

INF218

Struktur Data & Algoritma

Advanced Sorting:
Divide & Conquer, Merge Sort

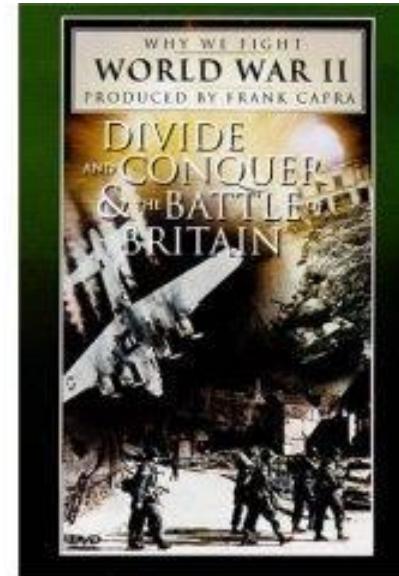
Alim Misbullah, S.Si., M.S.

The background of the image features a dynamic, abstract design. It consists of several overlapping, translucent layers. The top layer is a bright orange color, with darker orange and white splatters and streaks suggesting liquid being poured or splashed. Below this is a dark brown layer, followed by a deep blue layer at the bottom. The overall effect is one of motion and energy, with the colors blending and interacting with each other.

Divide & Conquer

Definisi Divide & Conquer

- Algoritma Divide & Conquer dulunya adalah strategi militer dengan nama *Divide et Imperes*

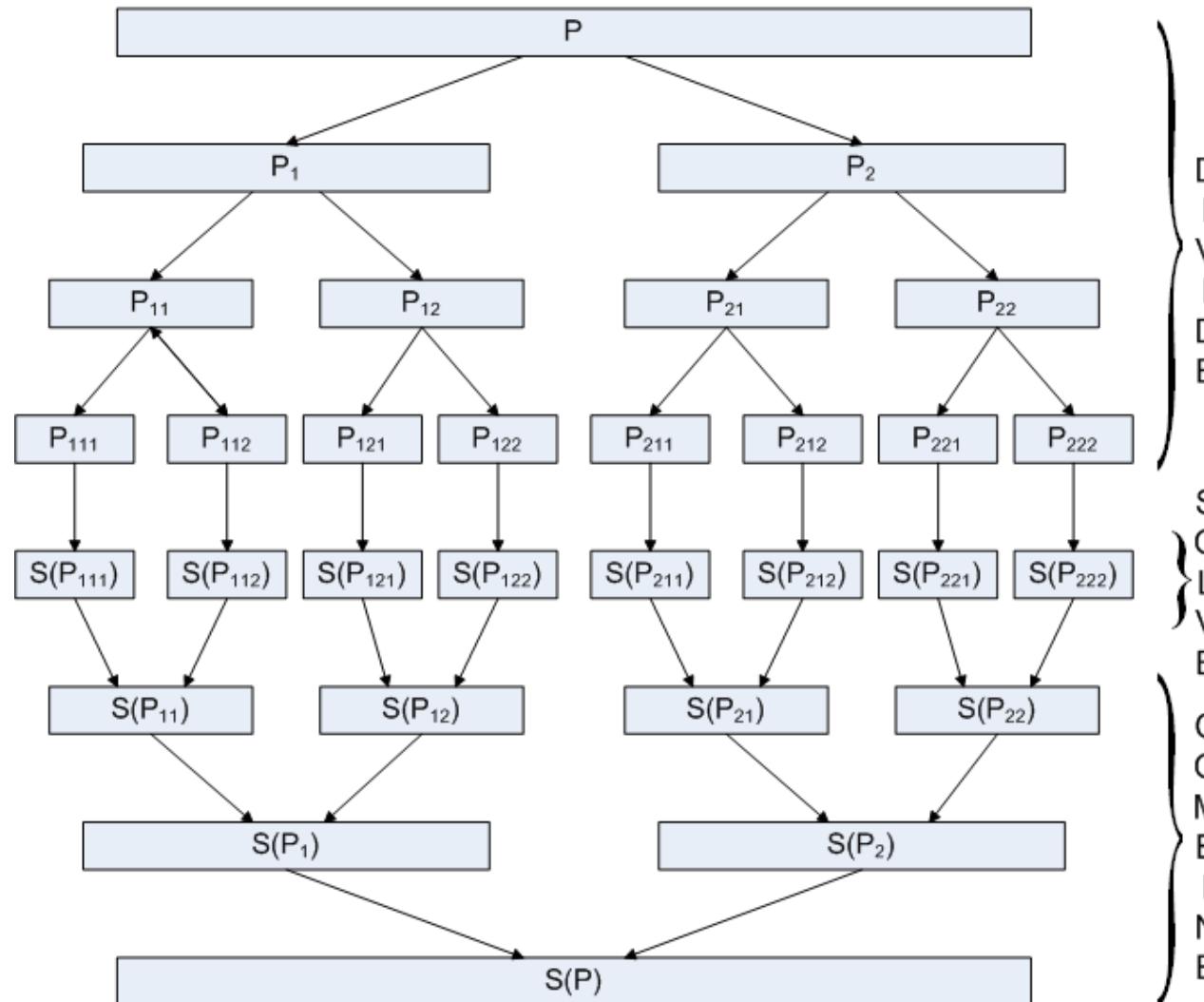


- Saat ini, strategi tersebut menjadi salah satu strategi fundamental di dalam ilmu komputer dengan nama *Divide and Conquer*

Definisi Divide & Conquer

- *Divide*: membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya berukuran hampir sama)
- *Conquer (Solve)*: menyelesaikan masing-masing sub-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar)
- *Combine*: menggabungkan solusi masing-masing sub-persoalan sehingga membentuk solusi persoalan semula

Skema Kerja Divide & Conquer



Keterangan:

P = persoalan
S = solusi

Persoalan Divide & Conquer

- Objek persoalan yang dibagi adalah masukan (input) atau instance persoalan yang berukuran n seperti:
 - Tabel (larik)
 - Matriks
 - Eksponen
 - Polinom
 - Dll, bergantung pada persoalannya
- Tiap-tiap sub-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakter persoalan semula
- Sehingga metode *Divide & Conquer* lebih cocok disajikan dalam skema rekursif

Skema Umum Algoritma Divide & Conquer

procedure DIVIDEandCONQUER(**input** P : problem, n : **integer**)
{ Menyelesaikan persoalan P dengan algoritma divide and conquer
Masukan: masukan persoalan P berukuran n
Luaran: solusi dari persoalan semula }

Deklarasi

r : **integer**

Algoritma

if $n \leq n_0$ then {ukuran persoalan P sudah cukup kecil}

SOLVE persoalan P yang berukuran n ini

else

DIVIDE menjadi r upa-persoalan, P_1, P_2, \dots, P_r , yang masing-masing berukuran n_1, n_2, \dots, n_r

for masing-masing P_1, P_2, \dots, P_r , **do**

DIVIDEandCONQUER(P_i, n_i)

endfor

COMBINE solusi dari P_1, P_2, \dots, P_r menjadi solusi persoalan semula

endif

Kompleksitas algoritma *divide and conquer*: $T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$

Skema Umum Algoritma Divide & Conquer

- Penjelasan:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

- $T(n)$: kompleksitas waktu penyelesaian persoalan P yang berukuran n
- $g(n)$: kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil
- $T(n_1) + T(n_2) + \dots + T(n_r)$: kompleksitas waktu untuk memproses setiap sub-persoalan
- $f(n)$: kompleksitas waktu untuk COMBINE solusi dari masing-masing sub-persoalan

Skema Umum Algoritma Divide & Conquer

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

procedure DIVIDEandCONQUER(**input** P : problem, n : **integer**)

{ Menyelesaikan persoalan dengan algoritma divide and conquer

Masukan: masukan yang berukuran n

Luaran: solusi dari persoalan semula

}

Deklarasi

r : **integer**

Algoritma

if $n \leq n_0$ then {ukuran persoalan sudah cukup kecil }

SOLVE persoalan P yang berukuran n ini

else

DIVIDE menjadi 2 upa-persoalan, P_1 dan P_2 , masing-masing berukuran $n/2$

DIVIDEandCONQUER(P_1 , $n/2$)

DIVIDEandCONQUER(P_2 , $n/2$)

COMBINE solusi dari P_1 dan P_2

endif

Kompleksitas algoritma *divide and conquer*: $T(n) = \begin{cases} g(n) & , n \leq n_0 \\ 2T(n/2) + f(n) & , n > n_0 \end{cases}$

Persoalan dengan solusi Divide & Conquer

- Beberapa persoalan yang diselesaikan dengan Divide & Conquer
 - Persoalan MinMaks(mencari nilai minimum dan nilai maksimum)
 - Menghitung perpangkatan
 - Persoalan pengurutan(*sorting*) – Merge Sort dan Quick Sort
 - Mencari sepasang titik terdekat(*closest pair problem*)
 - *Convex Hull*
 - Perkalian matriks
 - Perkalian bilangan bulat besar
 - Perkalian dua buah polinom

Persoalan MinMaks (Nilai Maksimum dan Minimum)

- Persoalan: Misalkan diberikan sebuah larik A yang berukuran n elemen dan sudah berisi nilai *integer*
- Carilah nilai minimum (min) dan nilai maksimum (max) sekaligus di dalam larik tersebut

- Contoh:

4	12	23	9	21	1	35	2	24
---	----	----	---	----	---	----	---	----
- $\text{Min} = 1$ dan $\text{Max} = 35$

Persoalan MinMaks (Nilai Maksimum dan Minimum)

- Penyelesaian dengan Algoritma Brute Force

```
procedure MinMaks1(input A : TabelInteger, n : integer, output min, maks : integer)
{ Mencari nilai minimum dan maksimum di dalam larik a yang berukuran n elemen, secara brute force.
Masukan: larik a yang sudah terdefinisi elemen-elemennya
Luaran: nilai maksimum dan nilai minimum tabel
}
Deklarasi
i : integer
```

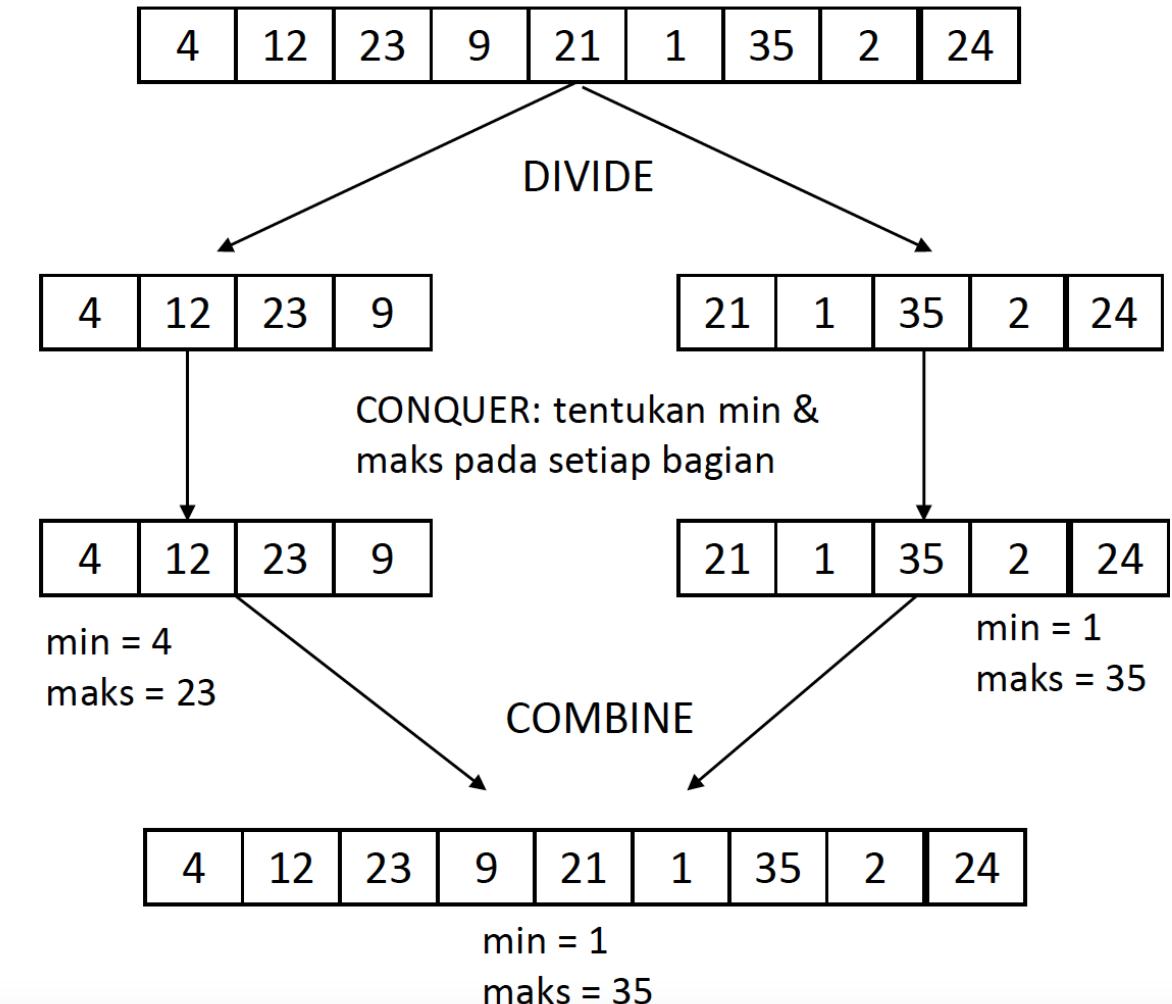
Algoritma:

```
min ← A[1]      { asumsikan elemen pertama sebagai nilai minimum sementara}
maks ← A[1]      {asumsikan elemen pertama sebagai nilai maksimum sementara}
for i ← 2 to n do
    if A[i] < min then
        min ← A[i]
    endif
    if A[i] > maks then
        maks ← A[i]
    endif
endfor
```

Jumlah perbandingan elemen larik: $T(n) = (n - 1) + (n - 1) = 2n - 2 = O(n)$

Persoalan MinMaks (Nilai Maksimum dan Minimum)

- Penyelesaian dengan Algoritma Divide & Conquer
 - Ukuran larik hasil pembagian dapat dibuat cukup kecil sehingga mencari minimum dan maksimum dapat diselesaikan (SOLVE) secara trivial
 - Dalam hal ini, ukuran “kecil” yang didefinisikan apabila larik hanya berisi 1 atau 2 elemen saja



Persoalan MinMaks (Nilai Maksimum dan Minimum)

Prosedur MinMaks($A[1..n]$, min, maks)

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 larik 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 $\max \leftarrow \max_1$

Persoalan MinMaks (Nilai Maksimum dan Minimum)

procedure *MinMaks2*(**input** A : LarikInteger, i, j : integer, **output** $min, maks$: integer)

{ Mencari nilai maksimum dan minimum di dalam larik A yang berukuran n elemen dengan algoritma divide and Conquer.

Masukan: larik A yang sudah terdefinisi elemen-elemennya

Luaran: nilai maksimum dan nilai minimum larik }

Deklarasi

$min1, min2, maks1, maks2$: integer

Algoritma:

if $i = j$ **then** { larik berukuran 1 elemen }
 $min \leftarrow A[i]; maks \leftarrow A[i]$

else

if $(i = j - 1)$ **then** { larik berukuran 2 elemen }
if $A[i] < A[j]$ **then**
 $min \leftarrow A[i]; maks \leftarrow A[j]$

else

$min \leftarrow A[j]; maks \leftarrow A[i]$

endif

else { larik berukuran lebih dari 2 elemen }
 $k \leftarrow (i + j) \text{ div } 2$ { bagidua larik pada posisi k }
MinMaks2($A, i, k, min1, maks1$)

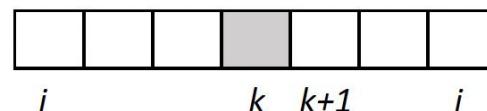
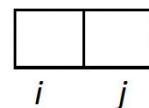
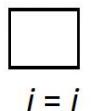
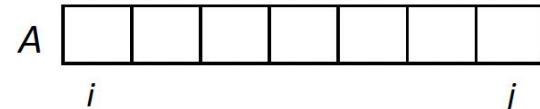
MinMaks2($A, k + 1, j, min2, maks2$)

if $min1 < min2$ **then** $min \leftarrow min1$ **else** $min \leftarrow min2$ **endif**

if $maks1 < maks2$ **then** $maks \leftarrow maks2$ **else** $maks \leftarrow maks1$ **endif**

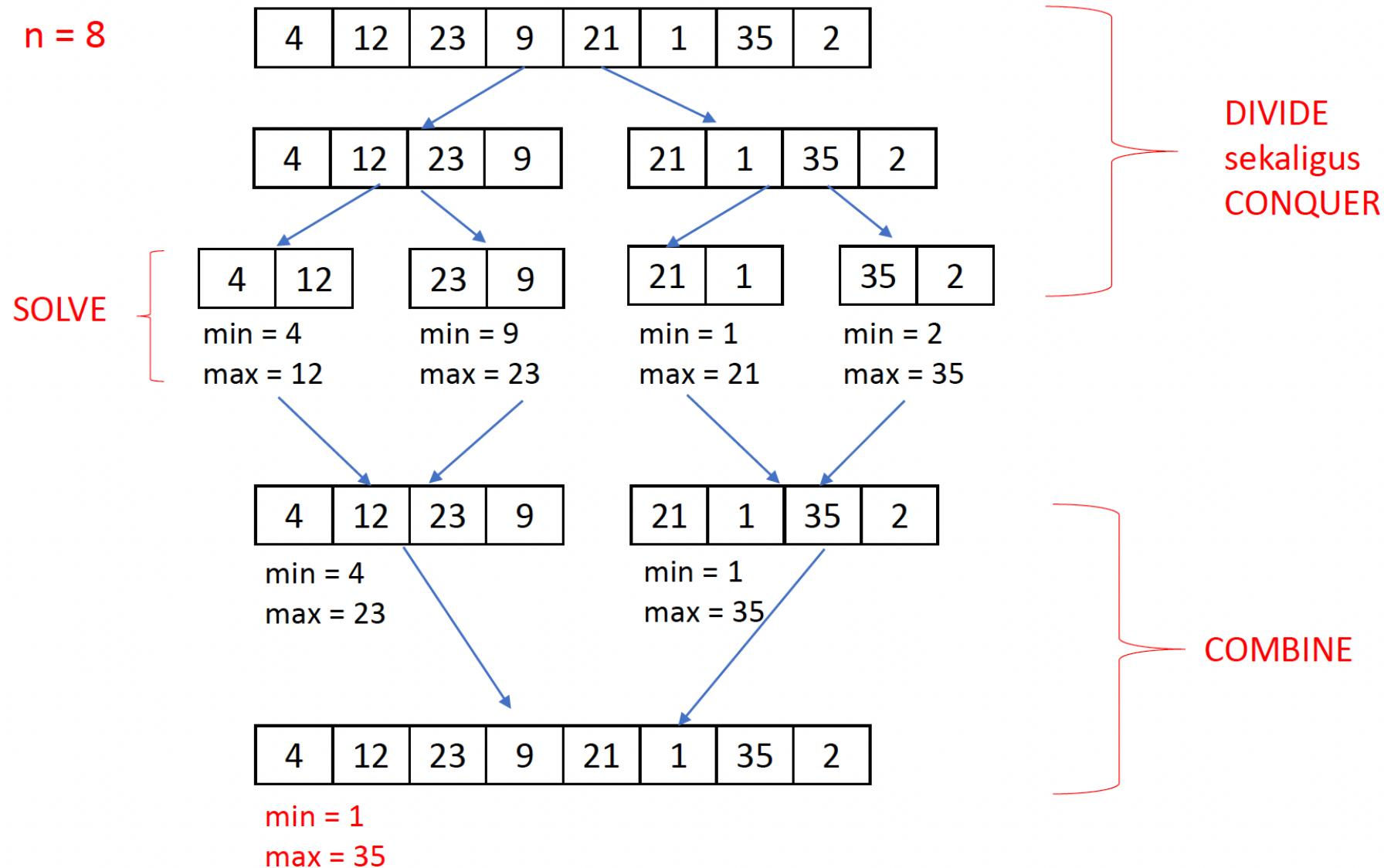
endif

endif



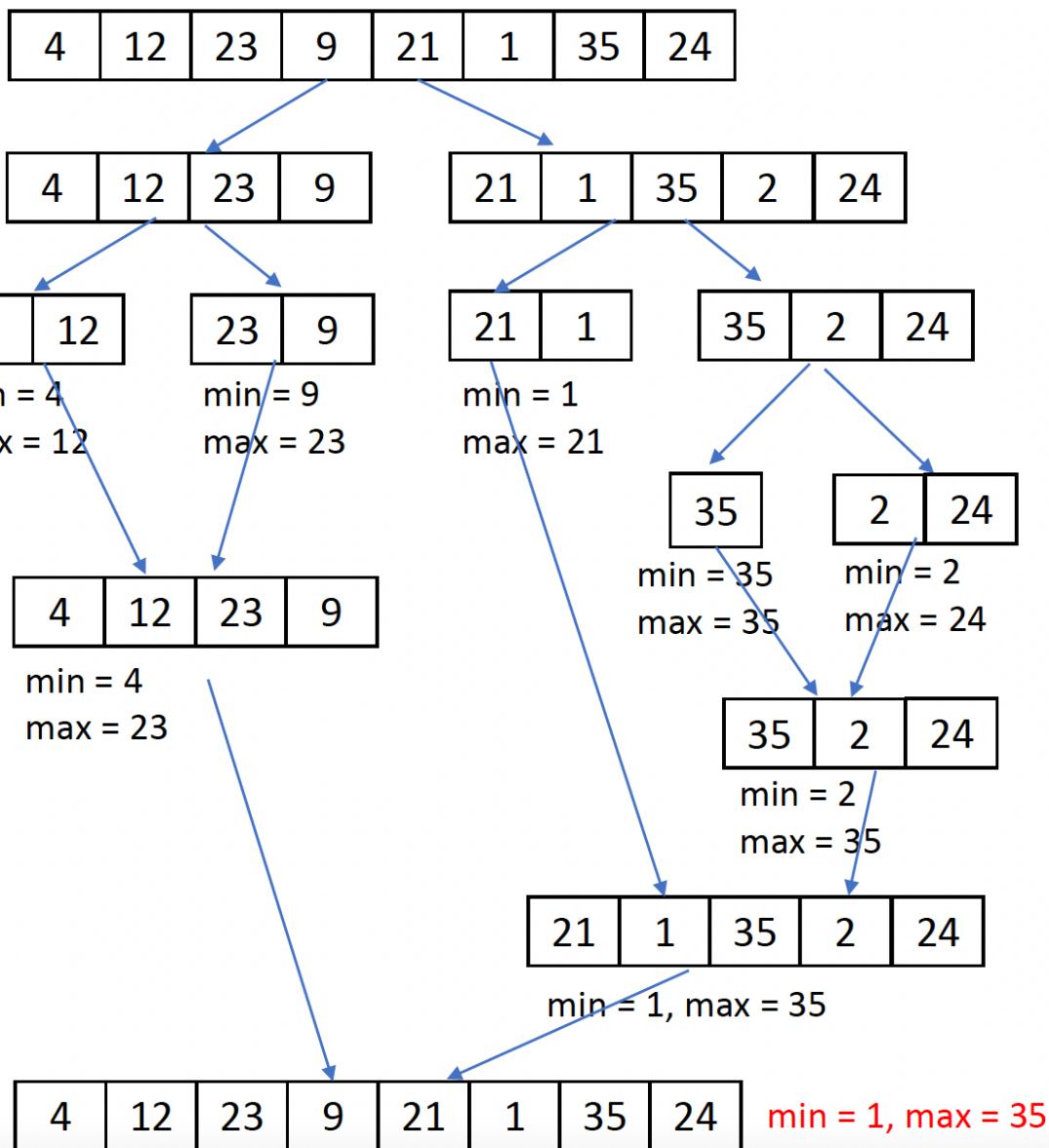
Pemanggilan pertama kali: *MinMaks2*($A, 1, n, min, maks$)

Contoh 1: MinMaks (Nilai Maksimum dan Minimum)



Contoh 2: MinMaks (Nilai Maksimum dan Minimum)

$n = 9$



Kompleksitas Waktu Algoritma MinMaks

Kompleksitas waktu algoritma *MinMaks2*, dihitung dari jumlah operasi perbandingan elemen-elemen larik:

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

Penyelesaian:

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

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

Kompleksitas Waktu Algoritma MinMaks

- Bandingkan:
 - MinMaks 1 secara brute force: $T(n) = 2n - 2$
 - MinMaks 2 secara divide & conquer: $T(n) = 3n/2 - 2$
 - Perhatikan bahwa $3n/2 - 2 < 2n - 2$
- Kesimpulan:
 - Persoalan MinMaks lebih cocok diselesaikan dengan algoritma Divide & Conquer
- Pesan dari contoh di atas adalah bahwa algoritma divide and conquer dapat membantu kita menghasilkan algoritma yang sesuai untuk persoalan yang sedang diselesaikan

Persoalan Perpangkatan a^n

- Misalkan a anggota bilangan R dan n adalah bilangan bulat tidak negative, maka perpangkatan a^n didefinisikan sebagai berikut:

$$a^n = \begin{cases} 1, & n = 0 \\ a \times a \times \cdots \times a, & n > 0 \end{cases}$$

- Bagaimana algoritma menghitung perpangkatan a^n secara *brute force* dan secara *divide & conquer*

Persoalan Perpangkatan a^n

- Penyelesaian dengan algoritma brute force

```
function Exp1(a : real, n : integer)→ real
{ Menghitung  $a^n$ , a > 0 dan n bilangan bulat tak-negatif }
```

Deklarasi

```
k : integer
hasil : real
```

Algoritma:

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

```
return hasil
```

Kompleksitas algoritma, dihitung dari jumlah operasi perkalian: $T(n) = n = O(n)$

Persoalan Perpangkatan a^n

- Penyelesaian dengan algoritma divide & conquer
 - Ide dasar: bagi dua pangkat n menjadi $n = n/2 + n/2$

$$a^n = a^{(n/2 + n/2)} = a^{n/2} \cdot a^{n/2}$$

- Algoritma divide & conquer untuk 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 $a^n = a^{n/2} \cdot a^{n/2} \cdot a$

Persoalan Perpangkatan a^n

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

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

→ Hanya membutuhkan enam operasi perkalian
(operasi perpangkatan dua = perkalian)

Persoalan Perpangkatan a^n

Pseudo-code menghitung a^n dengan divide and conquer:

```
function Exp2(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 { kasus n ganjil }
        return Exp2(a, n div 2) * Exp2(a, n div 2) * a      { $a^n = a^{n/2} \cdot a^{n/2} \cdot a$ }
    else          { kasus n genap }
        return Exp2(a, n div 2) * Exp2(a, n div 2)      { $a^n = a^{n/2} \cdot a^{n/2}$ }
    endif
endif
```

Fungsi *Exp2* tidak optimal, karena terdapat dua kali pemangkatan rekursif untuk nilai parameter yang sama
→ Exp2(*a*, *n* div 2) * Exp2(*a*, *n* div 2)

Persoalan Perpangkatan a^n

Perbaikan: simpan hasil $Exp2(a, n \text{ div } 2)$ di dalam sebuah peubah (misalkan x), lalu gunakan x untuk menghitung a^n pada kasus n genap dan n ganjil.

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

Deklarasi

x : real

Algoritma:

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

Persoalan Perpangkatan a^n

- Kompleksitas algoritma Exp3 dihitung dari jumlah operasi perkalian:

$$T(n) = \begin{cases} 0, & n = 0 \\ T\left(\frac{n}{2}\right) + 2, & n > 0 \text{ dan } n \text{ ganjil} \\ T\left(\frac{n}{2}\right) + 1, & n > 0 \text{ dan } n \text{ genap} \end{cases}$$

$x * x * a$
 $x * x$

- Dalam menghitung $T(n)$ ini ada sedikit kesulitan, yaitu nilai n mungkin ganjil atau genap, sehingga penyelesaian relasi rekuren menjadi lebih rumit
- Namun, perbedaan ini dianggap kecil sehingga dapat kita abaikan. Sebagai implikasinya, kita membuat asumsi penghampiran bahwa untuk n genap atau ganjil, jumlah operasi perkalian relative sama

Persoalan Perpangkatan a^n

- Sehingga kompleksitas algoritma Exp3 menjadi:

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

- Asumsikan n adalah perpangkatan dari 2, atau $n = 2^k$, maka

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

- Karena $n = 2^k$ maka $k = \log_2 n$, sehingga

$$\begin{aligned} &= k + T(n/2^k) = \log_2 n + T(1) \\ &= \log_2 n + (1 + T(0)) = \log_2 n + 1 + 0 \\ &= \log_2 n + 1 = O(\log_2 n) \rightarrow \text{lebih baik daripada algoritma } \textit{brute force}! \end{aligned}$$

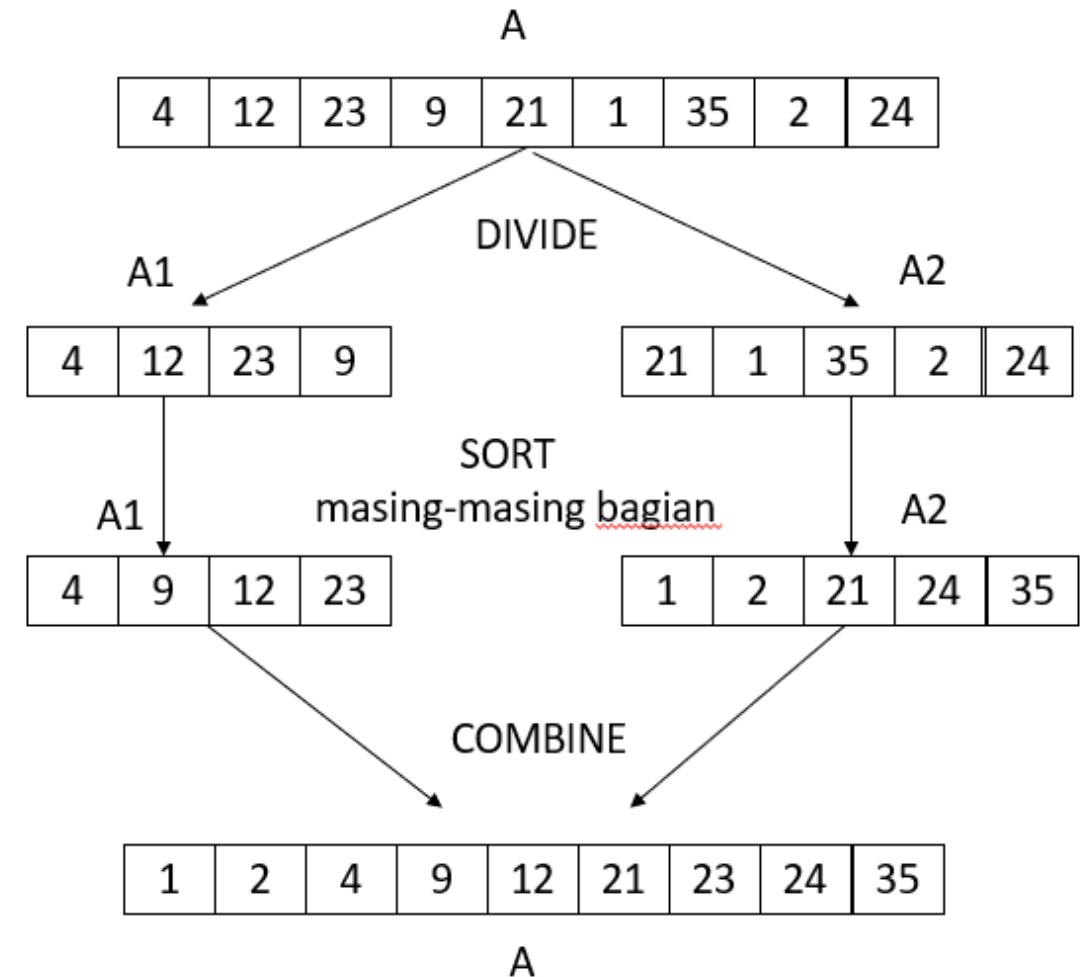
Merge Sort

Persoalan Pengurutan Secara Divide & Conquer

- Algoritma pengurutan secara *brute force*: algoritma *selection sort*, *bubble sort*, *insertion sort*
- Ketiganya memiliki kompleksitas algoritma $O(n^2)$
- Dengan metode divide and conquer, dapatkah dihasilkan algoritma pengurutan dengan kompleksitas lebih rendah dari n^2

Persoalan Pengurutan Secara Divide & Conquer

- Ide pengurutan larik secara divide & conquer:
 1. Jika ukuran larik = 1 elemen, larik sudah terurut dengan sendirinya
 2. Jika ukuran larik > 1, bagi larik menjadi 2 bagian, lalu urut masing-masing bagian
 3. Gabungkan hasil pengurutan masing-masing bagian menjadi sebuah larik yang terurut



Persoalan Pengurutan Secara Divide & Conquer

```
procedure Sort(input/output A : LarikInteger, input n : integer)
{ Mengurutkan larik A dengan metode Divide and Conquer
  Masukan: Larik A dengan n elemen
  Luaran: Larik A yang terurut
}
```

Algoritma:

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

Bagi A menjadi dua bagian, A_1 dan A_2 , masing-masing berukuran n_1 dan n_2 ($n = n_1 + n_2$)

$\text{Sort}(A_1, n_1)$ { urut larik bagian kiri yang berukuran n_1 elemen }

$\text{Sort}(A_2, n_2)$ { urut larik bagian kanan yang berukuran n_2 elemen }

$\text{Combine}(A_1, A_2, A)$ { gabung hasil pengurutan bagian kiri dan bagian kanan }

```
end
```

Persoalan Pengurutan Secara Divide & Conquer

Terdapat dua pendekatan melakukan pengurutan dengan *divide and conquer*:

1. Mudah membagi, tetapi sulit menggabung (*easy split/hard join*)
 - Pembagian larik menjadi dua bagian mudah secara komputasi (hanya membagi berdasarkan posisi atau indeks larik)
 - Penggabungan dua buah larik terurut menjadi sebuah larik terurut sukar secara komputasi (ditinjau dari kompleksitas algoritmanya)
2. Sulit membagi, tetapi mudah menggabung (*hard split/easy join*)
 - Pembagian larik menjadi dua bagian sukar secara komputasi (pembagiannya berdasarkan nilai elemen, bukan posisi elemen larik)
 - Penggabungan dua buah larik terurut menjadi sebuah larik terurut mudah dilakukan secara komputasi

Persoalan Pengurutan Secara Divide & Conquer

Contoh: Misalkan larik A adalah sebagai berikut:

A	8	1	4	6	9	3	5	7
-----	---	---	---	---	---	---	---	---

Dua pendekatan (*approach*) pengurutan:

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

Tabel A dibagi dua berdasarkan posisi elemen:

Divide: A_1

8	1	4	6
---	---	---	---

A_2

9	3	5	7
---	---	---	---

Sort: A_1

1	4	6	8
---	---	---	---

A_2

3	5	7	9
---	---	---	---

Combine: A_1

1	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---

Algoritma pengurutan yang termasuk jenis ini:

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

Persoalan Pengurutan Secara Divide & Conquer

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

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

A

8	1	4	6	9	3	5	7
---	---	---	---	---	---	---	---

Divide: $A1$

5	1	4	3
---	---	---	---

$A2$

9	6	8	7
---	---	---	---

Sort: $A1$

1	3	4	5
---	---	---	---

$A2$

6	7	8	9
---	---	---	---

Combine: A

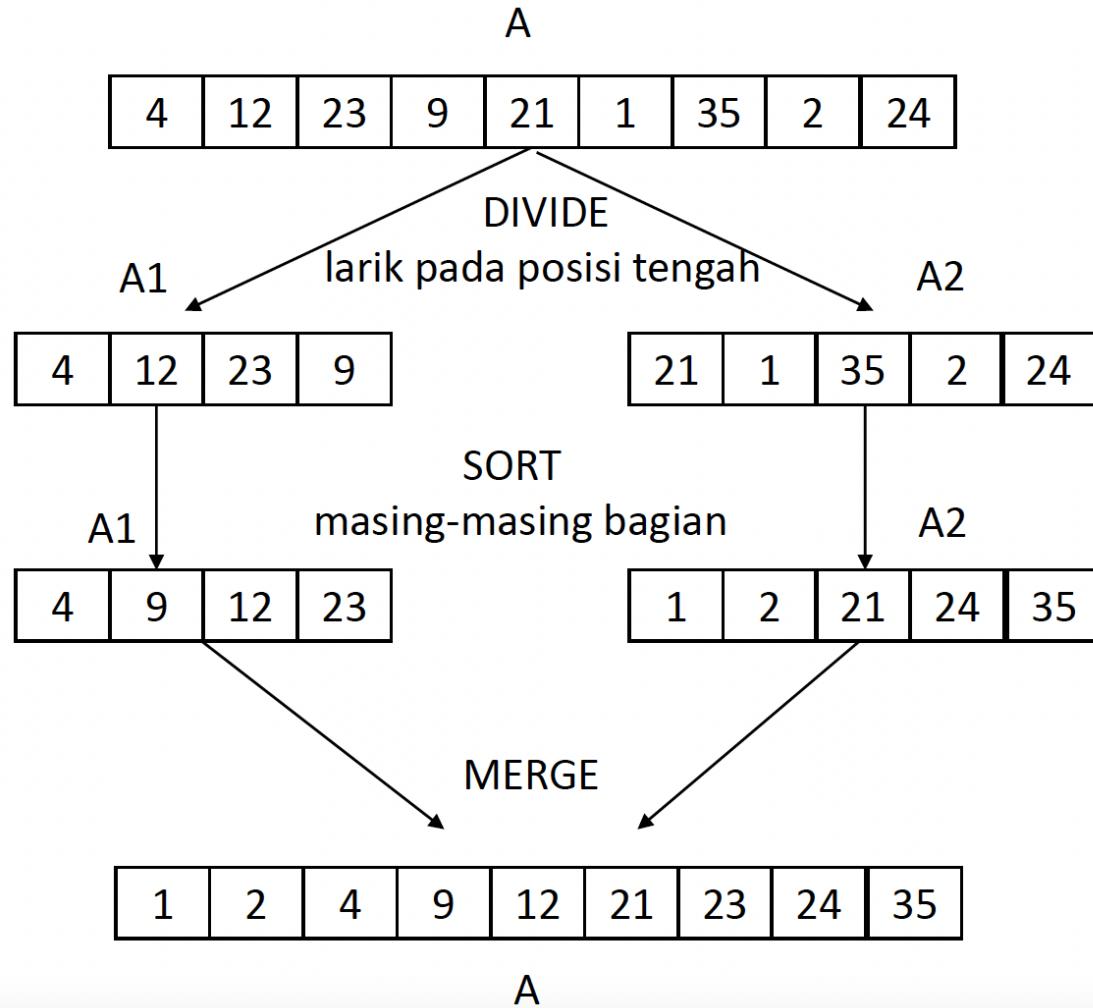
1	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---

Algoritma pengurutan yang termasuk jenis ini:

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

Divide & Conquer: Merge Sort

- Ide *merge sort*:



Pertanyaan:

1. Larik dibagi sampai ukurannya (n) tinggal berapa elemen?
2. Bagaimana menggabungkan dua larik terurut menjadi satu larik terurut?

Jawaban:

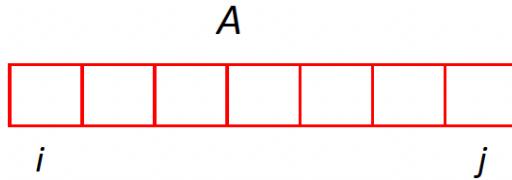
1. Sampai $n = 1$
2. Gunakan algoritma *merge*

Divide & Conquer: Merge Sort

- Algoritma Merge Sort (A, n):
 - Jika $n = 1$, maka larik A sudah terurut dengan sendirinya (langkah SOLVE)
 - Jika $n > 1$, maka
 - DIVIDE: bagi larik A menjadi dua bagian pada posisi pertengahan, masing-masing bagian berukuran $n/2$ elemen
 - CONQUER: secara rekursif, terapkan Merge Sort pada masing-masing bagian
 - MERGE: gabung hasil pengurutan kedua bagian sehingga diperoleh larik A yang terurut

Divide & Conquer: Merge Sort

- Dalam notasi *pseudo-code*:



```
procedure MergeSort(input/output A : LarikInteger, input i,j : integer)
{ Mengurutkan larik A[i..j] dengan algoritma Merge Sort.
  Masukan: Larik A[i..j] yang sudah terdefinisi elemen-elemennya
  Luaran: Larik A[i..j] yang terurut
}
```

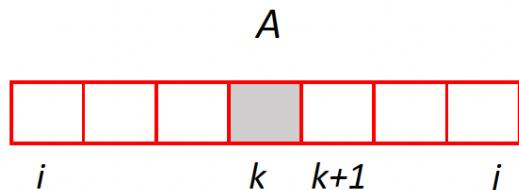
Deklarasi

k : integer

Algoritma:

```
if i < j then          { ukuran(A) > 1 }
  k ← (i + j) div 2   { bagi A pada posisi pertengahan }
  MergeSort(A, i, k)    { urut upalarik A[i..k] }
  MergeSort(A, k + 1, j) { urut upalarik A[k+1..j] }
  Merge(A, i, k, j)      { gabung hasil pengurutan A[i..k] dan A[k+1..j] menjadi A[i..j] }
```

end



Pemanggilan pertama kali: *MergeSort(A, 1, n)*

Divide & Conquer: Merge Sort

- Contoh Merge dua larik terurut menjadi satu larik terurut:

<i>A1</i>
1 13 24

<i>A2</i>
2 15 27

$1 < 2 \rightarrow 1$

<i>B</i>					
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
---	----	----

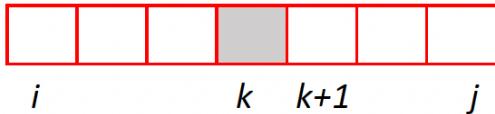
2	15	27
---	----	----

$27 \rightarrow$

1	2	13	15	24	27
---	---	----	----	----	----

Divide & Conquer: Merge Sort

```
procedure Merge(input/output A : LarikInteger, input i, k, j : integer)
{ Menggabung larik A[i..k] dan larik A[k+1..j] menjadi larik A[i..j] yang terurut menaik.    A
  Masukan: A[i..k] dan A[k+1..j] sudah terurut menaik.
  Luaran: A[k+1..j] yang terurut menaik. }
```



Deklarasi

B : LarikInteger { larik temporer untuk menyimpan hasil penggabungan }
p, q, r : integer

Algoritma:

```
p ← i          { A[i .. k] }
q ← k + 1      { A[k+1 .. j] }
r ← i
while (p ≤ k) and (q ≤ j) do
  if A[p] ≤ A[q] then
    B[r] ← A[p]      { salin elemen A[p] dari larik bagian kiri ke dalam larik B }
    p ← p + 1
  else
    B[r] ← A[q]      { salin elemen A[q] dari larik bagian kanan ke dalam larik B }
    q ← q + 1
  endif
  r ← r + 1
endwhile
{ p > k or q > j }
..... continued
```

Divide & Conquer: Merge Sort

{ salin sisa larik A bagian kiri ke larik B, jika masih ada }

while ($p \leq k$) **do**

$B[r] \leftarrow A[p]$

$p \leftarrow p + 1$

$r \leftarrow r + 1$

endwhile

{ $p > k$ }

{ salin sisa larik A bagian kanan ke larik B, jika masih ada }

while ($q \leq j$) **do**

$B[r] \leftarrow A[q]$

$q \leftarrow q + 1$

$r \leftarrow r + 1$

endwhile

{ $q > j$ }

{ salin kembali elemen-elemen larik B ke dalam A }

for $r \leftarrow i$ **to** j **do**

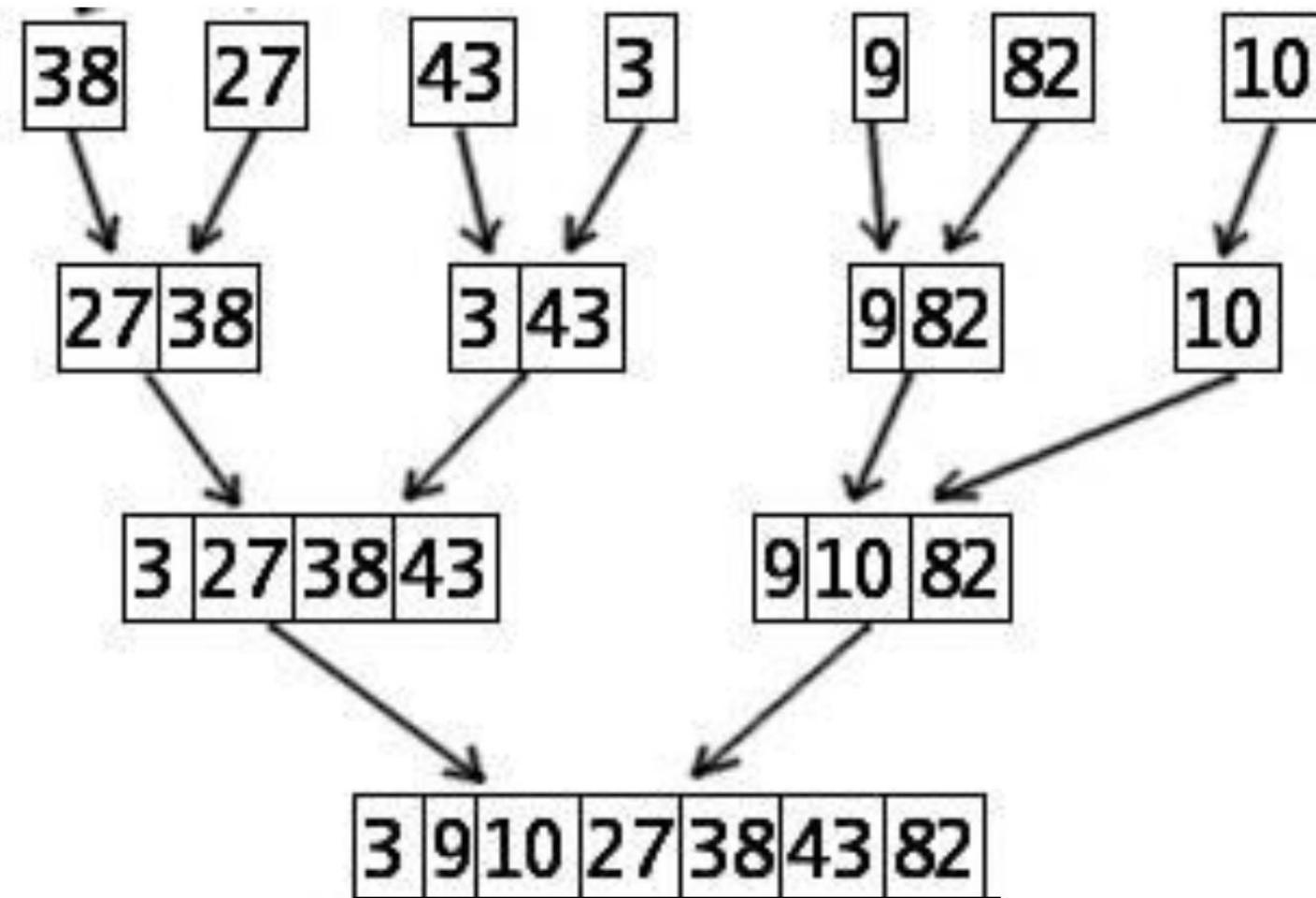
$A[r] \leftarrow B[r]$

endfor

{ diperoleh larik A yang terurut membesar }

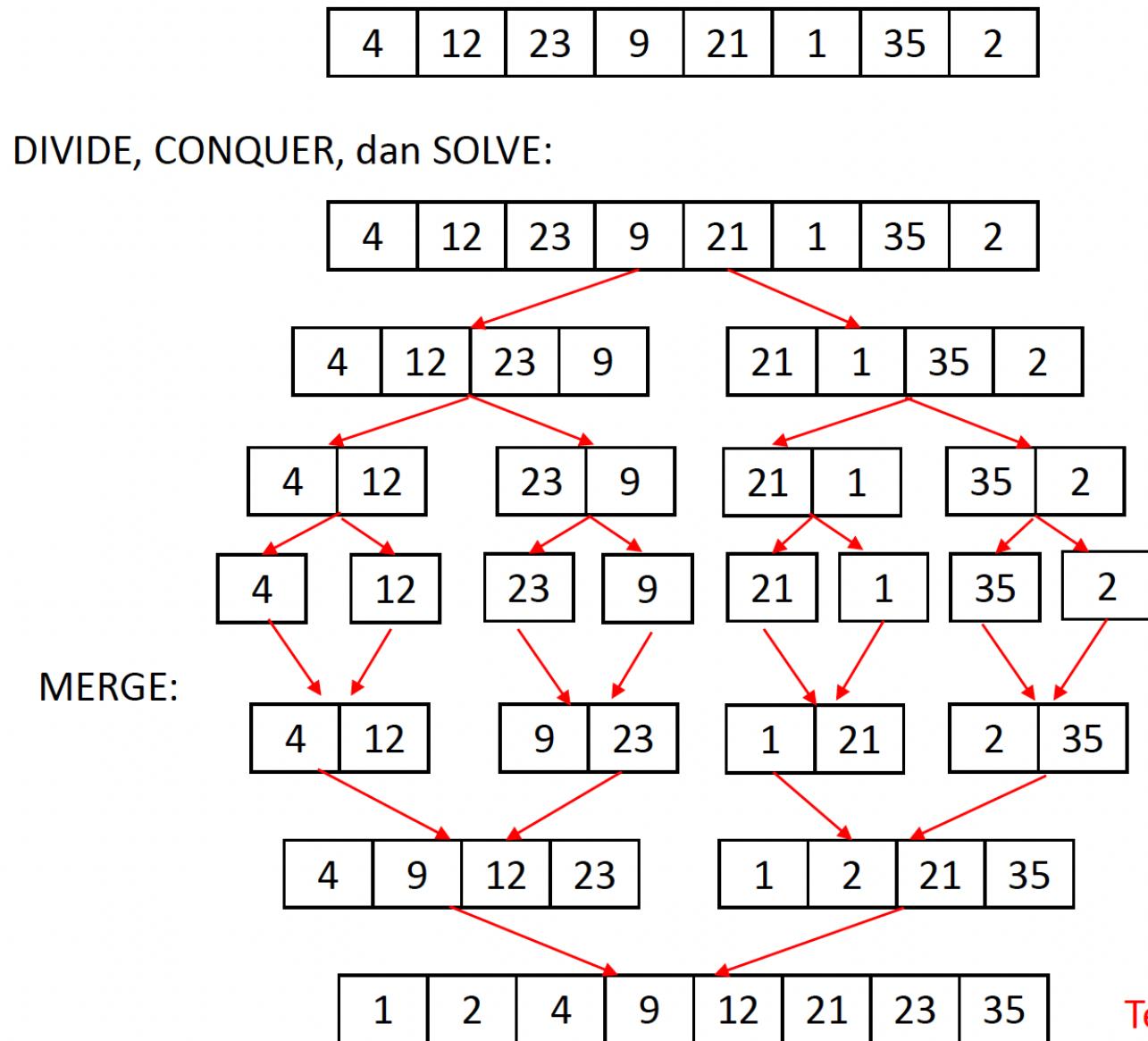
Divide & Conquer: Merge Sort

- Contoh proses Merge di dalam Merge Sort:



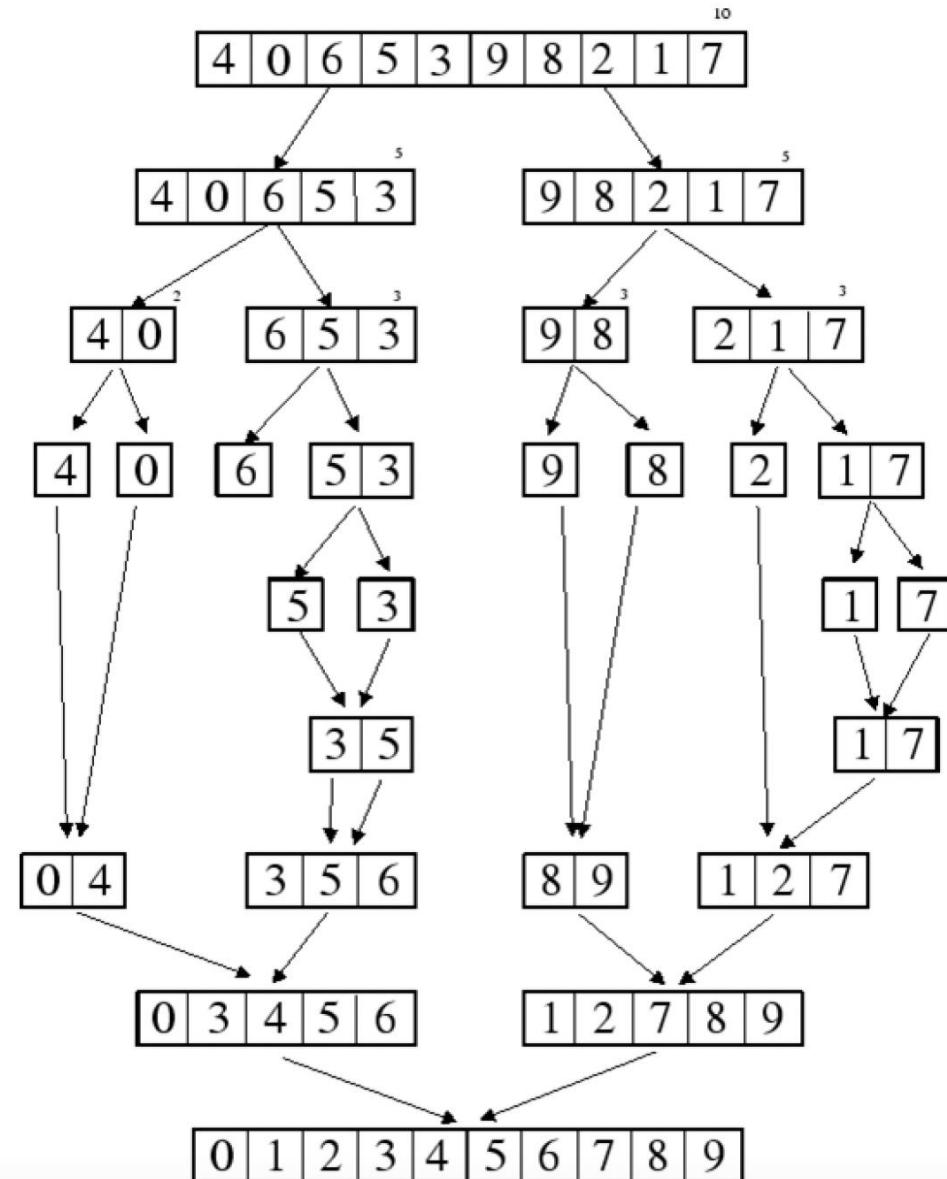
Divide & Conquer: Merge Sort

- Contoh: Pengurutan larik A disamping ini dengan Merge Sort



Divide & Conquer: Merge Sort

- Contoh: Pengurutan larik A disamping ini dengan Merge Sort



Divide & Conquer: Merge Sort

- Kompleksitas waktu Merge Sort
 - Kompleksitas algoritma Merge Sort diukur dari jumlah operasi perbandingan elemen-elemen larik
 - Jumlah perbandingan elemen-elemen larik di dalam prosedur Merge Sort adalah $O(n)$, yaitu berbanding lurus dengan jumlah elemen larik, atau cn , c adalah konstanta
 - Jumlah perbandingan elemen-elemen larik seluruhnya:

$$\begin{aligned} T(n) &= \text{Merge Sort untuk pengurutan dua buah sub-larik berukuran } n/2 + \\ &\quad \text{jumlah perbandingan elemen di dalam prosedur Merge} \\ &= 2T(n/2) + cn \end{aligned}$$

- Sehingga,

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

Divide & Conquer: Merge Sort

- Penyelesaian persamaan rekursif secara iteratif :

Untuk menyederhanakan perhitungan, asumsikan $n = 2^k$

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

$$n = 2^k \rightarrow k = \log n$$

sehingga

$$T(n) = nT(1) + cn \log n = an + cn \log n = O(n \log n)$$

- Jadi, kompleksitas algoritma *Merge Sort* adalah $O(n \log n)$, lebih baik daripada kompleksitas algoritma pengurutan secara *brute force*.

Implementasi Merge Sort

- Bahasa C
 - <https://www.programiz.com/dsa/merge-sort#c-code>
 - [https://www.tutorialspoint.com/data structures algorithms/merge sort program in c.htm](https://www.tutorialspoint.com/data_structures_algorithms/merge_sort_program_in_c.htm)
 - <https://www.geeksforgeeks.org/merge-sort/>

Q & A

Referensi

- <https://www.programiz.com/dsa/bubble-sort>
- <https://www.programiz.com/dsa/selection-sort>
- <https://www.programiz.com/dsa/insertion-sort>