

# **SORTING**

**TIB21**

# **TARGET:**

**Algoritma berdasarkan Priority Queue → Selection Sort & Heap Sort**

**Algoritma penyisipan dalam keterurutan → Insertion Sort & Tree Sort**

**Algoritma transposisi → Bubble Sort**

**Algoritma increment → Shell Sort**

**Algoritma dengan Divide & Conquer → Quick Sort & Merge Sort**

**Algoritma-algoritma penghitungan alamat → Radix Sort & Proximity Map Sort**

# **SORTING**

**Pengurutan data dalam struktur data baik data numerik ataupun karakter.**

**Metode:**

- ascending (urut naik)
- descending (urut turun)

**c/**

**Data Acak : 5 6 8 1 3 25 10**

**Ascending : 1 3 5 6 8 10 25 ; Amir Budi Badu Dudi**

**Descending : 25 10 8 6 5 3 1**

# **SORTING-1**

## **Bentuk:**

- Ascending → if N obyek disimpan dalam larik L, then menyusun elemen larik  
$$L[1] \leq L[2] \leq L[3] \leq \dots \leq L[N]$$
- Descending → if N obyek disimpan dalam larik L, then menyusun elemen larik  
$$L[1] \geq L[2] \geq L[3] \geq \dots \geq L[N]$$

# **(BUBBLE SORT/PENGURUTAN GELEMBUNG)**

**Metode sorting termudah**

**Bubble Sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya.**

**Ascending :**

**if elemen sekarang  $>$  dari elemen  
berikutnya then kedua elemen ditukar**

**Descending:**

**if elemen sekarang  $<$  dari elemen  
berikutnya then kedua elemen ditukar**

# BUBBLE SORT (2)

- Algoritma:

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 :

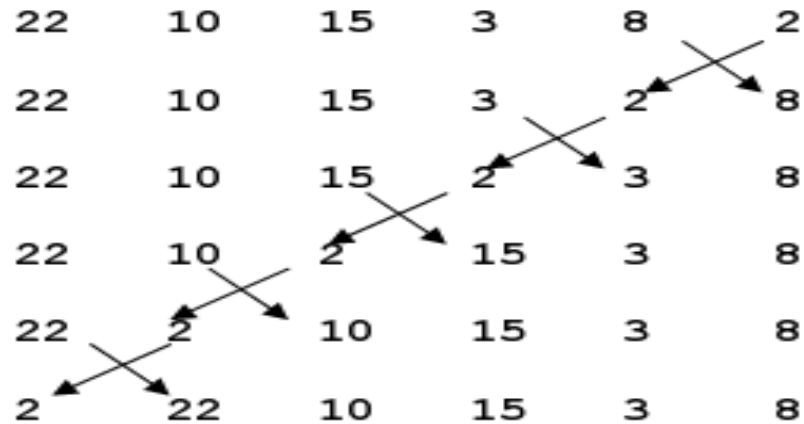
Periksalah nilai dua elemen mulai dari urutan ke-**n** sampai urutan ke-**2**. Jika nilai kiri < kanan, tukarkan kedua data itu.

step n-1 :

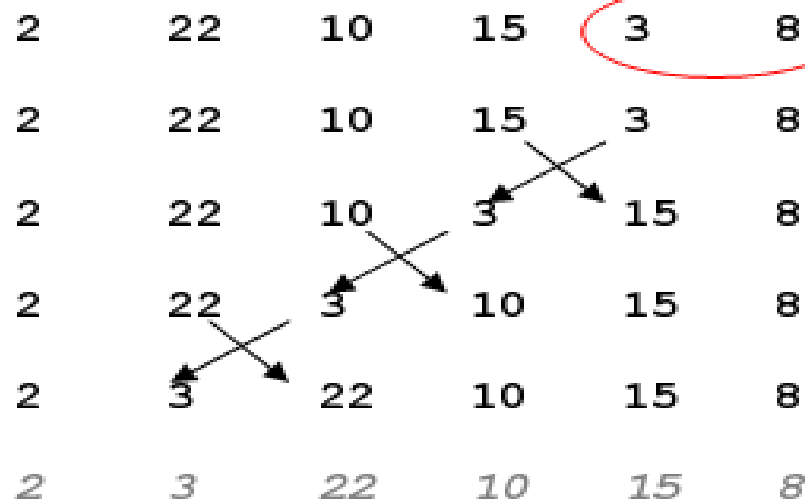
Periksalah nilai dua elemen mulai dari urutan ke-**n** sampai urutan ke-**n-1**. Jika nilai kiri < kanan, tukarkan kedua data itu.

# BUBBLE SORT (3)

## Proses 1



## Proses 2



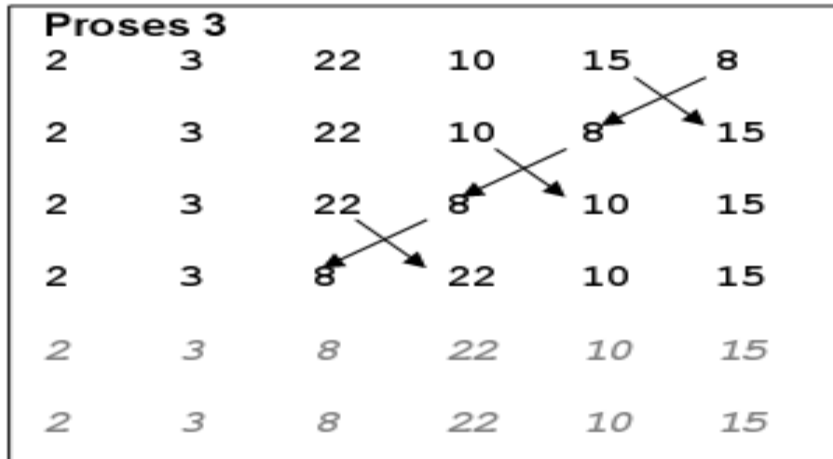
Tidak ada penukaran,  
karena  $3 < 8$

Pegurutan berhenti di sini!

# BUBBLE SORT (4)

**Proses 3**

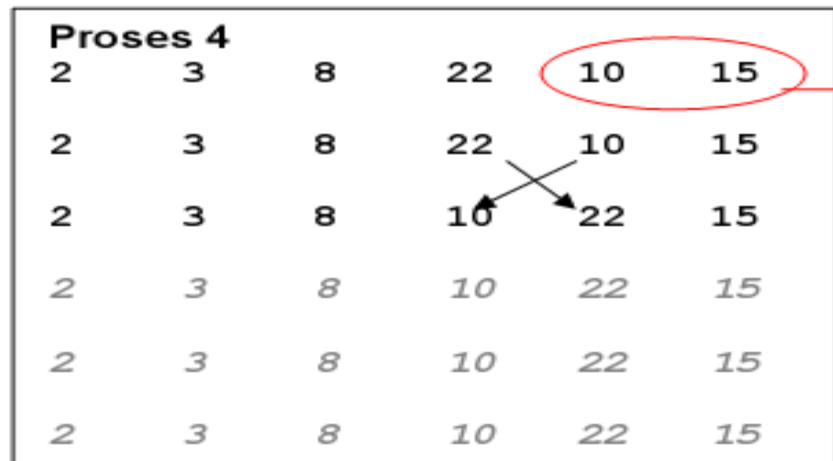
2	3	22	10	15	8
2	3	22	10	8	15
2	3	22	8	10	15
2	3	8	22	10	15
2	3	8	22	10	15
2	3	8	22	10	15



→ Pegurutan berhenti di sini!

**Proses 4**

2	3	8	22	10	15
2	3	8	22	10	15
2	3	8	10	22	15
2	3	8	10	22	15
2	3	8	10	22	15
2	3	8	10	22	15



Tidak ada penukaran, karena  $10 < 15$

→ Pegurutan berhenti di sini!



# BUBBLE SORT (5)

Proses 5					
2	3	8	10	22	15
2	3	8	10	15	22
<i>2</i>	<i>3</i>	<i>8</i>	<i>10</i>	<i>15</i>	<i>22</i>
<i>2</i>	<i>3</i>	<i>8</i>	<i>10</i>	<i>15</i>	<i>22</i>
<i>2</i>	<i>3</i>	<i>8</i>	<i>10</i>	<i>15</i>	<i>22</i>
<i>2</i>	<i>3</i>	<i>8</i>	<i>10</i>	<i>15</i>	<i>22</i>

→ Pegurutan berhenti di sini!

# ALGORITMA

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.

# STUDI KASUS BUBBLE SORT

Data 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



# BUBBLE SORT (6)

## Versi 1

```
void bubble_sort(int data[]){  
    for(int i=1;i<n;i++){  
        for(int j=n-1;j>=i;j--){  
            if(data[j]<data[j-1]) tukar(&data[j],&data[j-1]); //ascending  
        }  
    }  
}
```

if (data[j]<data[j-1])  
tukar(&data[j],&data[j-1]);  
  
if (data[j]>data[j-1])  
tukar(&data[j],&data[j-1]);

## Versi 2

```
void bubblesort2 (int data[]){  
    for(i=1;i<6;i++){  
        for(int j=0;j<6-i;j++){  
            if (data[j]>data[j+1])  
                tukar (&data[j] , &data[j+1]) ; //descending  
        }  
    }  
}
```

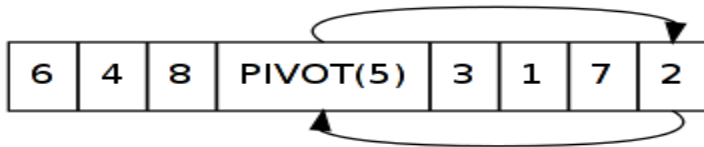
# ***QUICK SORT***

## **d/ pola algoritma divide-and-conquer:**

- **Divide**
  - Memilah rangkaian data menjadi dua sub-rangkaian  $A[p \dots q-1]$  dan  $A[q+1 \dots r]$
  - $A[p \dots q-1] \rightarrow < || == A[q]$
  - $A[q+1 \dots r] \rightarrow > || == A[q]$
  - $A[q]$  = elemen pivot
- **Conquer**
  - Mengurutkan elemen pada sub-rangkaian secara rekursif

6	4	8	5	3	1	7	2
---	---	---	---	---	---	---	---

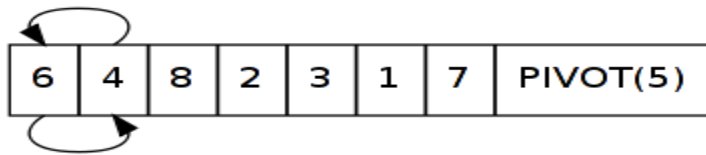
Saya pilih 5 sebagai pivot (sembarang elemen boleh menjadi pivot, saya pilih yang kira-kira di tengah). Tukarkan pivot ini dengan elemen terakhir.



Hasilnya seperti ini: 6\*, 4, 8, 2, 3, 1, 7, 5 (pivot). Pada elemen pertama saya beri tanda bintang (\*), akan saya jelaskan nanti gunanya.

*6	4	8	2	3	1	7	PIVOT(5)
----	---	---	---	---	---	---	----------

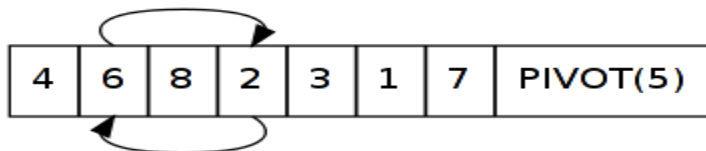
Sekarang kita akan mulai memproses list dari elemen pertama sampai elemen sebelum pivot. Setiap kali menemukan elemen yang kurang dari pivot, saya pindahkan (kita tukar) dengan elemen yang diberi tanda bintang. Lalu bintang dipindahkan satu elemen ke kanan. Elemen pertama (6) lebih besar dari pivot (5), jadi kita cek elemen berikutnya yaitu 4. Karena 4 kurang dari pivot, kita tukarkan 4 dengan elemen yang bertanda \*. Hasilnya:



Hasilnya: 4, \*6, 8, 2, 3, 1, 7, 5 (pivot). Ingat bahwa setelah menukar, tanda \* dipindah satu elemen ke kanan.

4	*6	8	2	3	1	7	PIVOT(5)
---	----	---	---	---	---	---	----------

Berikutnya kita lihat bahwa 8 lebih dari pivot, jadi kita biarkan. Elemen 2 kurang dari pivot, sehingga perlu ditukar dengan \*.

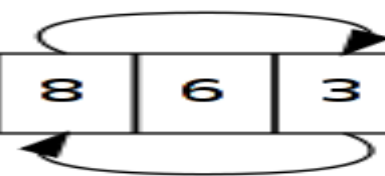


Dan hasilnya adalah: 4, 2, \*8, 6, 3, 1, 7, 5 (pivot).

4	2	*8	6	3	1	7	PIVOT(5)
---	---	----	---	---	---	---	----------

Elemen 3 juga kurang dari pivot, jadi kita perlu menukarnya.

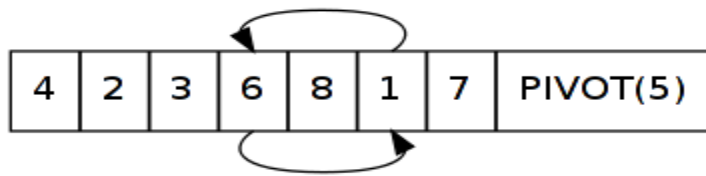
4	2	8	6	3	1	7	PIVOT(5)
---	---	---	---	---	---	---	----------



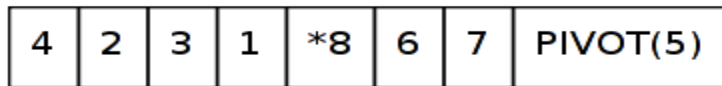
Hasilnya adalah: 4, 2, 3, \*6, 8, 1, 7, 5 (pivot).

4	2	3	*6	8	1	7	PIVOT(5)
---	---	---	----	---	---	---	----------

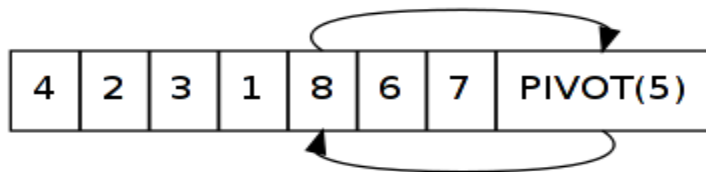
Dan yang terakhir yang kurang dari pivot adalah 1.



Hasilnya adalah: 4, 2, 3, 1, \*8, 6, 7, 5 (pivot).



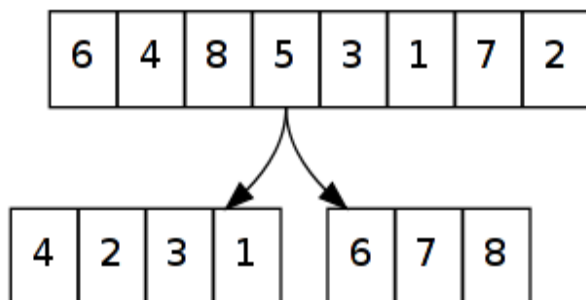
Dan langkah terakhir adalah menukar posisi \* dengan pivot:



Hasil akhirnya adalah list yang terbagi dua (semua elemen di kiri 5 lebih kecil dari 5 dan semua elemen di kanan 5 lebih besar dari 5): 4, 2, 3, 1, 5, 6, 7, 8:



Atau jika digambarkan, yang saya lakukan adalah seperti ini: memecah list awal, menjadi dua list, yang satu berisi elemen-elemen yang lebih kecil dari 5, dan list yang berisi elemen-elemen yang lebih besar atau sama dengan 5.



# QUICK SORT

Rangkaian data:

3	1	4	1	5	9	2	6	5	3	5	8
---	---	---	---	---	---	---	---	---	---	---	---

Pilih sebuah elemen yang akan menjadi elemen pivot.

<b>3</b>	1	4	1	5	9	2	6	5	3	5	8
----------	---	---	---	---	---	---	---	---	---	---	---

Inisialisasi elemen kiri sebagai elemen kedua dan elemen kanan sebagai elemen akhir.

	kiri										kanan
<b>3</b>	1	4	1	5	9	2	6	5	3	5	8

Geser elemen kiri ke arah kanan sampai ditemukan nilai yang lebih besar dari elemen pivot tersebut. Geser elemen kanan ke arah kiri sampai ditemukan nilai dari elemen yang tidak lebih besar dari elemen tersebut.

		kiri							kanan		
<b>3</b>	1	<u>4</u>	1	5	9	2	6	5	<u>3</u>	5	8

Tukarkan antara elemen kiri dan kanan

kiri

kanan

3	1	3	1	5	9	2	6	5	4	5	8
---	---	---	---	---	---	---	---	---	---	---	---

Geserkan lagi elemen kiri dan kanan.

				kiri		kanan					
3	1	3	1	5	9	2	6	5	4	5	8

Tukarkan antar elemen kembali.

				kiri		kanan					
3	1	3	1	2	9	5	6	5	4	5	8

Geserkan kembali elemen kiri dan kanan.

				kanan	kiri						
3	1	3	1	2	9	5	6	5	4	5	8

Terlihat bahwa titik kanan dan kiri telah digeser sehingga mendapatkan nilai elemen kanan < elemen kiri. Dalam hal ini tukarkan elemen pivot dengan elemen kanan.

				pivot							
2	1	3	1	3	9	5	6	5	4	5	8

# ***QUICK SORT → NEXT APLIKASI***

```
void quickSort(Object array[], int leftIdx, int rightIdx) {  
    int pivotIdx;  
    if (rightIdx > leftIdx) {  
        pivotIdx = partition(array, leftIdx, rightIdx);  
        quickSort(array, leftIdx, pivotIdx-1);  
        quickSort(array, pivotIdx+1, rightIdx);  
    }  
}
```



# ***MERGE SORT***

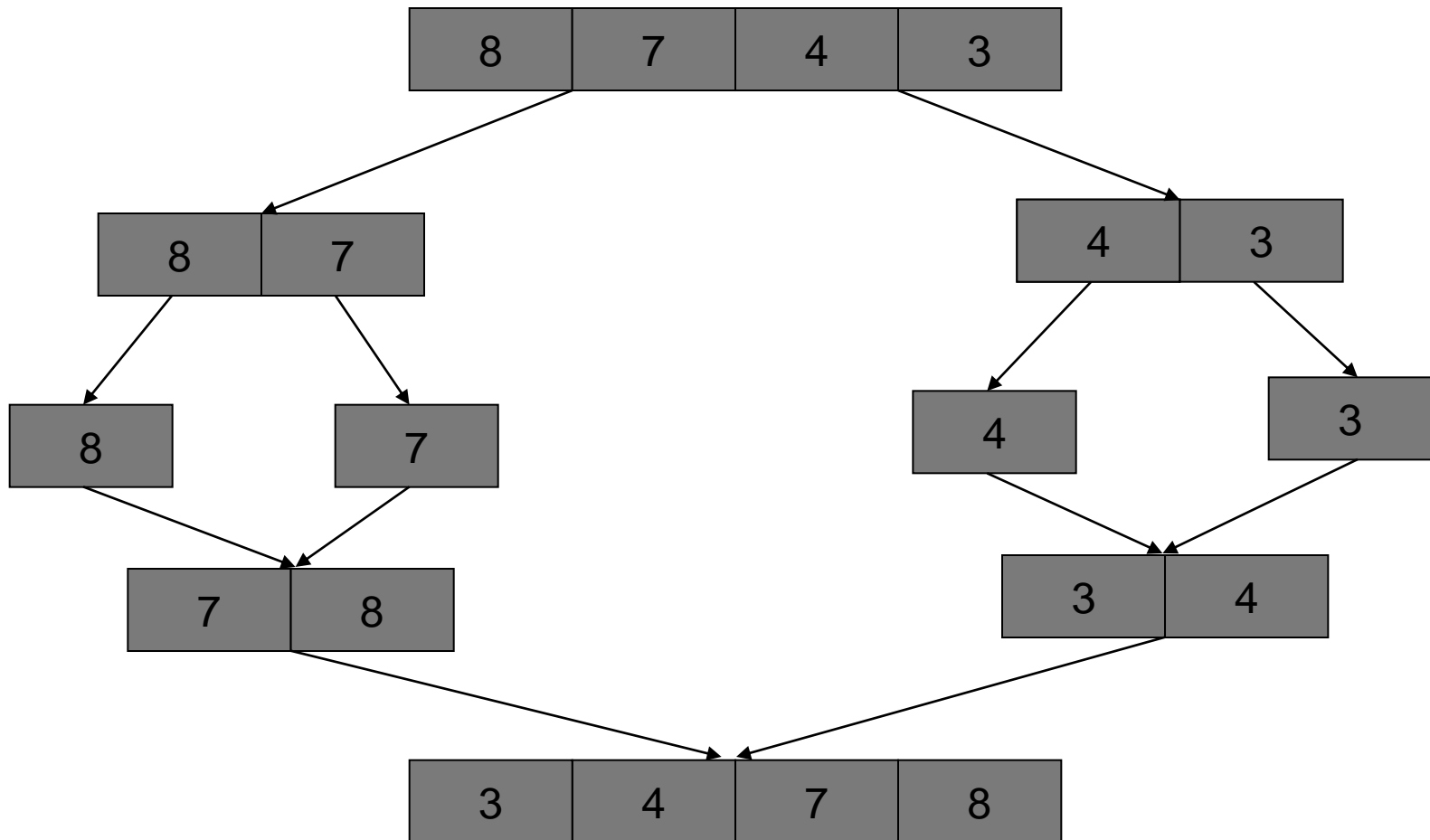
**d/ pola algoritma *divide and conquer*, langkah”:**

- **Divide**  
Memilah elemen” dari rangkaian data menjadi 2 bagian
- **Conquer**  
Selesaikan sub masalah tersebut secara rekursif d/ memanggil prosedur *merge sort* secara rekursif
- **Kombinasi**  
Mengkombinasikan dua bagian tersebut secara rekursif untuk mendapatkan rangkaian data berurutan

# MERGE SORT

Algoritma:

- Membagi data menjadi dua bagian (*LeftArr* ; *RightArr*)
- Membagi *LeftArr* menjadi dua bagian (*LeftArr* ; *RightArr*)
- Membagi *RightArr* menjadi dua bagian (*LeftArr* ; *RightArr*)
- Mengkombinasikan *LeftArr* dan *RightArr*.



# ***MERGE SORT***

```
void mergeSort(Object array[], int startIdx, int endIdx) {  
    if (array.length != 1) {  
        //Membagi rangkaian data, rightArr dan leftArr  
        mergeSort(leftArr, startIdx, midIdx);  
        mergeSort(rightArr, midIdx+1, endIdx);  
        combine(leftArr, rightArr);  
    }  
}
```

## **2. INSERTION SORT**

Algoritma *insertion sort* pada dasarnya memilah data yang akan diurutkan menjadi dua bagian, yang belum diurutkan (meja pertama) dan yang sudah diurutkan (meja kedua)

Elemen pertama diambil dari bagian array yang belum diurutkan dan kemudian diletakkan sesuai posisinya pada bagian lain dari array yang telah diurutkan. Langkah ini dilakukan secara berulang hingga tidak ada lagi elemen yang tersisa pada bagian array yang belum diurutkan.

c/ mengurutkan kartu dari kecil s/d besar

- Seluruh kartu meja 1, disusun dari kiri ke kanan dan atas ke bawah
- Meja 2 (tempat kartu yang diurutkan akan diletakkan)
- Ambil kartu ke-1 yang terletak pada pojok kiri atas meja 1 dan letakkan pada meja 2
- Ambil kartu ke-2 dari meja 1, bandingkan dengan kartu yang berada pada meja ke-2, kemudian letakkan pada urutan yang sesuai setelah perbandingan
- Proses akan berlangsung hingga seluruh kartu pada meja pertama telah diletakkan berurutan pada meja kedua

# INSERTIONSHORT

Putaran 1

Temp	[0]	[1]	[2]	[3]	[4]
<input type="text"/>	12	52	69	40	48

Putaran 2

Temp	[0]	[1]	[2]	[3]	[4]
<input type="text"/>	12	52	69	40	48

Putaran 3

Temp	[0]	[1]	[2]	[3]	[4]
<input type="text"/>	12	40	52	69	48

Putaran 4

Temp	[0]	[1]	[2]	[3]	[4]
<input type="text"/>	12	40	48	52	69

Result

12	40	48	52	69
----	----	----	----	----

# ***INSERTION SORT***

<b><i>Data</i></b>		<b><i>1<sup>st</sup> Pass</i></b>		<b><i>2<sup>nd</sup> Pass</i></b>		<b><i>3<sup>rd</sup> Pass</i></b>		<b><i>4<sup>th</sup> Pass</i></b>
Mango		Mango		Apple		Apple		Apple
Apple		Apple		Mango		Mango		Banana
Peach		Peach		Peach		Orange		Mango
Orange		Orange		Orange		Peach		Orange
Banana		Banana		Banana		Banana		Peach

# InsertionSort

- Ascending:

```
public static void Insertion(int[] data) {  
    int i, j, temp;  
    for(i=1; i<data.length; i++) {  
        temp = data[i];  
        for(j=i-1; (j>=0) && (data[j]>temp); j--)  
            data[j+1]=data[j];  
        data[j+1] = temp;  
    }  
}
```

- Descending:

```
public static void Insertion(int[] data) {  
    int i, j, temp;  
    for(i=1; i<data.length; i++) {  
        temp = data[i];  
        for(j=i-1; (j>=0) && (data[j]<temp); j--)  
            data[j+1]=data[j];  
        data[j+1] = temp;  
    }  
}
```

# ***INSERTION SORT***

```
void insertionSort(Object array[], int startIdx, int endIdx) {  
    for (int i = startIdx; i < endIdx; i++) {  
        int k = i;  
        for (int j = i + 1; j < endIdx; j++) {  
            if (((Comparable) array[k]).compareTo(array[j])>0) {  
                k = j;  
            }  
        }  
        swap(array[i],array[k]);  
    }  
}
```



## Mrk → **SELECTION SORT**

- Memilih elemen dengan nilai paling rendah dan menukar elemen yang terpilih dengan elemen ke- $i$
- Nilai dari  $i$  dimulai dari 1 ke  $n$ ,  $n \rightarrow$  jumlah total elemen dikurangi 1

**c/**

- Asumsikan bahwa kartu diurutkan secara *ascending*
- Kartu akan disusun secara linier pada sebuah meja dari kiri ke kanan, dan dari atas ke bawah

### **Algoritma:**

- Pilih nilai kartu yang paling rendah, tukarkan posisi kartu ini dengan kartu yang terletak pada pojok kiri atas meja
- Cari kartu dengan nilai paling rendah diantara sisa kartu yang tersedia
- Tukarkan kartu yang baru saja terpilih dengan kartu pada posisi kedua
- Ulangi proses tersebut hingga posisi kedua sebelum posisi terakhir dibandingkan dan dapat digeser dengan kartu yang bernilai lebih rendah

# SELECTION SORT

<b>Data</b>		<b>1<sup>st</sup> Pass</b>		<b>2<sup>nd</sup> Pass</b>		<b>3<sup>rd</sup> Pass</b>		<b>4<sup>th</sup> Pass</b>
Maricar		Hannah		Hannah		Hannah		Hannah
Vanessa		Vanessa		Margaux		Margaux		Margaux
Margaux		Margaux		Vanessa		Maricar		Maricar
Hannah		Maricar		Maricar		Vanessa		Rowena
Rowena		Rowena		Rowena		Rowena		Vanessa

# SELECTION SORT

## Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	Posisi
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (tukar idx)	4
21 < 40	4

Tukar data ke-0 (32) dengan data ke-4 (21)

0	1	2	3	4	5
21	75	69	58	32	40

## Proses 2

0	1	2	3	4	5
21	75	69	58	32	40

Pembanding	Posisi
75 > 69 (tukar idx)	2
69 > 58 (tukar idx)	3
58 > 32 (tukar idx)	4
32 < 40	4

Tukar data ke-1 (75) dengan data ke-4 (32)

0	1	2	3	4	5
21	32	69	58	75	40

## Proses 3

0	1	2	3	4	5
21	32	69	58	75	40

Pembanding	Posisi
69 > 58 (tukar idx)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (69) dengan data ke-5 (40)

0	1	2	3	4	5
21	32	40	58	75	69

## Proses 4

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (58) dengan data ke-3 (58)

0	1	2	3	4	5
21	32	40	58	75	69

## Proses 5

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding	Posisi
75 > 69	5

Tukar data ke-4 (75) dengan data ke-5 (69)

0	1	2	3	4	5
21	32	40	58	69	75

# ***SELECTION SORT***

```
void selectionSort(Object array[], int startIdx, int endIdx) {  
    int min;  
    for (int i = startIdx; i < endIdx; i++) {  
        min = i;  
        for (int j = i + 1; j < endIdx; j++) {  
            if (((Comparable)array[min]).compareTo(array[j])>0) {  
                min = j;  
            }  
        }  
        swap(array[min], array[i]);  
    }  
}
```

# EXCHANGE SORT

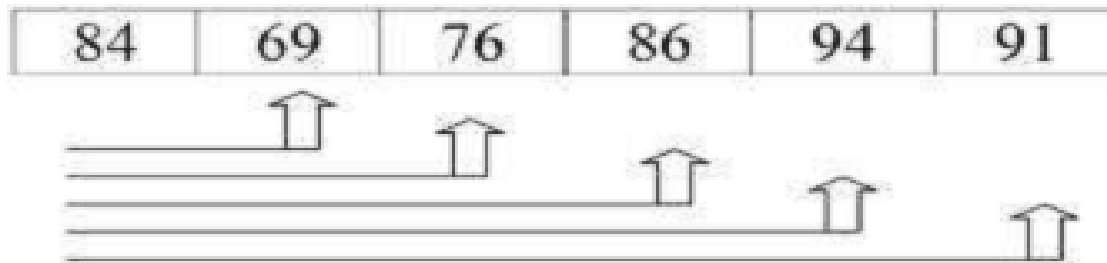
Sangat mirip dengan Bubble Sort

Banyak yang mengatakan Bubble Sort sama dengan Exchange Sort

Pebedaan : dalam hal bagaimana membandingkan antar elemen-elemennya.

- Exchange sort membandingkan **suatu elemen** dengan **elemen-elemen lainnya** dalam array tersebut, dan melakukan pertukaran elemen jika perlu. Jadi ada elemen yang selalu menjadi elemen **pusat (pivot)**.
- Sedangkan Bubble sort akan membandingkan **elemen pertama/terakhir** dengan **elemen sebelumnya/sesudahnya**, kemudian elemen tersebut itu akan menjadi **pusat (pivot)** untuk dibandingkan dengan elemen sebelumnya/sesudahnya lagi, begitu seterusnya.

# EXCHANGE SORT (2)



Proses 1

Pivot (Pusat) →

84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	86	94	91
86	69	76	84	94	91
94	69	76	84	86	91
94	69	76	84	86	91

# EXCHANGE SORT (3)

Proses 2

---

Pivot (Pusat)

94	69	76	84	86	91
94	76	69	84	86	91
94	84	69	76	86	91
94	86	69	76	84	91
94	91	69	76	84	86

Proses 3

Pivot (Pusat)

94	91	69	76	84	86
94	91	76	69	84	86
94	91	84	69	76	86
94	91	86	69	76	84

# EXCHANGE SORT (4)

Proses 4

The diagram for Proses 4 shows a 3x6 grid of numbers. The first row contains 94, 91, 86, 69, 76, 84. The second row contains 94, 91, 86, 76, 69, 84. The third row contains 94, 91, 86, 84, 69, 76. A red circle highlights the value 69 in the first row, with a red arrow pointing to it from the text 'Pivot (Pusat)'. The values 76 and 84 in the third row are also circled in red, indicating they are being compared or moved during this step.

94	91	86	69	76	84
94	91	86	76	69	84
94	91	86	84	69	76

Proses 5

The diagram for Proses 5 shows a 2x6 grid of numbers. The first row contains 94, 91, 86, 84, 69, 76. The second row contains 94, 91, 86, 84, 76, 69. A red circle highlights the value 69 in the first row, with a red arrow pointing to it from the text 'Pivot (Pusat)'. The value 76 in the second row is also circled in red, indicating it is being compared or moved during this step.

94	91	86	84	69	76
94	91	86	84	76	69



# EXCHANGE SORT (5)

## Prosedur Exchange Sort

```
.  
void exchange_sort(int data[]){  
    for (int i=0; i<n-1; i++){  
        for(int j = i+1; j<n; j++){  
            if(data [i] < data[j])  
                tukar (&data[i], &data[j]);  
        }  
    }  
}
```

---

# SELECTION SORT

**Merupakan kombinasi antara sorting dan searching**

**Untuk setiap proses, akan dicari elemen-elemen yang belum diurutkan yang memiliki nilai terkecil atau terbesar akan dipertukarkan ke posisi yang tepat di dalam array.**

**Misalnya untuk putaran pertama, akan dicari data dengan nilai terkecil dan data ini akan ditempatkan di indeks terkecil (`data[0]`), pada putaran kedua akan dicari data kedua terkecil, dan akan ditempatkan di indeks kedua (`data[1]`).**

**Selama proses, perbandingan dan pengubahan hanya dilakukan pada indeks perbandingan saja, pertukaran data secara fisik terjadi pada akhir proses.**

# SELECTION SORT (2)

## Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	Posisi
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (tukar idx)	4
21 < 40	4

Tukar data ke-0 (32) dengan data ke-4 (21)

0	1	2	3	4	5
21	75	69	58	32	40

## Proses 2

0	1	2	3	4	5
21	75	69	58	32	40

Pembanding	Posisi
75 > 69 (tukar idx)	2
69 > 58 (tukar idx)	3
58 > 32 (tukar idx)	4
32 < 40	4

Tukar data ke-1 (75) dengan data ke-4 (32)

0	1	2	3	4	5
21	32	69	58	75	40

## Proses 3

0	1	2	3	4	5
21	32	69	58	75	40

Pembanding	Posisi
69 > 58 (tukar idx)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (69) dengan data ke-5 (40)

0	1	2	3	4	5
21	32	40	58	75	69

## Proses 4

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding	Posisi
58 < 75	3
58 < 69	3

Tukar data ke-3 (58) dengan data ke-3 (58)

0	1	2	3	4	5
21	32	40	58	75	69

## Proses 5

0	1	2	3	4	5
21	32	40	58	75	69

Pembanding	Posisi
75 > 69	5

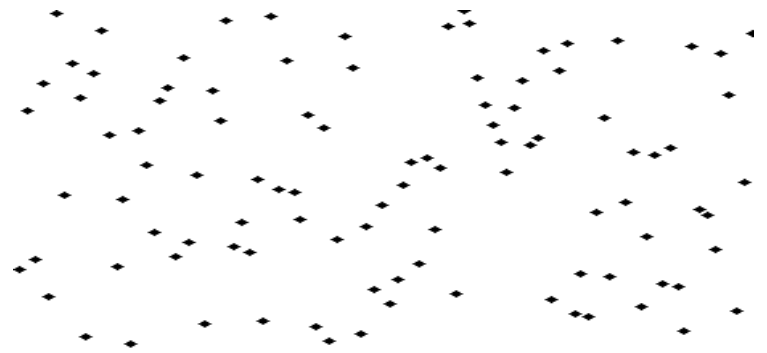
Tukar data ke-4 (75) dengan data ke-5 (69)

0	1	2	3	4	5
21	32	40	58	69	75

# SELECTION SORT (3)

## Prosedur Selection Sort

```
void selection_sort(int data[]){  
    for(int i=0;i<n-1;i++){  
        pos = i;  
        for(int j=i+1;j<n;j++){  
            if(data[j] < data[pos]) pos = j; //ascending  
        }  
        if(pos != i) tukar(&data[pos],&data[i]);  
    }  
}
```



# INSERTION SORT

Mirip dengan cara orang mengurutkan kartu, selembat demi selembat kartu diambil dan disisipkan (insert) ke tempat yang seharusnya.

Pengurutan dimulai dari data ke-2 sampai dengan data terakhir, jika ditemukan data yang lebih kecil, maka akan ditempatkan (diinsert) diposisi yang seharusnya.

Pada penyisipan elemen, maka elemen-elemen lain akan bergeser ke belakang



# INSERTION SORT (2)

## Proses 1

0	1	2	3	4	5
22	10	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2

## Proses 2

0	1	2	3	4	5
10	22	15	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	15	22	3	8	2

## Proses 3

0	1	2	3	4	5
10	15	22	3	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2

## Proses 4

0	1	2	3	4	5
3	10	15	22	8	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	8	10	15	22	2

# INSERTION SORT (3)

## Proses 5

0	1	2	3	4	5
3	8	10	15	22	2

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22

```
tion_sort(int data[]){
    temp;
    for (int i=1;i<n;i++){
        p = data[i];
        i--;
        while (data[j]>temp && j>=0){
            data[j+1] = data[j];
            j--;
        }
        data[j+1] = temp;
    }
}
```

# PERBANDINGAN

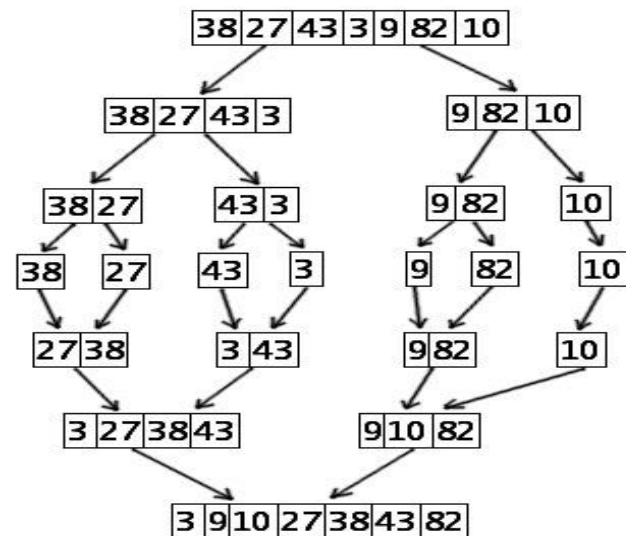
**Tabel Perbandingan Kecepatan Metode Pengurutan Data  
Untuk data sejumlah 10.000 data pada komputer Pentium II  
450 MHz**

Metode	Waktu (detik)		
	Data Acak	Data Ascending	Data Descending
Bubble Sort	11,2	1,32	19,77
Insertion Sort	1,09	0,00	2,25
Selection Sort	1,32	1,32	19,77

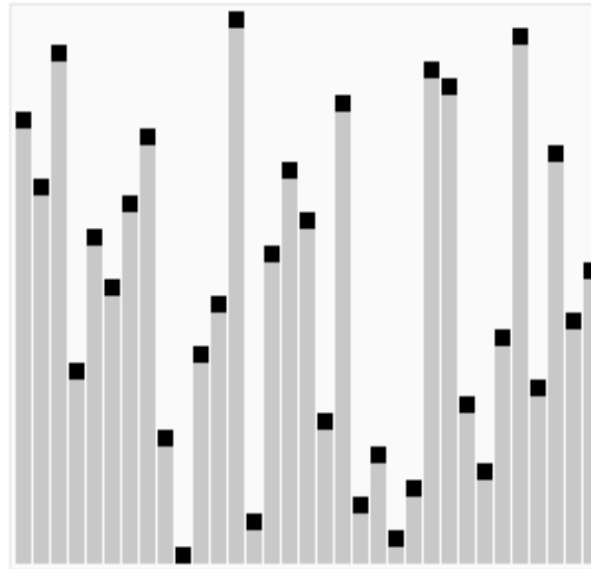


# MASIH BANYAK LAGI

## Merge Sort



## Heap Sort



## Quick Sort

