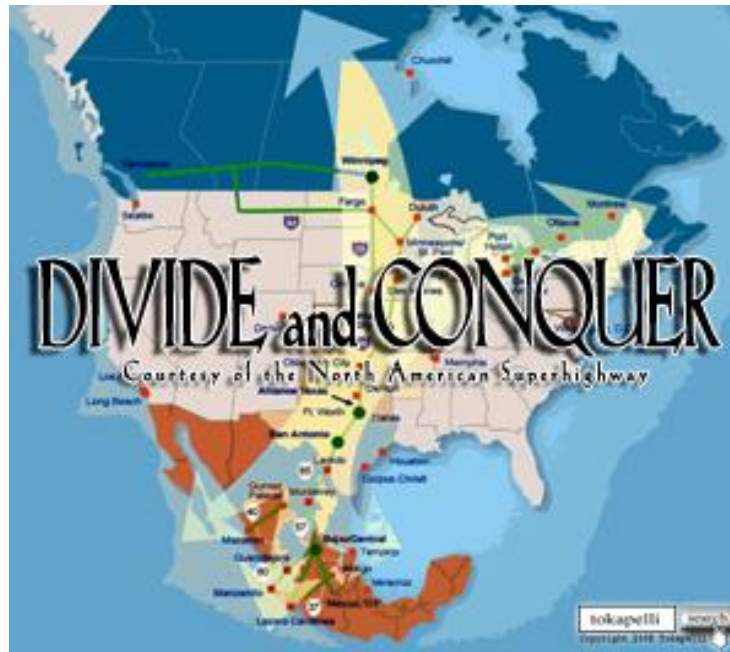


Algoritma *Divide and Conquer*

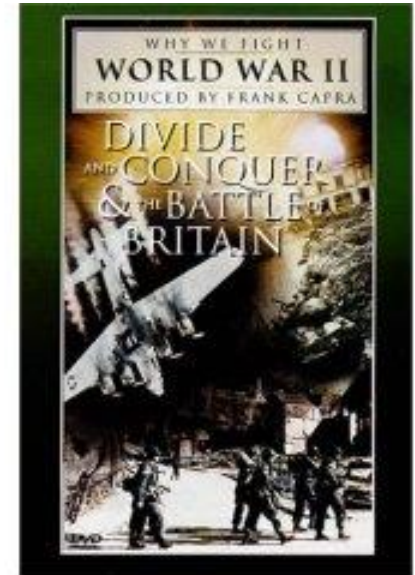
Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir



Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB

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

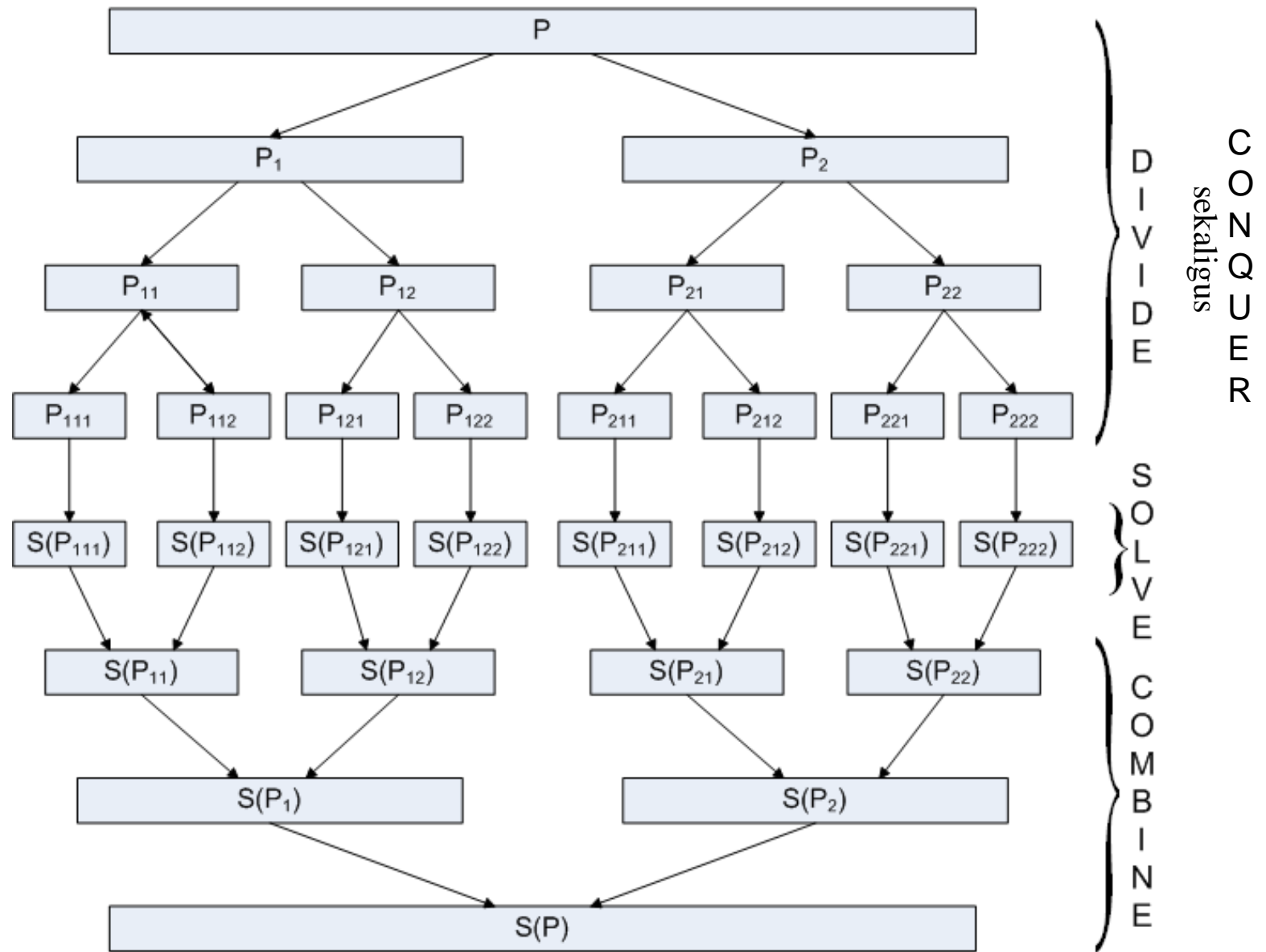


- Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama *Divide and Conquer*.



Definisi

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



Keterangan:

P = persoalan

S = solusi

- Obyek persoalan yang dibagi : masukan (*input*) atau *instances* persoalan yang berukuran n seperti:
 - tabel (larik),
 - matriks,
 - eksponen,
 - dll, bergantung persoalannya.
- Tiap-tiap upa-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal
- sehingga metode *Divide and Conquer* lebih natural diungkapkan dengan 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  $n/r$   
        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

```
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
```

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

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
```

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

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

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

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

4 12 23 9 21 1 5 2 24

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

```


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

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

Bandingkan:

- *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 algoritma *Divide and Conquer*.
- Algoritma *divide and conquer* dapat membantu kita menemukan algoritma yang mangkus.

Algoritma Pengurutan Secara *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:

<i>Divide:</i>	A1	4	12	3	9	A2	1	21	5	2
----------------	----	---	----	---	---	----	---	----	---	---

<i>Sort:</i>	A1	3	4	9	12	A2	1	2	5	21
--------------	----	---	---	---	----	----	---	---	---	----

<i>Combine:</i>	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 $A2$.

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

<i>Divide:</i>	A1	4	2	3	1
----------------	----	---	---	---	---

A2	9	21	5	12
----	---	----	---	----

<i>Sort:</i>	A1	1	2	3	4
--------------	----	---	---	---	---

A2	5	9	12	21
----	---	---	----	----

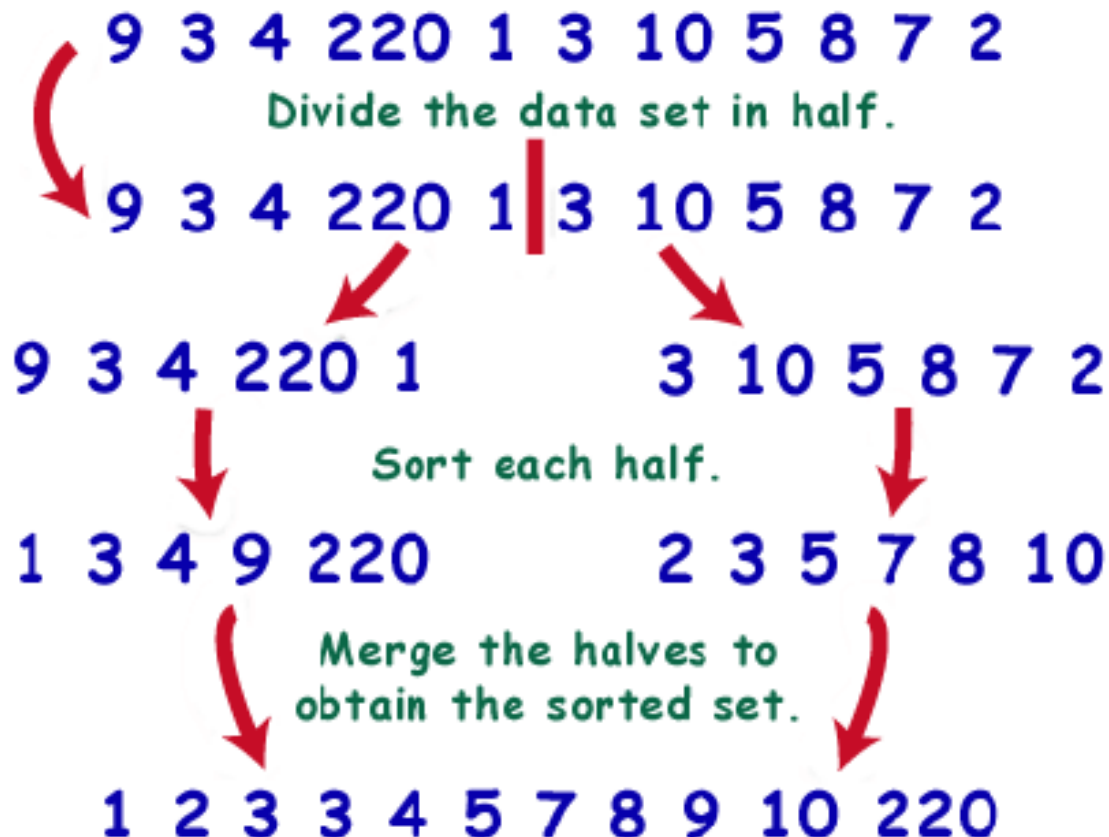
<i>Combine:</i>	A	1	2	3	4	5	9	12	21
-----------------	---	---	---	---	---	---	---	----	----

Algoritma pengurutan yang termasuk jenis ini:

- urut-cepat (*Quick Sort*)
- urut-seleksi (*Selection Sort*)

(a) Merge Sort

- Ide *merge sort*:

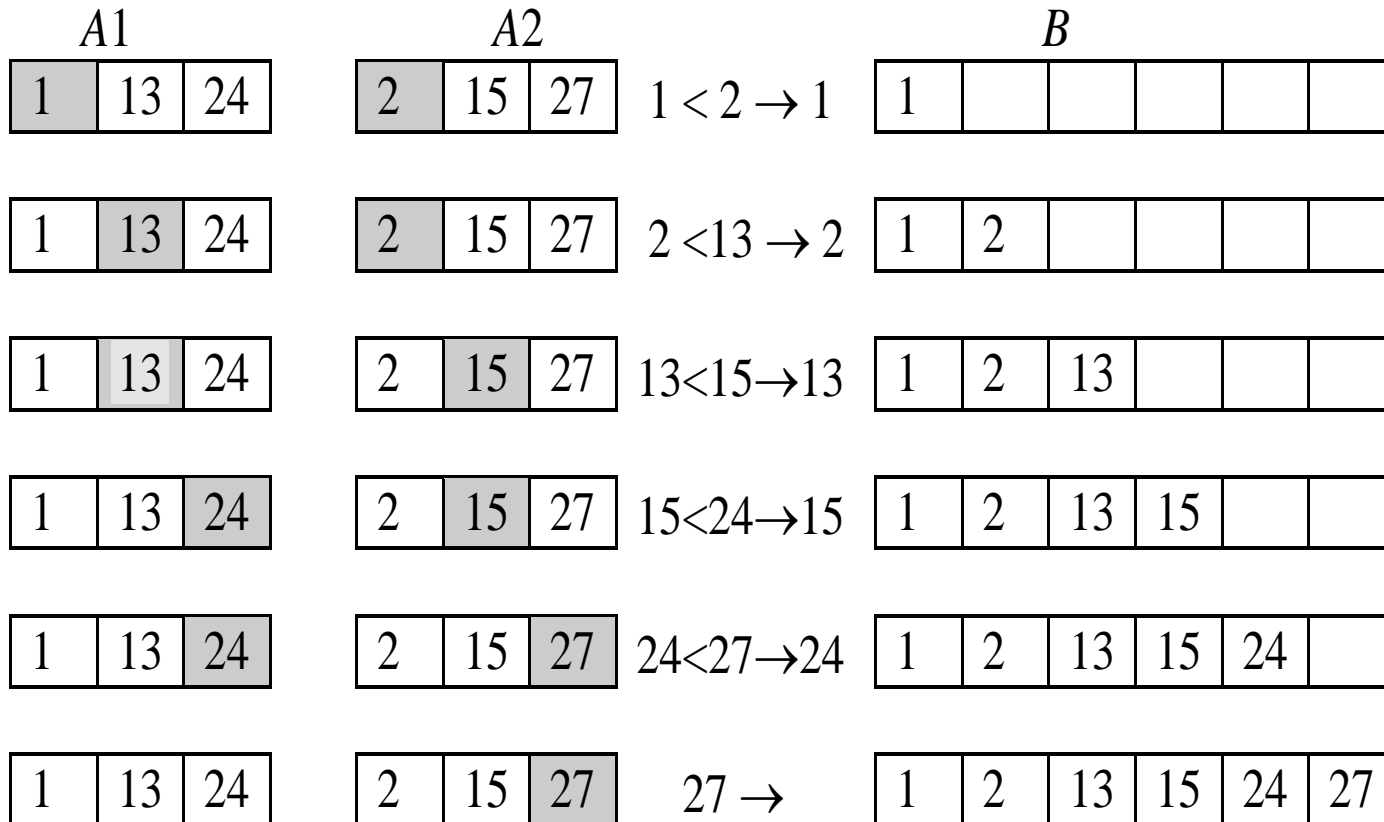


Merge Sort

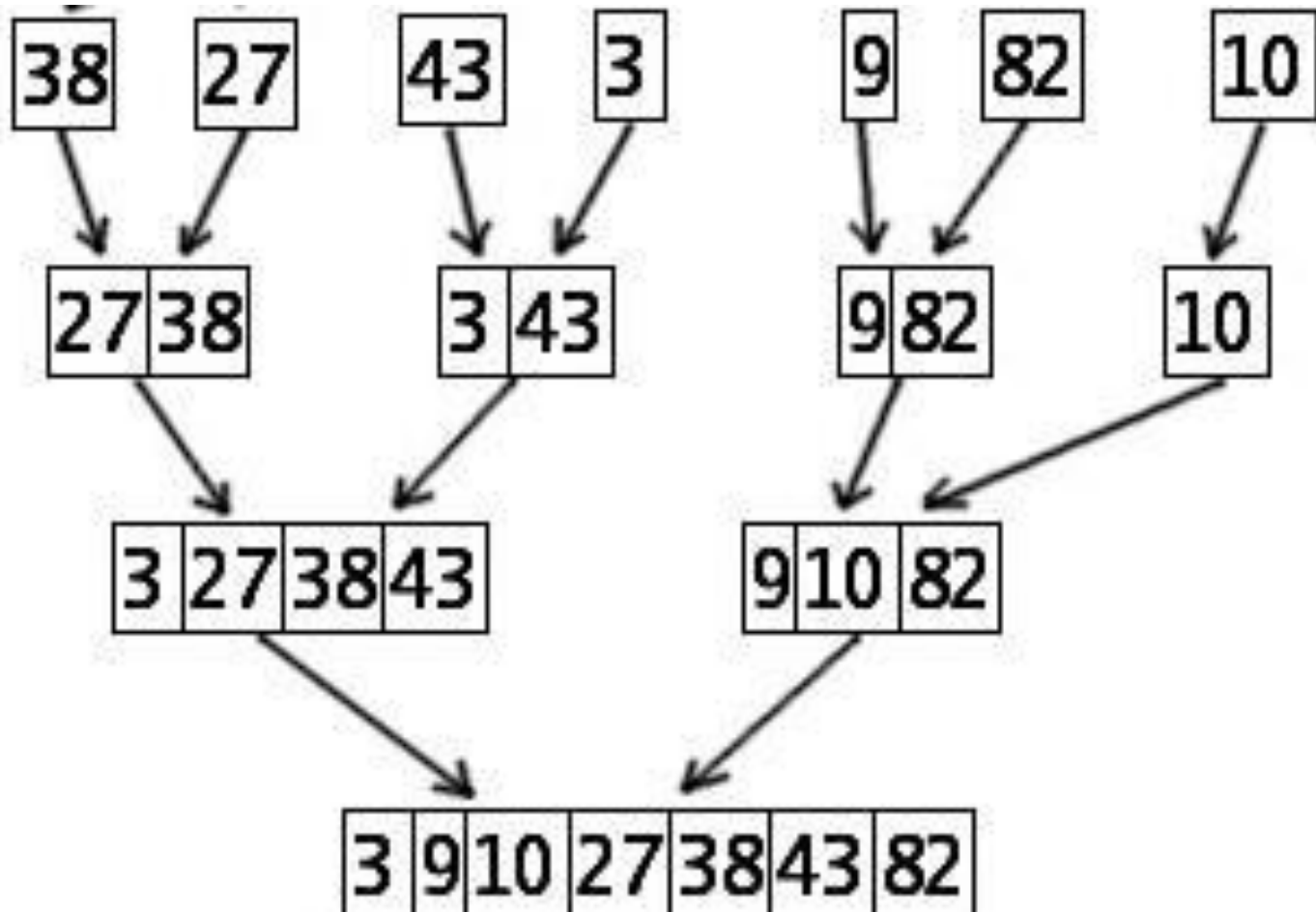
Algoritma:

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*:



Proses merge:



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

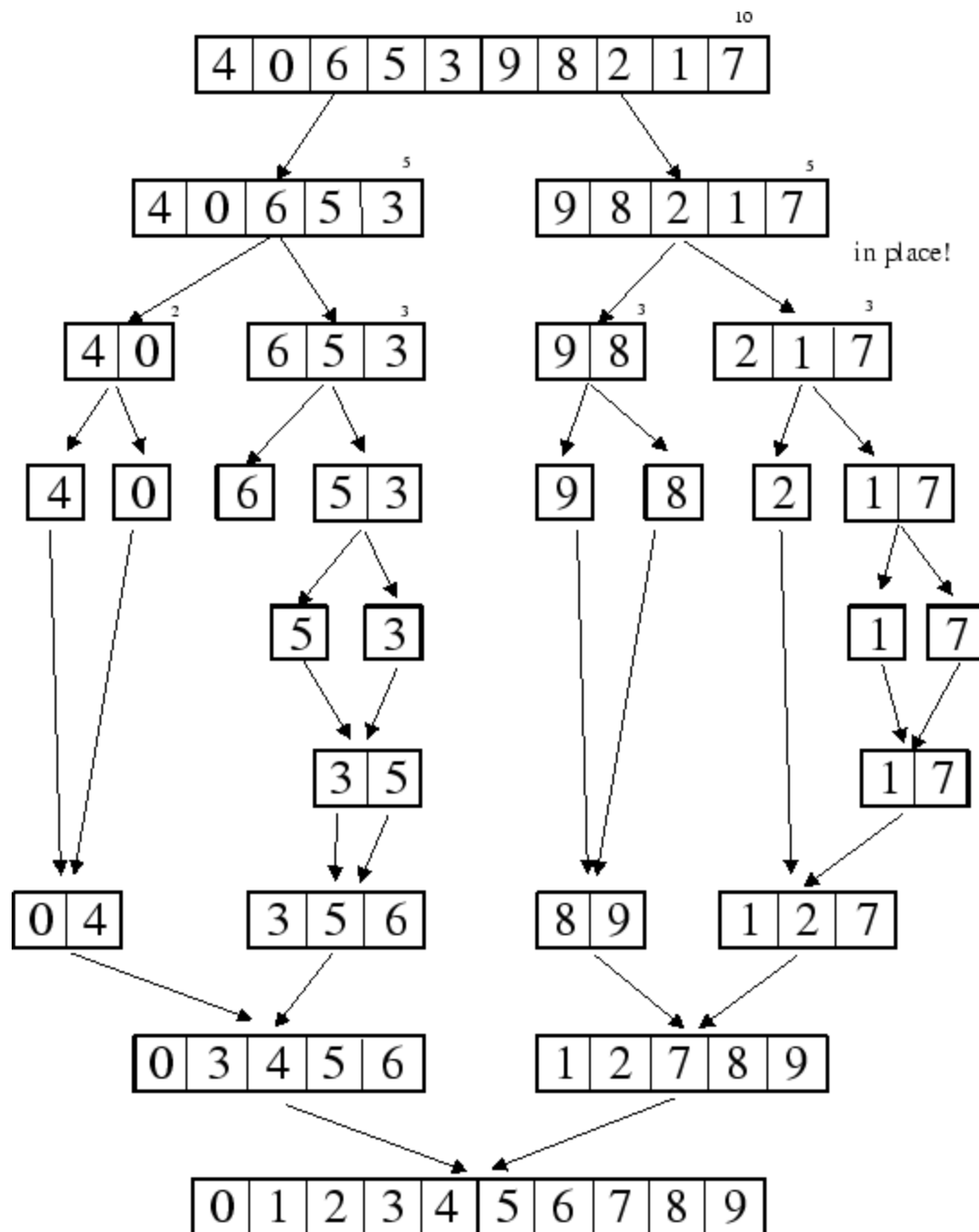
4 12 23 9 21 1 5 2

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  
    InsertionSort(A, k+1, j)  
    Merge(A, i, k, j)  
  endif
```

Selain menggunakan prosedur *Merge* yang sama seperti pada *MergeSort*, kita dapat mengganti *Merge* 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 2 5

4 12 3 9 1 2 5 21

4 12 3 9 1 2 5 21

4 12 3 1 2 5 9 21

4 12 1 2 3 5 9 21

4 1 2 3 5 9 12 21

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:

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

(c) Quick Sort

- Ditemukan oleh Tony Hoare tahun 1959 dan dipublikasikan tahun 1962.
- Termasuk pada pendekatan sulit membagi, mudah menggabung (*hard split/easy join*)
- Tabel A dibagi (istilahnya: dipartisi) menjadi A1 dan A2 sedemikian sehingga elemen-elemen $A1 \leq$ elemen-elemen A2.

Partisi: 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
---	---	---	---	---	---	----	----

9 3 4 220 1 3 10 5 8

Choose a pivot.

9 3 4 220 1 3 10 5 8

Partition data by pivot value.

3 4 1 3 5 8 9 220 10

Sort each partitioned set.

1 3 3 4 5 8 9 10 220

- Terdapat beberapa varian Algoritma Quicksort. Versi orisinal adalah dari Hoare seperti di bawah ini:

Teknik mem-partisi tabel:

- (i) pilih $x \in \{ A[1], A[2], \dots, A[n] \}$ sebagai *pivot*,
- (ii) pindai tabel dari kiri sampai ditemukan $A[p] \geq x$
- (iii) pindai tabel dari kanan sampai ditemukan $A[q] \leq x$
- (iv) pertukarkan $A[p] \Leftrightarrow A[q]$
- (v) ulangi (ii), dari posisi $p + 1$, dan (iii), dari posisi $q - 1$, sampai kedua pemindaian bertemu di tengah tabel ($p \geq q$)

Contoh 4.6. Misalkan tabel A berisi elemen-elemen berikut:

8 1 4 6 9 3 5 7

Langkah-langkah partisi:

(i): 8 1 4 6 9 3 5 7
 pivot


$$\begin{array}{cccccccc} & \rightarrow & & & & & & \leftarrow \\ \text{(ii) \& (iii):} & 8 & 1 & 4 & 6 & 9 & 3 & 5 & 7 \\ & \uparrow p & & & & & & \uparrow q & \end{array}$$

(iv):

5	1	4	6	9	3	8	7
↑						↑	
<hr/>							

$$\begin{array}{ccccccc} & & \rightarrow & & & \leftarrow & \\ \text{(ii) \& (iii):} & 5 & 1 & 4 & 6 & 9 & 3 & 8 & 7 \\ & & & & \uparrow_p & & \uparrow_q & & \end{array}$$

(iv): 5 1 4 3 9 6 8 7



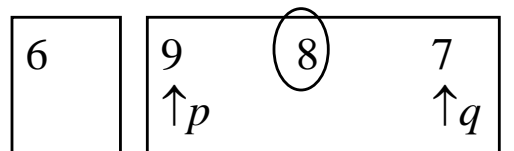
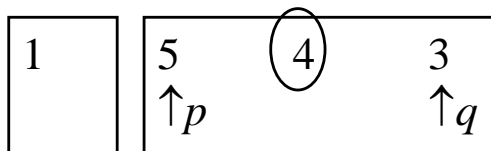
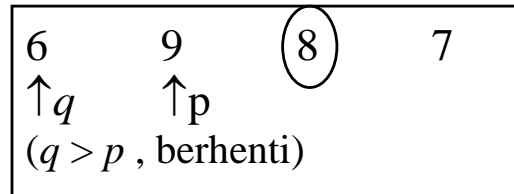
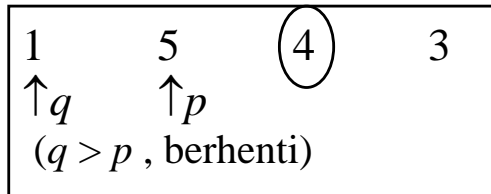
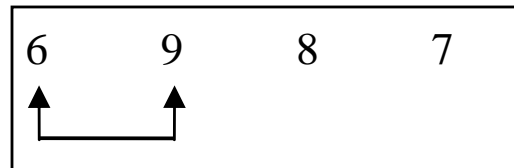
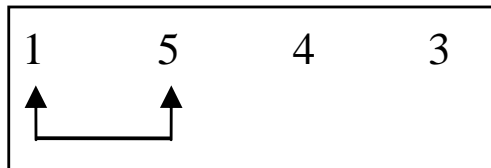
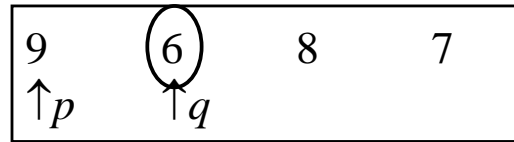
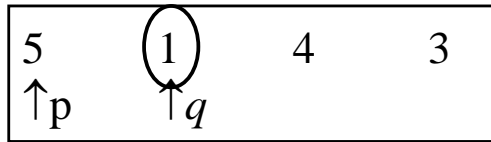
(ii) & (iii):

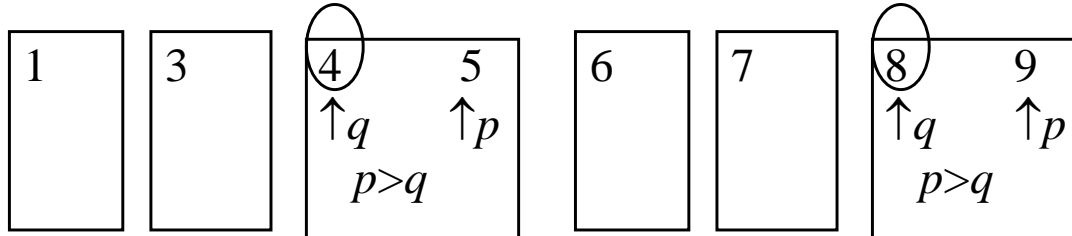
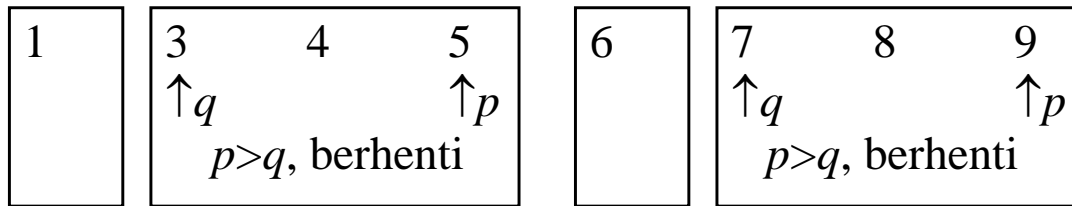
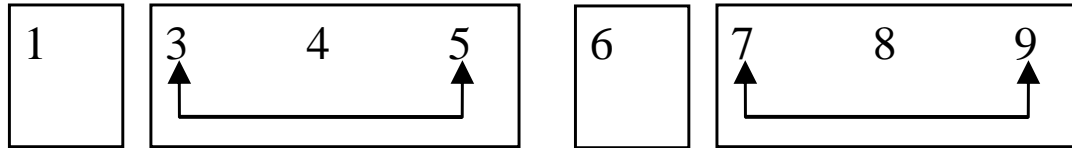
5	1	4	3	$\begin{array}{c} \rightarrow \leftarrow \\ \hline \end{array}$	9	6	8	7
			$\uparrow q$		$\uparrow p$	$(q < p, \text{berhenti})$		

Hasil partisi pertama:

kiri: 5 1 4 3 (< 6)

kanan: 9 6 8 7 (≥ 6)





(terurut)

Pseudo-code Quick Sort:

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

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

```
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.
```

```
  Keluaran: Tabel A[i..j] yang terurut menaik.
```

```
}
```

Deklarasi

```
  k : integer
```

Algoritma:

```
  if i < j then                                { Ukuran(A) > 1 }  
    Partisi(A, i, j, k)                          { Dipartisi pada indeks k }  
    QuickSort(A, i, k)                           { Urut A[i..k] dengan Quick Sort }  
    QuickSort(A, k+1, j)                         { Urut A[k+1..j] dengan Quick Sort }  
  endif
```

```

procedure Partisi(input/output A : TabelInt, input i, j : integer,
                  output q : integer)

{ Membagi tabel A[i..j] menjadi upatabel A[i..q] dan A[q+1..j]
  Masukan: Tabel A[i..j] yang sudah terdefinisi harganya.
  Keluaran upatabel A[i..q] dan upatabel A[q+1..j] sedemikian sehingga
    elemen tabel A[i..q] lebih kecil dari elemen tabel A[q+1..j]
}

Deklarasi
  pivot, temp : integer

Algoritma:
  pivot ← A[(i + j) div 2]    { pivot = elemen tengah }
  p ← i
  q ← j
  repeat
    while A[p] < pivot do
      p ← p + 1
    endwhile
    { A[p] >= pivot }

    while A[q] > pivot do
      q ← q - 1
    endwhile
    { A[q] <= pivot }

    if p < q then
      { pertukarkan A[p] dengan A[q] }
      swap(A[p], A[q])

      { tentukan awal pemindaian berikutnya }
      p ← p + 1
      q ← q - 1
    endif
  until p > q

```


Versi kedua: Partisi sedemikian rupa sehingga elemen-elemen larik kiri \leq pivot dan elemen-elemen larik kanan \geq dari pivot. Pivot = elemen pertama.

Contoh:

pivot
↓
5, 3, 1, 9, 8, 2, 4, 7

Partisi

pivot
↓
2, 3, 1, 4, **5**, 8, 9, 7

↔
Semua \leq pivot Semua \geq pivot

Pseudo-code Quick Sort versi 2:

```
procedure QuickSort2(input/output A : TabelInt, input i,j: integer)  
  
  { Mengurutkan tabel A[i..j] dengan algoritma Quick Sort.  
    Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.  
    Keluaran: Tabel A[i..j] yang terurut menaik.  
  }  
  
  Deklarasi  
    k : integer  
  
  Algoritma:  
    if i < j then                                { Ukuran(A) > 1 }  
      Partisi2(A, i, j, k)                          { Dipartisi pada indeks k }  
      QuickSort2(A, i, k-1)                        { Urut A[i..k-1] dengan Quick Sort }  
      QuickSort2(A, k+1, j)                        { Urut A[k+1..j] dengan Quick Sort }  
    endif
```

```
procedure Partisi2(input/output A : TabelInt, input i, j : integer,  
                   output q : integer)
```

```
{ Membagi tabel A[i..j] menjadi upatabel A[i..q] dan A[q+1..j]  
  Masukan: Tabel A[i..j] yang sudah terdefinisi harganya.  
  Keluaran upatabel A[i..q] dan upatabel A[q+1..j] sedemikian  
  sehingga elemen tabel A[i..q] lebih kecil dari pivot dan elemen  
  tabel A[q+1..j] lebih besar dari pivot  
}
```

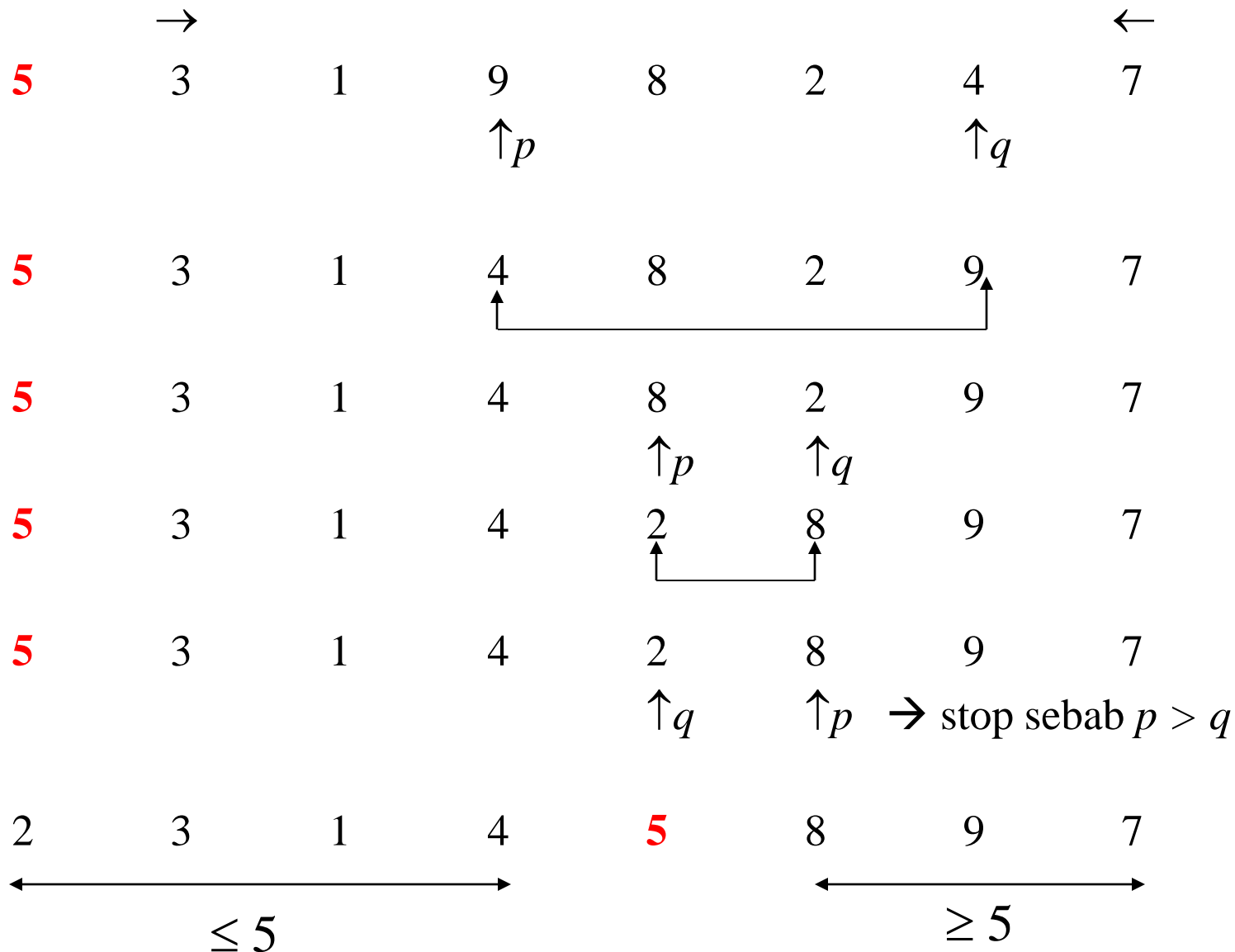
Deklarasi

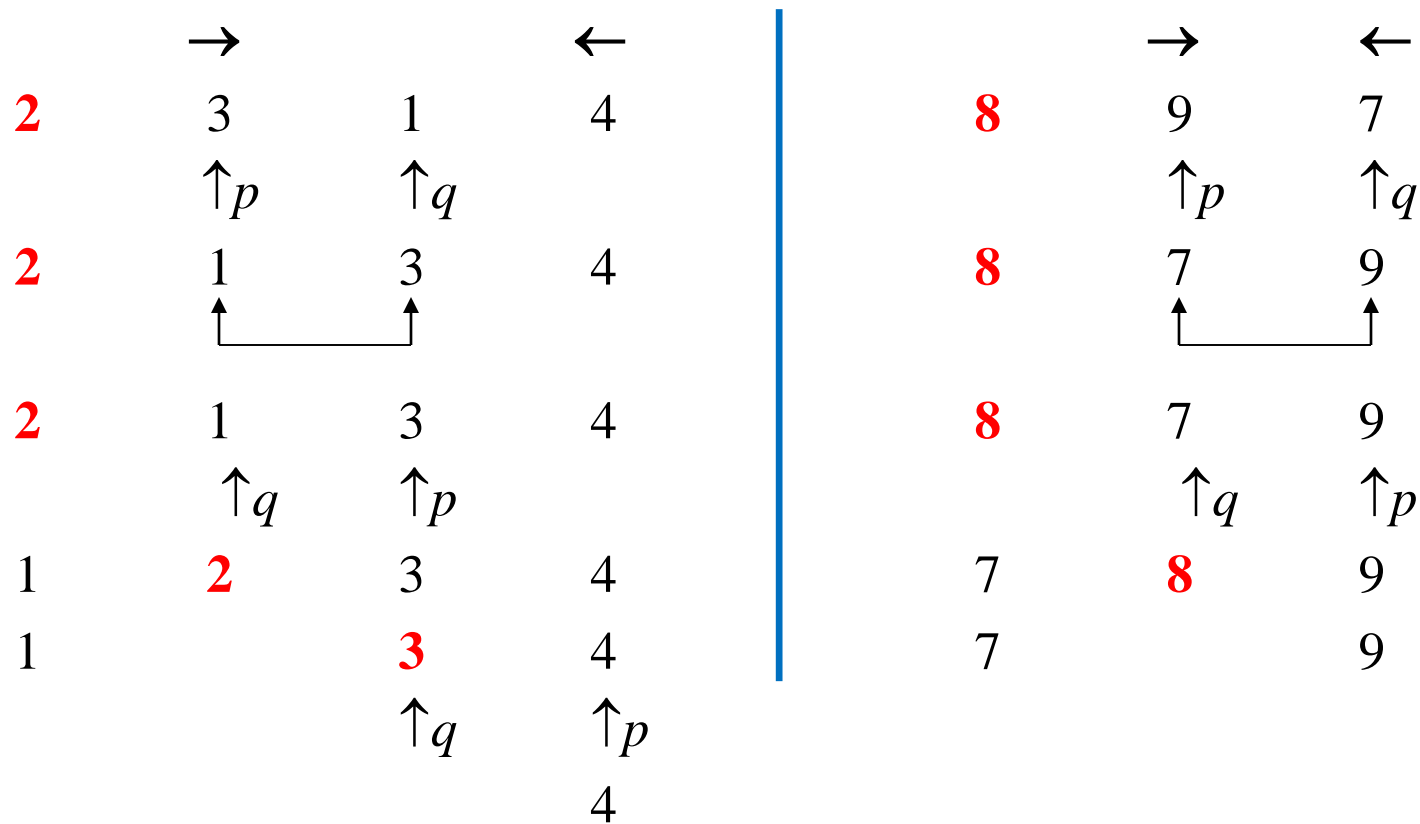
```
  pivot : integer  
  p : integer
```

Algoritma:

```
  pivot ← A[i] { missal pivot = elemen pertama }  
  p ← i  
  q ← j+1  
  repeat  
    repeat  
      p ← p + 1  
    until A[p] ≥ pivot  
  
    repeat  
      q ← q - 1  
    until A[q] ≤ pivot  
  
    swap(A[p], A[q]) {pertukarkan A[p] dengan A[q] }  
  until p ≥ q  
  
  swap(A[p], A[q]) { undo last swap when p ≥ q }  
  swap(A[i], A[q])
```

Contoh: (Levitin, 2003)





Terurut:

1 2 3 4 5 7 8 9

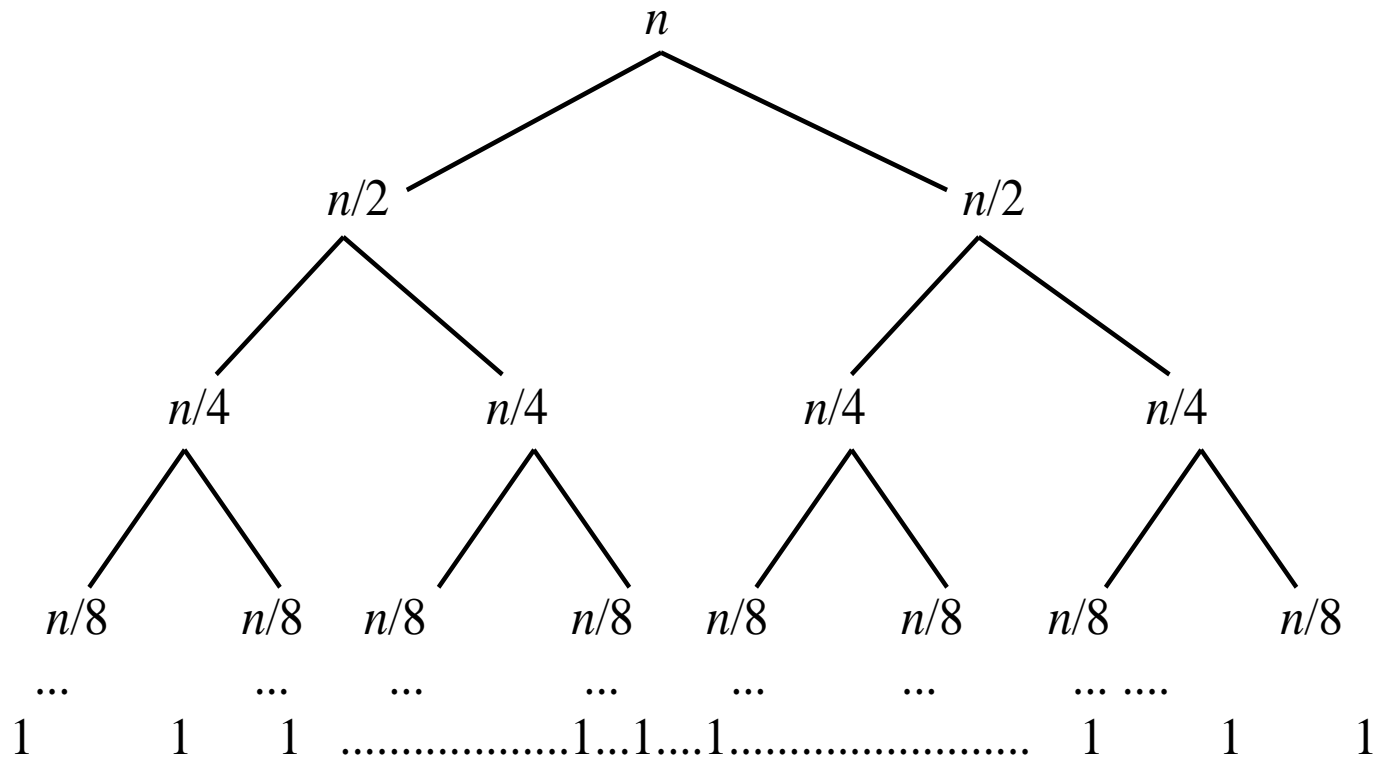
Cara pemilihan *pivot* (khusus pada Versi 1):

1. *Pivot* = elemen pertama/elemen terakhir/elemen tengah tabel
2. *Pivot* dipilih secara acak dari salah satu elemen tabel.
3. *Pivot* = elemen median tabel

Kompleksitas Algoritma *Quicksort*:

1. *Kasus terbaik (best case)*

- Kasus terbaik terjadi bila *pivot* adalah elemen median sedemikian sehingga kedua upatabel berukuran relatif sama setiap kali pempartisian.



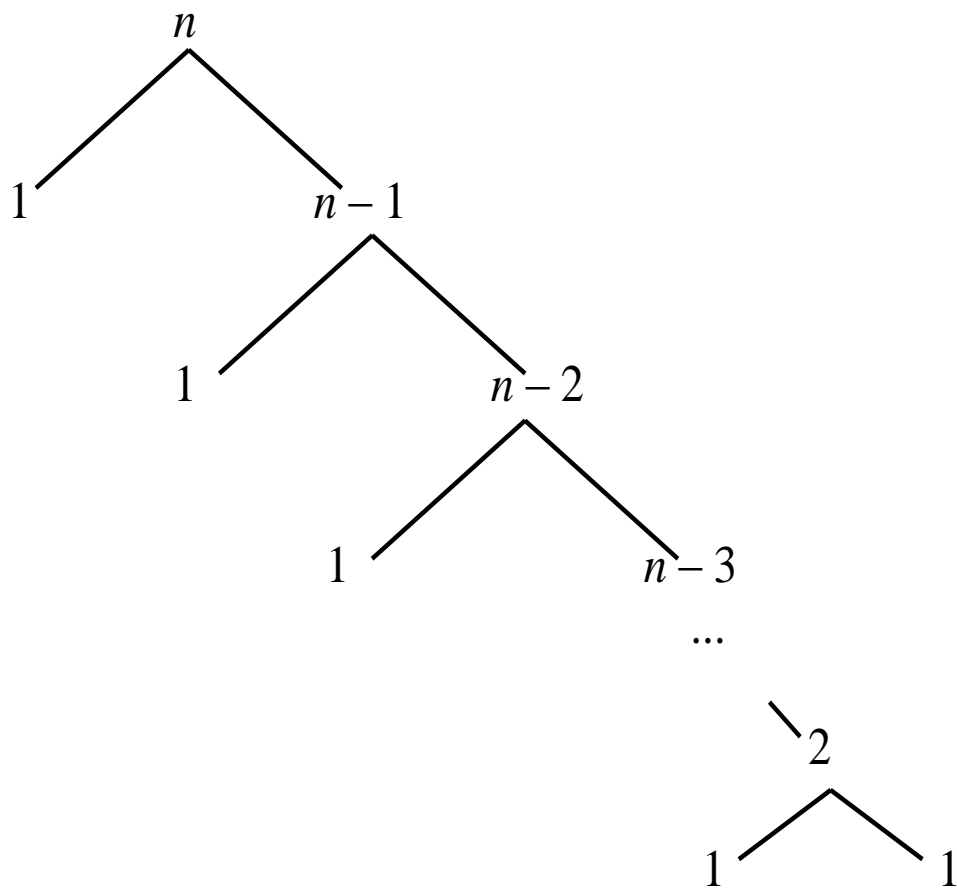
$$T(n) = \begin{cases} a & , n = 1 \\ 2T(n/2) + cn & , n > 1 \end{cases}$$

Penyelesaian (seperti pada *Merge Sort*):

$$T(n) = 2T(n/2) + cn = na + cn^2 \log n = O(n^2 \log n).$$

2. Kasus terburuk (*worst case*)

- Kasus ini terjadi bila pada setiap partisi *pivot* selalu elemen maksimum (atau elemen minimum) tabel.
- Kasus jika tabel sudah terurut menaik/menurun dan kita ingin terurut menurun/menaik.



Kompleksitas waktu pengurutan:

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

Penyelesaian (seperti pada *Insertion Sort*):

$$T(n) = T(n-1) + cn = O(n^2).$$

3. *Kasus rata-rata (average case)*

- Kasus ini terjadi jika *pivot* dipilih secara acak dari elemen tabel, dan peluang setiap elemen dipilih menjadi *pivot* adalah sama.
- $T_{\text{avg}}(n) = O(n^2 \log n)$.

(d) *Selection Sort*

```
procedure SelectionSort(input/output A : TabelInt, input i,j: integer)  
  
{ Mengurutkan tabel A[i..j] dengan algoritma Selection Sort.  
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.  
  Keluaran: Tabel A[i..j] yang terurut menaik.  
}
```

Algoritma:

```
  if i < j then { Ukuran(A) > 1 }  
    Bagi(A, i, j)  
    SelectionSort(A, i+1, j)  
  endif
```

```

procedure Bagi(input/output A : TabInt, input i,j: integer)

{ Mencari elemen terkecil di dalam tabel A[i..j], dan menempatkan
  elemen terkecil sebagai elemen pertama tabel.

  Masukan: A[i..j]
  Keluaran: A[i..j] dengan Ai adalah elemen terkecil.
}
Deklarasi
  idxmin, k, temp : integer

Algoritma:
  idxmin ← i
  for k ← i+1 to j do
    if Ak < Aidxmin then
      idxmin ← k
    endif
  endfor

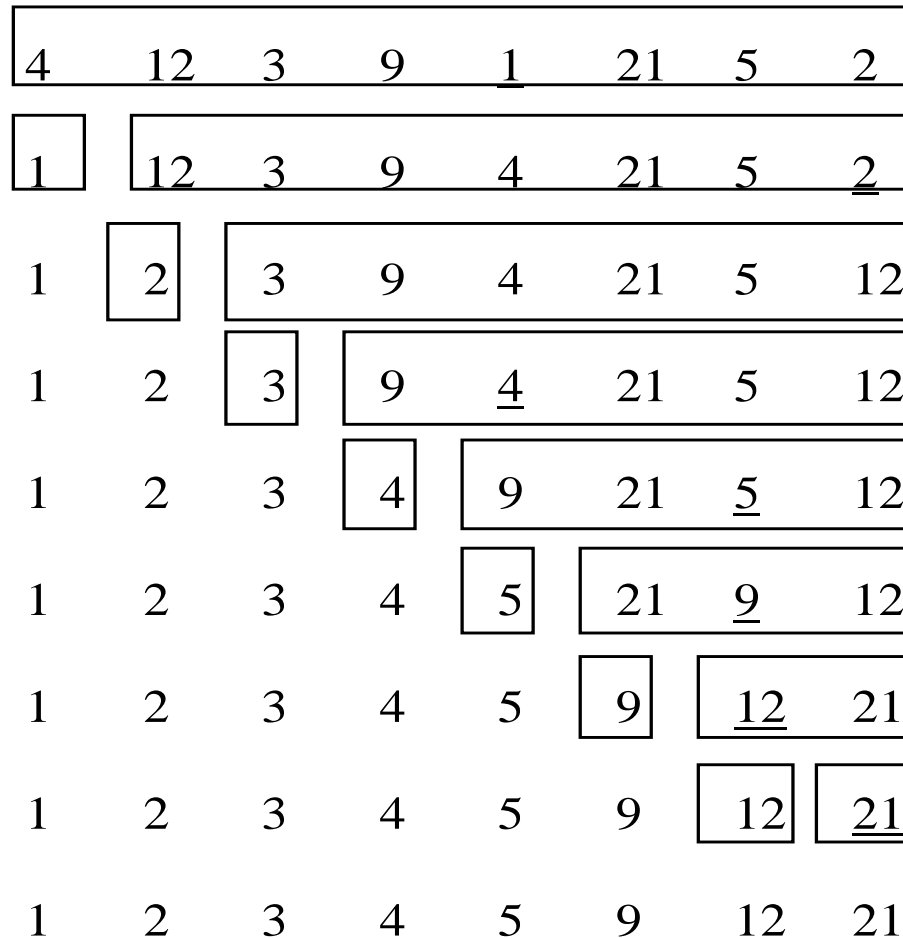
  { pertukarkan Ai dengan Aidxmin }
  temp ← Ai
  Ai ← Aidxmin
  Aidxmin ← temp

```

Contoh 4.5. Misalkan tabel *A* berisi elemen-elemen berikut:

4 12 3 9 1 21 5 2

Langkah-langkah pengurutan dengan *Selection Sort*:



Kompleksitas waktu algoritma:

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

Penyelesaian (seperti pada *Insertion Sort*):

$$T(n) = O(n^2).$$

Teorema Master

Misalkan $T(n)$ adalah fungsi menaik yang memenuhi relasi rekurens:

$$T(n) = aT(n/b) + cn^d$$

yang dalam hal ini $n = b^k$, $k = 1, 2, \dots$, $a \geq 1$, $b \geq 2$, dan c dan d adalah bilangan riil ≥ 0 , maka

$$T(n) \text{ adalah } \begin{cases} O(n^d) & \text{jika } a < b^d \\ O(n^d \log n) & \text{jika } a = b^d \\ O(n^{\log_b a}) & \text{jika } a > b^d \end{cases}$$

Contoh: Pada algoritma Mergesort/Quick Sort,

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

Menurut Teorema Master, $a = 2$, $b = 2$, $d = 1$, dan $a = b^d$, maka relasi rekurens:

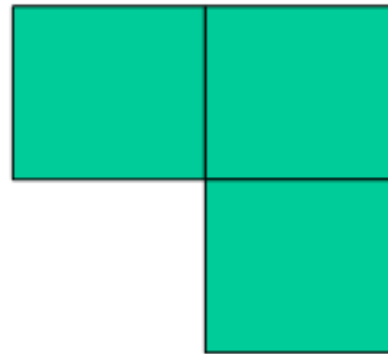
$$T(n) = 2T(n/2) + cn = O(n \log n)$$

Persoalan Pemasangan Ubin

Persoalan: Diberikan sebuah papan yang berukuran $2^k \times 2^k$. Tersedia sebuah ubin dan $2^{2k} - 1$ buah ubin yang terdiri dari kelompok 3-ubin berbentuk huruf L. Pasanglah semua ubin pada papan tersebut.



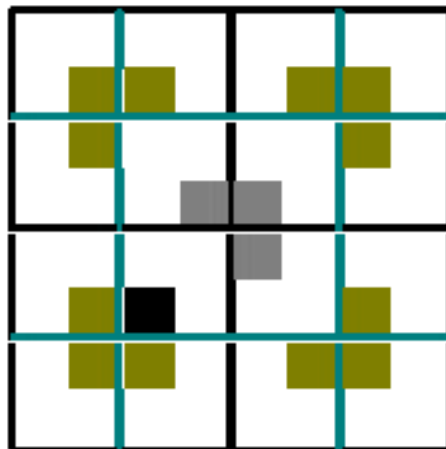
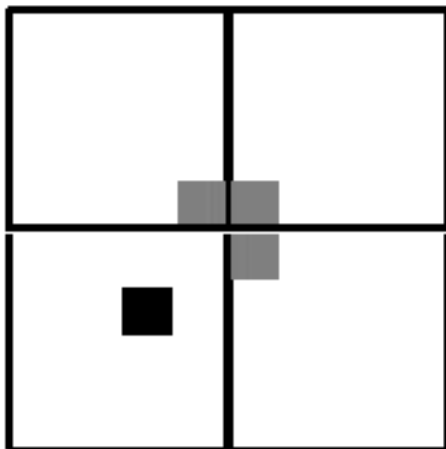
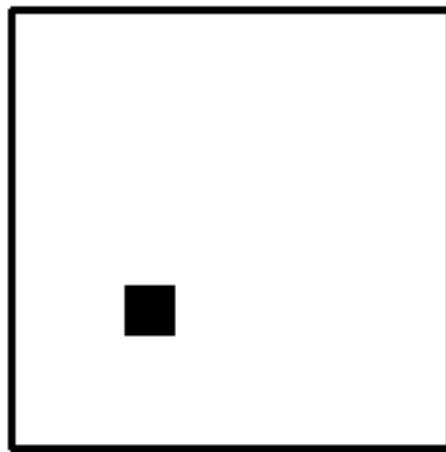
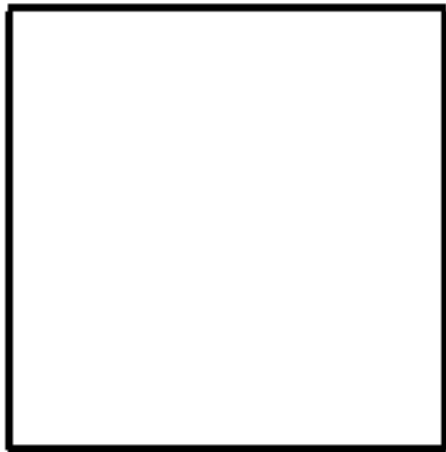
Ubin tunggal



Ubin berbentuk L (3-ubin)

Algoritma D & C:

- Bagi papan menjadi 4 bagian
- Ubin tunggal dapat ditaruh di mana saja.
- Tempatkan kelompok 3-ubin berbentuk L pada bagian tengah yang tidak ada ubin tunggal



Latihan

- (Soal UTS 2011) Misalkan anda mempunyai array $A[1..n]$ yang telah berisi n elemen *integer*. *Elemen mayoritas di dalam A adalah elemen yang terdapat pada lebih dari $n/2$ posisi* (jadi, jika $n = 6$ atau $n = 7$, elemen mayoritas terdapat pada paling sedikit 4 posisi). Rancanglah algoritma *divide and conquer* (tidak dalam bentuk *pseudo-code*, tapi dalam bentuk uraian deskriptif) untuk menemukan elemen mayoritas di dalam A (atau menentukan tidak terdapat elemen mayoritas). Jelaskan algoritma anda dengan contoh sebuah *array* berukuran 8 elemen. Selanjutnya, perkirakan kompleksitas algoritmanya dalam hubungan rekursif (misalnya $T(n) = bT(n/p) + h(n)$), lalu selesaikan $T(n)$ tersebut.

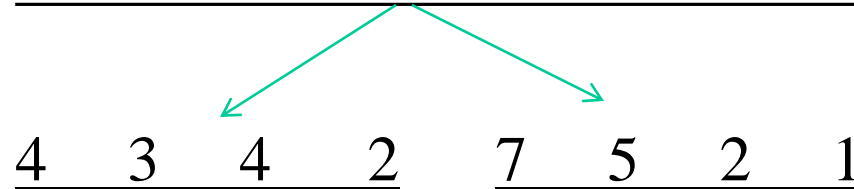
Solusi:

1. Jika $n = 1$, maka elemen tunggal tersebut adalah mayoritasnya sendiri.
2. Jika $n > 1$, maka bagi *array* menjadi dua bagian (kiri dan kanan) yang masing-masing berukuran sama ($n/2$).
3. Tahap *combine*. Ada empat kemungkinan kasus:

Kasus 1: tidak ada mayoritas pada setiap bagian, sehingga *array* gabungan keduanya tidak memiliki mayoritas.

Return: “no majority”

Contoh: 4 3 4 2 7 5 2 1



Ingat definisi
mayoritas!

no majority

no majority

4 3 4 2 7 5 2 1

“no majority”

Kasus 2: bagian kanan memiliki mayoritas, bagian kiri tidak. Pada *array* gabungan, hitung jumlah elemen yang sama dengan elemen mayoritas bagian kanan tersebut; Jika elemen tersebut mayoritas, *return* elemen tersebut, kalau tidak *return* “no majority”

Contoh: 4 3 4 2 7 4 4 4

Ingat definisi mayoritas!

4 3 4 2 7 4 4 4

no majority

majority = 4

4 3 4 2 7 4 4 4

Ingat definisi mayoritas!

Jumlah elemen 4 = 5 buah → mayoritas

“majority = 4”

Contoh lain (tidak ada mayoritas):

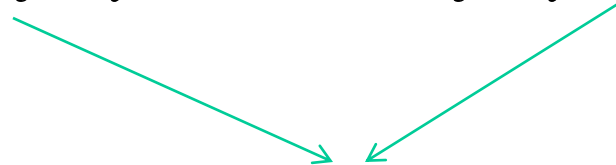
4 3 5 2 7 4 4 4



4 3 5 2 7 4 4 4

no majority

majority = 4



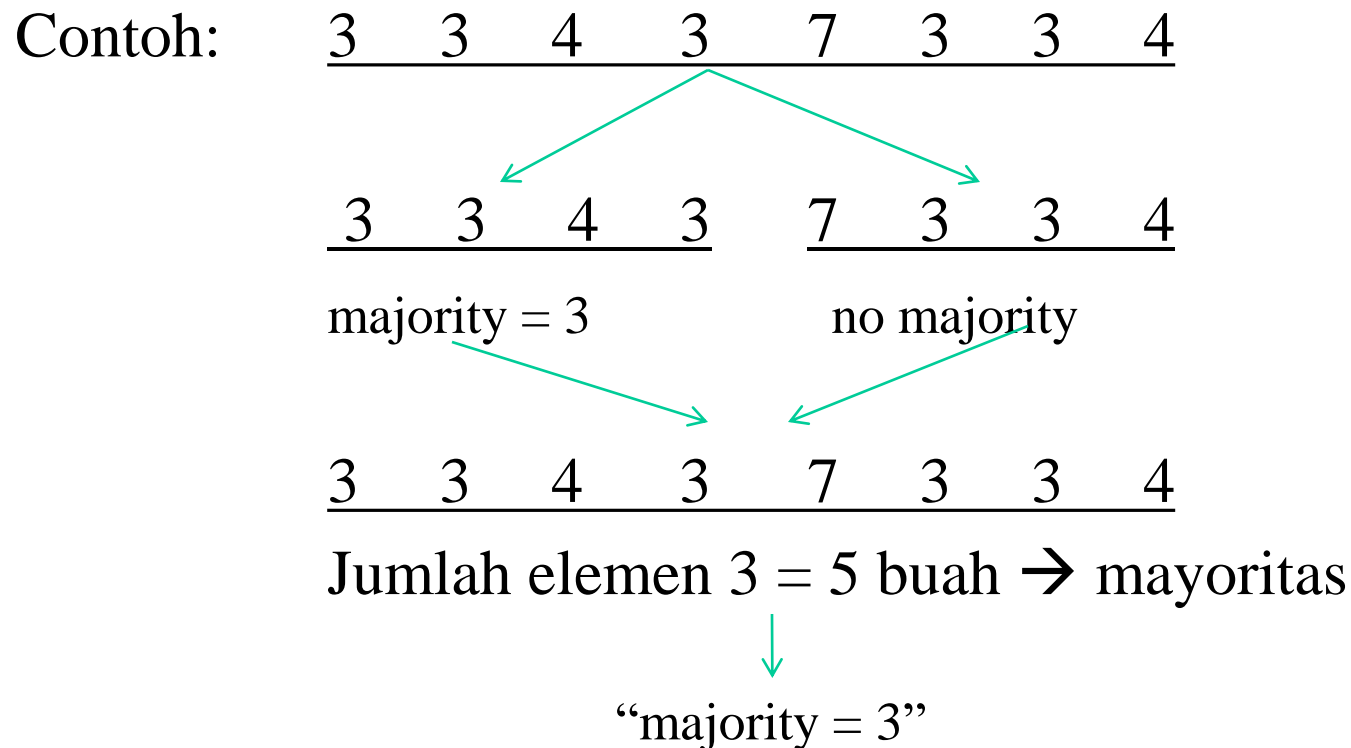
4 3 5 2 7 4 4 4

Jumlah elemen 4 = 4 buah → bukan mayoritas



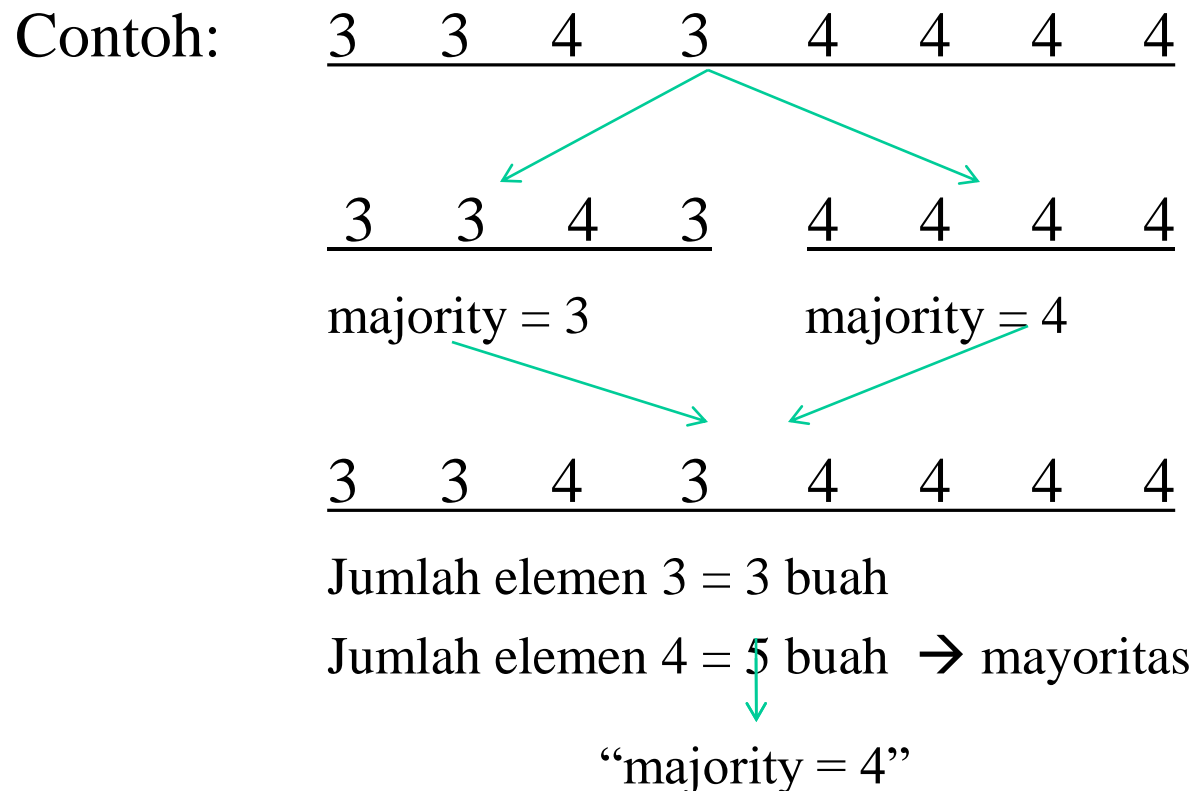
“no majority”

Kasus 3: bagian kiri memiliki mayoritas, bagian kanan tidak. Pada array gabungan, hitung jumlah elemen yang sama dengan elemen mayoritas bagian kiri tersebut. Jika elemen tersebut mayoritas, *return* elemen tersebut, kalau tidak *return* “no majority”



Kasus 4: bagian kiri dan bagian kanan memiliki mayoritas,
Pada *array* gabungan, hitung jumlah elemen yang sama
dengan kedua elemen kandidat mayoritas tersebut.

Jika salah satu kandidat adalah elemen mayoritas, *return*
elemen tersebut, kalau tidak *return* “no majority”



Contoh keseluruhan:

<u>4</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>5</u>	<u>4</u>	<u>3</u>	}	divide
<u>4</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>5</u>	<u>4</u>	<u>3</u>		
<u>4</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>5</u>	<u>4</u>	<u>3</u>		
<u>4</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>5</u>	<u>4</u>	<u>3</u>		
<u>4</u>	<u>3</u>	<u>4</u>	<u>4</u>	<u>4</u>	<u>5</u>	<u>4</u>	<u>3</u>	}	solve
m=4	m=3	m=4	m=4	m=4	m=5	m=4	m=3		

$\underline{4}$ $\underline{3}$ $\underline{4}$ $\underline{4}$ $\underline{4}$ $\underline{5}$ $\underline{4}$ $\underline{3}$
 m=4 m=3 m=4 m=4 m=4 m=5 m=4 m=3

$\underline{4 \quad 3}$ $\underline{4 \quad 4}$ $\underline{4 \quad 5}$ $\underline{4 \quad 3}$
 nm m = 4 nm nm

$\underline{4 \quad 3 \quad 4 \quad 4}$ $\underline{4 \quad 5 \quad 4 \quad 3}$
 m = 4 nm

$\underline{4 \quad 3 \quad 4 \quad 4 \quad 4 \quad 5 \quad 4 \quad 3}$
 m = 4

} combine

Kompleksitas waktu algoritma mayoritas:

$T(n)$ adalah jumlah operasi perbandingan yang terjadi (pada saat menghitung jumlah elemen yang sama dengan kandidat mayoritas)

Pada setiap level terdapat dua pemanggilan rekursif, masing-masing untuk $n/2$ elemen *array*.

Jumlah perbandingan yang terjadi paling banyak $2n$ (*upper bound*) yaitu pada kasus 4, untuk *array* berukuran n .

Secara umum jumlah perbandingan = cn .

Untuk $n = 1$, jumlah perbandingan = 0, secara umum = a .

Jadi,

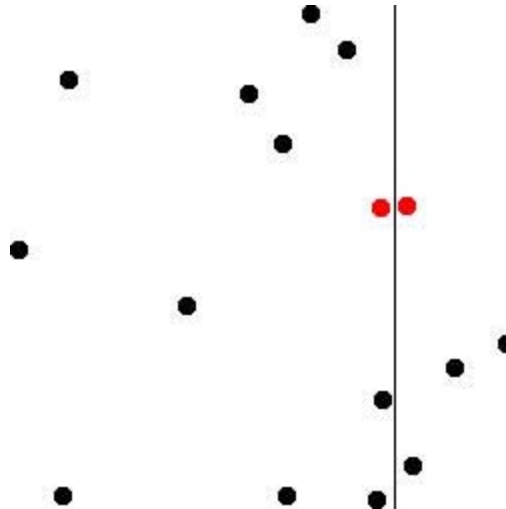
$$T(n) = \begin{cases} a & , n = 1 \\ 2T(n/2) + cn & , n > 1 \end{cases}$$

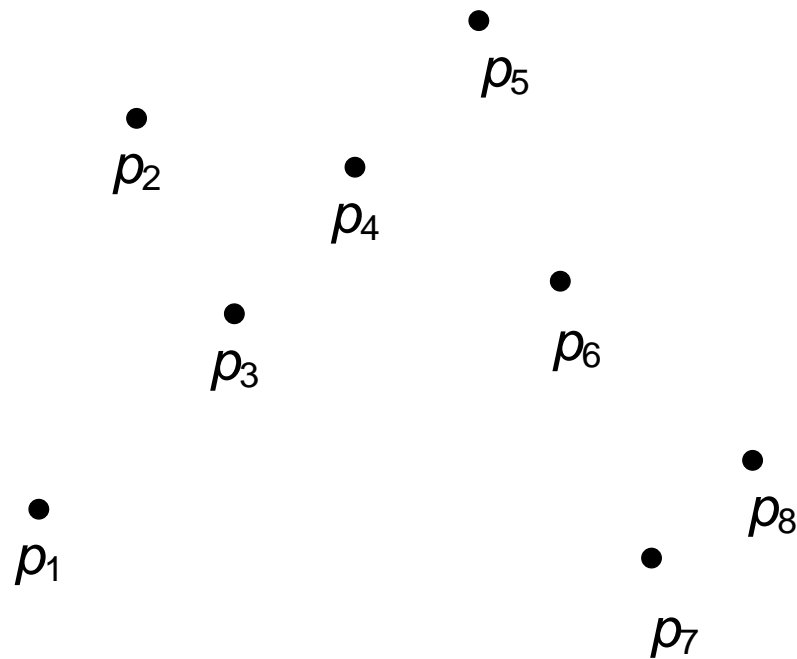
Menurut Teorema Master,

$$T(n) = 2T(n/2) + cn = O(n \log n)$$

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 sepasang titik di dalam P yang jaraknya terdekat satu sama lain.





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 secara *Brute Force*

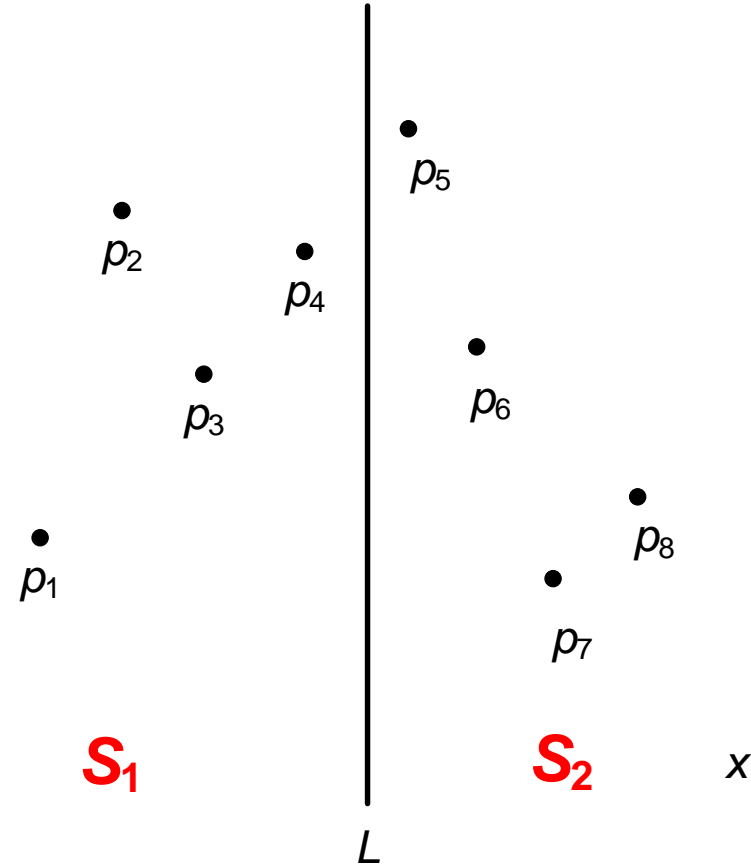
- Hitung jarak setiap pasang titik. Ada sebanyak $C(n, 2) = n(n - 1)/2$ pasangan titik
- Pilih pasangan titik yang mempunyai jarak terkecil.
- Kompleksitas algoritma adalah $O(n^2)$.

Penyelesaian secara *Divide and Conquer*

- Asumsi: $n = 2^k$ dan titik-titik sudah 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, S_1 dan S_2 , setiap bagian mempunyai jumlah titik^y yang sama. L adalah garis maya yang membagi dua himpunan titik ke dalam dua sub-himpunan, masing-masing $n/2$ titik.

Garis L dapat dihampiri sebagai $y = x_{n/2}$ dengan asumsi titik-titik diurut menaik berdasarkan absis.



3. CONQUER: Secara rekursif, terapkan algoritma *D-and-C* pada masing-masing bagian.
4. COMBINE: Pasangan titik yang jaraknya terdekat ada tiga kemungkinan letaknya:
 - (a) Pasangan titik terdekat terdapat di bagian S_1 .
 - (b) Pasangan titik terdekat terdapat di bagian S_2 .
 - (c) Pasangan titik terdekat dipisahkan oleh garis batas L , yaitu satu titik di S_1 dan satu titik di S_2 .

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

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

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

Deklarasi:

```
d1, d2 : real
```

Algoritma:

```
if n = 2 then
```

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

```
else
```

```
    S1 ← {p1, p2 , ..., pn/2 }
```

```
    S2 ← {pn/2+1, pn/2+2 , ..., pn }
```

```
    FindClosestPair2(S1, n/2, d1)
```

```
    FindClosestPair2(S2, n/2, d2)
```

```
    d ← MIN(d1,d2)
```

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

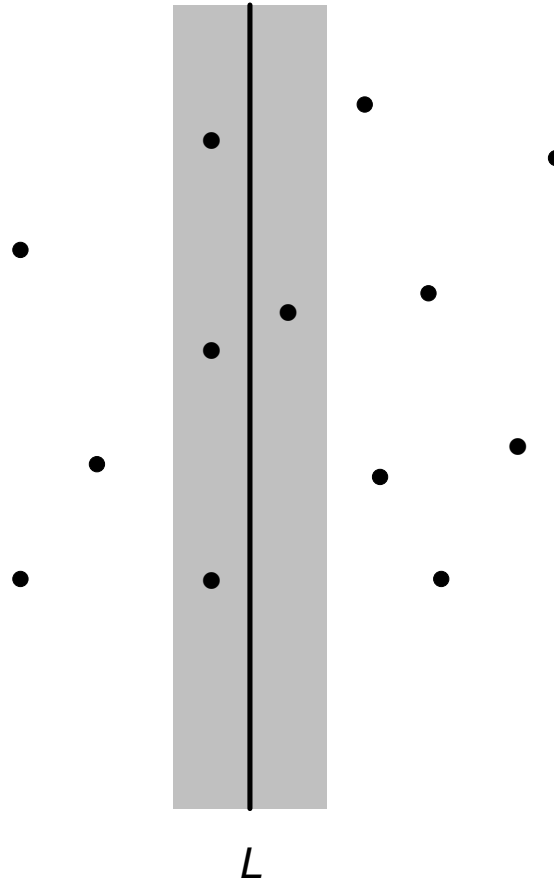
```
Tentukan apakah terdapat titik p1 di S1 dan pr di  
S2 dengan jarak(p1, pr) < d. Jika ada, set  
d dengan jarak terkecil tersebut.
```

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

```
endif
```

- Jika terdapat pasangan titik p_l and p_r yang jaraknya lebih kecil dari d , maka kasusnya adalah:
 - (i) Absis x dari p_l dan p_r berbeda paling banyak sebesar d .
 - (ii) Ordinat y dari p_l dan p_r berbeda paling banyak sebesar d .

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

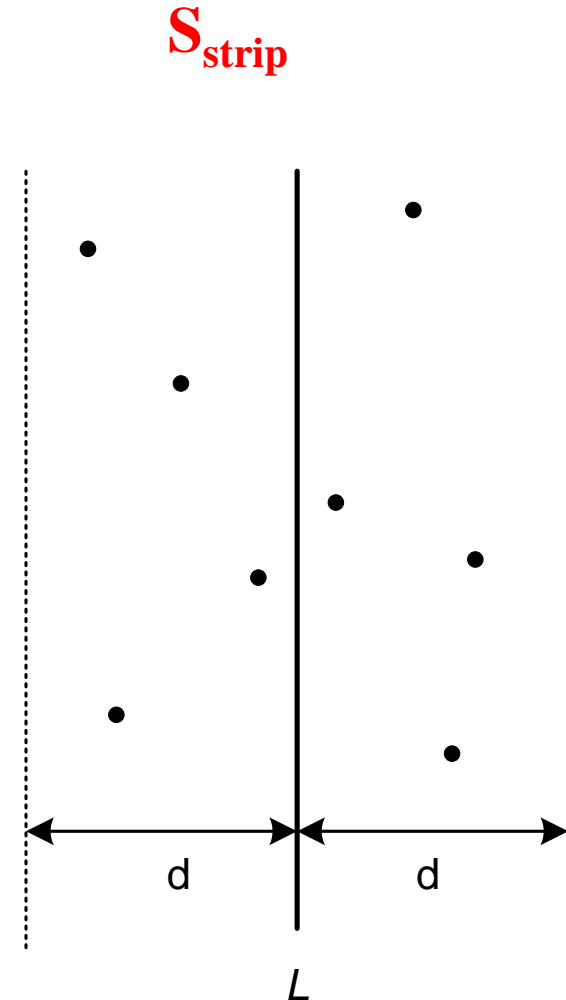


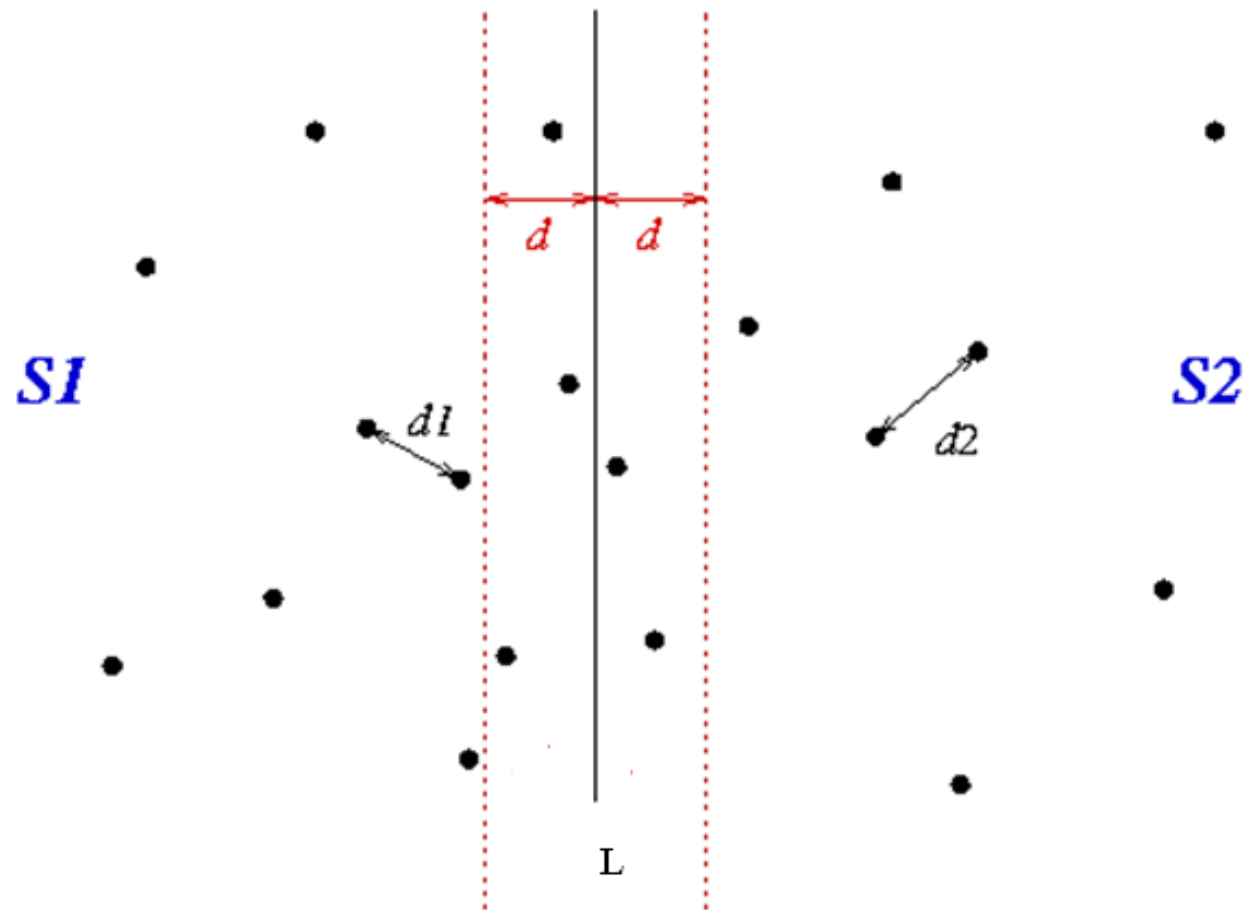
- Berapa lebar strip abu-abu tersebut?

- Kita membatasi titik-titik di dalam *strip* selebar $2d$
- Oleh karena itu, implementasi tahap ketiga adalah sbb:
 - Temukan semua titik di $S1_t$ yang memiliki absis x minimal $x_{n/2} - d$.
 - Temukan semua titik di $S2$ yang memiliki absis x maksimal $x_{n/2} + d$.

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

Urutkan titik-titik tersebut dalam urutan ordinat y yang menaik. Misalkan $q1, q2, \dots, qs$ menyatakan hasil pengurutan.



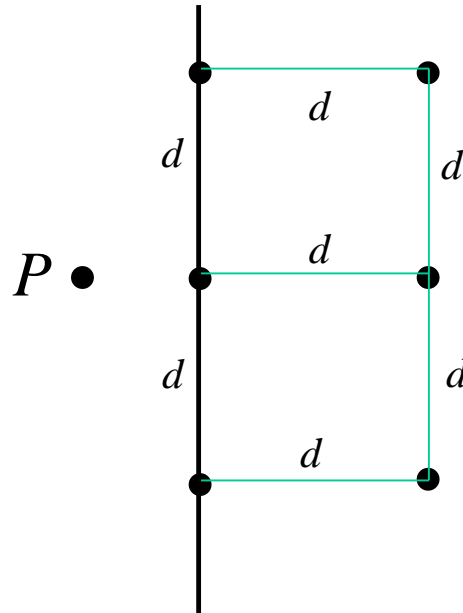


```

for i ← 1 to s do
  for j ← i+1 to s do
    if ( $|q_i.x - q_j.x| > d$  or  $|q_i.y - q_j.y| > d$  then
      tidak diproses
    else
       $d3 \leftarrow \text{EUCLIDEAN}(q_i, q_j)$ 
      if  $d3 < d$  then
         $d \leftarrow d3$ 
      endif
    endif
  endfor
endfor

```

- Jika diamati, kita tidak perlu memeriksa semua titik di dalam area strip abu-abu tersebut. Untuk sebuah titik P di sebelah kiri garis L , kita hanya perlu memeriksa paling banyak enam buah titik saja yang jaraknya sebesar d dari ordinat P (ke atas dan ke bawah), serta titik-titik yang berjarak d dari garis L .



- Pengurutan titik-titik dalam absis x dan ordinat y dilakukan sebelum menerapkan algoritma *Divide and Conquer*.
- Pemrosesan titik-titik di dalam S_{strip} memerlukan waktu $t(n) = cn = O(n)$.
- Kompleksitas algoritma:

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

Solusi dari persamaan di atas adalah $T(n) = O(n \log n)$, sesuai dengan Teorema Master

Perpangkatan a^n

Misalkan $a \in R$ dan n adalah bilangan bulat tidak negatif:

$$\begin{aligned} a^n &= a \times a \times \dots \times a \quad (n \text{ kali}), \text{ jika } n > 0 \\ &= 1 \quad \quad \quad , \text{ jika } n = 0 \end{aligned}$$

Penyelesaian dengan Algoritma Brute Force

```
function Exp1(input a, n : integer)→integer  
{ Menghitung  $a^n$ ,  $a > 0$  dan  $n$  bilangan bulat tak-negatif  
  Masukan: a, n  
  Keluaran: nilai perpangkatan.  
}
```

Deklarasi

```
k, hasil : integer
```

Algoritma:

```
hasil←1  
for k←1 to n do  
    hasil←hasil * a  
endfor  
  
return hasil
```

Kompleksitas waktu algoritma:

$$T(n) = n = O(n)$$

Penyelesaian dengan Divide and Conquer

Algoritma menghitung a^n :

1. Untuk kasus $n = 0$, maka $a^n = 1$.
2. Untuk kasus $n > 0$, bedakan menjadi dua kasus lagi:
 - (i) jika n genap, maka $a^n = a^{n/2} \cdot a^{n/2}$
 - (ii) jika n ganjil, maka $a^n = a^{n/2} \cdot a^{n/2} \cdot a$

Contoh 4.6. Menghitung 3^{16} dengan metode *Divide and Conquer*:

$$\begin{aligned} 3^{16} &= 3^8 \cdot 3^8 = (3^8)^2 \\ &= ((3^4)^2)^2 \\ &= (((3^2)^2)^2)^2 \\ &= (((3^1)^2))^2)^2)^2 \\ &= (((3^0)^2 \cdot 3)^2)^2)^2)^2 \\ &= (((1)^2 \cdot 3)^2)^2)^2)^2 \\ &= (((3)^2))^2)^2)^2 \\ &= ((9)^2)^2)^2 \\ &= (81)^2)^2 \\ &= (6561)^2 \\ &= 43046721 \end{aligned}$$

```
function Exp2(input a :real, n : integer) → real  
{ mengembalikan nilai  $a^n$ , dihitung dengan metode Divide and Conquer }
```

Algoritma:

```
if n = 0 then  
    return 1  
else  
    if odd(n) then    { fungsi ganjil }  
        return Exp2(a, n div 2) * Exp2(a, n div 2) * a  
    else  
        return Exp2(a, n div 2) * Exp2(a, n div 2)  
    endif  
endif
```

Tidak mangkus, karena ada dua kali pemanggilan rekursif untuk nilai parameter yang sama → Exp2(a, n div 2)

Perbaikan:

```
function Exp3(input a :real, n : integer) → real  
{ mengembalikan nilai  $a^n$ , dihitung dengan metode Divide and Conquer }
```

Algoritma:

```
if n = 0 then  
    return 1  
else  
    x ← Exp3(a, n div 2)  
    if odd(n) then { fungsi n ganjil }  
        return x * x * a  
    else  
        return x * x  
    endif  
endif
```

Kompleksitas algoritma:

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

Penyelesaian:

$$\begin{aligned} T(n) &= 1 + T(\lfloor n/2 \rfloor) \\ &= 1 + (1 + T(\lfloor n/4 \rfloor)) = 2 + T(\lfloor n/4 \rfloor) \\ &= 2 + (1 + T(\lfloor n/8 \rfloor)) = 3 + T(\lfloor n/8 \rfloor) \\ &= \dots \\ &= k + T(\lfloor n/2^k \rfloor) \end{aligned}$$

Persamaan terakhir diselesaikan dengan membuat $n/2^k = 1$,

$$\begin{aligned}(n/2^k) = 1 &\rightarrow \log (n/2^k) = \log 1 \\ \log n - \log 2^k &= 0 \\ \log n - k \log 2 &= 0 \\ \log n &= k \log 2 \\ k &= \log n / \log 2 = {}^2\log n\end{aligned}$$

sehingga

$$\begin{aligned}T(n) &= \lfloor {}^2\log n \rfloor + T(1) \\ &= \lfloor {}^2\log n \rfloor + 1 + T(0) \\ &= \lfloor {}^2\log n \rfloor + 1 + 0 \\ &= \lfloor {}^2\log n \rfloor + 1 \\ &= O({}^2\log n)\end{aligned}$$

Perkalian Matriks

- Misalkan A dan B dua buah matrik berukuran $n \times n$.
- Perkalian matriks: $C = A \times B$

Elemen-elemen hasilnya: $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$

Penyelesaian dengan Algoritma Brute Force

```
function KaliMatriks1(input A,B: Matriks, input n : integer)→ Matriks  
{ Memberikan hasil kali matriks A dan B yang berukuran  $n \times n$ .  
  Masukan: matriks integer A dan B, ukuran matriks (n)  
  Keluaran: matriks  $C = A \times B$ .  
}
```

Deklarasi

```
i, j, k : integer  
C : Matriks
```

Algoritma:

```
for i←1 to n do  
  for j←1 to n do  
     $C_{i,j} \leftarrow 0$     { inisialisasi penjumlah }  
    for k ← 1 to n do  
       $C_{i,j} \leftarrow C_{i,j} + A_{i,k} * B_{k,j}$   
    endfor  
  endfor  
endfor  
  
return C
```

Kompleksitas algoritma: $T(n) = n^3 + n^3 = O(n^3)$.

Penyelesaian dengan Algoritma Divide and Conquer

Matriks A dan B dibagi menjadi 4 buah matriks bujur sangkar. Masing-masing matriks bujur sangkar berukuran $n/2 \times n/2$:

$$\begin{array}{ccc} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} & \times & \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \\ \mathbf{A} & & \mathbf{B} \qquad \qquad \mathbf{C} \end{array}$$

Elemen-elemen matriks C adalah:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

Contoh 4.7. Misalkan matriks A adalah sebagai berikut:

$$A = \begin{bmatrix} 3 & 4 & 8 & 16 \\ 21 & 5 & 12 & 10 \\ 5 & 1 & 2 & 3 \\ 45 & 9 & 0 & -1 \end{bmatrix}$$

Matriks A dibagi menjadi 4 upa-matriks 2×2 :

$$A_{11} = \begin{bmatrix} 3 & 4 \\ 21 & 5 \end{bmatrix} \quad A_{12} = \begin{bmatrix} 8 & 16 \\ 12 & 10 \end{bmatrix} \quad A_{21} = \begin{bmatrix} 5 & 1 \\ 45 & 9 \end{bmatrix} \quad A_{22} = \begin{bmatrix} 2 & 3 \\ 0 & -1 \end{bmatrix}$$

```

function KaliMatriks2(input A,B: Matriks, input n : integer) → Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran  $n \times n$ .
  Masukan: matriks integer A dan B, ukuran matriks (n)
  Keluaran: matriks  $C = A \times B$ .
}

```

Deklarasi

```

i, j, k : integer
A11, A12, A21, A22,
B11, B12, B21, B22,
C11, C12, C21, C22 : Matriks

```

Algoritma:

```

  if n = 1 then
    return A × B    { perkalian biasa }
  else
    Bagi A menjadi A11, A12, A21, dan A22 yang masing-masing
    berukuran  $n/2 \times n/2$ 
    Bagi B menjadi B11, B12, B21, dan B22 yang masing-masing
    berukuran  $n/2 \times n/2$ 
    C11 ← KaliMatriks2(A11, B11, n/2) + KaliMatriks2(A12, B21, n/2)
    C12 ← KaliMatriks2(A11, B12, n/2) + KaliMatriks2(A12, B22, n/2)
    C21 ← KaliMatriks2(A21, B11, n/2) + KaliMatriks2(A22, B21, n/2)
    C22 ← KaliMatriks2(A21, B12, n/2) + KaliMatriks2(A22, B22, n/2)
    return C    { C adalah gabungan C11, C12, C13, C14 }
  endif

```

Pseudo-code algoritma penjumlahan (+), $C = A + B$:

```
function Tambah(input A, B : Matriks, input n : integer) → Matriks  
{ Memberikan hasil penjumlahan dua buah matriks, A dan B, yang  
  berukuran  $n \times n$ .  
  Masukan: matriks integer A dan B, ukuran matriks (n)  
  Keluaran: matriks  $C = A + B$   
}
```

Deklarasi

```
i, j, k : integer
```

Algoritma:

```
for i←1 to n do  
  for j←1 to n do  
     $C_{i,j} \leftarrow A_{i,j} + B_{i,j}$   
  endfor  
endfor  
return C
```

Kompleksitas waktu perkalian matriks seluruhnya adalah:

$$T(n) = \begin{cases} a & , n = 1 \\ 8T(n/2) + cn^2 & , n > 1 \end{cases}$$

yang bila diselesaikan, hasilnya adalah:

$$T(n) = O(n^3)$$

Hasil ini tidak memberi perbaikan kompleksitas dibandingkan dengan algoritma *brute force*.

Dapatkah kita membuat algoritma perkalian matriks yang lebih baik?

Algoritma Perkalian Matriks Strassen

Hitung matriks antara:

$$M1 = (A12 - A22)(B21 + B22)$$

$$M2 = (A11 + A22)(B11 + B22)$$

$$M3 = (A11 - A21)(B11 + B12)$$

$$M4 = (A11 + A12)B22$$

$$M5 = A11 (B12 - B22)$$

$$M6 = A22 (B21 - B11)$$

$$M7 = (A21 + A22)B11$$

maka,

$$C11 = M1 + M2 - M4 + M6$$

$$C12 = M4 + M5$$

$$C21 = M6 + M7$$

$$C22 = M2 - M3 + M5 - M7$$



- **Volker Strassen** (born April 29, 1936) is a German mathematician, a professor emeritus in the department of mathematics and statistics at the University of Konstanz.

- In 2008 he was awarded the Knuth Prize for "seminal and influential contributions to the design and analysis of efficient algorithms."[\[5\]](#)

Kompleksitas waktu algoritma perkalian matriks Strassen:

$$T(n) = \begin{cases} a & , n = 1 \\ 7T(n/2) + cn^2 & , n > 1 \end{cases}$$

yang bila diselesaikan, hasilnya adalah

$$T(n) = O(n^{\log 7}) = O(n^{2.81})$$

Perkalian Dua Buah Bilangan Bulat yang Besar

Persoalan: Misalkan bilangan bulat X dan Y
yang panjangnya n angka

$$X = x_1x_2x_3 \dots x_n$$

$$Y = y_1y_2y_3 \dots y_n$$

Hitunglah hasil kali X dengan Y .

Contoh 4.8. Misalkan,

$$X = 1234 \quad (n = 4)$$

$$Y = 5678 \quad (n = 4)$$

Cara klasik mengalikan X dan Y :

$$X \times Y = 1234$$

$$\begin{array}{r} \\ \underline{5678 \times} \\ 9872 \\ 8368 \\ 7404 \\ \underline{6170} \quad + \\ 7006652 \quad (7 \text{ angka}) \end{array}$$

Pseudo-code algoritma perkalian matriks:

```
function Kali1(input X, Y : LongInteger, n : integer) → LongInteger  
{ Mengalikan X dan Y, masing-masing panjangnya n digit dengan algoritma  
brute force.
```

Masukan: X dan Y yang panjangnya n angka

Keluaran: hasil perkalian

```
}
```

Deklarasi

```
temp, AngkaSatuan, AngkaPuluhan : integer
```

Algoritma:

```
for setiap angka  $y_i$  dari  $y_n, y_{n-1}, \dots, y_1$  do
```

```
AngkaPuluhan ← 0
```

```
for setiap angka  $x_j$  dari  $x_n, x_{n-1}, \dots, x_1$  do
```

```
temp ←  $x_j * y_i$ 
```

```
temp ← temp + AngkaPuluhan
```

```
AngkaSatuan ← temp mod 10
```

```
AngkaPuluhan ← temp div 10
```

```
tuliskan AngkaSatuan
```

```
endfor
```

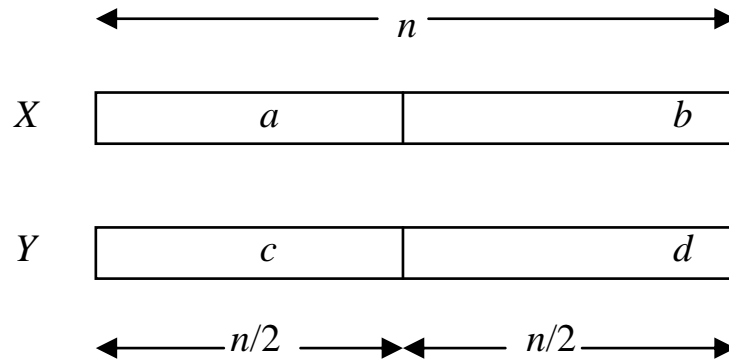
```
endfor
```

```
Z ← Jumlahkan semua hasil perkalian dari atas ke bawah
```

```
return Z
```

Kompleksitas algoritma: $O(n^2)$.

Penyelesaian dengan Algoritma Divide and Conquer



$$s = n \text{ \underline{div} } 2$$

$$a = X \text{ \underline{div} } 10^s$$

$$b = X \text{ \underline{mod} } 10^s$$

$$c = Y \text{ \underline{div} } 10^s$$

$$d = Y \text{ \underline{mod} } 10^s$$

X dan Y dapat dinyatakan dalam a , b , c , d , dan s sebagai

$$X = a \cdot 10^s + b$$

$$Y = c \cdot 10^s + d$$

Contoh,

$$X = 346769 = 346 \cdot 10^3 + 769$$

$$Y = 279431 = 279 \cdot 10^3 + 431$$

Perkalian X dengan Y dinyatakan sebagai

$$\begin{aligned} X \cdot Y &= (a \cdot 10^s + b) \cdot (c \cdot 10^s + d) \\ &= ac \cdot 10^{2s} + ad \cdot 10^s + bc \cdot 10^s + bd \\ &= ac \cdot 10^{2s} + (ad + bc) \cdot 10^s + bd \end{aligned}$$

Pseudo-code perkalian X dan Y :

```
function Kali2(input  $X, Y$  : LongInteger,  $n$  : integer)  $\rightarrow$  LongInteger
{ Mengalikan  $X$  dan  $Y$ , masing-masing panjangnya  $n$  digit dengan algoritma
  Divide and Conquer.
  Masukan:  $X$  dan  $Y$ 
  Keluaran: hasil perkalian  $X$  dan  $Y$ 
}
Deklarasi
   $a, b, c, d$  : LongInteger
   $s$  : integer

Algoritma:
  if  $n = 1$  then
    return  $X * Y$       { perkalian biasa }
  else
     $s \leftarrow n \text{ div } 2$       { bagidua pada posisi  $s$  }
     $a \leftarrow X \text{ div } 10^s$ 
     $b \leftarrow X \text{ mod } 10^s$ 
     $c \leftarrow Y \text{ div } 10^s$ 
     $d \leftarrow Y \text{ mod } 10^s$ 
    return  $Kali2(a, c, s) * 10^{2s} + Kali2(b, c, s) * 10^s +$ 
       $Kali2(a, d, s) * 10^s + Kali2(b, d, s)$ 
  endif
```

Kompleksitas waktu algoritma:

$$T(n) = \begin{cases} a & , n = 1 \\ 4T(n/2) + cn & , n > 1 \end{cases}$$

- Penyelesaian:

$$T(n) = O(n^2).$$

- Ternyata, perkalian dengan algoritma *Divide and Conquer* seperti di atas belum memperbaiki kompleksitas waktu algoritma perkalian secara *brute force*.
- Adakah algoritma perkalian yang lebih baik?

Perbaikan (A.A Karatsuba, 1962):

Misalkan

$$r = (a + b)(c + d) = ac + (ad + bc) + bd$$

maka,

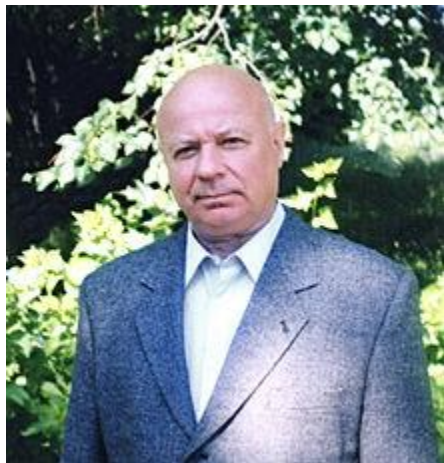
$$(ad + bc) = r - ac - bd = (a + b)(c + d) - ac - bd$$

Dengan demikian, perkalian X dan Y dimanipulasi menjadi

$$X \cdot Y = ac \cdot 10^{2s} + (ad + bc) \cdot 10^s + bd$$

$$= \underbrace{ac}_p \cdot 10^{2s} + \underbrace{\{(a + b)(c + d) - \underbrace{ac}_p - \underbrace{bd}_q\}}_r \cdot 10^s + \underbrace{bd}_q$$

Anatolii Alexevich Karatsuba



Anatolii Alexeevitch Karatsuba (Russian: Анато́лий Алексе́евич Карацúба; Grozny, January 31, 1937 — Moscow, September 28, 2008) was a Russian mathematician, who authored the first fast multiplication method: the Karatsuba algorithm, a fast procedure for multiplying large numbers. (Sumber: Wikipedia)

```

function Kali3(input X, Y : LongInteger, n : integer) → LongInteger
{ Mengalikan X dan Y, masing-masing panjangnya n digit dengan algoritma
Divide and Conquer.
  Masukan: X dan Y
  Keluaran: hasil perkalian X dan Y
}

```

Deklarasi

```

a, b, c, d : LongInteger
s : integer

```

Algoritma:

```

if n = 1 then
  return X * Y      { perkalian biasa }
else
  s ← n div 2        { bagidua pada posisi s }
  a ← X div 10s
  b ← X mod 10s
  c ← Y div 10s
  d ← Y mod 10s
  p ← Kali3(a, c, s)
  q ← Kali3(b, d, s)
  r ← Kali3(a + b, c + d, s)
  return p*102s + (r - p - q)*10s + q

```

```

endif

```

Kompleksitas waktu algoritmanya:

$T(n)$ = waktu perkalian integer yang berukuran $n/2$ +
waktu untuk perkalian dengan 10^s dan 10^{2s} dan waktu
untuk penjumlahan

$$T(n) = \begin{cases} a & , n = 1 \\ 3T(n/2) + cn & , n > 1 \end{cases}$$

Bila relasi rekurens diselesaikan, diperoleh $T(n) = O(n^{\log_2 3}) = O(n^{1.59})$, lebih baik daripada kompleksitas waktu dua algoritma perkalian sebelumnya.

Lecture 3: The Polynomial Multiplication Problem

A More General Divide-and-Conquer Approach

Divide: Divide a given problem into subproblems (ideally of approximately equal size).

No longer only TWO subproblems

Conquer: Solve each subproblem (directly or recursively), and

Combine: Combine the solutions of the subproblems into a global solution.

The Polynomial Multiplication Problem

another divide-and-conquer algorithm

Problem:

Given two polynomials of degree n

$$A(x) = a_0 + a_1x + \cdots + a_nx^n$$

$$B(x) = b_0 + b_1x + \cdots + b_nx^n,$$

compute the product $A(x)B(x)$.

Example:

$$A(x) = 1 + 2x + 3x^2$$

$$B(x) = 3 + 2x + 2x^2$$

$$A(x)B(x) = 3 + 8x + 15x^2 + 10x^3 + 6x^4$$

Question: How can we efficiently calculate the coefficients of $A(x)B(x)$?

Assume that the coefficients a_i and b_i are stored in arrays $A[0 \dots n]$ and $B[0 \dots n]$.

Cost of any algorithm is number of scalar multiplications and additions performed.

Convolutions

Let $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^m b_i x^i$.

Set $C(x) = \sum_{k=0}^{n+m} c_k x^k = A(x)B(x)$.

Then

$$c_k = \sum_{i=0}^k a_i b_{k-i}$$

for all $0 \leq k \leq m + n$.

Definition: The vector $(c_0, c_1, \dots, c_{m+n})$
is the **convolution** of the vectors
 (a_0, a_1, \dots, a_n) and (b_0, b_1, \dots, b_m) .

Calculating convolutions (and thus polynomial multiplication) is a major problem in digital signal processing.

Convolutions

Let $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^m b_i x^i$.

Set $C(x) = \sum_{k=0}^{n+m} c_k x^k = A(x)B(x)$.

Then

$$c_k = \sum_{i=0}^k a_i b_{k-i}$$

for all $0 \leq k \leq m+n$.

Definition: The vector $(c_0, c_1, \dots, c_{m+n})$ is the [convolution](#) of the vectors (a_0, a_1, \dots, a_n) and (b_0, b_1, \dots, b_m) .

Calculating convolutions (and thus polynomial multiplication) is a major problem in digital signal processing.

The Direct (Brute Force) Approach

Let $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^n b_i x^i$.

Set $C(x) = \sum_{k=0}^{2n} c_k x^k = A(x)B(x)$ with

$$c_k = \sum_{i=0}^k a_i b_{k-i}$$

for all $0 \leq k \leq 2n$.

The direct approach is to compute all c_k using the formula above. The total number of multiplications and additions needed are $\Theta(n^2)$ and $\Theta(n^2)$ respectively. Hence the complexity is $\Theta(n^2)$.

Questions: Can we do better?

Can we apply the divide-and-conquer approach to develop an algorithm?

The Divide-and-Conquer Approach

The Divide Step: Define

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1}x^{\lfloor \frac{n}{2} \rfloor - 1},$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_nx^{n - \lfloor \frac{n}{2} \rfloor}.$$

Then $A(x) = A_0(x) + A_1(x)x^{\lfloor \frac{n}{2} \rfloor}$.

Similarly we define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x)x^{\lfloor \frac{n}{2} \rfloor}.$$

Then

$$\begin{aligned} A(x)B(x) = & A_0(x)B_0(x) + A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} + \\ & A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} + A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}. \end{aligned}$$

Remark: The original problem of size n is divided into 4 problems of input size $\frac{n}{2}$.

Example:

$$\begin{aligned}A(x) &= 2 + 5x + 3x^2 + x^3 - x^4 \\B(x) &= 1 + 2x + 2x^2 + 3x^3 + 6x^4 \\A(x)B(x) &= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 \\&\quad + 19x^6 + 3x^7 - 6x^8\end{aligned}$$

$$\begin{aligned}A_0(x) &= 2 + 5x, & A_1(x) &= 3 + x - x^2, \\A(x) &= A_0(x) + A_1(x)x^2 \\B_0(x) &= 1 + 2x, & B_1(x) &= 2 + 3x + 6x^2, \\B(x) &= B_0(x) + B_1(x)x^2\end{aligned}$$

$$\begin{aligned}A_0(x)B_0(x) &= 2 + 9x + 10x^2 \\A_1(x)B_1(x) &= 6 + 11x + 19x^2 + 3x^3 - 6x^4 \\A_0(x)B_1(x) &= 4 + 16x + 27x^2 + 30x^3 \\A_1(x)B_0(x) &= 3 + 7x + x^2 - 2x^3 \\A_0(x)B_1(x) + A_1(x)B_0(x) &= 7 + 23x + 28x^2 + 28x^3\end{aligned}$$

$$\begin{aligned}&A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4 \\&= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 + 19x^6 + 3x^7 - 6x^8\end{aligned}$$

The Divide-and-Conquer Approach

The Conquer Step: Solve the four subproblems, i.e., computing

$$\begin{aligned} &A_0(x)B_0(x), \quad A_0(x)B_1(x), \\ &A_1(x)B_0(x), \quad A_1(x)B_1(x) \end{aligned}$$

by recursively calling the algorithm 4 times.

The Divide-and-Conquer Approach

The Combining Step: Adding the following four polynomials

$$\begin{aligned} &A_0(x)B_0(x) \\ &+ A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} \\ &+ A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} \\ &+ A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}. \end{aligned}$$

takes $\Theta(n)$ operations. Why?

The First Divide-and-Conquer Algorithm

PolyMulti1($A(x), B(x)$)

{

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1}x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_nx^{n - \lfloor \frac{n}{2} \rfloor};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1}x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + b_nx^{n - \lfloor \frac{n}{2} \rfloor};$$

$$U(x) = \text{PolyMulti1}(A_0(x), B_0(x));$$

$$V(x) = \text{PolyMulti1}(A_0(x), B_1(x));$$

$$W(x) = \text{PolyMulti1}(A_1(x), B_0(x));$$

$$Z(x) = \text{PolyMulti1}(A_1(x), B_1(x));$$

$$\text{return } \left(U(x) + [V(x) + W(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x)x^{2\lfloor \frac{n}{2} \rfloor} \right)$$

}

Running Time of the Algorithm

Assume n is a power of 2, $n = 2^h$. By substitution (expansion),

$$\begin{aligned}T(n) &= 4T\left(\frac{n}{2}\right) + cn \\&= 4\left[4T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\&= 4^2T\left(\frac{n}{2^2}\right) + (1+2)cn \\&= 4^2\left[4T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + (1+2)cn \\&= 4^3T\left(\frac{n}{2^3}\right) + (1+2+2^2)cn \\&\vdots \\&= 4^iT\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j cn \quad (\text{induction}) \\&\vdots \\&= 4^hT\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} 2^j cn \\&= n^2T(1) + cn(n-1) \\&\quad (\text{since } n = 2^h \text{ and } \sum_{j=0}^{h-1} 2^j = 2^h - 1 = n - 1) \\&= \Theta(n^2).\end{aligned}$$

The same order as the brute force approach!

Comments on the Divide-and-Conquer Algorithm

Comments: The divide-and-conquer approach makes no essential improvement over the brute force approach!

Question: Why does this happen.

Question: Can you improve this divide-and-conquer algorithm?

Problem: Given 4 numbers

$$A_0, A_1, B_0, B_1$$

how many multiplications are needed to calculate the three values

$$A_0B_0, A_0B_1 + A_1B_0, A_1B_1?$$

This can obviously be done using 4 multiplications but there is a way of doing this using only the following 3:

$$Y = (A_0 + A_1)(B_0 + B_1)$$

$$U = A_0B_0$$

$$Z = A_1B_1$$

U and Z are what we originally wanted and

$$A_0B_1 + A_1B_0 = Y - U - Z.$$

Improving the Divide-and-Conquer Algorithm

Define

$$Y(x) = (A_0(x) + A_1(x)) \times (B_0(x) + B_1(x))$$

$$U(x) = A_0(x)B_0(x)$$

$$Z(x) = A_1(x)B_1(x)$$

Then

$$Y(x) - U(x) - Z(x) = A_0(x)B_1(x) + A_1(x)B_0(x).$$

Hence $A(x)B(x)$ is equal to

$$U(x) + [Y(x) - U(x) - Z(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x) \times x^{2\lfloor \frac{n}{2} \rfloor}$$

Conclusion: You need to call the multiplication procedure 3, rather than 4 times.

The Second Divide-and-Conquer Algorithm

PolyMulti2($A(x), B(x)$)

{

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1}x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_nx^{n - \lfloor \frac{n}{2} \rfloor};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1}x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + b_nx^{n - \lfloor \frac{n}{2} \rfloor};$$

$$Y(x) = \text{PolyMulti2}(A_0(x) + A_1(x), B_0(x) + B_1(x))$$

$$U(x) = \text{PolyMulti2}(A_0(x), B_0(x));$$

$$Z(x) = \text{PolyMulti2}(A_1(x), B_1(x));$$

$$\text{return } \left(U(x) + [Y(x) - U(x) - Z(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x)x^{2\lfloor \frac{n}{2} \rfloor} \right);$$

}

Running Time of the Modified Algorithm

Assume $n = 2^h$. Let $\lg x$ denote $\log_2 x$.
By the substitution method,

$$\begin{aligned}
 T(n) &= 3T\left(\frac{n}{2}\right) + cn \\
 &= 3\left[3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\
 &= 3^2T\left(\frac{n}{2^2}\right) + \left(1 + \frac{3}{2}\right)cn \\
 &= 3^2\left[3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + \left(1 + \frac{3}{2}\right)cn \\
 &= 3^3T\left(\frac{n}{2^3}\right) + \left(1 + \frac{3}{2} + \left[\frac{3}{2}\right]^2\right)cn \\
 &\vdots \\
 &= 3^hT\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} \left[\frac{3}{2}\right]^j cn.
 \end{aligned}$$

We have

$$3^h = (2^{\lg 3})^h = 2^{h \lg 3} = (2^h)^{\lg 3} = n^{\lg 3} \approx n^{1.585},$$

and

$$\sum_{j=0}^{h-1} \left[\frac{3}{2}\right]^j = \frac{(3/2)^h - 1}{3/2 - 1} = 2 \cdot \frac{3^h}{2^h} - 2 = 2n^{\lg 3 - 1} - 2.$$

Hence

$$T(n) = \Theta(n^{\lg 3}T(1) + 2cn^{\lg 3}) = \Theta(n^{\lg 3}).$$

Comments

- The divide-and-conquer approach doesn't always give you the best solution.
Our original D-A-C algorithm was just as bad as brute force.
- There is actually an $O(n \log n)$ solution to the polynomial multiplication problem.
It involves using the *Fast Fourier Transform* algorithm as a subroutine.
The FFT is another classic D-A-C algorithm (Chapt 30 in CLRS gives details).
- The idea of using 3 multiplications instead of 4 is used in large-integer multiplications.
A similar idea is the basis of the classic [Strassen matrix multiplication algorithm](#) (CLRS, Chapter 28).