

Algoritma *Divide and Conquer*

(Bagian 1)






- *Divide and Conquer* dulunya adalah strategi militer yang dikenal dengan nama *divide ut imperes*.

- Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama *Divide and Conquer*.





Definisi

- *Divide*: membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),
 - *Conquer*: memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan
 - *Combine*: menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.
- 
- 
- 

- 
- Obyek permasalahan yang dibagi : masukan (*input*) atau *instances* yang berukuran n seperti:

- tabel (larik),
- matriks,
- eksponen,
- dll, bergantung pada masalahnya.

- Tiap-tiap upa-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal, sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

Skema Umum Algoritma *Divide and Conquer*

```
procedure DIVIDE_and_CONQUER(input n : integer)  
{ Menyelesaikan masalah dengan algoritma D-and-C.  
  Masukan: masukan yang berukuran n  
  Keluaran: solusi dari masalah semula  
}
```

Deklarasi

r, k : integer

Algoritma

```
if  $n \leq n_0$  then {ukuran masalah sudah cukup kecil }  
  SOLVE upa-masalah yang berukuran n ini  
else  
  Bagi menjadi r upa-masalah, masing-masing berukuran  $n/k$   
  for masing-masing dari r upa-masalah do  
    DIVIDE_and_CONQUER( $n/k$ )  
  endfor  
  COMBINE solusi dari r upa-masalah menjadi solusi masalah semula }  
endif
```

Jika pembagian selalu menghasilkan dua upa-masalah yang berukuran sama:

```
procedure DIVIDE_and_CONQUER(input n : integer)  
{ Menyelesaikan masalah dengan algoritma D-and-C.  
  Masukan: masukan yang berukuran n  
  Keluaran: solusi dari masalah semula  
}
```

Deklarasi

r, k : integer

Algoritma

```
if n ≤ n0 then {ukuran masalah sudah cukup kecil }  
  SOLVE upa-masalah yang berukuran n ini  
else  
  Bagi menjadi 2 upa-masalah, masing-masing berukuran n/2  
  DIVIDE_and_CONQUER(upa-masalah pertama yang berukuran n/2)  
  DIVIDE_and_CONQUER(upa-masalah kedua yang berukuran n/2)  
  COMBINE solusi dari 2 upa-masalah  
endif
```

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ 2T(n/2) + f(n) & , n > n_0 \end{cases}$$



Contoh-contoh masalah

1. Mencari Nilai Minimum dan Maksimum (MinMaks)

Persoalan: Misalkan diberikan tabel A yang berukuran n elemen dan sudah berisi nilai *integer*.

Carilah nilai minimum dan nilai maksimum sekaligus di dalam tabel tersebut.

Penyelesaian dengan *Algoritma Brute Force*

```
procedure MinMaks1(input A : TabelInt, n : integer,  
                  output min, maks : integer)  
{ Mencari nilai minimum dan maksimum di dalam tabel A yang berukuran n  
  elemen, secara brute force.  
Masukan: tabel A yang sudah terdefinisi elemen-elemennya  
Keluaran: nilai maksimum dan nilai minimum tabel  
}  
Deklarasi  
  i : integer  
  
Algoritma:  
  min ← A1 { inisialisasi nilai minimum}  
  maks ← A1 { inisialisasi nilai maksimum }  
  for i ← 2 to n do  
    if Ai < min then  
      min ← Ai  
    endif  
  
    if Ai > maks then  
      maks ← Ai  
    endif  
  
  endfor
```

$$T(n) = (n - 1) + (n - 1) = 2n - 2 = O(n)$$

Penyelesaian dengan *Divide and Conquer*

Contoh 4.1. Misalkan tabel A berisi elemen-elemen sebagai berikut:

4 12 23 9 21 1 35 2 24

Ide dasar algoritma secara *Divide and Conquer*:

4 12 23 9 21 1 35 2 24

DIVIDE

4 12 23 9 21 1 35 2 24

SOLVE: tentukan min &
maks pada tiap bagian

4 12 23 9 21 1 35 2 24



min = 4
maks = 23

min = 1
maks = 35

COMBINE

4 12 23 9 21 1 35 2 24

min = 1
maks = 35

- 
- 
- Ukuran tabel hasil pembagian dapat dibuat cukup kecil sehingga mencari minimum dan maksimum dapat diselesaikan (SOLVE) secara lebih mudah.

- 
- 
- Dalam hal ini, ukuran kecil yang dipilih adalah 1 elemen atau 2 elemen.



MinMaks(A , n , \min , \max)

Algoritma:

1. Untuk kasus $n = 1$ atau $n = 2$,

SOLVE: Jika $n = 1$, maka $\min = \max = A[n]$

Jika $n = 2$, maka bandingkan kedua elemen untuk menentukan \min dan \max .



2. Untuk kasus $n > 2$,

(a) **DIVIDE:** Bagi dua tabel A menjadi dua bagian yang sama, A_1 dan A_2




(b) **CONQUER:**

MinMaks(A_1 , $n/2$, \min_1 , \max_1)

MinMaks(A_2 , $n/2$, \min_2 , \max_2)

(c) **COMBINE:**



if $\min_1 < \min_2$ then $\min \leftarrow \min_1$ else $\min \leftarrow \min_2$

if $\max_1 < \max_2$ then $\max \leftarrow \max_2$ else $\max \leftarrow \max_1$

Contoh 4.2. Tinjau kembali Contoh 4.1 di atas.

DIVIDE dan CONQUER:

4 12 23 9 21 1 35 2 24

4 12 23 9 21 1 35 2 24

4 12 23 9 21 1 35 2 24

SOLVE dan COMBINE:

<u>4 12</u>	<u>23 9</u>	<u>21 1</u>	<u>35</u>	<u>2 24</u>
min = 4	min = 9	min = 1	min = 35	min = 2
maks = 12	maks = 23	maks = 21	maks = 35	maks = 24

<u>4 12</u>	<u>23 9</u>	<u>21 1</u>	<u>35 2 24</u>
min = 4		min = 1	min = 2
maks = 23		maks = 21	maks = 35

<u>4 12</u>	<u>23 9</u>	<u>21 1</u>	<u>35 2 24</u>
min = 4		min = 1	
maks = 23		maks = 35	

<u>4 12</u>	<u>23 9</u>	<u>21 1</u>	<u>5 2 24</u>
min = 1			
maks = 35			

```

procedure MinMaks2(input A : TabelInt, i, j : integer,
                    output min, maks : integer)
{ Mencari nilai maksimum dan minimum di dalam tabel A yang berukuran n
  elemen secara Divide and Conquer.
Masukan: tabel A yang sudah terdefinisi elemen-elemennya
Keluaran: nilai maksimum dan nilai minimum tabel
}

```

Deklarasi

```

    min1, min2, maks1, maks2 : integer

```

Algoritma:

```

    if i=j then                                { 1 elemen  }
        min←Ai
        maks←Ai
    else
        if (i = j-1) then                        { 2 elemen  }
            if Ai < Aj then
                maks←Aj
                min←Ai
            else
                maks←Ai
                min←Aj
            endif
        else                                    { lebih dari 2 elemen }
            k←(i+j) div 2                        { bagidua tabel pada posisi k }
            MinMaks2(A, i, k, min1, maks1)
            MinMaks2(A, k+1, j, min2, maks2)
            if min1 < min2 then
                min←min1
            else
                min←min2
            endif

            if maks1 < maks2 then
                maks←maks2
            else
                maks←maks1
            endif
        endif
    endif

```

Kompleksitas waktu asimptotik:

$$T(n) = \begin{cases} 0 & , n = 1 \\ 1 & , n = 2 \\ 2T(n/2) + 2 & , n > 2 \end{cases}$$

Penyelesaian:

Asumsi: $n = 2^k$, dengan k bilangan bulat positif, maka


$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 = 4T(n/4) + 4 + 2 \\ &= 4T(2T(n/8) + 2) + 4 + 2 = 8T(n/8) + 8 + 4 + 2 \\ &= \dots \\ &= 2^{k-1} T(2) + \sum_{i=1}^{k-1} 2^i \\ &= 2^{k-1} \cdot 1 + 2^k - 2 \\ &= n/2 + n - 2 \\ &= 3n/2 - 2 \\ &= O(n) \end{aligned}$$

- 
- MinMaks1 secara *brute force* :

$$T(n) = 2n - 2$$

- 
- MinMaks2 secara *divide and conquer*.

$$T(n) = 3n/2 - 2$$

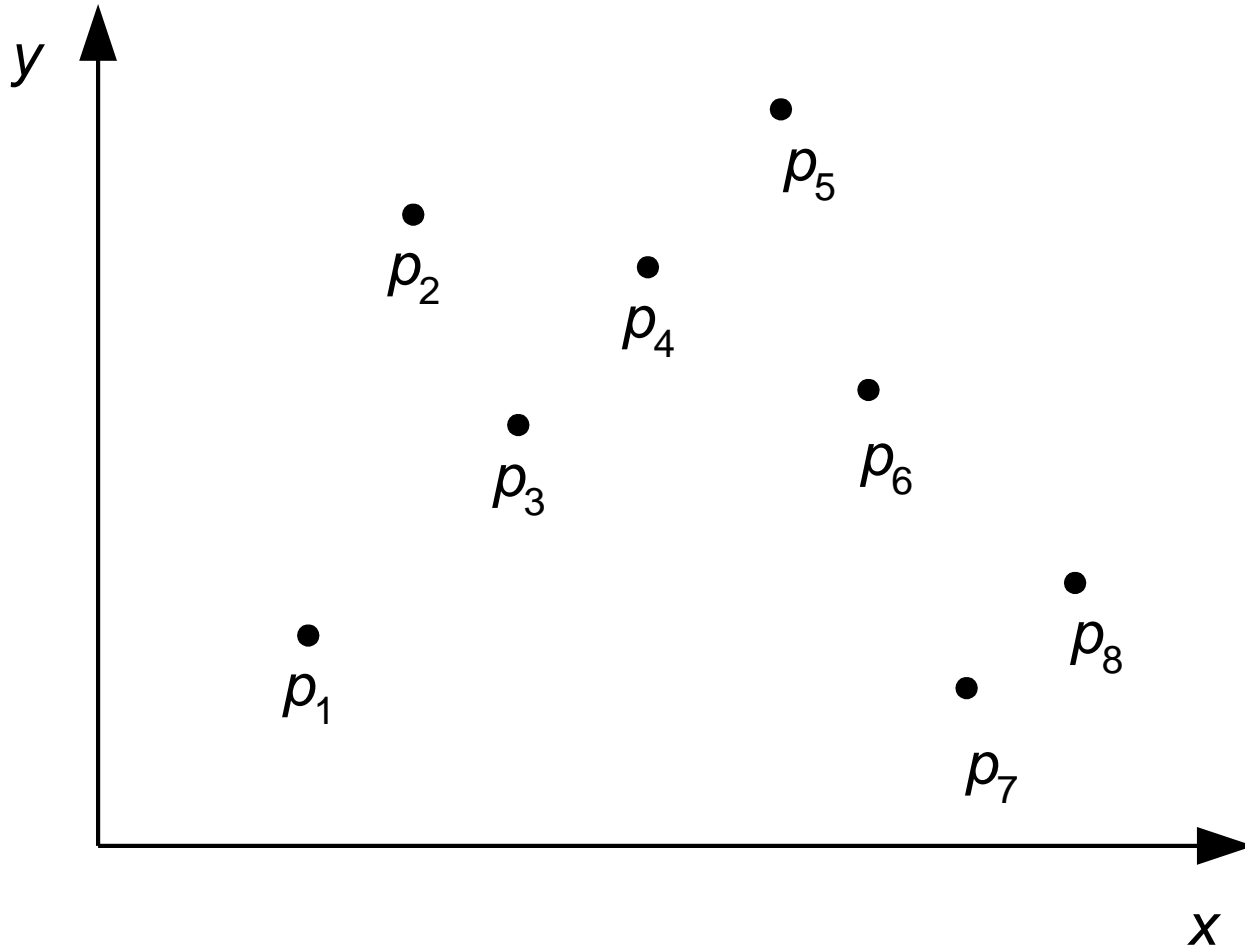
- 
- Perhatikan: $3n/2 - 2 < 2n - 2$, $n \geq 2$.

- 
- Kesimpulan: algoritma MinMaks lebih mangkus dengan metdoe *Divide and Conquer*.



2. Mencari Pasangan Titik yang Jaraknya Terdekat (*Closest Pair*)

Persoalan: Diberikan himpunan titik, P , yang terdiri dari n buah titik, (x_i, y_i) , pada bidang 2-D. Tentukan jarak terdekat antara dua buah titik di dalam himpunan P .



Jarak dua buah titik $p_1 = (x_1, y_1)$ dan $p_2 = (x_2, y_2)$:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



Penyelesaian dengan Algoritma Brute Force

- Hitung jarak setiap pasang titik. Ada sebanyak

$$C(n, 2) = n(n - 1)/2$$

pasangan titik

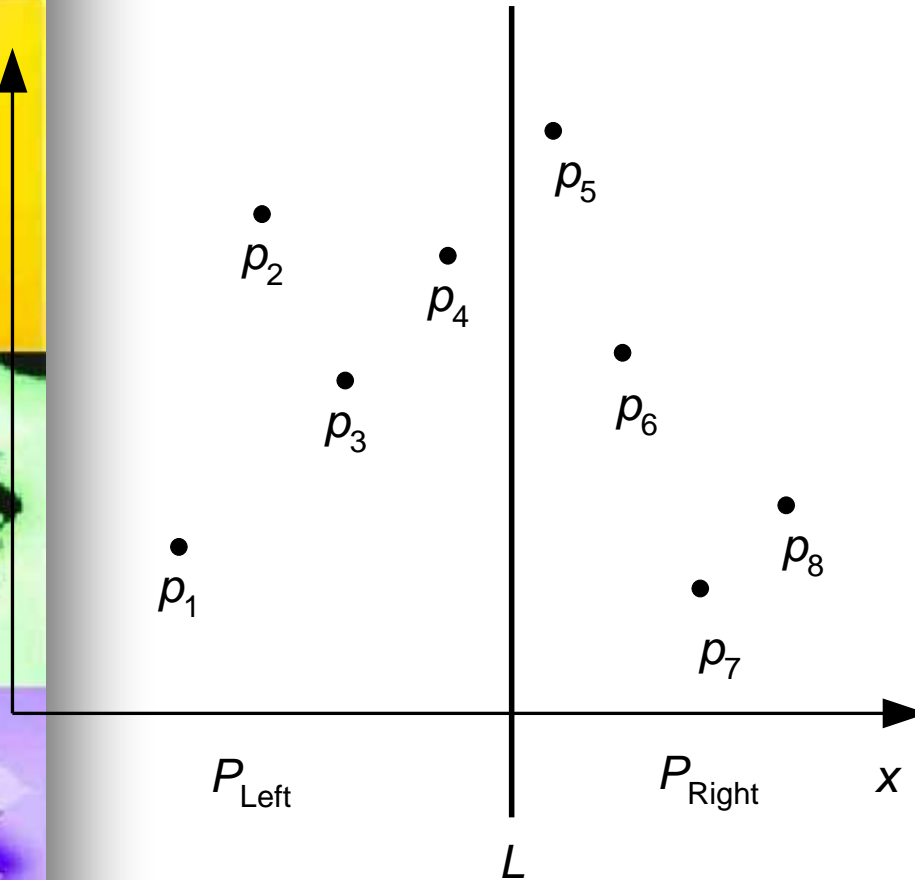
- Pilih pasangan titik yang mempunyai jarak terkecil.
- Kompleksitas algoritma adalah $O(n^2)$.



Penyelesaian dengan Divide and Conquer

- Asumsi: $n = 2^k$ dan titik-titik diurut berdasarkan absis (x).
- Algoritma *Closest Pair*.
 1. SOLVE: jika $n = 2$, maka jarak kedua titik dihitung langsung dengan rumus Euclidean.

2. DIVIDE: Bagi himpunan titik ke dalam dua bagian, P_{left} dan P_{right} , setiap bagian mempunyai jumlah titik yang sama.






3. CONQUER: Secara rekursif, terapkan algoritma *D-and-C* pada masing-masing bagian.

4. Pasangan titik yang jaraknya terdekat ada tiga kemungkinan letaknya:

- 
- (a) Pasangan titik terdekat terdapat di bagian P_{Left} .
 - (b) Pasangan titik terdekat terdapat di bagian P_{Right} .
 - (c) Pasangan titik terdekat dipisahkan oleh garis batas L , yaitu satu titik di P_{Left} dan satu titik di P_{Right} .



Jika kasusnya adalah (c), maka lakukan tahap COMBINE untuk mendapatkan jarak dua titik terdekat sebagai solusi persoalan semula.




```
procedure FindClosestPair2(input P: SetOfPoint, n : integer,  
                           output delta : real)
```

```
{ Mencari jarak terdekat sepasang titik di dalam himpunan P. }
```

Deklarasi:

```
DeltaLeft, DeltaRight : real
```

Algoritma:

```
if n = 2 then
```

```
    delta ← jarak kedua titik dengan rumus Euclidean
```

```
else
```

```
    P-Left ← {p1, p2 , ..., pn/2 }
```

```
    P-Right ← {pn/2+1, pn/2+2 , ..., pn }
```

```
    FindClosestPair2(P-Left, n/2, DeltaLeft)
```

```
    FindClosestPair2(P-Right, n/2, DeltaRight)
```

```
    delta ← minimum(DeltaLeft, DeltaRight)
```


```
{--*****--}
```

```
Tentukan apakah terdapat titik pl di P-Left dan pr di P-Right
```

```
Dengan jarak(pl, pr) < delta. Jika ada, set delta dengan jarak  
terkecil tersebut.
```

```
{--*****--}
```

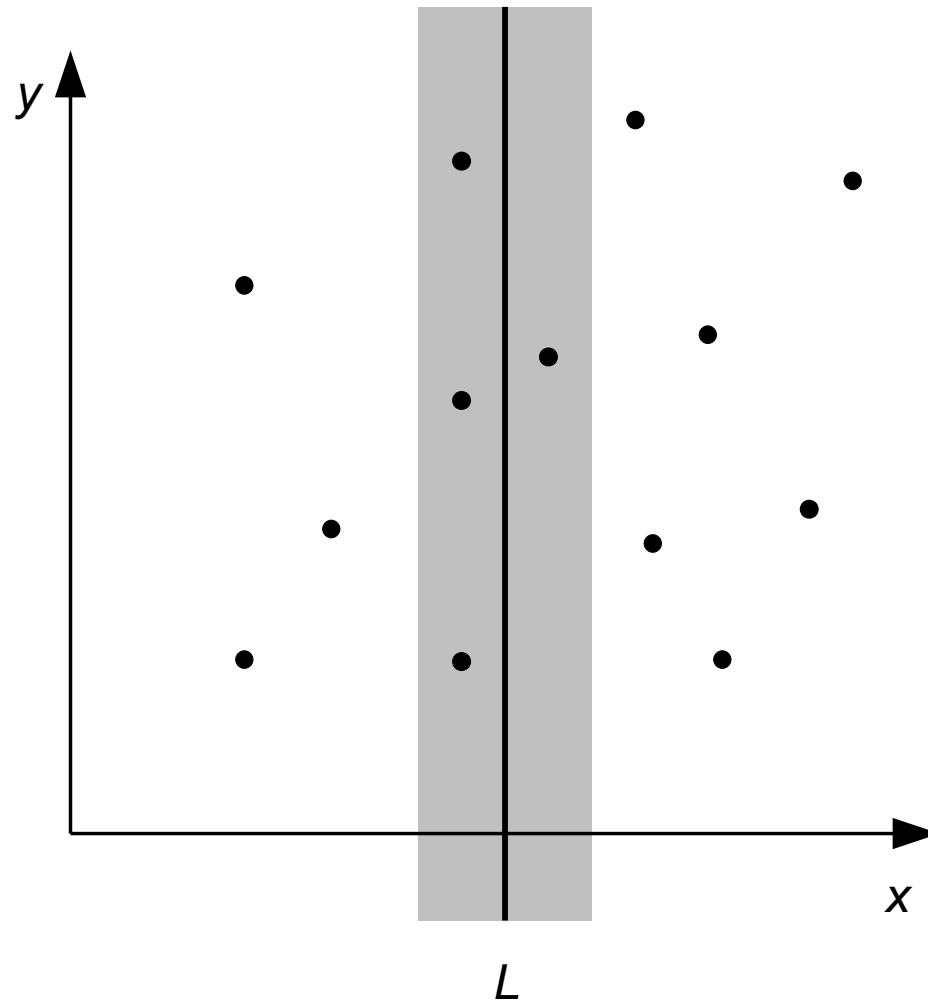
```
endif
```


- 
- Jika terdapat pasangan titik p_l and p_r yang jaraknya lebih kecil dari *delta*, maka kasusnya adalah:

(i) Absis x dari p_l dan p_r berbeda paling banyak sebesar *delta*.

(ii) Ordinat y dari p_l dan p_r berbeda paling banyak sebesar *delta*.

- Ini berarti p_l and p_r adalah sepasang titik yang berada di daerah sekitar garis vertikal L :

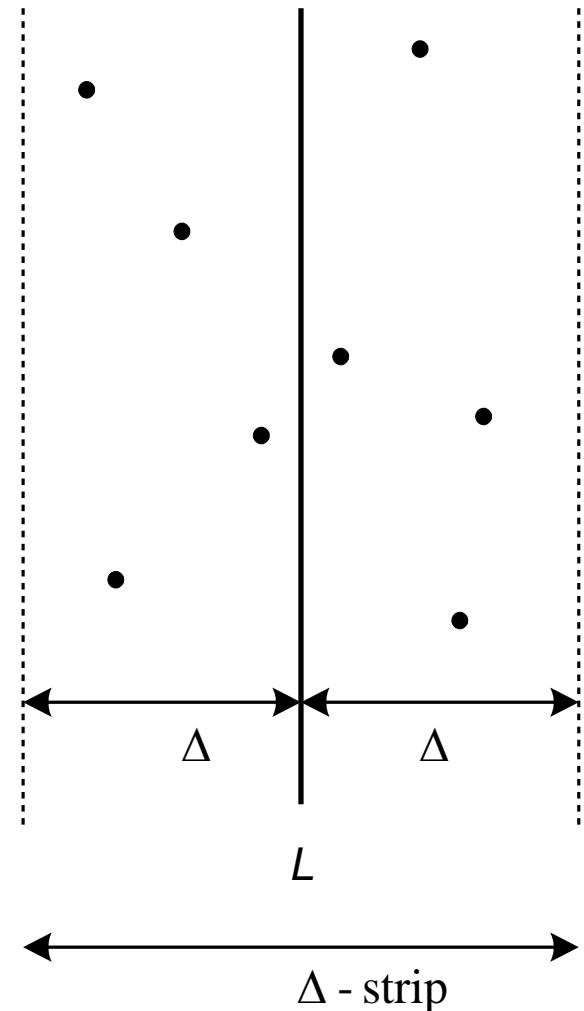


Oleh karena itu, implementasi tahap COMBINE sbb:

- (i) Temukan semua titik di P_{Left} yang memiliki absis x minimal $x_{n/2} - \delta$.
- (ii) Temukan semua titik di P_{Right} yang memiliki absis x maksimal $x_{n/2} + \delta$.

Sebut semua titik-titik yang ditemukan pada langkah (i) dan (ii) tersebut sebagai himpunan P_{strip} yang berisi s buah titik.

Urut titik-titik tersebut dalam urutan absis y yang menaik. Misalkan q_1, q_2, \dots, q_s menyatakan hasil pengurutan.



Langkah COMBINE:

```
for i←1 to s do  
  for j←i+1 to s do  
    exit when ( $|q_i.x - q_j.x| > \Delta$  or  $|q_i.y - q_j.y| > \Delta$ )  
    if jarak ( $q_i, q_j$ ) <  $\Delta$  then  
       $\Delta \leftarrow \text{jarak}(q_i, q_j)$  { dihitung dengan rumus Euclidean }  
    endif  
  endfor  
endfor
```



Kompleksitas algoritma:

$$T(n) = \begin{cases} 2T(n/2) + cn & , n > 2 \\ a & , n = 2 \end{cases}$$

Solusi dari persamaan di atas adalah $T(n) = O(n \log n)$.

3. Algoritma Pengurutan dengan Metode *Divide and Conquer*

```
procedure Sort(input/output A : TabelInt, input n : integer)
```

```
{ Mengurutkan tabel A dengan metode Divide and Conquer
```

```
  Masukan: Tabel A dengan n elemen
```

```
  Keluaran: Tabel A yang terurut
```

```
}
```

Algoritma:

```
  if Ukuran(A) > 1 then
```

```
    Bagi A menjadi dua bagian, A1 dan A2, masing-masing berukuran n1  
    dan n2 (n = n1 + n2)
```

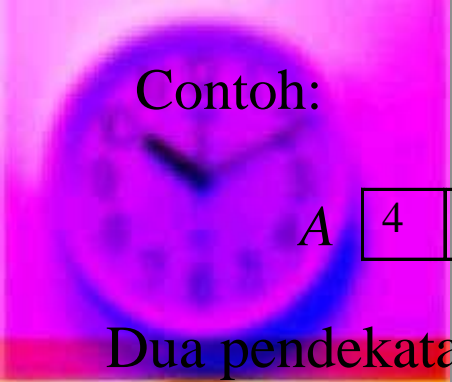
```
    Sort(A1, n1)    { urut bagian kiri yang berukuran n1 elemen }
```

```
    Sort(A2, n2)    { urut bagian kanan yang berukuran n2 elemen }
```

```
    Combine(A1, A2, A) { gabung hasil pengurutan bagian kiri dan  
                        bagian kanan }
```

```
  end
```


Contoh:



A	4	12	3	9	1	21	5	2
---	---	----	---	---	---	----	---	---

Dua pendekatan (*approach*) pengurutan:



1. Mudah membagi, sulit menggabung (*easy split/hard join*)

Tabel A dibagidua berdasarkan posisi elemen:

Divide:

A1	4	12	3	9
----	---	----	---	---

A2	1	21	5	2
----	---	----	---	---

Sort:


A1	3	4	9	12
----	---	---	---	----

A2	1	2	5	21
----	---	---	---	----

Combine:

A1	1	2	3	4	5	9	12	21
----	---	---	---	---	---	---	----	----

Algoritma pengurutan yang termasuk jenis ini:

- 
- a. urut-gabung (*Merge Sort*)
 - b. urut-sisip (*Insertion Sort*)



2. Sulit membagi, mudah menggabung (*hard split/easy join*)

Tabel A dibagidua berdasarkan nilai elemennya. Misalkan elemen-elemen $A1 \leq$ elemen-elemen $A2$.



Divide:

A1

4	2	3	1
---	---	---	---

A2

9	21	5	12
---	----	---	----


Sort:

A1

1	2	3	4
---	---	---	---

A2

5	9	12	21
---	---	----	----



Combine:


A

1	2	3	4	5	9	12	21
---	---	---	---	---	---	----	----

Algoritma pengurutan yang termasuk jenis ini:

a. urut-cepat (*Quick Sort*)

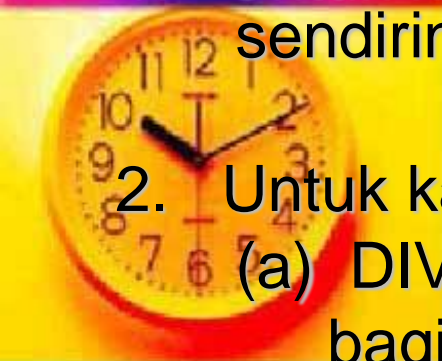
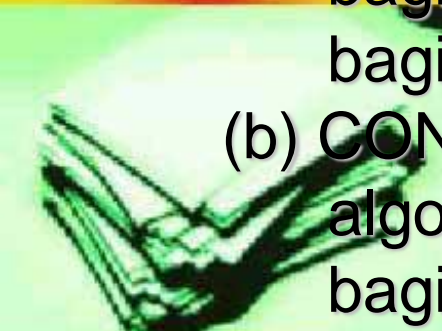

b. urut-seleksi (*Selection Sort*)





(a) Merge Sort

Algoritma:

1. Untuk kasus $n = 1$, maka tabel A sudah terurut dengan sendirinya (langkah SOLVE).
 2. Untuk kasus $n > 1$, maka
 - (a) DIVIDE: bagi tabel A menjadi dua bagian, bagian kiri dan bagian kanan, masing-masing bagian berukuran $n/2$ elemen.
 - (b) CONQUER: Secara rekursif, terapkan algoritma *D-and-C* pada masing-masing bagian.
 - (c) MERGE: gabung hasil pengurutan kedua bagian sehingga diperoleh tabel A yang terurut.
- 
- 
- 



Contoh Merge:

A1	A2		B
1 13 24	2 15 27	$1 < 2 \rightarrow 1$	1
1 13 24	2 15 27	$2 < 13 \rightarrow 2$	1 2
1 13 24	2 15 27	$13 < 15 \rightarrow 13$	1 2 13
1 13 24	2 15 27	$15 < 24 \rightarrow 15$	1 2 13 15
1 13 24	2 15 27	$24 < 27 \rightarrow 24$	1 2 13 15 24
1 13 24	2 15 27	$27 \rightarrow$	1 2 13 15 24 27

Contoh 4.3. Misalkan tabel A berisi elemen-elemen berikut:

4 12 23 9 21 1 5 2

DIVIDE, CONQUER, dan SOLVE:

4 12 23 9 21 1 5 2

4 12 23 9 21 1 5 2

4 12 23 9 21 1 5 2

4 12 23 9 21 1 5 2

MERGE: 4 12 9 23 1 21 2 5

4 9 12 23 1 2 5 21

1 2 4 5 9 12 21 23



```
procedure MergeSort(input/output A : TabelInt, input i, j : integer)
```

```
{ Mengurutkan tabel A[i..j] dengan algoritma Merge Sort
```

```
  Masukan: Tabel A dengan n elemen
```

```
  Keluaran: Tabel A yang terurut
```

```
}
```

```
Deklarasi:
```

```
  k : integer
```

```
Algoritma:
```

```
  if i < j then           { Ukuran(A) > 1 }
```

```
    k ← (i+j) div 2
```

```
    MergeSort(A, i, k)
```

```
    MergeSort(A, k+1, j)
```

```
    Merge(A, i, k, j)
```

```
  endif
```

Prosedur Merge:

```
procedure Merge(input/output A : TabelInt, input kiri,tengah,kanan :  
integer)  
{ Menggabung tabel A[kiri..tengah] dan tabel A[tengah+1..kanan]  
menjadi tabel A[kiri..kanan] yang terurut menaik.  
Masukan: A[kiri..tengah] dan tabel A[tengah+1..kanan] yang sudah  
terurut menaik.  
Keluaran: A[kiri..kanan] yang terurut menaik.  
}
```

Deklarasi

```
B : TabelInt  
i, kidal1, kidal2 : integer
```

Algoritma:

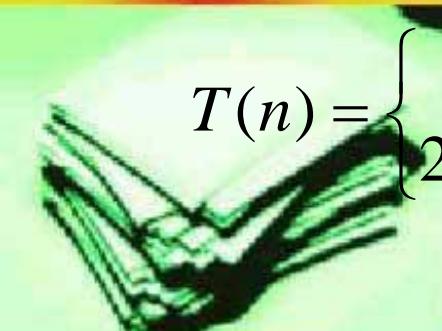
```
kidall←kiri          { A[kiri .. tengah] }  
kidal2←tengah + 1    { A[tengah+1 .. kanan] }  
i←kiri  
while (kidall ≤ tengah) and (kidal2 ≤ kanan) do  
    if Akidall ≤ Akidal2 then  
        Bi←Akidall  
        kidall←kidall + 1  
    else  
        Bi←Akidal2  
        kidal2←kidal2 + 1  
    endif  
    i←i + 1  
endwhile  
{ kidall > tengah or kidal2 > kanan }  
  
{ salin sisa A bagian kiri ke B, jika ada }  
while (kidall ≤ tengah) do  
    Bi←Akidall  
    kidall←kidall + 1  
    i←i + 1  
endwhile  
{ kidall > tengah }  
  
{ salin sisa A bagian kanan ke B, jika ada }  
while (kidal2 ≤ kanan) do  
    Bi←Akidal2  
    kidal2←kidal2 + 1  
    i←i + 1  
endwhile  
{ kidal2 > kanan }  
  
{ salin kembali elemen-elemen tabel B ke A }  
for i←kiri to kanan do  
    Ai←Bi  
endfor  
{ diperoleh tabel A yang terurut membesar }
```

- 
- Kompleksitas waktu:



Asumsi: $n = 2^k$

$T(n)$ = jumlah perbandingan pada pengurutan dua buah upatabel + jumlah perbandingan pada prosedur *Merge*


$$T(n) = \begin{cases} a & , n = 1 \\ 2T(n/2) + cn & , n > 1 \end{cases}$$





Penyelesaian:

$$\begin{aligned}T(n) &= 2T(n/2) + cn \\&= 2(2T(n/4) + cn/2) + cn = 4T(n/4) + 2cn \\&= 4(2T(n/8) + cn/4) + 2cn = 8T(n/8) + 3cn \\&= \dots \\&= 2^k T(n/2^k) + kcn\end{aligned}$$

Berhenti jika ukuran tabel terkecil, $n = 1$:

$$n/2^k = 1 \rightarrow k = {}^2\log n$$

sehingga

$$\begin{aligned}T(n) &= nT(1) + cn {}^2\log n \\&= na + cn {}^2\log n \\&= O(n {}^2\log n)\end{aligned}$$

(b) *Insertion Sort*

```
procedure InsertionSort(input/output A : TabelInt,  
                        input i, j : integer)  
{ Mengurutkan tabel A[i..j] dengan algoritma Insertion Sort.  
  Masukan: Tabel A dengan n elemen  
  Keluaran: Tabel A yang terurut  
}
```

Deklarasi:

k : integer

Algoritma:

```
if i < j then                { Ukuran(A) > 1 }  
  k ← i  
  InsertionSort(A, i, k)  
  InsertionSort(A, k+1, j)  
  Merge(A, i, k, j)  
endif
```

Perbaikan:

```
procedure InsertionSort(input/output A : TabelInt,  
                        input i, j : integer)  
{ Mengurutkan tabel A[i..j] dengan algoritma Insertion Sort.  
  Masukan: Tabel A dengan n elemen  
  Keluaran: Tabel A yang terurut  
}
```

Deklarasi:

k : integer

Algoritma:

```
if i < j then                { Ukuran(A) > 1 }  
  k ← i  
  Insertion(A, k+1, j)  
  Merge(A, i, k, j)  
endif
```

Prosedur *Merge* dapat diganti dengan prosedur penyisipan sebuah elemen pada tabel yang sudah terurut (lihat algoritma *Insertion Sort* versi iteratif).

Contoh 4.4. Misalkan tabel A berisi elemen-elemen berikut:

4 12 23 9 21 1 5 2

DIVIDE, CONQUER, dan SOLVE::

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2



MERGE: 4 12 3 9 1 21 5 2



3 4 12 9 1 21 5 2

3 4 9 12 1 21 5 2

1 3 4 9 12 21 5 2



1 3 4 9 12 21 5 2

1 3 4 5 9 12 21 2



1 2 3 4 5 9 12 21

Kompleksitas waktu algoritma *Insertion Sort*:

$$T(n) = \begin{cases} a & , n = 1 \\ T(n-1) + cn & , n > 1 \end{cases}$$

Penyelesaian:

$$\begin{aligned} T(n) &= cn + T(n-1) \\ &= cn + \{ c \cdot (n-1) + T(n-2) \} \\ &= cn + c(n-1) + \{ c \cdot (n-2) + T(n-3) \} \\ &= cn + c \cdot (n-1) + c \cdot (n-2) + \{ c(n-3) + T(n-4) \} \\ &= \dots \\ &= cn + c \cdot (n-1) + c(n-2) + c(n-3) + \dots + c2 + T(1) \\ &= c \{ n + (n-1) + (n-2) + (n-3) + \dots + 2 \} + a \\ &= c \{ (n-1)(n+2)/2 \} + a \\ &= cn^2/2 + cn/2 + (a-c) \\ &= O(n^2) \end{aligned}$$