

# Algoritma *Divide and Conquer*

(Bagian 1)

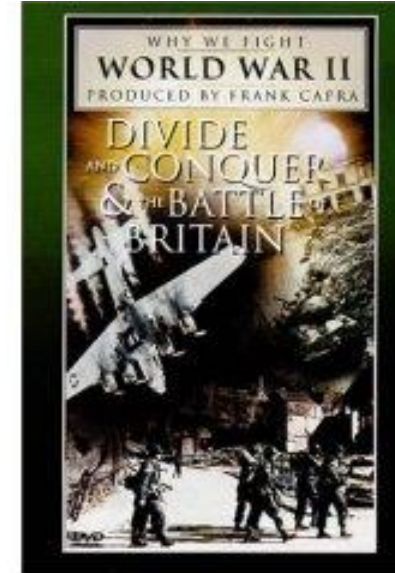
Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi M



Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika ITB  
2026

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

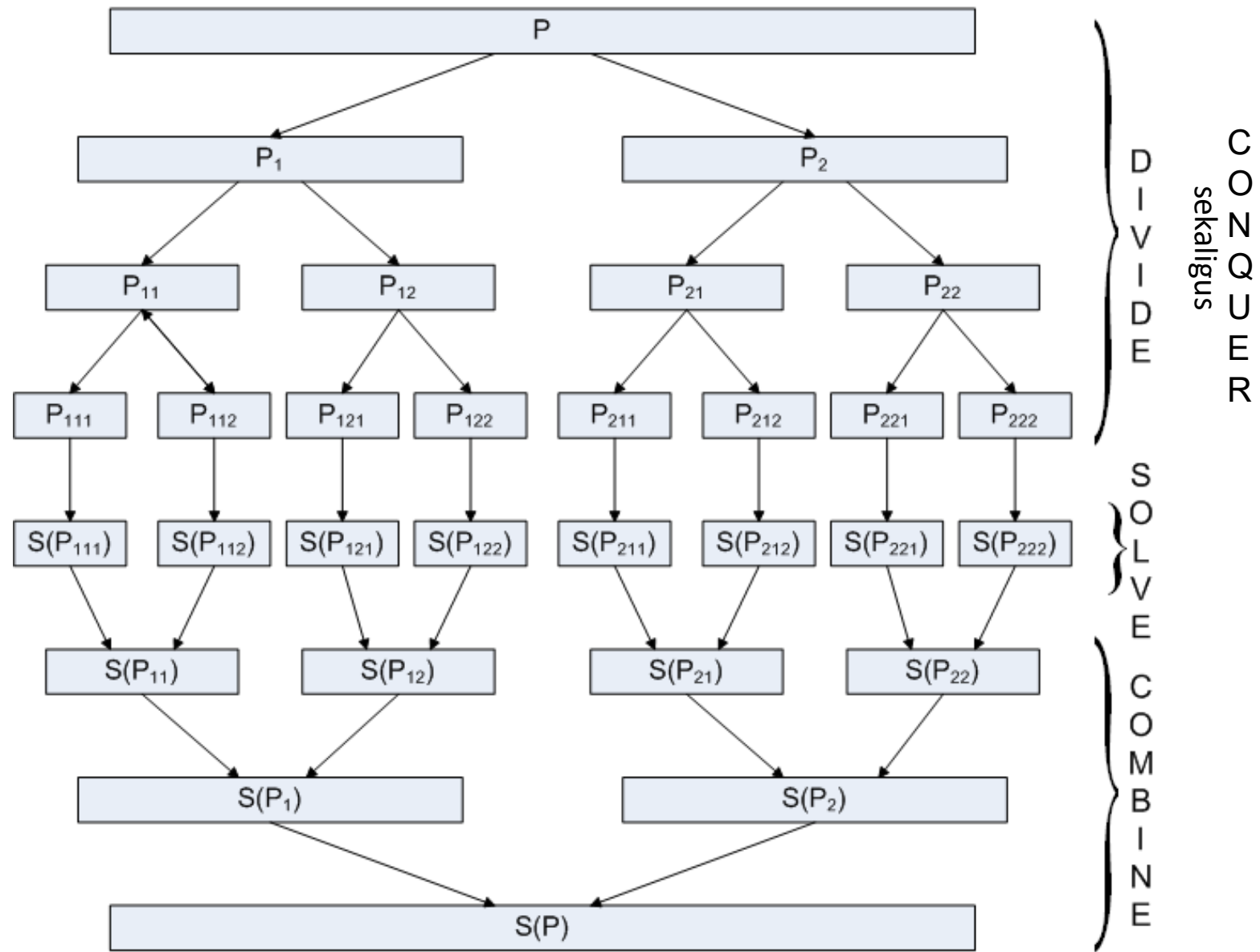


- Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama *Divide and Conquer*.



# Definisi Divide and Conquer

- *Divide*: membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama),
- *Conquer*: menyelesaikan (*solve*) masing-masing upa-persoalan ( secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
- *Combine*: menggabungkan solusi masing-masing upa-persoalan 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,
  - polinom,
  - dll, bergantung persoalannya.
- Tiap-tiap upa-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakteristik persoalan semula namun berukuran lebih kecil
- sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

# Skema Umum Algoritma *Divide and 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 }
```

## Kamus

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

Penjelasan:

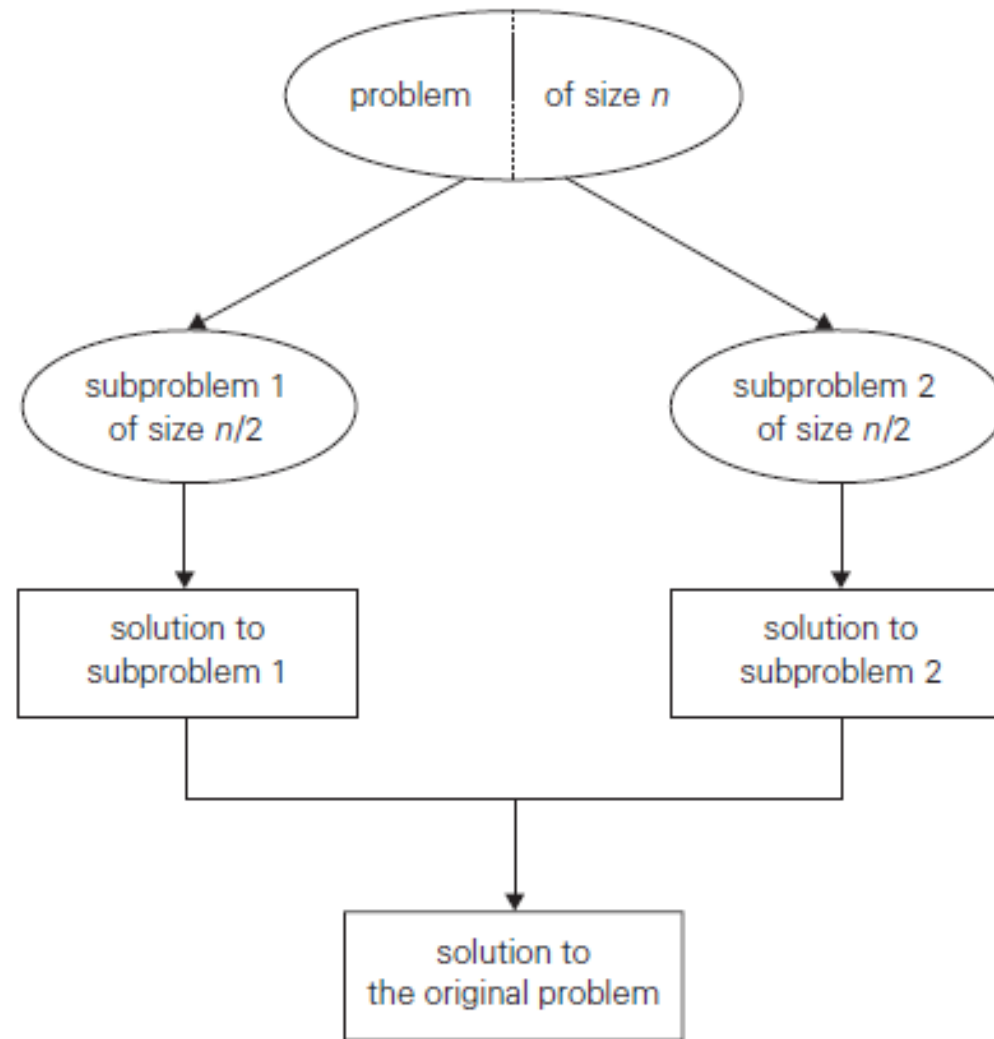
$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

- $T(n)$  : 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 upa-persoalan
- $f(n)$  : kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan
- Tahap DIVIDE dapat dilakukan dalam  $O(1)$ , sehingga tidak dimasukkan ke dalam formula

Kasus sederhana: Jika pembagian selalu menghasilkan **dua** upa-persoalan 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  
}  
Kamus  
   $r$  : 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*: 
$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ 2T(n/2) + f(n) & , n > n_0 \end{cases}$$



**FIGURE 5.1** Divide-and-conquer technique (typical case).

# Beberapa persoalan klasik yang diselesaikan dengan D&C

1. Persoalan MinMaks (mencari nilai minimum dan nilai maksimum)
2. Menghitung perpangkatan
3. Persoalan pengurutan (*sorting*) – *Mergesort* dan *Quicksort*
4. Mencari sepasang titik terdekat (*closest pair problem*)
5. *Convex Hull*
6. Perkalian matriks
7. Perkalian bilangan bulat besar
8. Perkalian dua buah polinom
9. *Skyline* problem
10. Dll

# 1. Persoalan *MinMaks*: Mencari Nilai Minimum dan Maksimum

**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 |
|---|----|----|---|----|---|----|---|----|

min = 1

max = 35

# Penyelesaian dengan *algoritma brute force*

**procedure** *MinMaks1*(**input**  $A : \text{Larik}, n : \text{integer}$ , **output**  $min, maks : \text{integer}$ )

*{ 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*

*}*

**Kamus**

$i : \text{integer}$

**Algoritma:**

$min \leftarrow A[1]$       *{ asumsikan elemen pertama sebagai nilai minimum sementara }*

$maks \leftarrow A[1]$       *{ asumsikan elemen pertama sebagai nilai maksimum sementara }*

**for**  $i \leftarrow 2$  **to**  $n$  **do**

**if**  $A[i] < min$  **then**

$min \leftarrow A[i]$

**endif**

**if**  $A[i] > maks$  **then**

$maks \leftarrow A[i]$

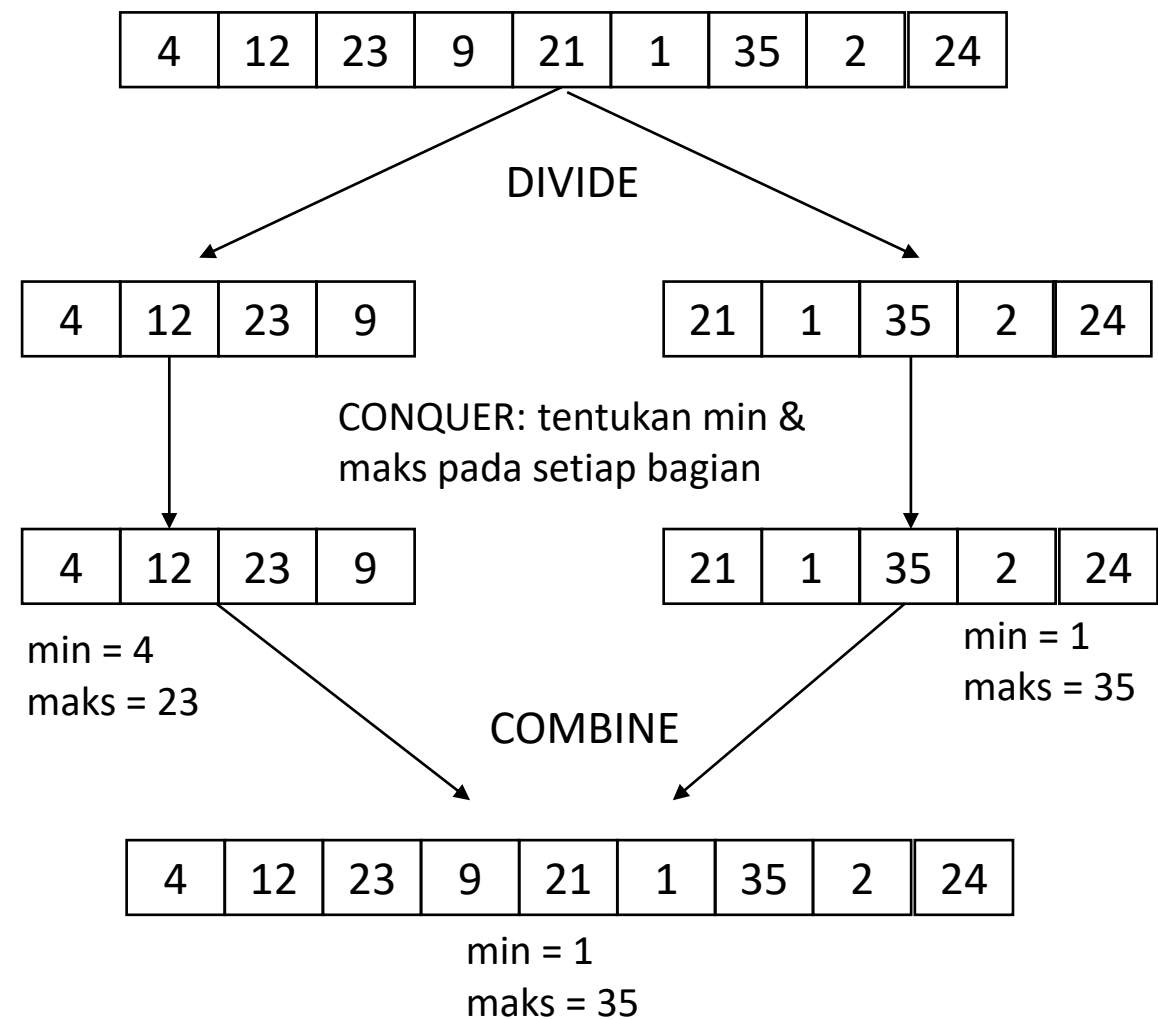
**endif**

**endfor**

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

# Penyelesaian dengan *algorithm divide and conquer*

Ide dasar secara *divide and conquer*:



- Ukuran larik hasil pembagian dapat dibuat cukup kecil sehingga mencari minimum dan maksimum dapat diselesaikan (SOLVE) secara trivial.
- Dalam hal ini, ukuran “kecil” didefinisikan apabila larik hanya berukuran 1 elemen atau 2 elemen.

## Prosedur MinMaks2( $A$ , $n$ , $\min$ , $\max$ )

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 larik  $A$  menjadi dua bagian yang sama,  $A1$  dan  $A2$ , masing2  $n/2$  elemen
  - (b) CONQUER:  
MinMaks2( $A1$ ,  $n/2$ ,  $\min1$ ,  $\max1$ )  
MinMaks2( $A2$ ,  $n/2$ ,  $\min2$ ,  $\max2$ )
  - (c) COMBINE:  
if  $\min1 < \min2$  then  $\min \leftarrow \min1$  else  $\min \leftarrow \min2$   
if  $\max1 < \max2$  then  $\max \leftarrow \max2$  else  $\max \leftarrow \max1$

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

{ 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 }

**Kamus**

$\text{min1}, \text{min2}, \text{maks1}, \text{maks2} : \text{integer}$

**Algoritma:**

**if**  $i = j$  **then** { larik berukuran 1 elemen }

$\text{min} \leftarrow A[i]; \text{maks} \leftarrow A[i]$



$i = j$

**else**

**if**  $(i = j - 1)$  **then** { larik berukuran 2 elemen }

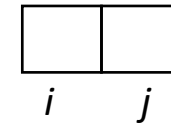
**if**  $A[i] < A[j]$  **then**

$\text{min} \leftarrow A[i]; \text{maks} \leftarrow A[j]$

**else**

$\text{min} \leftarrow A[j]; \text{maks} \leftarrow A[i]$

**endif**



**else** { larik berukuran lebih dari 2 elemen }

$k \leftarrow (i + j) \text{ div } 2$  { bagidua larik pada posisi  $k$  }

$\text{MinMaks2}(A, i, k, \text{min1}, \text{maks1})$

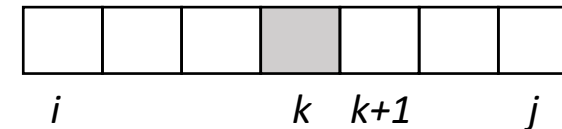
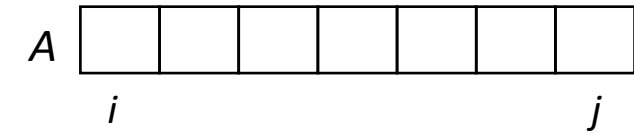
$\text{MinMaks2}(A, k + 1, j, \text{min2}, \text{maks2})$

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

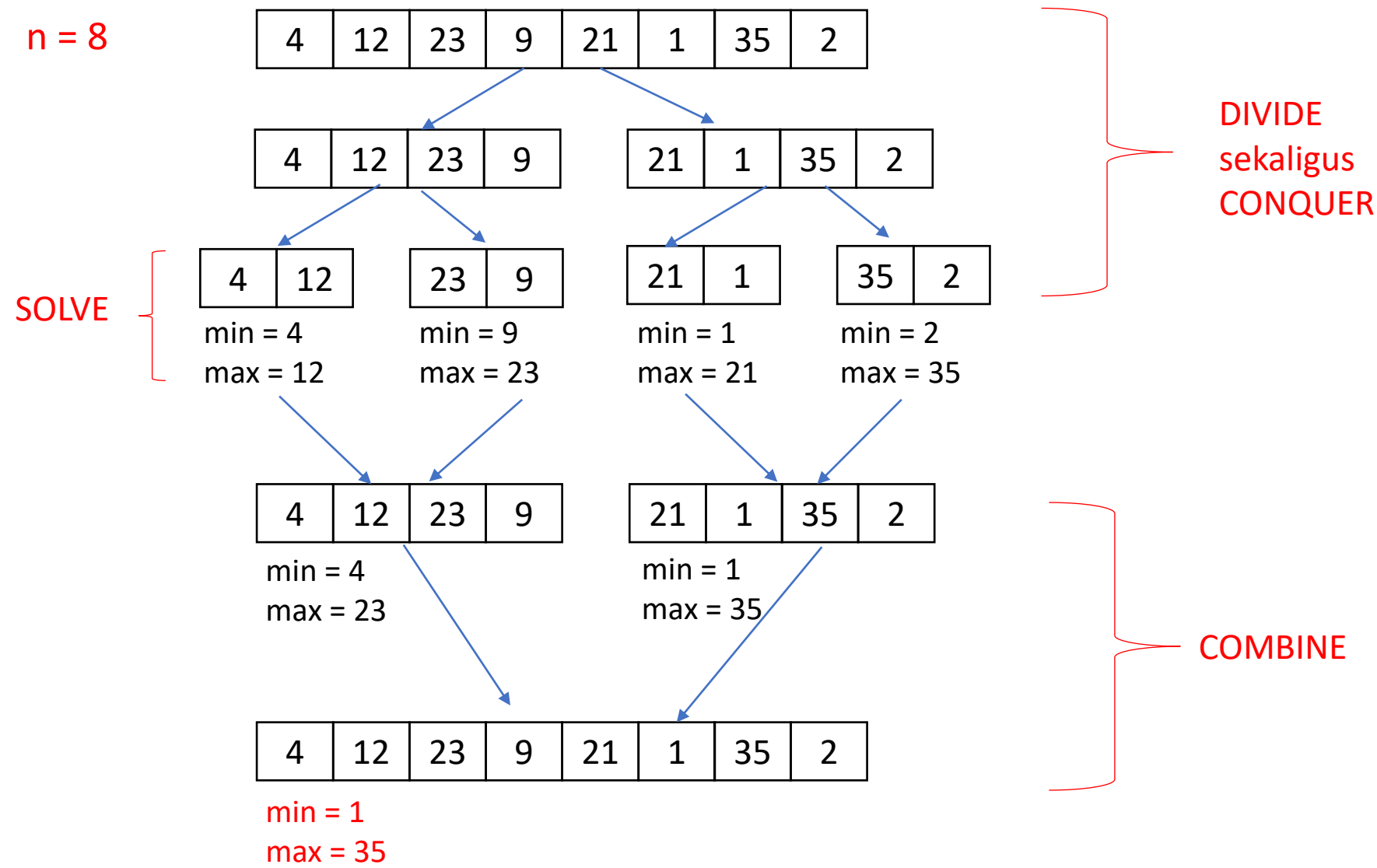
**if**  $\text{maks1} < \text{maks2}$  **then**  $\text{maks} \leftarrow \text{maks2}$  **else**  $\text{maks} \leftarrow \text{maks1}$  **endif**

**endif**

**endif**

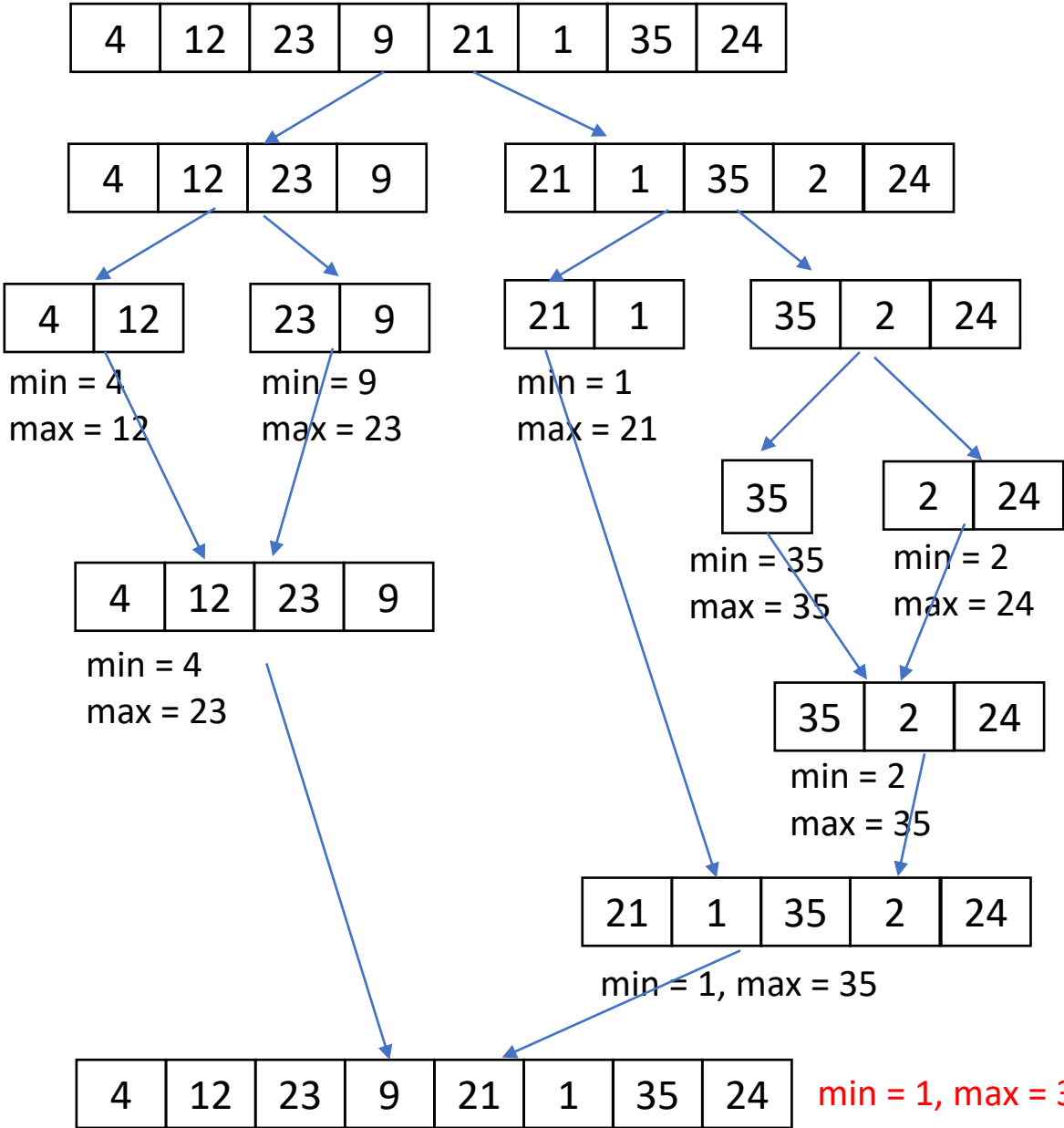


**Contoh 1:** Mencari nilai minimum dan maksimum di dalam larik berikut (kasus  $n = 2^k$ )



**Contoh 2:** Mencari nilai minimum dan maksimum di dalam larik berikut (kasus  $n \neq 2^k$ )

$n = 9$



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

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

## **Bandingkan:**

- *MinMaks1* secara *brute force* :  $T(n) = 2n - 2$
- *MinMaks2* secara *divide and conquer*:  $T(n) = 3n/2 - 2$

Perhatikan bahwa  $3n/2 - 2 < 2n - 2$  untuk  $n \geq 2$ .

- Kesimpulan: persoalan *MinMaks* lebih sangkil diselesaikan dengan algoritma *Divide and Conquer* untuk  $n$  yang besar
- Moral dari contoh ini adalah bahwa algoritma *divide and conquer* dapat membantu kita menghasilkan algoritma yang sangkil.

## 2. Perpangkatan $a^n$

- Misalkan  $a \in R$  dan  $n$  adalah bilangan bulat tidak negatif, 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 and conquer*?

## Penyelesaian dengan *algorithm brute force*

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

### **Kamus**

```
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:  $T(n) = n = O(n)$

## Penyelesaian dengan algoritma *Divide and Conquer*

Ide dasar: bagi dua pangkat  $n$  menjadi  $n = n/2 + n/2$

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

Contoh:  $25^{12} = 25^6 \times 25^6$

Algoritma *divide and 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  $a^n = a^{n/2} \cdot a^{n/2} \cdot a$

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

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

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

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

**function** *Exp2*(*a* : **real**, *n* : **integer**)  $\rightarrow$  **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*,  $n \text{ div } 2$ ) \* *Exp2*(*a*,  $n \text{ div } 2$ ) \* *a*  $\{a^n = a^{n/2} \cdot a^{n/2} \cdot a\}$

**else** *{ kasus n genap }*

**return** *Exp2*(*a*,  $n \text{ div } 2$ ) \* *Exp2*(*a*,  $n \text{ div } 2$ )  $\{a^n = a^{n/2} \cdot a^{n/2}\}$

**endif**

**endif**

Fungsi *Exp2* tidak sangkil, sebab terdapat dua kali pemanggilan rekursif untuk nilai parameter yang sama  $\rightarrow \text{Exp2}(a, n \text{ div } 2) * \text{Exp2}(a, n \text{ div } 2)$

**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**  $Exp3(a : \text{real}, n : \text{integer}) \rightarrow \text{real}$

*{ mengembalikan nilai  $a^n$ , dihitung dengan metode Divide and Conquer }*

**Deklarasi**

$x : \text{real}$

**Algoritma:**

**if**  $n = 0$  **then**

**return** 1

**else**

$x \leftarrow Exp3(a, n \text{ div } 2)$

**if**  $odd(n)$  **then**    *{ kasus  $n$  ganjil }*

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

**else**                    *{ kasus  $n$  genap }*

**return**  $x * x$

**endif**

**endif**

- 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 rekurens menjadi lebih rumit.
- Namun, perbedaan ini dianggap cukup kecil sehingga dapat kita abaikan. Sebagai implikasinya, kita membuat asumsi penghampiran bahwa untuk  $n$  genap atau ganjil, jumlah operasi perkalian relatif sama.

- 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

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

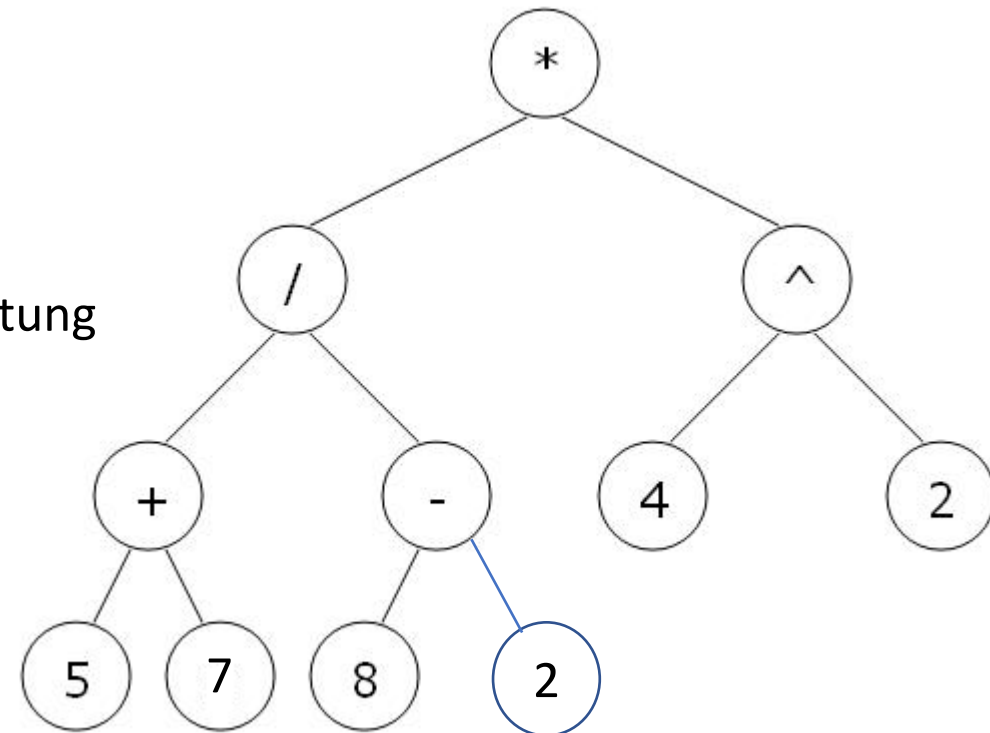
### 3. Mengevaluasi Pohon Ekspresi

- Di dalam *compiler* bahasa pemrograman, ekspresi aritmetika direpresentasikan dalam pohon biner yaitu pohon ekspresi (*expression tree*)

Contoh:  $(5 + 7) / (8 - 2) * (4 ^ 2)$

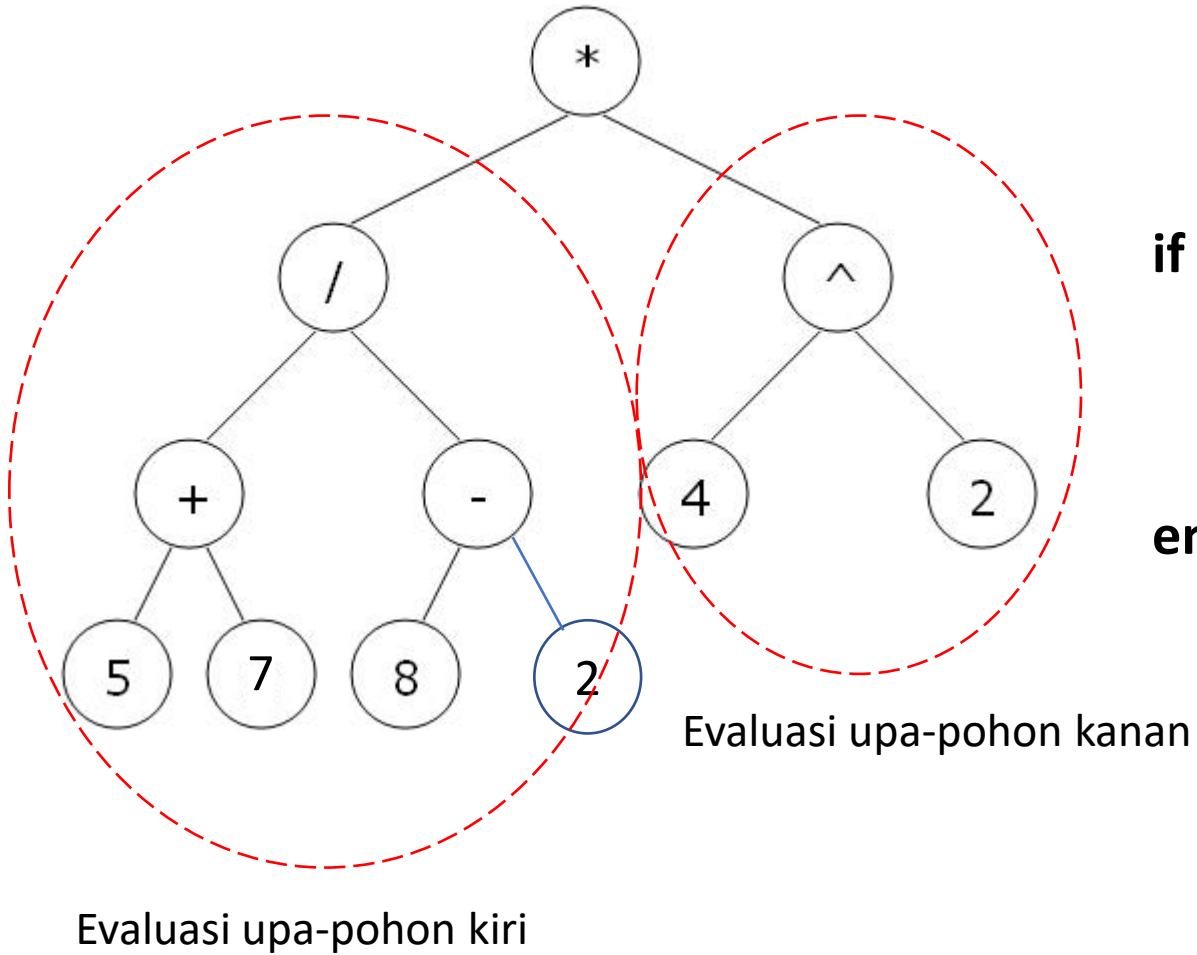
- Mengevaluasi pohon ekspresi artinya menghitung nilai ekspresi aritmetika yang dinyatakannya.

Contoh:  $(5 + 7) / (8 - 2) * (4 ^ 2) = 32$



Pohon ekspresi

- Algoritma *divide and conquer*:



```
if pohon tidak kosong then  
    nilai1 ← Evaluasi(upa-pohon kiri)  
    nilai2 ← Evaluasi(upa-pohon kanan)  
    Gabungkan nilai1 dan nilai2 dengan operatornya  
end
```

- Misalkan pohon ekspresi direpresentasikan dengan senarai berkait (*linked list*).

Simpul daun  $\rightarrow$  *operand*, contoh: 4, -2, 0, dst

Simpul dalam  $\rightarrow$  operator, contoh: +, -, \*, /, ^

Struktur setiap simpul: 

|      |      |       |
|------|------|-------|
| left | info | right |
|------|------|-------|

info: *operand* atau operator

Pada simpul daun  $\rightarrow$  left = NIL dan right = NIL

- Algoritma *divide and conquer*:

**if** simpul adalah daun **then**

**return** info

**else**

secara rekursif evaluasi upa-pohon kiri dan return nilainya

secara rekursif evaluasi upa-pohon kanan dan return nilainya

gabungkan kedua nilai tersebut sesuai dengan operator dan return nilainya

**endif**

- Algoritma evaluasi pohon ekspresi dalam bentuk prosedur:

**procedure** *EvaluasiPohon*(**input**  $T : \text{Pohon}$ , **output**  $\text{nilai} : \text{real}$ )

{ Mengevaluasi pohon ekspresi  $T$

Masukan: Pohon ekspresi  $T$ , asumsik  $T$  tidak kosong

Luaran: nilai berisi hasil evaluasi ekspresi

}

**Kamus**

$\text{nilai1}, \text{nilai2} : \text{real}$

Algoritma:

**if**  $\text{left}(T) = \text{NIL}$  **and**  $\text{right}(T) = \text{NIL}$  { simpul daun }

$\text{nilai} \leftarrow \text{info}(T)$

**else** { simpul dalam }

*EvaluasiPohon*( $\text{left}(T)$ ,  $\text{nilai1}$ );

*EvaluasiPohon*( $\text{right}(T)$ ,  $\text{nilai2}$ );

**case**  $\text{info}(T)$  **of**

“+” :  $\text{nilai} \leftarrow \text{nilai1} + \text{nilai2}$

“-” :  $\text{nilai} \leftarrow \text{nilai1} - \text{nilai2}$

“\*” :  $\text{nilai} \leftarrow \text{nilai1} * \text{nilai2}$

“/” :  $\text{nilai} \leftarrow \text{nilai1} / \text{nilai2}$  {dengan syarat  $\text{nilai2} \neq 0$  }

“^” :  $\text{nilai} \leftarrow \text{nilai1} ^ \text{nilai2}$  {dengan syarat  $\text{nilai1} \neq 0$  dan  $\text{nilai2} \neq 0$  }

**end**

**end**

- Algoritma evaluasi pohon ekspresi dalam bentuk fungsi:

**function** *EvaluasiPohon*( $T : Pohon$ )  $\rightarrow$  **real**

*{ mengevaluasi pohon ekspresi T }*

**Deklarasi**

*nilai1, nilai2 : real*

**Algoritma:**

**if**  $left(T) = \mathbf{NIL}$  **and**  $right(T) = \mathbf{NIL}$  *{ simpul daun }*

**return** *info*( $T$ )

**else** *{ simpul dalam }*

**case** *info*( $T$ ) **of**

“+” : **return** *EvaluasiPohon*( $left(T)$ , *nilai1*) + *EvaluasiPohon*( $right(T)$ , *nilai2*);

“-” : **return** *EvaluasiPohon*( $left(T)$ , *nilai1*) – *EvaluasiPohon*( $right(T)$ , *nilai2*);

“\*” : **return** *EvaluasiPohon*( $left(T)$ , *nilai1*) \* *EvaluasiPohon*( $right(T)$ , *nilai2*);

“/” : **return** *EvaluasiPohon*( $left(T)$ , *nilai1*) / *EvaluasiPohon*( $right(T)$ , *nilai2*); *{ nilai2  $\neq$  0 }*

“^” : **return** *EvaluasiPohon*( $left(T)$ , *nilai1*) ^ *EvaluasiPohon*( $right(T)$ , *nilai2*); *{ nilai1  $\neq$  0 dan nilai2  $\neq$  0 }*

**end**

**end**

BERSAMBUNG