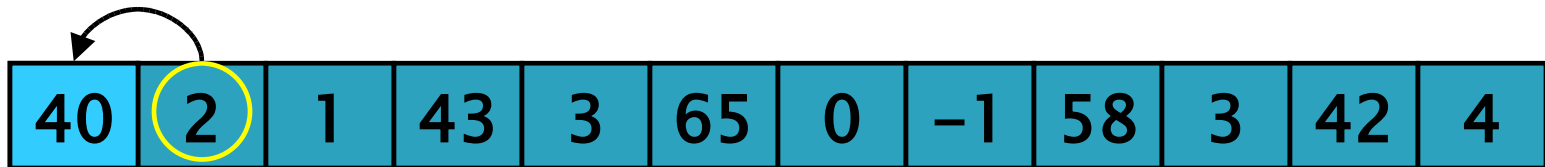
Selection sort: Contoh (lanj.)



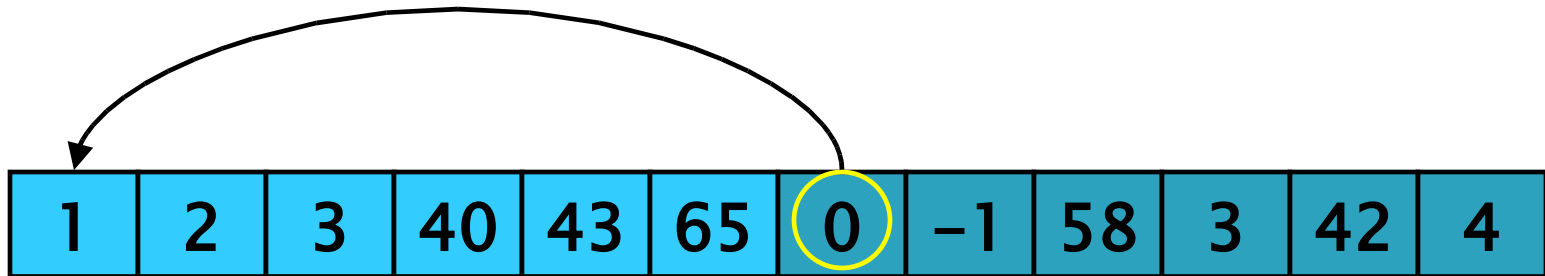
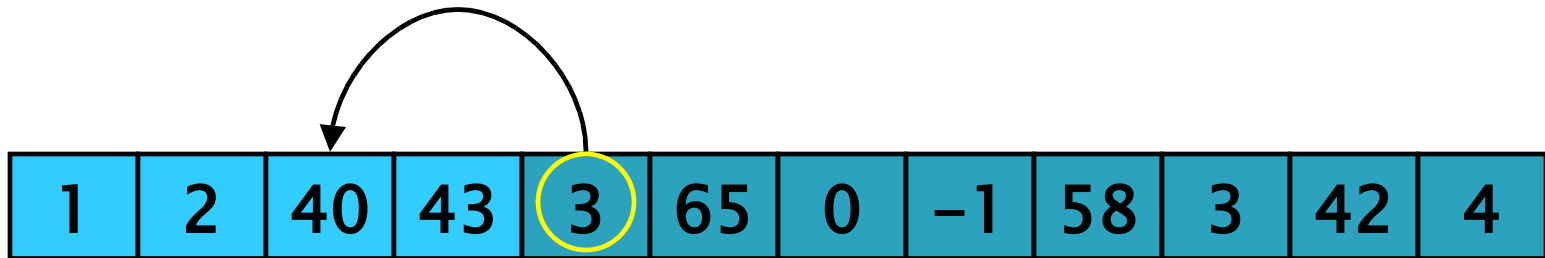
Insertion sort: Ide dasar

- ▶ Kondisi awal:
 - Unsorted list = data
 - Sorted list = kosong
 - ▶ Ambil **sembarang** elemen dari unsorted list, sisipkan (**insert**) pada posisi yang benar dalam sorted list.
 - ▶ Lakukan terus sampai unsorted list habis.
 - ▶ Bayangkan anda mengurutkan kartu.
- 

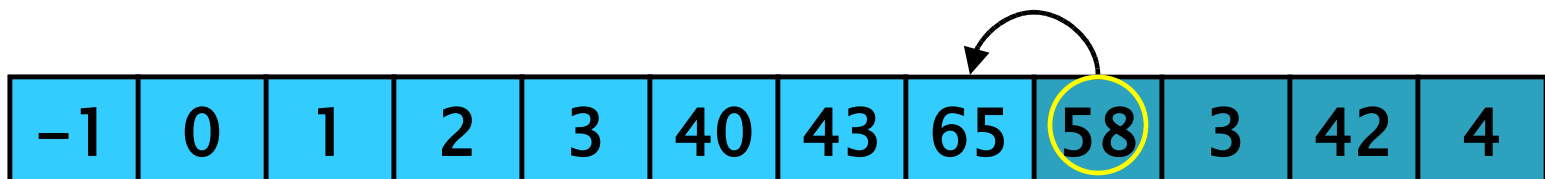
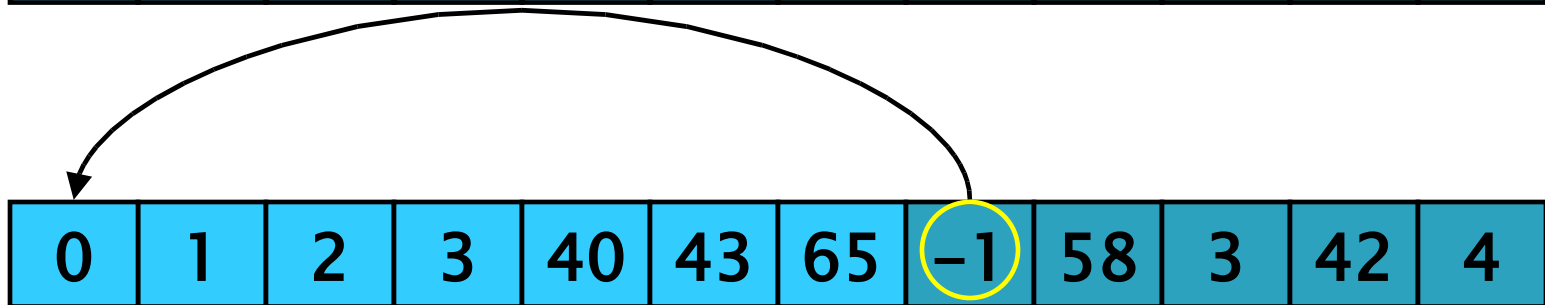
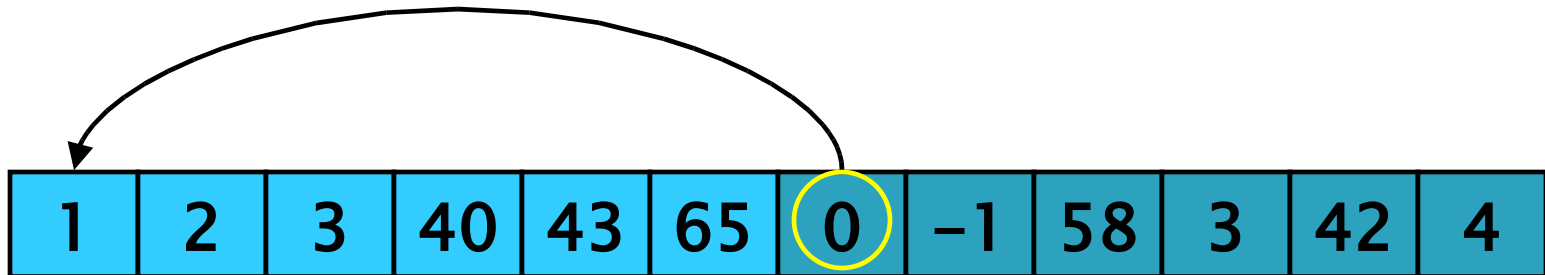
Insertion sort: Contoh



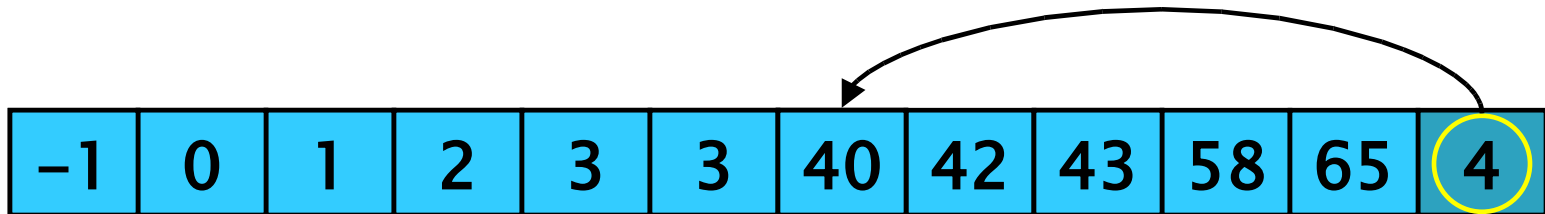
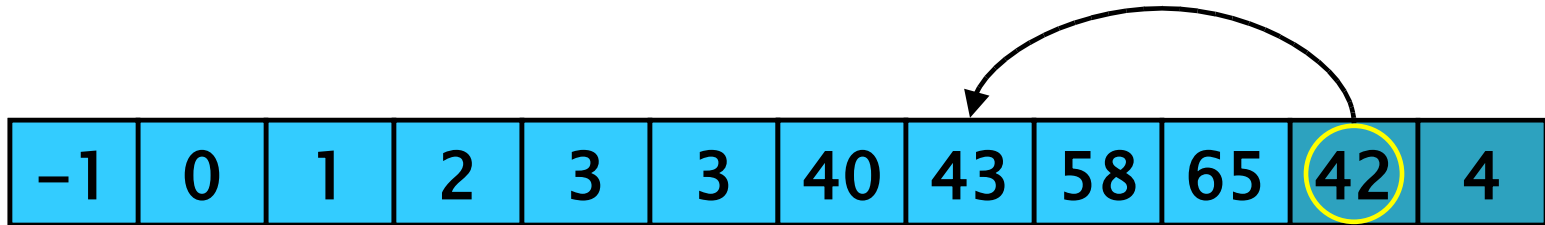
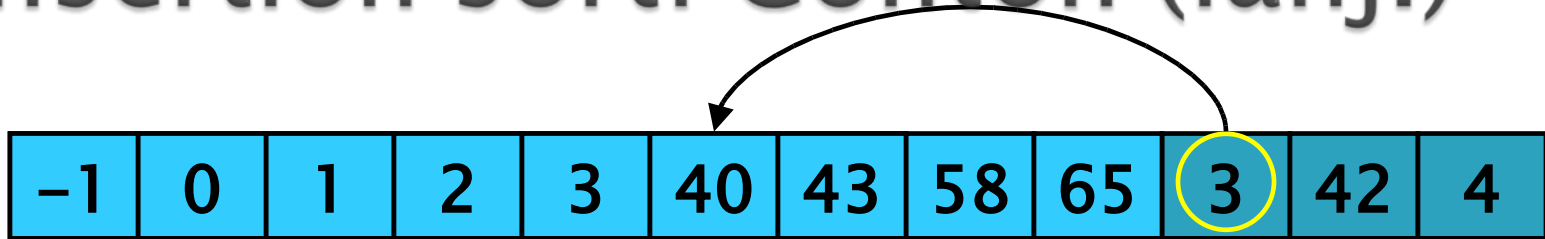
Insertion sort: Contoh (lanj.)



Insertion sort: Contoh (lanj.)



Insertion sort: Contoh (lanj.)



Metode Shell Sort

- perluasan dari pada metode sisipan
- perbandingan bukan pada rekaman yang berdekatan, tetapi pada rekaman dengan spasi tertentu, misal_

babak I = spasi 5

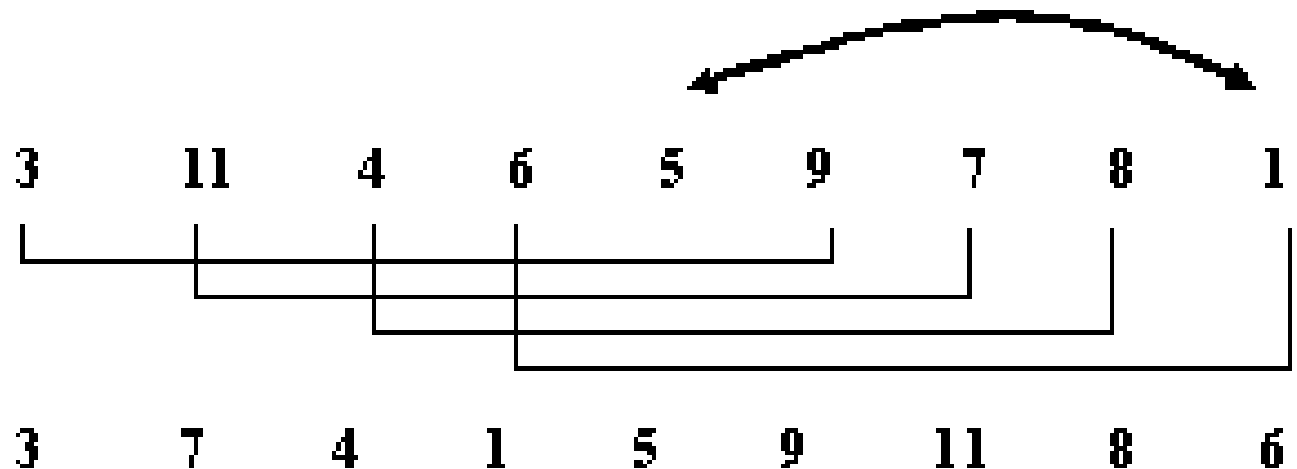
babak II = spasi 2

babak III = spasi 1

SORTING

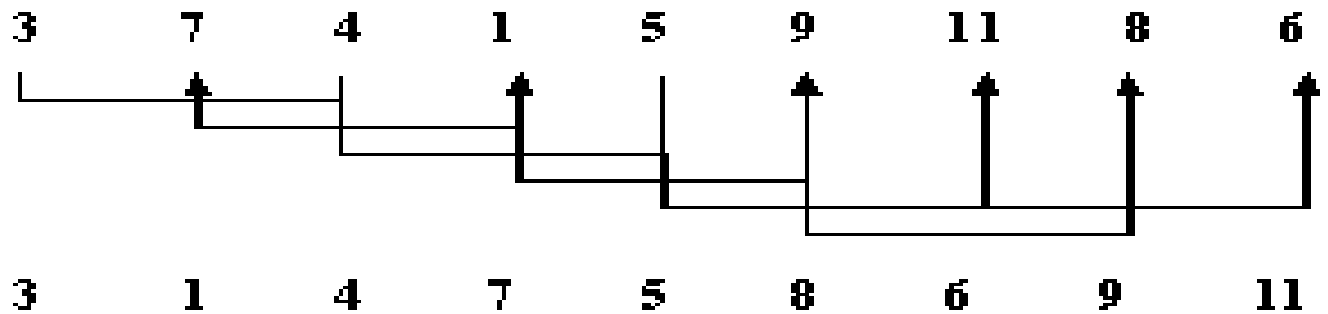
Metode Shell

babak I

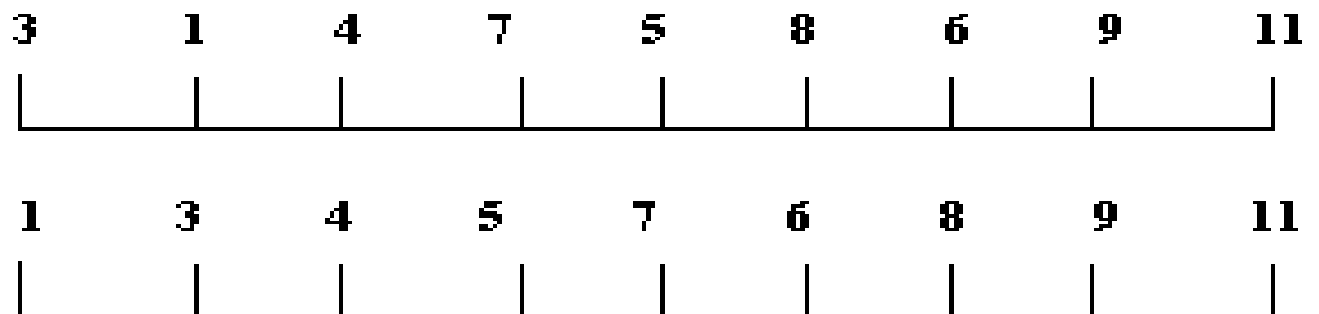


SORTING

babak II



babak III

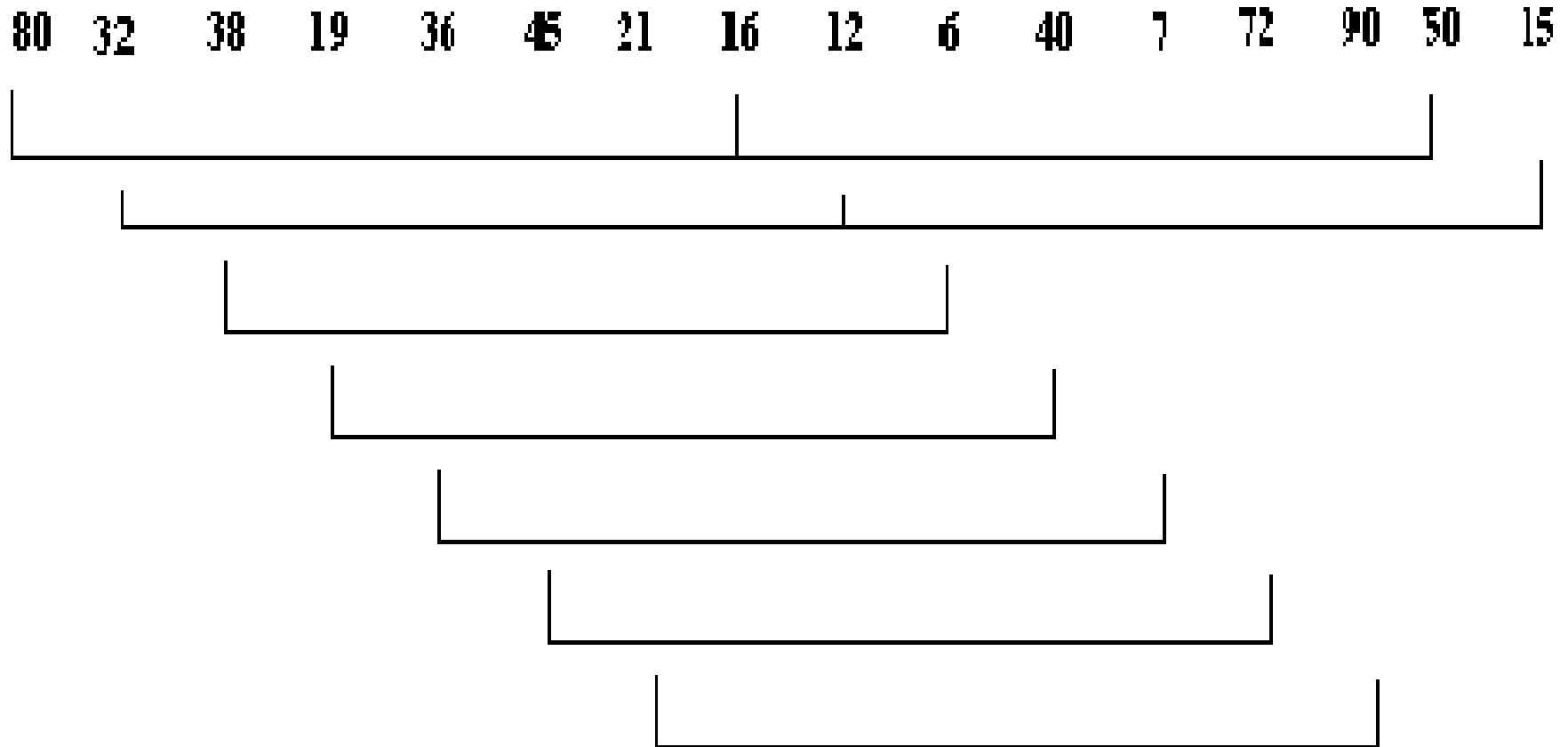


hasil

1, 3, 4, 5, 6, 7, 8, 9, 11

Gambar 4. Metode Shell

Tentukan shell-Sort dengan babak 1: spasi 7 , babak 2: spasi 3, babak 3 spasi : 1



Gambar 5. Shellsort

SORTING

Metode Shellsort

- output babak I, spasi $k = 7$:

3 12 6 19 7 45 21 50 15 38 40 36 72 90
 80 32

- output babak II, spasi $k = 3$:

5 7 6 19 12 15 21 40 36 38 50 45 32 90
 80 72

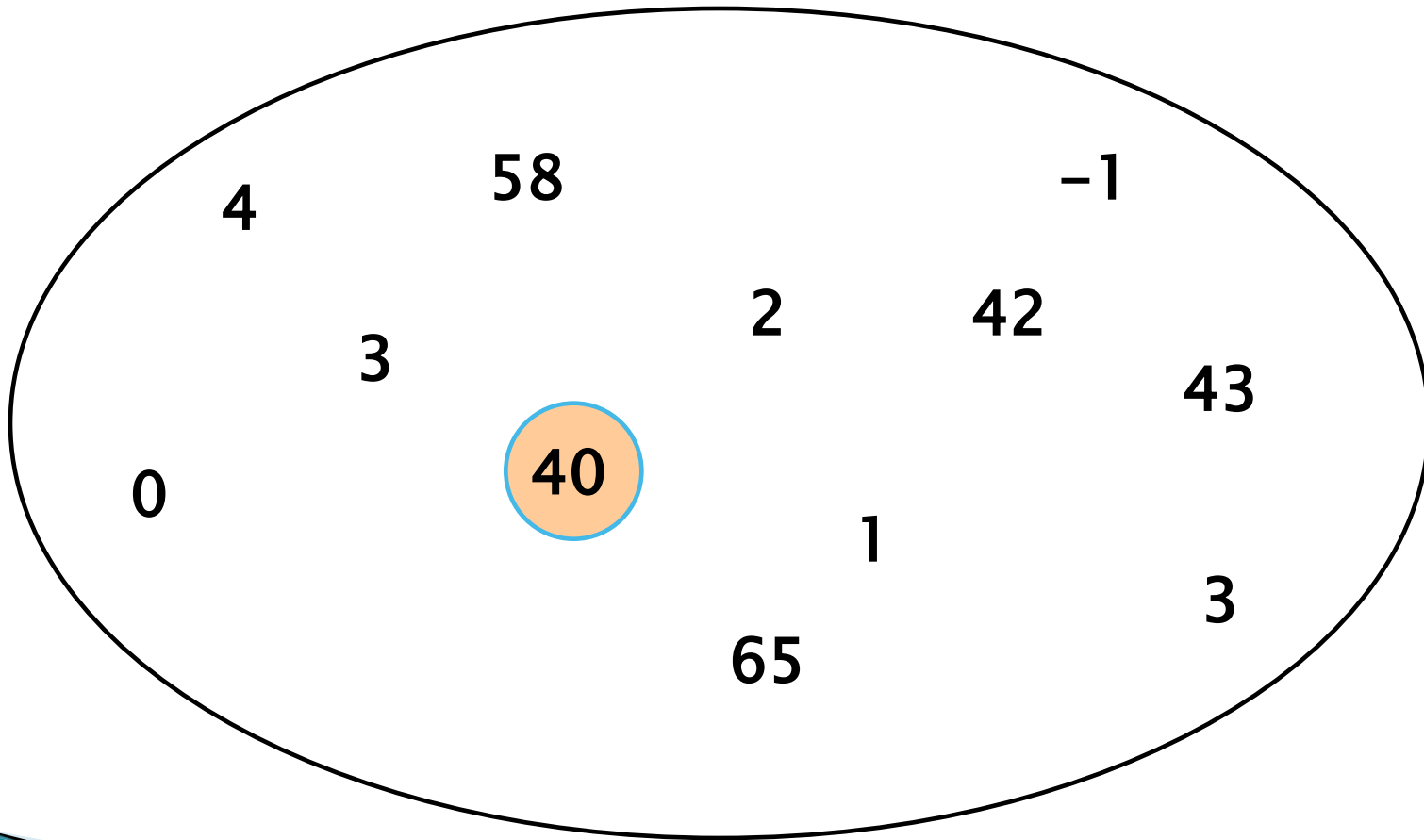
- output babak III, spasi $k = 1$:

6 7 12 15 16 19 21 32 36 38 40 45 50 72
 80 90

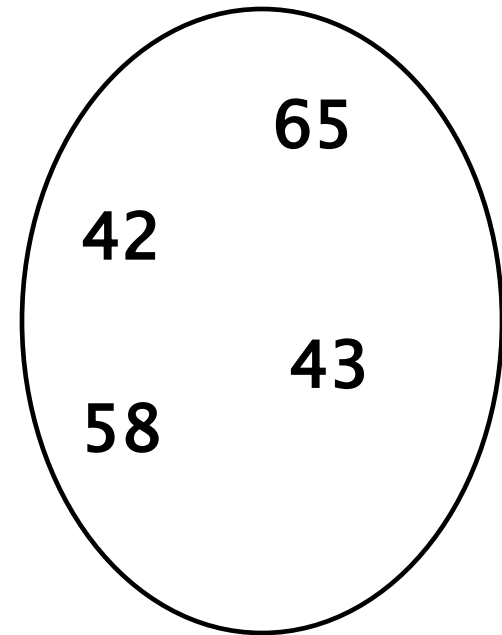
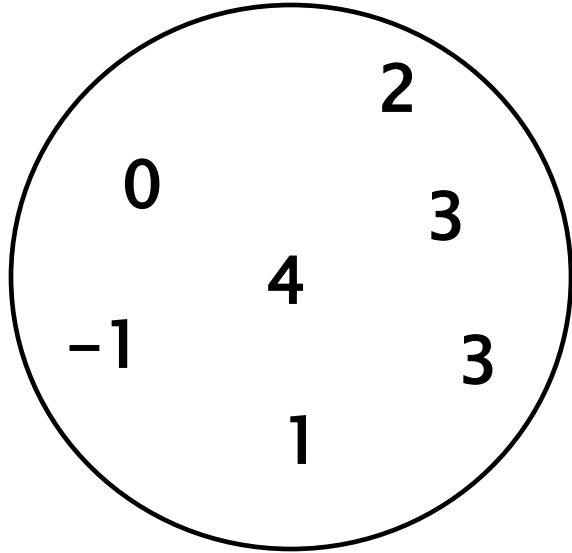
Quick sort: Ide dasar

- ▶ Divide and conquer approach
- ▶ Algoritma *quickSort(S)*:
 - Jika jumlah elemen dalam $S = 0$ atau 1 , return.
 - Pilih sembarang elemen $v \in S$ – sebutlah **pivot**.
 - Partisi $S - \{v\}$ ke dalam 2 bagian:
 - $L = \{x \in S - \{v\} \mid x \leq v\}$
 - $R = \{x \in S - \{v\} \mid x \geq v\}$
 - Kembalikan nilai *quickSort(S)*, diikuti v , diikuti *quickSort(S)*.

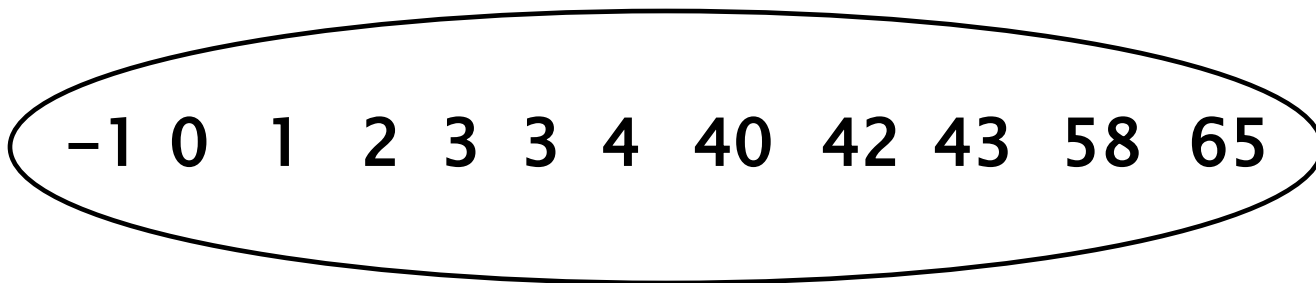
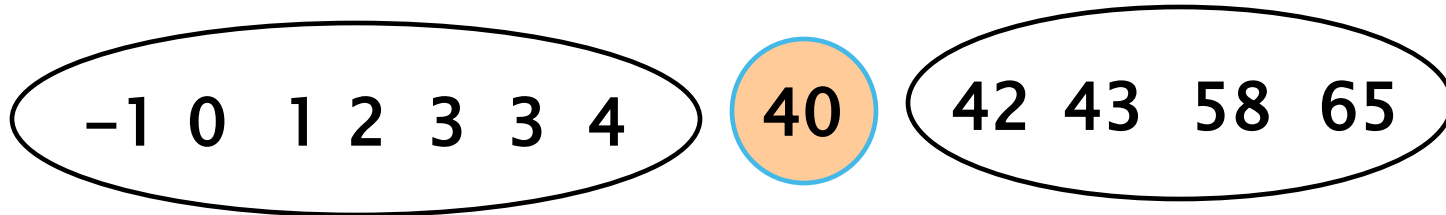
Quick sort: Pilih elemen *pivot*



Quick sort: Partition



Quick sort: Sort scr. rekursif, gabungkan



SORTING

Pohon Biner

Contoh:

Sebuah pohon biner harus dibuat dengan setiap simpulnya mengandung bilangan integer. Nilai-nilai pada setiap simpul harus dicetak secara berurutan, sehingga berurutan mulai dari kecil ke nilai yang makin besar.

Susunannya adalah sebagai berikut;

47, 94, 23, 87, 35, 71, 66, 98, 12, 16, 2, 46, 38

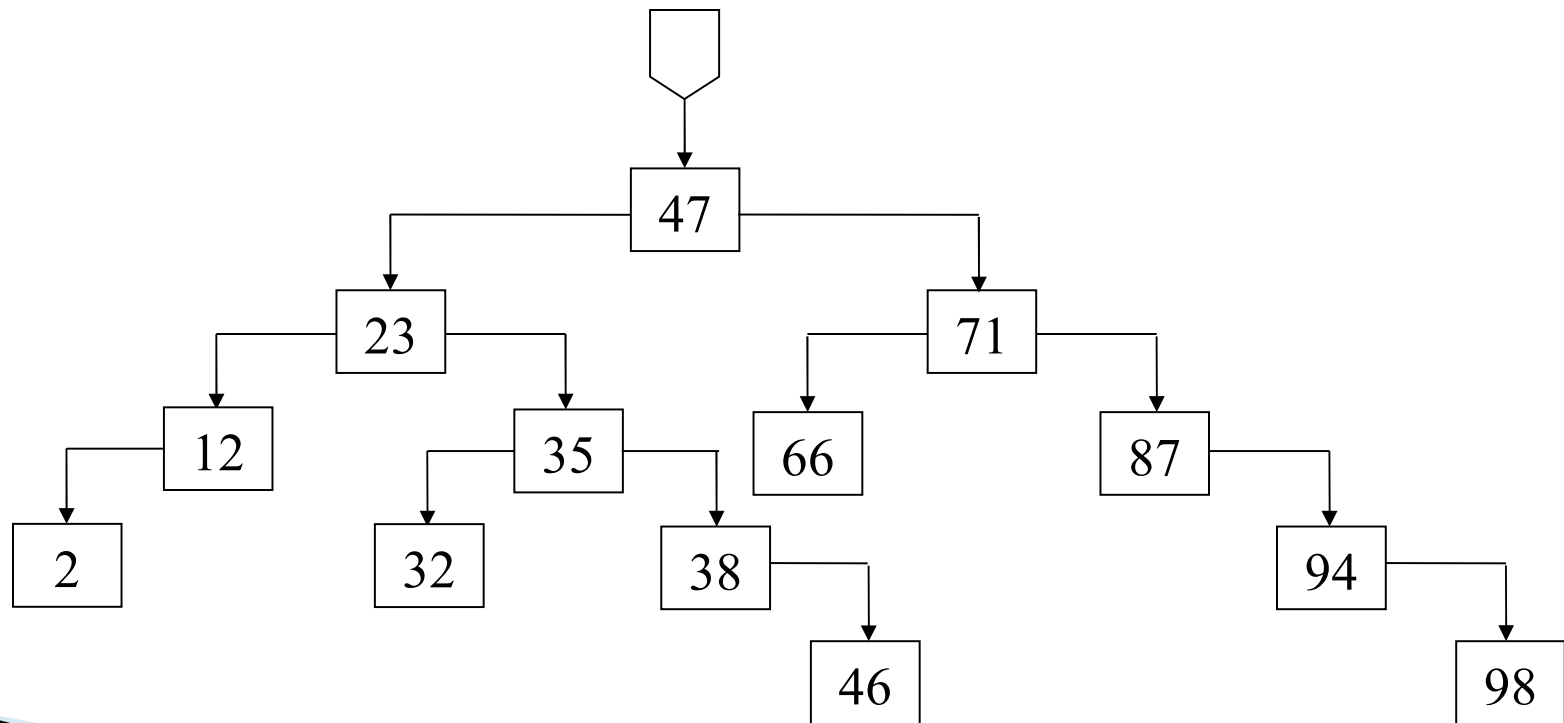
Sajikan programnya secara lengkap.

Penyelesaian;

SORTING

Pohon Biner

Penyusunan pohon biner tersebut akan menghasilkan nilai;
2, 12, 23, 32, 35, 38, 46, 47, 66, 71, 87, 94, 98



SORTING

Metode Radix

Umumnya digunakan untuk pengurutan kartu mekanis standard 80 kolom;

Contoh;

Bilangan pada tiap kartu;

42, 23, 65, 57, 94, 36, 99, 87, 70, 81, 61, 11, 74

pada urutan pertama, bilangan-bilangan dikelompokkan dalam saku tingkat digit satuan.

Anggota tiap saku:

Saku (0) = 70

Saku (1) = 81, 61, 11

Saku (2) = 42

Saku (3) = 23

Saku (4) = 94, 74

saku (5) = 65

saku (6) = 36

saku (7) = 57, 87

saku (8) =

saku (9) = 99

SORTING

Metode Radix

Saku-saku dikombinasikan dari saku (0) pada dasar sampai dengan saku (9) pada puncak:

70, 81, 61, 11, 42, 23, 94, 74, 65, 36, 57, 87, 99

pada urutan ke dua, deret di atas dikelompokkan dalam saku tingkat digit puluhan.

Anggota tiap saku:

Saku (0) =

Saku (1) = 11

Saku (2) = 23

Saku (3) = 36

Saku (4) = 42

saku (5) = 57

saku (6) = 61, 65

saku (7) = 70, 74

saku (8) = 81, 87

saku (9) = 99

Saku-saku dikombinasikan dari saku (0) pada dasar sampai dengan saku (9) pada puncak;

11, 23, 36, 42, 57, 61, 65, 70, 74, 81, 87, 99

SORTING

Metode Merge

Banyak digunakan untuk mengurutkan dua atau lebih tabel yang sudah terurutkan menjadi satu tabel yang terurutkan juga.

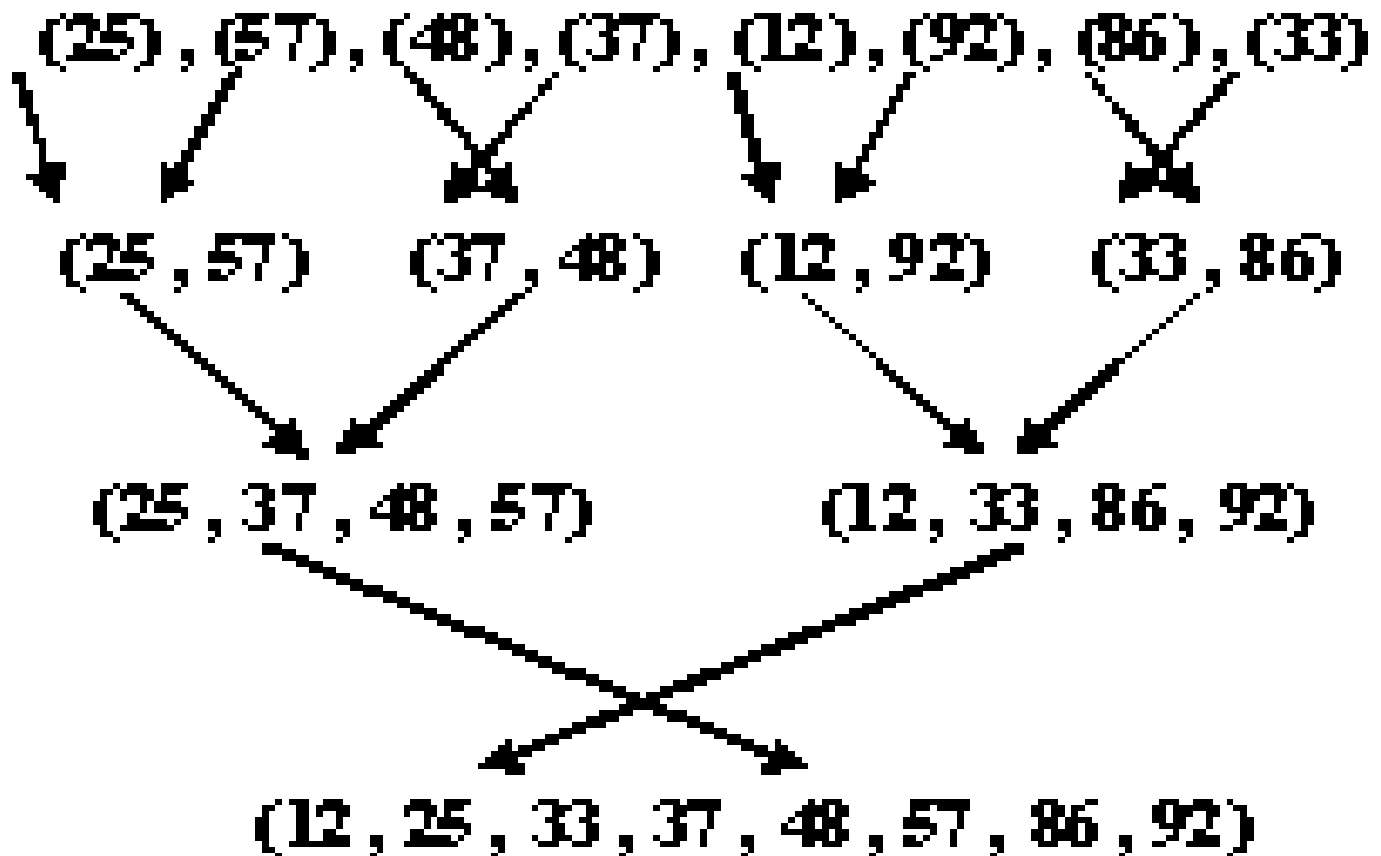
Tabel yang beranggotakan n dibagi menjadi n subtabel, kemudian digabungkan menjadi satu tabel. Contoh:

Tabel asli [25, 57, 48, 37, 12, 92, 86, 33]

SORTING

Metode Merge

Diubah menjadi 8 subtabel;



urutan 1
menjadi
4 tabel

urutan 2
menjadi
2 tabel

urutan 3
menjadi
1 tabel

SEARCHING

Proses pencarian adalah menemukan harga (data) tertentu di dalam sekumpulan harga yang bertipe sama (tipe dasar atau tipe bentukan).

Contoh:

Untuk menghapus (mengubah) harga tertentu di dalam kumpulannya, langkah pertama yang dilakukan adalah mencari apakah harga tersebut terdapat di dalam kumpulan yang dimaksud. Jika ada, harga tersebut dapat dihapus atau diubah nilainya. Dengan cara yang sama untuk penyisipan, jika data sudah ada, dan mempertahankan tidak ada duplikasi data, maka data tersebut tidak disisipkan, dan jika belum ada disisipkan.

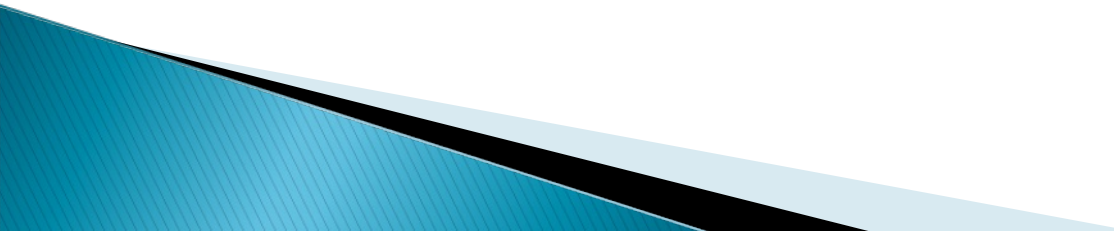
Pencarian terbagi Dua

1. Pencarian Internal

adalah pencarian terhadap sekumpulan data yang disimpan di dalam memori utama, struktur penyimpanan data yang umum adalah berupa larik atau tabel (*array*);

2. Pencarian Eksternal

adalah pencarian terhadap sekumpulan data yang disimpan di dalam memori sekunder seperti tape atau disk, struktur penyimpanan data berupa arsip (*file*).



Larik atau tabel

Larik merupakan tipe data terstruktur.

elemen-elemen larik disusun horizontal, sedangkan elemen-elemen larik yang disusun secara vertikal disebut tabel.

Contoh Algoritma pendeklarasian larik

DEKLARASI

D : array[1..11] of integer

Kar: array[1..8] of character

const n = 5 { *jumlah data mahasiswa* }

type Data = record <Nama: string, Usia:integer>

Mahasiswa : array[1..n] of Data

SEARCHING


Data searching (pencarian data) meliputi;

FETCH – pencarian lokasi posisi record
– pembacaan rekaman

NEXT – memperoleh rekaman berikutnya
– membaca seluruh record dalam berkas

Algoritma searching sangat erat hubungannya dengan sistem berkas (organisasi berkas).

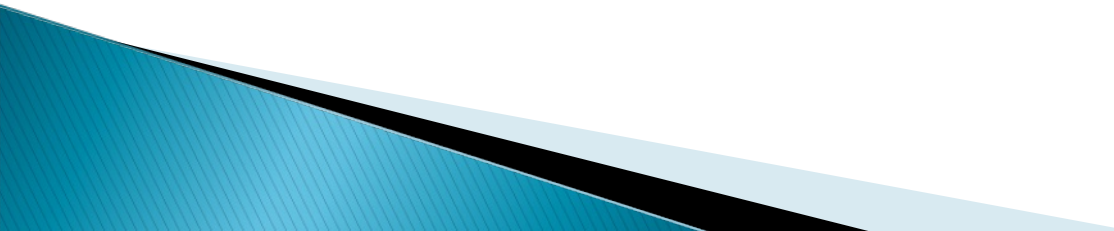
Macam metode pencarian:

1. Pencarian Beruntun (Sequential Search); algoritma pencarian yang paling sederhana
 2. Pencarian Beruntun dengan Sentinel;
 3. Pencarian Bagi dua (Binary Search), algoritma pencarian yang lebih maju.
- 

Pencarian Beruntun (Sequential Search)

Pencarian beruntun adalah proses membandingkan setiap elemen larik satu per satu secara beruntun, mulai dari elemen pertama sampai elemen yang dicari ditemukan atau seluruh elemen sudah diperiksa. Disebut juga dengan pencarian lurus (*linear search*)

Pencarian beruntun terbagi dua:

1. Pencarian beruntun pada larik tidak terurut;
 2. Pencarian beruntun pada larik terurut.
- 

Pencarian beruntun pada larik tidak terurut

Pencarian dilakukan dengan memeriksa setiap elemen larik mulai dari elemen pertama sampai elemen yang dicari ditemukan atau sampai seluruh elemen sudah diperiksa.

Contoh:

10	23	18	21	25	30
----	----	----	----	----	----

Misal nilai yang dicari adalah $x = 21$, maka elemen yang diperiksa : 10, 23, 18, 21 (ditemukan!)

Indeks larik yang dikembalikan: $idx = 4$

Misal nilai yang dicari adalah $x = 17$, maka elemen yang diperiksa : 10, 23, 18, 21, 25, 30 (tidak ditemukan!)

Indeks larik yang dikembalikan: $idx = 0$.

Misal anda diminta membuat algoritma dan program dari beberapa data yang telah diketahui dengan menggunakan metode pencarian sekuensial.

Data-data tersebut adalah sebagai berikut: 25, 36, 2, 48, 0, 69, 14, 22, 7, 19. Data yang akan dicari diinputkan.

Algoritma dari permasalahan di atas adalah:

1. Tentukan dan simpan data di dalam suatu larik;
 2. Tentukan fungsi pencarian sekuensial;
- 

Algoritma Pencarian Beruntun pada larik yang tidak terurut

```
procedure SeqSearch1(input L : LarikInt, input n : integer,  
                    input x : integer, output ketemu: boolean)  
  
{ Mencari keberadaan nilai x di dalam larik L[1..n]. }  
{ K.Awal: x dan larik L[1..n] sudah terdefinisi nilainya. }  
{ K.Akhir: ketemu bernilai true jika x ditemukan. Jika x tidak ditemukan,  
  ketemu bernilai false. }
```

DEKLARASI

i : integer { pencatat indeks larik }

ALGORITMA:

```
i ← 1  
while (i < n ) and (L[i] ≠ x) do  
    i ← i + 1  
endwhile  
{ i = n or L[i] = x }  
  
if L[i] = x then           { x ditemukan }  
    ketemu ← true  
else  
    ketemu ← false       { x tidak ada di dalam larik L }  
endif
```

Pencarian Beruntun pada Larik yang Terurut

Jika larik sudah terurut (misal terurut menaik, yaitu untuk setiap $I=1..N$, Nilai $[I-1] < \text{Nilai}[I]$ atau terurut mengecil, yaitu untuk setiap $I=1..N$, Nilai $[I-1] > \text{Nilai}[I]$), maka proses pencarian lebih singkat dibandingkan pencarian larik yang tidak terurut.

Larik yang elemen-elemennya sudah terurut dapat meningkatkan kinerja algoritma pencarian beruntun. Jika pada larik tidak terurut jumlah perbandingan elemen larik maksimum n kali, maka pada larik terurut (dengan asumsi distribusi elemen-elemen larik adalah seragam) hanya dibutuhkan rata-rata $n/2$ kali perbandingan.

Contoh Pencarian pada larik terurut

Diberikan larik L tidak terurut :

13	16	14	21	76	15
1	2	3	4	5	6

untuk mencari 15, dibutuhkan perbandingan sebanyak 6 kali

Misalkan larik L di atas sudah diurut menaik :

13	14	15	16	21	76
1	2	3	4	5	6

maka, untuk mencari 15, dibutuhkan perbandingan hanya 3 kali (secara rata-rata).



Algoritma Pencarian beruntun pada larik terurut

procedure SeqSearch (input L : LarikInt, input n : integer,
 input x : integer, output idx : integer)
*{ Mencari keberadaan nilai X di dalam larik L[1..n] yang elemen-elemennya
 sudah terurut menaik. }*
*{ K.Awal: nilai x dan elemen-elemen larik L[1..n] sudah terdefinisi.
 Elemen-elemen larik L sudah terurut menaik. }*
*{ K.Akhir: idx berisi indeks larik L yang berisi nilai x. Jika x tidak ditemukan, maka
 idx diisi dengan nilai -1. }*

```
DEKLARASI
  i : integer           { pencatat indeks larik }

ALGORITMA:
  i ← 1
  while (i < n ) and (L[i] < x) do
    i ← i + 1
  endwhile
  { i = N or L[i] = x }

  if L[i] = x then      { x ditemukan }
    idx ← i
  else
    idx ← -1
  endif
```

Pencarian Beruntun dengan Sentinel

Yang dimaksud dengan sentinel adalah elemen fiktif yang sengaja ditambahkan sesudah elemen terakhir larik. Jika elemen larik terakhir $L[N]$, maka sentinel dipasang pada elemen $L[N+1]$.

Sentinel ini harganya sama dengan elemen yang dicari. Akibatnya proses pencarian selalu berhasil menemukan data yang dicari. Walaupun demikian harus diperiksa lagi letak data tersebut ditemukan, apakah:

1. Di antara elemen–elemen larik sesungguhnya, yaitu $L[1] \dots L[N]$
2. Pada elemen fiktif ($L[N+1]$) berarti X sesungguhnya tidak terdapat di dalam larik L.

Pencarian Beruntun dengan Sentinel

$x = 18$

13	16	14	21	76	15	18	← sentinel
1	2	3	4	5	$n = 6$	7	

18 ditemukan pada elemen ke- $n+1$. Sentinel otomatis sudah ditambahkan ke dalam larik. Ukuran larik sekarang = 7.

$x = 21$

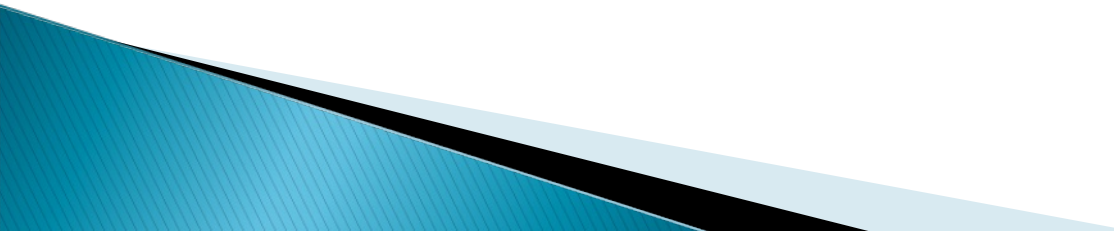
13	16	14	21	76	15	21	← sentinel
1	2	3	4	5	$N = 6$	7	

21 ditemukan pada elemen ke-4. Sentinel batal menjadi elemen yang ditambahkan ke dalam larik. Ukuran larik tetap 6.

Algoritma pencarian beruntun dengan Sentinel

```
procedure SeqSearchWithSentinel(input L : LarikInt, input n : integer,  
                                input x : integer, output idx : integer)  
  
  { Mencari x di dalam larik L[1..n] dengan menggunakan sentinel }  
  { K.Awal: x dan elemen-elemen larik L[1..N] sudah terdefinisi nilainya. }  
  { K.Akhir: idx berisi indeks larik L yang berisi nilai x. }  
  { Jika x tidak ditemukan, maka idx diisi dengan nilai -1. }  
  
  DEKLARASI  
    i : integer    { pencatat indeks larik }  
  
  ALGORITMA:  
    L[n + 1] ← x    { sentinel }  
    i ← 1  
    while (L[i] ≠ x) do  
      i ← i + 1  
    endwhile  
    { L[i] = x }    { Pencarian selalu berhasil menemukan x }  
  
    { Kita harus menyimpulkan apakah x ditemukan pada elemen  
      sentinel atau bukan }  
  
    if idx = n + 1 then { x ditemukan pada elemen sentinel }  
      idx ← -1    { berarti x belum ada pada larik L semula }  
    else    { x ditemukan pada indeks < n + 1 }  
      idx ← i  
    endif
```

Pencarian bagi dua

- ❖ Merupakan metode pencarian pada data terurut yang paling mangkus (*efficient*)
 - ❖ Pencarian bagi dua atau pencarian biner adalah metode pencarian yang diterapkan pada sekumpulan data yang sudah terurut (terurut menaik atau terurut menurun).
 - ❖ Data yang terurut syarat mutlak penerapan algoritma ini.
 - ❖ Salah satu keuntungan data terurut adalah memudahkan pencarian dalam hal ini pencarian bagi dua.
- 

Langkah 1: Bagi dua elemen larik pada elemen tengah.

Elemen tengah adalah elemen dengan indeks $k = (i + j) \text{ div } 2$.
(Elemen tengah, $L[k]$, membagi larik menjadi dua bagian, yaitu bagian kiri $L[i..j]$ dan bagian kanan $L[k+1..j]$)

Langkah 2: Periksa apakah $L[k] = x$.

Jika $L[k] = x$, pencarian selesai sebab x sudah ditemukan.

Tetapi, jika $L[k] \neq x$, harus ditentukan apakah pencarian akan dilakukan di larik bagian kiri atau di bagian kanan.

Jika $L[k] < x$, maka pencarian dilakukan lagi pada larik bagian kiri.

Sebaliknya, jika $L[k] > x$, pencarian dilakukan lagi pada larik bagian kanan.

Langkah 3: Ulangi Langkah 1 hingga x ditemukan atau $i > j$ (yaitu, ukuran larik sudah nol!)

Contoh Pencarian elemen dengan metode bagidua

Misalkan diberikan larik L dengan delapan buah elemen yang sudah terurut menurun seperti di bawah ini:

81	76	21	18	16	13	10	7
$i = 1$	2	3	4	5	6	7	$8 = j$

Misalkan elemen yang dicari adalah $x = 18$.

Langkah 1:

$i = 1$ dan $j = 8$

Indeks elemen tengah $k = (1 + 8) \text{ div } 2 = 4$ (elemen yang diarsir)

81	76	21	18	16	13	10	7
1	2	3	4	5	6	7	8

kiri

kanan

Langkah 2:

Pembandingan: $L[4] = 18$? Ya! (x ditemukan, proses pencarian selesai)