

Menganalisa Algoritma

```
Procedure BubbleSort (input/output L: TabelInt, input n: integer)
{Mengurutkan table L[1..N] sehingga terurut menaik dengan metode pengurutan bubble sort.
Masukkan: Tabel L yang sudah terdefinisi nilai-nilainya
Keluaran: Tabel L yang terurut menaik sedemekian sehingga
          L[1] ≤ L[2] ≤ ⋯ ≤ L[N].
}
```

Deklarasi

```
i      Integer    (pencacah untuk jumlah Langkah)
k      Integer    (pencacah, untuk penjumlahan pada setiap langkah)
temp   Integer    (peubah bantu untuk pertukaran)
```

Algoritma

```
For i ← 1 to n-1 do
    For k ← n n downto i+1 do
        If L[k] =L[k-1] then
            {pertukaran L[k] dengan L[k-1]}
            temp ← L[k]
            L[k] ← L[k - 1]
            L[k - 1] ← temp
        endif
    endfor
endfor
```

Jika ada larik L dengan 10 buah elemen yang berisi angka-angka random:

7	10	16	13	4	12	3	81	75	26
1	2	3	4	5	6	7	8	9	10

Jelaskan bagaimana proses kerja tahapan algoritma di atas sampai dicapai angka yang berurutan

```
Array awal: [7, 10, 16, 13, 4, 12, 3, 81, 75, 26]
Iterasi ke-1
j = 1: [7, 10, 16, 13, 4, 12, 3, 81, 75, 26] (Tidak ada pertukaran)
j = 2: [7, 10, 16, 13, 4, 12, 3, 81, 75, 26] (Tidak ada pertukaran)
j = 3: [7, 10, 13, 16, 4, 12, 3, 81, 75, 26]
j = 3: [7, 10, 13, 16, 4, 12, 3, 81, 75, 26] (Tidak ada pertukaran)
j = 4: [7, 10, 13, 4, 16, 12, 3, 81, 75, 26]
j = 4: [7, 10, 13, 4, 16, 12, 3, 81, 75, 26] (Tidak ada pertukaran)
j = 5: [7, 10, 13, 4, 12, 16, 3, 81, 75, 26]
j = 5: [7, 10, 13, 4, 12, 16, 3, 81, 75, 26] (Tidak ada pertukaran)
j = 6: [7, 10, 13, 4, 12, 3, 16, 81, 75, 26]
j = 6: [7, 10, 13, 4, 12, 3, 16, 81, 75, 26] (Tidak ada pertukaran)
j = 7: [7, 10, 13, 4, 12, 3, 16, 81, 75, 26] (Tidak ada pertukaran)
j = 8: [7, 10, 13, 4, 12, 3, 16, 75, 81, 26]
j = 8: [7, 10, 13, 4, 12, 3, 16, 75, 81, 26] (Tidak ada pertukaran)
j = 9: [7, 10, 13, 4, 12, 3, 16, 75, 26, 81]
j = 9: [7, 10, 13, 4, 12, 3, 16, 75, 26, 81] (Tidak ada pertukaran)
Iterasi ke-2
j = 1: [7, 10, 13, 4, 12, 3, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 2: [7, 10, 13, 4, 12, 3, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 3: [7, 10, 4, 13, 12, 3, 16, 75, 26, 81]
j = 3: [7, 10, 4, 13, 12, 3, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 4: [7, 10, 4, 12, 13, 3, 16, 75, 26, 81]
j = 4: [7, 10, 4, 12, 13, 3, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 5: [7, 10, 4, 12, 3, 13, 16, 75, 26, 81]
j = 5: [7, 10, 4, 12, 3, 13, 16, 75, 26, 81] (Tidak ada pertukaran)
```

```

j = 6: [7, 10, 4, 12, 3, 13, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 7: [7, 10, 4, 12, 3, 13, 16, 75, 26, 81] (Tidak ada pertukaran)
j = 8: [7, 10, 4, 12, 3, 13, 16, 26, 75, 81]
j = 8: [7, 10, 4, 12, 3, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Iterasi ke-3
j = 1: [7, 10, 4, 12, 3, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 2: [7, 4, 10, 12, 3, 13, 16, 26, 75, 81]
j = 2: [7, 4, 10, 12, 3, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 3: [7, 4, 10, 12, 3, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 4: [7, 4, 10, 3, 12, 13, 16, 26, 75, 81]
j = 4: [7, 4, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 5: [7, 4, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 6: [7, 4, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 7: [7, 4, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Iterasi ke-4
j = 1: [4, 7, 10, 3, 12, 13, 16, 26, 75, 81]
j = 1: [4, 7, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 2: [4, 7, 10, 3, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 3: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81]
j = 3: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 4: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 5: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 6: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Iterasi ke-5
j = 1: [4, 7, 3, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 2: [4, 3, 7, 10, 12, 13, 16, 26, 75, 81]
j = 2: [4, 3, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 3: [4, 3, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 4: [4, 3, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 5: [4, 3, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Iterasi ke-6
j = 1: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81]
j = 1: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 2: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 3: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 4: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Iterasi ke-7
j = 1: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 2: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
j = 3: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81] (Tidak ada pertukaran)
Array terurut: [3, 4, 7, 10, 12, 13, 16, 26, 75, 81]

```

Menganalisis Algoritma

```

For i ← 1 to n do
For k ← n downto 1+1 do
    If data[k] < data[k-1] then
        temp ← data[k]
        data[k] ← data[k - 1]
        data[k - 1] ← temp
    endif
endfor
endfor

```

Berdasarkan larik data dengan 6 buah elemen yang berisi angka-angka yang random
(7,10,3,6,2,8)

Tuliskan hasil proses dari algoritma di atas pada setiap perulangan nilai i!

Data awal: [7, 10, 3, 6, 2, 8]	k = 4
Iterasi ke-1	2 < 6 TRUE (swap)
k = 5	Data sementara: [7, 10, 3, 2, 6, 8]
8 < 2 FALSE	k = 3
Data sementara: [7, 10, 3, 6, 2, 8]	2 < 3 TRUE (swap)

```

Data sementara: [7, 10, 2, 3, 6, 8]
k = 2
2 < 10 TRUE (swap)
Data sementara: [7, 2, 10, 3, 6, 8]
k = 1
2 < 7 TRUE (swap)
Data sementara: [2, 7, 10, 3, 6, 8]
Iterasi ke-2
k = 5
8 < 6 FALSE
Data sementara: [2, 7, 10, 3, 6, 8]
k = 4
6 < 3 FALSE
Data sementara: [2, 7, 10, 3, 6, 8]
k = 3
3 < 10 TRUE (swap)
Data sementara: [2, 7, 3, 10, 6, 8]
k = 2
3 < 7 TRUE (swap)
Data sementara: [2, 3, 7, 10, 6, 8]
Iterasi ke-3
k = 5

8 < 6 FALSE
Data sementara: [2, 3, 7, 10, 6, 8]
k = 4
6 < 7 TRUE (swap)
Data sementara: [2, 3, 7, 6, 10, 8]
k = 3
6 < 7 TRUE (swap)
Data sementara: [2, 3, 6, 7, 10, 8]
Iterasi ke-4
k = 5
8 < 10 TRUE (swap)
Data sementara: [2, 3, 6, 7, 8, 10]
k = 4
8 < 7 FALSE
Data sementara: [2, 3, 6, 7, 8, 10]
Iterasi ke-5
k = 5
10 < 8 FALSE
Data sementara: [2, 3, 6, 7, 8, 10]
Iterasi ke-6
Data akhir: [2, 3, 6, 7, 8, 10]

```

- ➊ Terdapat penugasan 4 orang ke 4 pekerjaan (job) dengan matriks biaya terlihat pada gambar di bawah. Setiap orang akan ditugasi mengerjakan satu pekerjaan. Persoalannya adalah bagaimana melakukan penugasan tersebut sehingga menghasilkan total biaya penugasan seminimal mungkin.

Job 1	Job 2	Job 3	Job 4	
9	2	7	8	Orang a
6	4	3	7	Orang b
5	8	1	4	Orang c
7	6	9	4	Orang d

- a) Bila menggunakan strategi Brute Force tentukan berapa banyak percobaan yang harus dicoba

$$\text{Complexity} = O(n!) = O(4!) = O(24)$$

- b) Bila menggunakan strategi greedy tentukan caranya dan tentukan hasilnya

Job 1	Job 2	Job 3	Job 4	
9	2	7	8	Orang a
6	4	3	7	Orang b
5	8	1	4	Orang c
7	6	9	4	Orang d

Untuk menyelesaikan permasalahan penugasan dengan menggunakan strategi greedy, kita dapat mengikuti langkah-langkah berikut:

- 1) Mengidentifikasi $e_{terkecil}$ dalam matriks biaya dan menempatkan orang pada pekerjaan dengan biaya terkecil

- 2) Menghapus baris dan kolom yang bersesuaian dengan penugasan tersebut
- 3) Ulangi Langkah 1-3 hingga semua orang telah ditugaskan

c -> Job 3 (biaya: 1)

a -> Job 2 (biaya: 2)

d -> Job 4 (biaya: 4)

b -> Job 1 (biaya: 6)

Total biaya penugasan: 13

 Terdapat penugasan 5 orang ke 5 pekerjaan (job) dengan matriks produktivitas terlihat pada gambar di bawah. Setiap orang hanya ditugasi satu pekerjaan. Persoalannya adalah bagaimana melakukan penugasan tersebut sehingga menghasilkan produktivitas **semaksimal mungkin**. Dengan menggunakan strategi Brute Force

	P1	P2	P3	P4	P5
Job1	8	7	4	9	5
Job2	4	2	4	5	7
Job3	9	5	8	7	8
Job4	4	3	7	6	7
Job5	6	7	4	6	5

- a) Tentukan berapa banyak percobaan yang harus dicoba (20 point)

$$\text{Complexity} = O(n!) = O(5!) = O(120)$$

- b) Tentukan nilai produktivitas maksimal yang didapat!

	P1	P2	P3	P4	P5
Job1	8	7	4	9	5
Job2	4	2	4	5	7
Job3	9	5	8	7	8
Job4	4	3	7	6	7
Job5	6	7	4	6	5

P4 -> Job1 (produktivitas: 9)

P5 -> Job2 (produktivitas: 7)

P1 -> Job3 (produktivitas: 9)

P3 -> Job4 (produktivitas: 7)

P2 -> Job5 (produktivitas: 7)

Produktivitas maksimum adalah: 39

Sebuah kotak dapat diisi dengan fraksi obyek-obyek. Kapasitas kotak adalah 30 kg, sedangkan 6 obyek yang akan dimasukkan masing-masing mempunyai massa (satuan kg) 5,12,15,18, 20, 25.

- a) Tentukan komposisi obyek-obyek yang dimasukkan ke dalam kotak sehingga total nilai obyek di dalamnya maksimum!

$$\sum n_{berat} = \text{capacity}$$

Komposisi obyek

{5,25}

{12,18}

- b) Tuliskan algoritma greedy untuk penyelesaian kasus di atas!

```
Procedure AssignJobs (input/output cost_matrix: 2D array of integers, input/output people: array of strings, input/output jobs: array of strings)
{Mengalokasikan pekerjaan kepada orang dengan biaya minimum menggunakan pendekatan greedy.
Masukkan: Matriks biaya yang berisi biaya penugasan setiap pekerjaan ke setiap orang,
Daftar orang dan pekerjaan yang belum ditugaskan.
Keluaran: Daftar penugasan yang berisi pasangan orang dan pekerjaan serta biaya penugasannya.}

Deklarasi
i      Integer (indeks untuk baris dalam matriks biaya)
j      Integer (indeks untuk kolom dalam matriks biaya)
min_costInteger (biaya minimum dalam matriks biaya)
assignments   array of tuples (daftar penugasan)

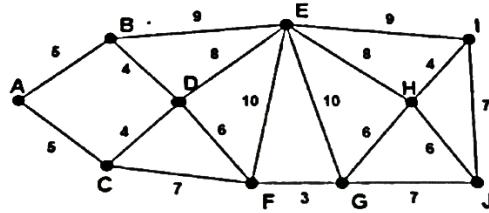
Algoritma
    While cost_matrix is not empty do
        {Menemukan biaya minimum dalam matriks biaya}
        min_cost ← minimum value in cost_matrix
        For i from 0 to length of cost_matrix do
            If min_cost is in cost_matrix[i] then
                j ← index of min_cost in cost_matrix[i]
                Break the loop
            Endif
        Endfor

        {Menambahkan penugasan ke dalam daftar}
        Add (people[i], jobs[j], min_cost) to assignments

        {Menghapus baris dan kolom yang bersesuaian dengan penugasan}
        Remove people[i] and jobs[j] from the list
        Remove i-th row and j-th column from cost_matrix
    Endwhile

    Return assignments
EndProcedure
```

Algoritma Minimum Spanning Tree (MST)



a. Buatlah MST dengan *Algoritma Kruskal* dan tuliskan algoritmanya pada Graph di atas

```

Procedure Kruskal (input G: graf, output T: pohon)
(Membentuk pohon merentang minimum T dari graf terhubung - berbobot G.
Masukan: graf-berbobot terhubung G=(V,E), dengan |v|=n
Keluaran: pohon rentang minimum T = (V,E')
Deklarasi
I,p,q,u,v: integer
Algoritma
(Asumsi: sisi-sisi dari graf sudah diurut menaik berdasarkan bobotnya - dari bobot kecil ke bobot besar)
T<={}
while jumlah sisi T<n-1 do
    pilih sisi (u,v) dari E yang bobotnya terkecil
    if (u,v) tidak membentuk siklus di T then
        T ← TU{(u,v)}
    Endif
Endfor

```

Langkah	Sisi	Bobot	Hutan Merentang
0			
1	(F,G)	3	
2	(B,D)	4	
3	(D,C)	4	
4	(H,I)	4	

5	(A,B)	5	
6	(D,F)	6	
7	(G,H)	6	
8	(H,J)	6	
9	(D,E)	8	

Sisi	(F,G)	(B,D)	(D,C)	(H,I)	(A,B)	(D,F)	(G,H)	(H,J)	(D,E)
Bobot	3	4	4	4	5	6	6	6	8

$$Bobot = 3 + 4 + 4 + 4 + 5 + 6 + 6 + 6 + 8 = 46$$

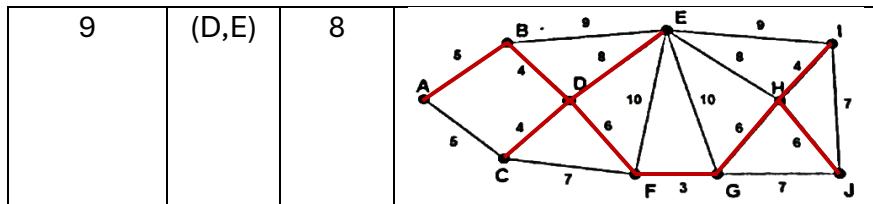
b. Buatlah MST dengan Algoritma Prim's dan tuliskan algoritmanya pada Graph di atas

```

Procedure Prim(input G: graf, output T: pohon)
{Memebentuk pohon merentang minimum T dari graf terhubung-berbobot G.
Masukan: graf-berbobot terhubung G=(V,E), dengan |v|=n
Keluaran: pohon rentang minimum T=(V,E')
}
Deklarasi
I, p, q, u, v : integer
Cari sisi (p,q) dari E yang berbobot terkecil
T  $\leftarrow \{(p,q)\}$ 
for i  $\leftarrow 1$  to n-1 do
    pilih ssi (u,v) dari E yang bobotnya terkecil namun bersisian dengan simpul T
    T  $\leftarrow TU\{(u,v)\}$ 
Endfor

```

Langkah	Sisi	Bobot	Pohon Rentang
0			
1	(F,G)	3	
2	(D,F)	6	
3	(B,D)	4	
4	(D,C)	4	
5	(A,B)	5	
6	(G,H)	6	
7	(H,I)	4	
8	(H,J)	6	



Sisi	(F,G)	(D,F)	(B,D)	(D,C)	(A,B)	(G,H)	(H,I)	(H,J)	(D,E)
Bobot	3	6	4	4	5	6	4	6	8

$$Bobot = 3 + 6 + 4 + 4 + 5 + 6 + 4 + 6 + 8 = 46$$

💡 Jika diketahui nilai bobot(W), keuntungan(b), Kapasitas knapsack W-6:

- N=4.
 - w1=3.5; b1=7
 - w2=2.5; b2=5
 - w3=5.5; b3=10
 - w4=4.5; b4=8
- a. Hitunglah keuntungan maksimum secara manual, menggunakan metode Brute force.

$$\text{Complexity} = O(2^n) = O(2^4) = 16$$

Kombinasi	Bobot	Keuntungan
Tidak ada item	0	0
Item 1	3.5	7
Item 2	2.5	5
Item 3	5.5	10
Item 4	4.5	8
Item 1, 2	6	12 (maksimum)
Item 1, 3	9 (Tidak Memenuhi)	17
Item 1, 4	8 (Tidak Memenuhi)	15
Item 2, 3	8 (Tidak Memenuhi)	15
Item 2, 4	7 (Tidak Memenuhi)	13
Item 3, 4	10 (Tidak Memenuhi)	18
Item 1, 2, 3	11.5 (Tidak Memenuhi)	22
Item 1, 2, 4	10.5 (Tidak Memenuhi)	20
Item 1, 3, 4	13.5 (Tidak Memenuhi)	25
Item 2, 3, 4	12.5 (Tidak Memenuhi)	23
Semua item	16 (Tidak Memenuhi)	30

b. Buatlah program soal di atas sehingga ditampilkan keuntungan maksimumnya.

```
from itertools import combinations
```

```
def knapsack_problem(weights, profits, W):
    comb = [combinations(range(len(weights)), r) for r in range(1, len(weights) + 1)]
```

```

max_profit, best_comb = max(((sum(profits[i] for i in c), c) for combo in comb for c in combo if
sum(weights[i] for i in c) <= W), default=(0, None))
best_items = [weights[i] for i in best_comb]
return max_profit, best_comb, best_items

weights = [3.5, 2.5, 5.5, 4.5]
profits = [7, 5, 10, 8]
knapsack_capacity = 6

max_profit, best_comb, best_items = knapsack_problem(weights, profits, knapsack_capacity)

print(f"Keuntungan maksimum: {max_profit}")
print(f"Kombinasi barang terbaik: {best_comb}")
print(f"Berat barang terbaik: {best_items}")
Output:
Keuntungan maksimum: 12
Kombinasi barang terbaik: (0, 1)
Berat barang terbaik: [3.5, 2.5]

```

💡 Jika diketahui table A berisi elemen-elemen berikut: 5 12 3 9 1 20 7 2

Urutkan table di atas menggunakan:

- Metode Divide and Conquer secara manual.

```

Urutkan array: [5, 12, 3, 9, 1, 20, 7, 2]
Pivot yang dipilih: 5
Elemen kurang dari atau sama dengan pivot: [3, 1, 2]
Elemen lebih besar dari pivot: [12, 9, 20, 7]
Menggabungkan [3, 1, 2], 5, dan [12, 9, 20, 7]
Urutkan array: [3, 1, 2]
Pivot yang dipilih: 3
Elemen kurang dari atau sama dengan pivot: [1, 2]
Elemen lebih besar dari pivot: []
Menggabungkan [1, 2], 3, dan []
Urutkan array: [1, 2]
Pivot yang dipilih: 1
Elemen kurang dari atau sama dengan pivot: []
Elemen lebih besar dari pivot: [2]
Menggabungkan [], 1, dan [2]
Urutkan array: []
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.
Urutkan array: [2]
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.
Array yang telah diurutkan: [1, 2]
Urutkan array: []
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.
Array yang telah diurutkan: [1, 2, 3]
Urutkan array: [12, 9, 20, 7]
Pivot yang dipilih: 12
Elemen kurang dari atau sama dengan pivot: [9, 7]
Elemen lebih besar dari pivot: [20]
Menggabungkan [9, 7], 12, dan [20]
Urutkan array: [9, 7]
Pivot yang dipilih: 9
Elemen kurang dari atau sama dengan pivot: [7]
Elemen lebih besar dari pivot: []
Menggabungkan [7], 9, dan []
Urutkan array: [7]
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.
Urutkan array: []
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.
Array yang telah diurutkan: [7, 9]
Urutkan array: [20]
Array berisi 1 atau kurang elemen, kembalikan seperti adanya.

```

```

Array yang telah diurutkan: [7, 9, 12, 20]
Array yang telah diurutkan: [1, 2, 3, 5, 7, 9, 12, 20]
[1, 2, 3, 5, 7, 9, 12, 20]
b. Buatlah program sehingga ditampilkan elemen-elemen di atas secara urut (Ascending).

def quick_sort(arr):

    if len(arr) <= 1:
        return arr

    else:

        pivot = arr[0]

        less_than_pivot = [x for x in arr[1:] if x <= pivot]
        greater_than_pivot = [x for x in arr[1:] if x > pivot]

        return quick_sort(less_than_pivot) + [pivot] + quick_sort(greater_than_pivot)

arr = [5, 12, 3, 9, 1, 20, 7, 2]
print(quick_sort(arr))

```

Kompleksitas Waktu'

Sebuah algoritma tidak saja harus benar sesuai spesifikasi persoalan tetapi juga harus sangkil efisien

Algoritma yang bagus adalah algoritma yang sangkil efficient

Kesangkilan algoritma diukur dari waktu time yang diperlukan untuk menjalankan algoritma dan ruang space memori yang dibutuhkan oleh algoritma tersebut

Algoritma yang sangkil ialah algoritma yang meminimumkan kebutuhan waktu dan ruang memori

Kebutuhan waktu dan ruang memori suatu algoritma bergantung pada ukuran masukan n yang menyatakan ukuran data yang diproses oleh algoritma

Kesangkilan algoritma dapat digunakan untuk menilai algoritma yang bagus dari sejumlah algoritma penyelesaian persoalan

Sebab sebuah persoalan dapat memiliki banyak algoritma penyelesaian Contoh persoalan pengurutan sort ada puluhan algoritma pengurutan sort insertion sort bubble sort dll

Model Perhitungan Kebutuhan Waktu

Menghitung kebutuhan waktu algoritma dengan mengukur waktu eksekusi riil nya dalam satuan detik ketika program (yang merepresentasikan sebuah algoritma dijalankan oleh komputer bukanlah cara yang tepat

Alasan

Setiap komputer dengan arsitektur berbeda memiliki bahasa mesin yang

berbeda

waktu setiap operasi antara satu komputer dengan komputer lain tidak sama

Compiler bahasa pemrograman yang berbeda menghasilkan kode Bahasa mesin yang berbeda

waktu setiap operasi antara compiler dengan compiler lain tidak sama

Kompleksitas waktu, $T(n)$, diukur dari jumlah tahapan komputasi yang dilakukan di dalam algoritma sebagai fungsi dari ukuran masukan n .

Kompleksitas ruang, $S(n)$, diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari ukuran masukan n .

Dengan menggunakan besaran kompleksitas waktu/ruang algoritma, kita dapat menentukan laju peningkatan waktu (ruang) yang diperlukan algoritma dengan meningkatnya ukuran masukan n .

Di dalam kuliah ini kita hanya membatasi bahasan kompleksitas waktu saja, karena dua alasan:

- 1) Materi struktur data diluar lingkup mata kuliah matematika diskrit
- 2) Saat ini memori komputer bukan persoalan yang kritis dibandingkan waktu

Pekerjaan utama di dalam kompleksitas waktu adalah menghitung (counting) jumlah tahapan komputasi di dalam algoritma .

Jumlah tahapan komputasi dihitung dari berapa kali suatu operasi dilakukan sebagai fungsi ukuran masukan (n).

Di dalam sebuah algoritma terdapat banyak jenis operasi:

- Operasi baca/tulis (input a, print a)
- Operasi aritmetika (+, -, *, /) (a + b, M * N)
- Operasi pengisian nilai (assignment) (a <- 10)
- Operasi perbandingan (a < b, k >= 10)
- Operasi pengaksesan elemen larik, pemanggilan prosedur/fungsi, dll

Untuk menyederhanakan perhitungan, kita tidak menghitung semua jenis operasi, tetapi kita hanya menghitung jumlah operasi khas (tipikal) yang mendasari suatu algoritma.

Kategorisasi Kompleksitas Waktu

- $O(1)$ - Konstan: tidak bergantung pada ukuran masukan.
- $O(\log n)$ - Logaritmik: seperti algoritma pencarian biner.
- $O(n)$ - Linier: seperti pencarian linier, memproses setiap elemen masukan.
- $O(n \log n)$ - Linier logaritmik: divide-and-conquer, seperti merge sort.
- $O(n^2)$ - Kuadratik: seperti selection sort, insertion sort.
- $O(n^3)$ - Kubik: seperti perkalian matriks.
- $O(2^n)$ - Eksponensial: seperti mencari solusi dengan mencoba semua kemungkinan.
- $O(n!)$ - Faktorial: seperti algoritma persoalan pedagang keliling.

Made with by Rendie

Baik, berikut penjelasan yang lebih rinci tentang analisis kompleksitas waktu algoritma:

1. Memahami Konsep Dasar

- Kompleksitas waktu adalah analisis berapa lama waktu yang dibutuhkan oleh sebuah algoritma untuk menyelesaikan tugasnya sebagai fungsi dari ukuran masukan.
- Notasi asimptotik digunakan untuk menggambarkan kompleksitas waktu untuk ukuran masukan yang besar (mendekati tak hingga).

2. Notasi Big-O (O-besar)

- Notasi O-besar memberikan batas atas (upper bound) dari kompleksitas waktu algoritma.
- Jika $T(n) = O(f(n))$, artinya ada konstanta C dan nilai n_0 sehingga $T(n) \leq C.f(n)$ untuk setiap $n \geq n_0$.

- Contoh:

$$T(n) = 2n^2 + 6n + 1.$$

Karena $2n^2 + 6n + 1 \leq 9n^2$ untuk $n \geq 1$ ($C = 9$, $f(n) = n^2$, $n_0 = 1$), maka $T(n) = O(n^2)$.

- Mencari nilai maksimum dari n elemen memiliki $T(n) = n - 1 = O(n)$.

3. Notasi Big-Omega (Ω -besar)

- Notasi Ω -besar memberikan batas bawah (lower bound) dari kompleksitas waktu algoritma.
- Jika $T(n) = \Omega(g(n))$, artinya ada konstanta C dan nilai n_0 sehingga $T(n) \geq C.g(n)$ untuk setiap $n \geq n_0$.
- Contoh: $T(n) = 2n^2 + 6n + 1 = \Omega(n^2)$, karena $2n^2 + 6n + 1 \geq 2n^2$ untuk $n \geq 1$.

Notasi O-Besar (Big-O)

- Notasi "O" disebut notasi "O-Besar" (Big-O) yang merupakan notasi kompleksitas waktu asimptotik.

- DEFINISI 1. $T(n) = O(f(n))$ (dibaca " $T(n)$ adalah $O(f(n))$ ", yang artinya $T(n)$ berorde paling besar $f(n)$) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C f(n)$$

untuk $n \geq n_0$.

DEFINISI. $T(n) = O(f(n))$ (dibaca " $T(n)$ adalah $O(f(n))$ " yang artinya $T(n)$ berorde paling besar $f(n)$) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C f(n)$$

untuk $n \geq n_0$.

- Catatan: Ada tak-berhingga nilai C dan n_0 yang memenuhi $T(n) \leq C f(n)$, kita cukup menunjukkan satu pasang (C, n_0) yang memenuhi definisi sehingga $T(n) = O(f(n))$

Contoh 7. Tunjukkan bahwa $2n^2 + 6n + 1 = O(n^2)$. (tanda '=' dibaca 'adalah')

Penyelesaian:

$$2n^2 + 6n + 1 = O(n^2) \text{ karena}$$

$$2n^2 + 6n + 1 \leq 2n^2 + 6n^2 + n^2 = 9n^2 \text{ untuk semua } n \geq 1 \quad (C=9, f(n) = n^2, n_0 = 1).$$

atau karena

$$2n^2 + 6n + 1 \leq n^2 + n^2 + n^2 = 3n^2 \text{ untuk semua } n \geq 7 \quad (C=3, f(n) = n^2, n_0 = 7).$$

DEFINISI. $T(n) = O(f(n))$ (dibaca " $T(n)$ adalah $O(f(n))$ " yang artinya $T(n)$ berorde paling besar $f(n)$) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C f(n)$$

untuk $n \geq n_0$.

DEFINISI. $T(n) = O(f(n))$ (dibaca " $T(n)$ adalah $O(f(n))$ " yang artinya $T(n)$ berorde paling besar $f(n)$) bila terdapat konstanta C dan n_0 sedemikian sehingga

$$T(n) \leq C f(n)$$

untuk $n \geq n_0$.

Contoh-contoh Lain

1. Tunjukkan bahwa $5 = O(1)$.

Jawaban:

$$5 = O(1) \text{ karena } 5 \leq 6 \cdot 1 \text{ untuk } n \geq 1 \quad (C = 6, f(n) = 1, \text{ dan } n_0 = 1)$$

Kita juga dapat memperlihatkan bahwa

$$5 = O(1) \text{ karena } 5 \leq 10 \cdot 1 \text{ untuk } n \geq 1 \quad (C = 10, f(n) = 1, \text{ dan } n_0 = 1)$$

Contoh 8. Tunjukkan bahwa $3n + 2 = O(n)$.

Penyelesaian:

$$3n + 2 = O(n)$$

karena

$$3n + 2 \leq 3n + 2n = 5n \text{ untuk semua } n \geq 1$$

$$(C = 5, f(n) = n, \text{ dan } n_0 = 1).$$

3. Tunjukkan $6 \cdot 2^n + 2n^2 = O(2^n)$

Jawaban:

$$6 \cdot 2^n + 2n^2 = O(2^n)$$

karena

$$6 \cdot 2^n + 2n^2 \leq 6 \cdot 2^n + 2 \cdot 2^n = 8 \cdot 2^n$$

$$\text{untuk semua } n \geq 4 \quad (C = 8, f(n) = 2^n, \text{ dan } n_0 = 4).$$

Jawaban:

$$\frac{n(n-1)}{2} = O(n^2)$$

karena

$$\frac{n(n-1)}{2} \leq \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2$$

untuk $n \geq 1$

$$(C = 1, f(n) = n^2, \text{ dan } n_0 = 1).$$

4. Tunjukkan $1 + 2 + \dots + n = O(n^2)$

Jawaban:

Cara 1: $1 + 2 + \dots + n \leq n + n + \dots + n = n^2$ untuk $n \geq 1$

Cara 2: $1 + 2 + \dots + n = \frac{1}{2}n(n+1) \leq \frac{1}{2}n^2 + \frac{1}{2}n^2 = n^2$ untuk $n \geq 1$

5. Tunjukkan $n! = O(n^n)$

Jawaban:

$n! = 1 \cdot 2 \cdot \dots \cdot n \leq n \cdot n \cdot \dots \cdot n = n^n$ untuk $n \geq 1$

Teorema 1: Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_1 n + a_0$ adalah polinom derajat $\leq m$ maka $T(n) = O(n^m)$.

- Jadi, untuk menentukan notasi Big-Oh, cukup melihat suku (term) yang mempunyai pangkat terbesar di dalam $T(n)$.

• Contoh 8:

$$T(n) = 5 = 5n^0 = O(n^0) = O(1)$$

$$T(n) = 2n + 3 = O(n)$$

$$T(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

$$T(n) = 3n^3 + 2n^2 + 10 = O(n^3)$$

Tunjukkan bahwa $T(n) = 5n^2 = O(n^3)$, tetapi $T(n) = n^3 \neq O(n^2)$.

Jawaban:

- $5n^2 = O(n^3)$ karena $5n^2 \leq n^3$ untuk semua $n \geq 5$.

- Tetapi, $T(n) = n^3 \neq O(n^2)$ karena tidak ada konstanta C dan n_0 sedemikian sehingga $n^3 \leq Cn^2 \Leftrightarrow n \leq C$ untuk semua n_0 karena n dapat berupa sembarang bilangan yang besar.

• Menuliskan

$O(2n)$ tidak standard, seharusnya $O(n)$

$O(n-1)$ tidak standard, seharusnya $O(n)$

$O(\frac{n^2}{2})$ tidak standard, seharusnya $O(n^2)$

$O((n-1)!)$ tidak standard, seharusnya $O(n!)$

- Ingat, di dalam notasi Big-Oh tidak ada koefisien atau suku-suku lainnya, hanya berisi fungsi-fungsi standard seperti $1, n^2, n^3, \dots, \log n, n \log n, 2^n, n!$, dan sebagainya

Contoh 11: Tentukan notasi O -besar untuk $T(n) = (n+1)\log(n^2+1) + 3n^2$.

Jawaban:

Cara 1: • $n+1 = O(n)$

- $\log(n^2+1) \leq \log(2n^2) = \log(2) + \log(n^2)$
 $= \log(2) + 2\log(n)$
 $\leq \log(n) + 2\log(n) = 3\log(n)$ untuk $n \geq 2$
 $= O(\log n)$
- $(n+1)\log(n^2+1) = O(n)O(\log n) = O(n \log n)$
- $3n^2 = O(n^2)$
- $(n+1)\log(n^2+1) + 3n^2 = O(n \log n) + O(n^2) = O(\max(n \log n, n^2)) = O(n^2)$

Cara 2: suku yang dominan di dalam $(n+1)\log(n^2+1) + 3n^2$ untuk n yang besar adalah $3n^2$, sehingga $(n+1)\log(n^2+1) + 3n^2 = O(n^2)$

6. Tunjukkan $\log n! = O(n \log n)$

Jawaban:

Dari soal 5 sudah diperoleh bahwa $n! \leq n^n$ untuk $n \geq 1$ maka
 $\log n! \leq \log n^n = n \log n$ untuk $n \geq 1$ maka
sehingga $\log n! = O(n \log n)$

7. Tunjukkan $8n^2 = O(n^3)$

Jawaban:

$8n^2 = O(n^3)$ karena $8n^2 \leq n^3$ untuk $n \geq 8$

- Teorema 1 tersebut digeneralisasi untuk suku-suku dominan lainnya:

- Eksponensial mendominasi sembarang perpangkatan (yaitu, $y^n > n^p$, $y > 1$)
- Perpangkatan mendominasi $\ln(n)$ (yaitu $n^p > \ln n$)
- Sebuah logaritma tumbuh pada laju yang sama (yaitu $a \log(n) = b \log(n)$)
- $n \log n$ tumbuh lebih cepat daripada n tetapi lebih lambat daripada n^2

Contoh 9: $T(n) = 2^n + 2n^2 = O(2^n)$.

$$T(n) = 2n \log(n) + 3n = O(n \log n)$$

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

$$T(n) = 2n \log n + 3n^2 = O(n^2)$$

- Definisi: $T(n) = O(f(n))$ jika terdapat C dan n_0 sedemikian sehingga $T(n) \leq C f(n)$ untuk $n \geq n_0$

→ tidak menyiratkan seberapa atas fungsi f itu.

- Jadi, menyatakan bahwa

$$T(n) = 2n^2 = O(n^2) \rightarrow \text{benar}$$

$$T(n) = 2n^2 = O(n^3) \rightarrow \text{juga benar, karena } 2n^2 \leq 2n^3 \text{ untuk } n \geq 1$$

$$T(n) = 2n^2 = O(n^4) \rightarrow \text{juga benar, karena } 2n^2 \leq 2n^4 \text{ untuk } n \geq 1$$

- Namun, untuk alasan praktikal kita memilih fungsi yang sekecil mungkin agar $O(f(n))$ memiliki makna

- Jadi, kita menulis $2n^2 = O(n^2)$, bukan $O(n^3)$ atau $O(n^4)$

TEOREMA 2. Misalkan $T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$, maka

$$(a) T_1(n) + T_2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

$$(b) T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$$

$$(c) O(cf(n)) = O(f(n)), c \text{ adalah konstanta}$$

$$(d) f(n) = O(f(n))$$

Contoh 9. Misalkan $T_1(n) = O(n)$ dan $T_2(n) = O(n^2)$, maka

$$(a) T_1(n) + T_2(n) = O(\max(n, n^2)) = O(n^2)$$

$$(b) T_1(n)T_2(n) = O(nn^2) = O(n^3)$$

Contoh 10. $O(5n^2) = O(n^2)$

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

Contoh 3:

```
for i ← 1 to n do
    for j ← 1 to i do
        a ← a + 1      O(1)
        b ← b - 2      O(1)
    endfor
endfor
```

Kompleksitas untuk $a \leftarrow a + 1$	$= O(1)$
Kompleksitas untuk $b \leftarrow b - 2$	$= O(1)$
Kompleksitas total keduanya	$= O(1) + O(1) = O(1)$
Jumlah pengulangan seluruhnya	$= 1 + 2 + \dots + n = n(n+1)/2$
Kompleksitas seluruhnya	$= n(n+1)/2 O(1) = O(n(n+1)/2)$ $= O(n^2/2 + n/2)$ $= O(n^2)$

1. Apa yang dimaksud dengan algoritma brute force?
 - a) Algoritma yang menggunakan kecerdasan buatan untuk menyelesaikan masalah
 - b) Algoritma yang mencari solusi optimal secara langsung
 - c) **Algoritma yang mencoba semua kemungkinan solusi secara sistematis**
 - d) Algoritma yang hanya dapat digunakan untuk masalah kecil
2. Ketika menggunakan algoritma brute force, apa yang terjadi jika jumlah kemungkinan solusi sangat besar?
 - a) Algoritma akan memberikan solusi optimal
 - b) Algoritma akan berhenti dan tidak memberikan solusi
 - c) **Algoritma akan memakan waktu yang sangat lama untuk mencari solusi**
 - d) Algoritma akan memberikan solusi yang salah
3. Apa kelemahan utama dari algoritma brute force?
 - a) Memerlukan perangkat keras yang kuat
 - b) Memerlukan pemrogram yang sangat ahli
 - c) **Memerlukan waktu yang sangat lama untuk menyelesaikan masalah dengan ukuran yang besar**
 - d) Tidak dapat menyelesaikan masalah apapun
4. Bagaimana cara mengidentifikasi apakah algoritma brute force merupakan solusi yang tepat untuk suatu masalah?
 - a) Dengan mencoba algoritma brute force terlebih dahulu
 - b) **Dengan menganalisis kompleksitas waktu algoritma brute force**
 - c) Dengan memilih algoritma brute force jika tidak ada solusi yang lain
 - d) Dengan menggunakan algoritma brute force untuk setiap masalah
5. Kapan algoritma brute force biasanya digunakan?
 - a) Ketika solusi optimal diperlukan dengan cepat
 - b) Ketika masalah memiliki jumlah kemungkinan solusi yang terbatas
 - c) Ketika tidak ada algoritma yang lain yang dapat digunakan
 - d) **Ketika memerlukan solusi untuk masalah dengan jumlah kemungkinan solusi yang kecil**
6. Apa perbedaan utama antara algoritma brute force dan greedy?
 - a) **Algoritma brute force mencari solusi optimal, sedangkan greedy mencari solusi yang cukup baik**
 - b) Algoritma greedy mencoba semua kemungkinan solusi, sedangkan brute force hanya mencoba solusi yang terdekat
 - c) Algoritma brute force lebih cepat daripada greedy
 - d) Tidak ada perbedaan antara keduanya
7. Bagaimana kompleksitas waktu dari algoritma brute force dan greedy?
 - a) Brute force memiliki kompleksitas waktu yang lebih rendah daripada greedy
 - b) **Greedy memiliki kompleksitas waktu yang lebih rendah daripada brute force**
 - c) Keduanya memiliki kompleksitas waktu yang sama
 - d) Tergantung pada masalah yang diselesaikan
8. Ketika memilih solusi, algoritma brute force akan:
 - a) **Memilih solusi yang optimal pada setiap langkah**
 - b) Memilih solusi yang terlihat paling baik pada setiap langkah
 - c) Memilih solusi yang paling sederhana pada setiap langkah
 - d) Memilih solusi secara acak pada setiap langkah
9. Apa yang dilakukan oleh algoritma greedy ketika memilih solusi?
 - a) Menimbang semua kemungkinan solusi
 - b) Memilih solusi yang paling optimal secara global pada setiap langkah
 - c) **Memilih solusi yang paling baik secara lokal pada setiap langkah**
 - d) Memilih solusi yang paling kompleks pada setiap langkah
10. Algoritma brute force lebih cocok digunakan ketika:
 - a) **Masalah memiliki jumlah kemungkinan solusi yang sangat besar**
 - b) Masalah memiliki jumlah kemungkinan solusi yang terbatas
 - c) Masalah hanya memiliki satu solusi yang mungkin
 - d) Masalah memiliki solusi yang tidak terdefinisi
11. Dalam hal kecepatan, algoritma greedy biasanya lebih cepat daripada brute force karena:
 - a) Greedy hanya mempertimbangkan solusi yang terbaik pada setiap langkah
 - b) Greedy tidak perlu melakukan backtracking
 - c) Greedy tidak perlu mempertimbangkan semua kemungkinan solusi
 - d) **Semua jawaban di atas benar**
12. Algoritma greedy cenderung menghasilkan solusi yang optimal global?
 - a) Ya, karena selalu memilih solusi terbaik pada setiap langkah
 - b) **Tidak, karena hanya mempertimbangkan keuntungan lokal pada setiap langkah**
 - c) Tergantung pada kompleksitas masalah
 - d) Hanya dalam beberapa kasus khusus
13. Dalam hal kompleksitas ruang (space complexity), algoritma brute force biasanya:
 - a) Memiliki kompleksitas ruang yang lebih rendah daripada greedy
 - b) **Memiliki kompleksitas ruang yang lebih tinggi daripada greedy**
 - c) Keduanya memiliki kompleksitas ruang yang sama
 - d) Tergantung pada implementasi masing-masing algoritma
14. Kapan algoritma greedy biasanya digunakan?
 - a) Ketika solusi optimal diperlukan
 - b) Ketika masalah memiliki jumlah kemungkinan solusi yang besar
 - c) **Ketika solusi yang cukup baik sudah memadai**
 - d) Ketika masalah memerlukan semua kemungkinan solusi untuk dijelajahi
15. Algoritma brute force sering digunakan dalam konteks:
 - a) Kriptografi
 - b) Perutean (Routing)
 - c) Optimasi fungsi matematika
 - d) **Semua jawaban di atas benar**
16. Apa yang menjadi fokus utama dari algoritma Divide and Conquer?
 - a) Mengurangi masalah menjadi ukuran yang lebih kecil pada setiap iterasi
 - b) **Memecah masalah menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah, dan menggabungkan solusinya**
 - c) Mengurangi kompleksitas waktu dengan mengurangi jumlah langkah yang diperlukan untuk menyelesaikan masalah
 - d) Tidak ada perbedaan antara algoritma Divide and Conquer dengan Decrease and Conquer

17. Apa prinsip dasar dari algoritma Decrease and Conquer?
- Membagi masalah menjadi submasalah yang lebih kecil, menyelesaikan setiap submasalah, dan menggabungkan solusinya
 - Mengurangi masalah menjadi ukuran yang lebih kecil pada setiap iterasi
 - Mencari pola di dalam masalah dan mengurangi ukurannya secara berurutan**
 - Tidak ada perbedaan antara keduanya, keduanya mengacu pada prinsip yang sama
18. Pada tahap mana algoritma Divide and Conquer membagi masalah menjadi submasalah yang lebih kecil?
- Tahap "divide"**
 - Tahap "conquer"
 - Tahap "combine"
 - Tahap "merge"
19. Pada tahap mana algoritma Decrease and Conquer mengurangi masalah menjadi ukuran yang lebih kecil?
- Tahap "divide"
 - Tahap "conquer"
 - Tahap "reduce"**
 - Tahap "merge"
20. Algoritma Decrease and Conquer cenderung menyelesaikan masalah dengan cara
- Memecah masalah menjadi submasalah yang lebih kecil
 - Mengurangi masalah menjadi ukuran yang lebih kecil pada setiap iterasi
 - Menggabungkan solusi dari submasalah yang lebih kecil
 - Menyelesaikan masalah secara langsung tanpa membaginya**
21. Algoritma Divide and Conquer lebih cocok digunakan untuk masalah yang memiliki:
- Struktur hierarkis dan dapat didekomposisi ke dalam submasalah yang lebih kecil**
 - Masalah yang dapat dipecah menjadi bagian-bagian yang berbeda
22. Proses reduksi dalam algoritma Decrease and Conquer dilakukan pada tahap:
- "divide"
 - "conquer"
 - "reduce"**
 - "combine"
23. Dalam algoritma Decrease and Conquer, proses "decrease" mengacu pada:
- Membagi masalah menjadi submasalah yang lebih kecil
 - Mengurangi masalah menjadi ukuran yang lebih kecil pada setiap iterasi**
 - Menyelesaikan setiap submasalah
 - Menggabungkan solusi dari submasalah yang lebih kecil
24. Algoritma Divide and Conquer sering memiliki kompleksitas waktu yang:
- Lebih rendah daripada Decrease and Conquer
 - Lebih tinggi daripada Decrease and Conquer**
 - Sama dengan Decrease and Conquer
 - Bergantung pada masalah yang dihadapi
25. Algoritma Decrease and Conquer cenderung lebih efisien daripada Divide and Conquer dalam kasus-kasus di mana:
- Masalah dapat dibagi menjadi bagian-bagian yang sama besar
 - Masalah memiliki struktur hierarkis yang kompleks
 - Masalah memiliki pola yang jelas dan dapat ditempatkan dalam Langkah langkah diskrit**
 - Tidak ada perbedaan efisiensi antara keduanya

BRUTE FORCE

Pendahuluan

- Algoritma Brute Force adalah pendekatan lurus (straightforward) untuk memecahkan suatu persoalan.
- Didasarkan pada pernyataan masalah (problem statement) dan definisi/konsep yang terlibat.
- Memecahkan masalah dengan sangat sederhana, langsung, dan jelas caranya.

Contoh Algoritma Brute Force

- **Mencari elemen terbesar/terkecil dalam larik**

```
procedure CariElemenTerbesar(input a1, a2, ..., an : integer, output maks : integer)
    maks ← a[1]
    for k ← 2 to n do
        if a[k] > maks then
            maks ← a[k]
        endif
    endfor
endprocedure
```

- **Pencarian beruntun (sequential search)**

```
procedure PencarianBeruntun(input a1, a2, ..., an : integer, x : integer, output idx : integer)
    k ← 1
    while (k <= n) and (ak != x) do
        k ← k + 1
    endwhile
    if ak = x then
        idx ← k
    else
        idx ← -1
    endif
endprocedure
```

- **Menghitung perpangkatan (a^n)**

```
function pangkat(a : real, n : integer) → real
    hasil ← 1
    for i ← 1 to n do
        hasil ← hasil * a
    endfor
    return hasil
endfunction
```

- **Menghitung faktorial ($n!$)**

```
function faktorial(n : integer) → integer
    fak ← 1
    for k ← 1 to n do
        fak ← fak * k
    endfor
    return fak
endfunction
```

- **Perkalian matriks**

```
procedure PerkalianMatriks(input A, B : Matriks, input n : integer, output C : Matriks)
    for i ← 1 to n do
        for j ← 1 to n do
            C[i, j] ← 0
            for k ← 1 to n do
                C[i, j] ← C[i, j] + A[i, k]*B[k, j]
```

```

        endfor
    endfor
endfor
endprocedure

```

- **Uji bilangan prima**

```

function IsPrima(n : integer) → boolean
    if n < 2 then
        return false
    else
        test ← true
        k ← 2
        while test and (k <= √n) do
            if n mod k = 0 then
                test ← false
            else
                k ← k + 1
            endif
        endwhile
        return test
    endif
endfunction

```

- **Pengurutan dengan selection sort dan bubble sort**

```

procedure SelectionSort(input/output s1, s2, ..., sn : integer)
    for i ← 1 to n-1 do
        imin ← i
        for j ← i+1 to n do
            if s[j] < s[imin] then
                imin ← j
            endif
        endfor
        temp ← s[i]
        s[i] ← s[imin]
        s[imin] ← temp
    endfor
endprocedure

```

```

procedure BubbleSort(input/output s1, s2, ..., sn : integer, input n : integer)
    for i ← n-1 downto 1 do
        for k ← 1 to i do
            if s[k+1] < s[k] then
                temp ← s[k]
                s[k] ← s[k+1]
                s[k+1] ← temp
            endif
        endfor
    endfor
endprocedure

```

- **Mengevaluasi nilai polinom**

```

function polinom(t : real) → real
    p ← 0
    for i ← n downto 0 do
        pangkat ← 1
        for j ← 1 to i do

```

```

    pangkat ← pangkat * t
endfor
p ← p + a[i] * pangkat
endfor
return p
• endfunction

```

- **Pencocokan string**

```

function PencocokanString(input P : string, T : string, m, n : integer, output idx : integer)
    i ← 0
    ketemu ← false
    while (i <= n-m) and (not ketemu) do
        j ← 1
        while (j <= m) and (Pj = Ti+j) do
            j ← j + 1
        endwhile
        if j > m then
            ketemu ← true
        else
            i ← i + 1
        endif
    endwhile
    if ketemu then
        return i+1
    else
        return -1
    endif
endfunction

```

Karakteristik Algoritma Brute Force

- Umumnya tidak "cerdas" dan tidak efisien, membutuhkan komputasi besar
- Lebih cocok untuk persoalan dengan ukuran masukan kecil
- Hampir semua persoalan dapat diselesaikan dengan brute force

Kekuatan dan Kelemahan Algoritma Brute Force

Kekuatan:

- Dapat diterapkan untuk hampir semua masalah
- Sederhana dan mudah dimengerti
- Menghasilkan algoritma standar untuk tugas komputasi dasar

Kelemahan:

- Jarang menghasilkan algoritma efisien
- Lambat untuk masukan berukuran besar
- Tidak sekonstruktif strategi pemecahan masalah lainnya

Contoh Aplikasi

- Sudoku
- Cryptarithmetic
- Permainan 24
- Teka-teki silang (crossword puzzle)
- Kakurasu
- Futoshiki
- Kakuro

Exhaustive Search

- Teknik pencarian solusi secara brute force untuk persoalan kombinatorik
- Mengenumerasi semua kemungkinan solusi, evaluasi satu per satu, simpan solusi terbaik
- Contoh: Travelling Salesperson Problem (TSP), 0/1 Knapsack Problem

Teknik Heuristik

- Teknik untuk mempercepat pencarian solusi pada exhaustive search
- Mengeliminasi beberapa kemungkinan solusi tanpa mengeksplorasi seluruhnya
- Menggunakan pendekatan tidak formal, intuisi, terkaan, atau common sense
- Tidak menjamin solusi optimal, tapi seringkali cukup baik dan lebih cepat
- Banyak digunakan di bidang Kecerdasan Buatan

Aplikasi Teknik Heuristik

- Anagram
- Kubus ajaib 3x3
- 8-puzzle

ALGORITMA GREEDY

Pendahuluan

- Algoritma greedy merupakan metode yang populer dan sederhana untuk memecahkan persoalan optimasi.
- Persoalan optimasi terdiri dari dua jenis: maksimasi dan minimasi.
- Algoritma greedy membentuk solusi langkah per langkah dengan memilih optimum lokal pada setiap langkah, berharap akan mengarah ke optimum global.

Definisi Algoritma Greedy

- Pada setiap langkah, algoritma greedy mengambil pilihan terbaik yang dapat diperoleh saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "take what you can get now!").
- Elemen-elemen algoritma greedy: himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, fungsi objektif.

Contoh Persoalan dengan Algoritma Greedy

- Persoalan penukaran uang (coin exchange problem)
- Persoalan memilih aktivitas (activity selection problem)
- Minimisasi waktu di dalam sistem
- Persoalan knapsack (knapsack problem)
- Penjadwalan job dengan tenggat waktu (job scheduling with deadlines)

Persoalan Penukaran Uang

- Strategi greedy: Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa.
- Tidak selalu memberikan solusi optimal untuk semua kasus.

```
function CoinExchange(C: set of coins, A: integer) → set of coins
    S ← {}
    while ((sum of coin values in S) < A) and (C ≠ {}) do
        x ← coin with the largest value in C
        C ← C - {x}
        if ((sum of coin values in S) + value of x ≤ A) then
            S ← S ∪ {x}
        end if
    end while
```

```

if ((sum of coin values in S) = A) then
    return S
else
    print("No solution")
end if
end function

```

Persoalan Memilih Aktivitas

- Strategi greedy: Urutkan aktivitas berdasarkan waktu selesai, lalu pilih aktivitas dengan waktu mulai terbesar yang tidak bertabrakan dengan aktivitas terpilih sebelumnya.
- Algoritma greedy menghasilkan solusi optimal.

```

function GreedyActivitySelector(start_times, finish_times) → set of activities
    n ← length(start_times)
    A ← {1} // Activity 1 is always selected
    j ← 1
    for i ← 2 to n do
        if start_times[i] ≥ finish_times[j] then
            A ← A ∪ {i}
            j ← i
        end if
    end for
    return A
end function

```

Minimisasi Waktu di dalam Sistem

- Strategi greedy: Pada setiap langkah, pilih pelanggan dengan waktu pelayanan terkecil.
- Algoritma greedy menghasilkan solusi optimal jika pelanggan diurutkan berdasarkan waktu pelayanan yang menaik.

```

function ScheduleCustomers(C: set of customers) → sequence of customers
    S ← {}
    while C ≠ {} do
        i ← customer with the smallest service time in C
        C ← C - {i}
        S ← S ∪ {i}
    end while
    return S
end function

```

Persoalan Knapsack

- Integer Knapsack: Strategi greedy tidak selalu memberikan solusi optimal.
- Fractional Knapsack: Strategi greedy dengan memilih objek berdasarkan densitas terbesar selalu memberikan solusi optimal.

```

function FractionalKnapsack(C: set of objects, K: real) → solution vector
    for i ← 1 to n do
        x[i] ← 0 // Initialize solution vector
    end for
    i ← 0
    total_weight ← 0
    can_fit_whole ← true
    while (i < n) and can_fit_whole do
        i ← i + 1
        if total_weight + weights[i] ≤ K then

```

```

x[i] ← 1 // Include the entire object
total_weight ← total_weight + weights[i]
else
    can_fit_whole ← false
    x[i] ← (K - total_weight) / weights[i] // Include a fraction
end if
end while
return x
end function

```

Penjadwalan Job dengan Tenggat Waktu

- Strategi greedy: Pada setiap langkah, pilih job dengan profit terbesar yang masih memenuhi tenggat waktu.
- Algoritma greedy dapat menghasilkan solusi optimal jika job diurutkan berdasarkan profit dari besar ke kecil.

```

function JobScheduling(deadlines, profits) → set of jobs
    J ← {1} // Job 1 is always selected
    for i ← 2 to n do
        if all jobs in J ∪ {i} meet their deadlines then
            J ← J ∪ {i}
        end if
    end for
    return J
end function

```

Pohon Merentang Minimum

- Pohon merentang minimum adalah pohon merentang dari graf berbobot dengan total bobot sisi-sisinya minimum.
- Ada dua algoritma greedy untuk mencari pohon merentang minimum:
 - a. Algoritma Prim
 - Strategi: Pada setiap langkah, pilih sisi dengan bobot terkecil yang bersisian dengan simpul di pohon terbentuk tetapi tidak membentuk sirkuit.
 - Kompleksitas: $O(n^2)$
 - Langkah 1: ambil sisi dari graf G yang berbobot minimum, masukkan ke dalam T.
 - Langkah 2: pilih sisi (u, v) yang mempunyai bobot minimum dan bersisian dengan simpul di T, tetapi (u, v) tidak membentuk sirkuit di T. Masukkan (u, v) ke dalam T.
 - Langkah 3: ulangi langkah 2 sebanyak $n - 2$ kali.

```

procedure Prim(input G: weighted_graph, output T: tree)
    Cari sisi  $(p, q)$  dari G yang berbobot terkecil
    T ← {(p, q)}
    for i = 1 to n - 2 do
        Pilih sisi  $(u, v)$  dari G yang bobotnya terkecil namun bersisian dengan simpul di T
        T ← T ∪ {(u, v)}
    end for
end procedure

```

Pada algoritma Prim, kita membangun pohon merentang minimum T dengan menambahkan sisi berbobot minimum yang bersisian dengan simpul di T tetapi tidak membentuk sirkuit.

b. Algoritma Kruskal

- Sisi-sisi diurutkan berdasarkan bobot dari kecil ke besar.
- Strategi: Pada setiap langkah, pilih sisi dengan bobot terkecil yang tidak membentuk sirkuit dengan pohon terbentuk.

- Kompleksitas: $O(|E| \log |E|)$

```

procedure Kruskal(input G: weighted_graph, output T: tree)
    // Assume edges are sorted in non-decreasing order of weights
    T ← {}
    while the number of edges in T < n - 1 do
        Select the smallest edge (u, v) from G
        if (u, v) does not form a cycle in T then
            T ← T ∪ {(u, v)}
        end if
    end while
end procedure

```

Pada algoritma Kruskal, kita membangun pohon merentang minimum T dengan menambahkan sisi berbobot minimum yang tidak membentuk sirkuit di T.

Lintasan Terpendek (Shortest Path)

- Terdapat beberapa jenis persoalan lintasan terpendek seperti lintasan terpendek antara dua buah simpul tertentu, semua pasangan simpul, dari simpul tertentu ke semua simpul lain, dll.
- Algoritma Greedy sederhana tidak selalu memberikan solusi optimal.
- Algoritma Dijkstra digunakan untuk mencari lintasan terpendek dari simpul asal ke semua simpul lain.
- Strategi: Pada setiap langkah, pilih lintasan berbobot minimum yang menghubungkan simpul terpilih dengan simpul belum terpilih. Lintasan dari asal ke simpul baru harus terpendek.
- Kompleksitas: $O(n^2)$
- Aplikasi Algoritma Dijkstra: routing pada jaringan komputer untuk menentukan lintasan terpendek antar router.

```

procedure Dijkstra(input G: weighted_graph, input a: initial_vertex, output L: array of real)
    for each vertex v in G do
        L[v] ← infinity
    end for
    L[a] ← 0
    S ← {}
    for k = 1 to n do
        u ← vertex not in S with minimum L[u]
        S ← S ∪ {u}
        for each vertex v not in S do
            if L[u] + weight(u, v) < L[v] then
                L[v] ← L[u] + weight(u, v)
            end if
        end for
    end for
end procedure

```

Pada algoritma Dijkstra, kita mencari lintasan terpendek dari simpul asal a ke semua simpul lain dengan memilih simpul u yang belum terpilih dengan $L[u]$ minimum, lalu memperbarui jarak $L[v]$ untuk setiap simpul v yang belum terpilih.

Perlu diingat bahwa algoritma greedy tidak selalu memberikan solusi optimal untuk semua kasus, tetapi dapat menjadi solusi hampiran yang cukup baik. Pembuktian optimalitas algoritma greedy untuk suatu persoalan merupakan tantangan tersendiri.

ALGORITMA DIVIDE AND CONQUER

Pengenalan Algoritma Divide and Conquer

- Divide and Conquer merupakan strategi fundamental dalam ilmu komputer yang awalnya berasal dari strategi militer "divide ut imperes".
- Konsep dasar Divide and Conquer:
 - Divide: Membagi persoalan menjadi upa-persoalan yang lebih kecil dengan karakteristik yang mirip.
 - Conquer: Menyelesaikan masing-masing upa-persoalan secara rekursif jika masih besar atau secara langsung jika sudah kecil.
 - Combine: Menggabungkan solusi dari masing-masing upa-persoalan untuk membentuk solusi persoalan semula.

Skema Umum Algoritma Divide and Conquer

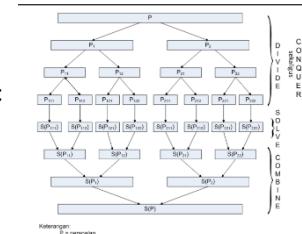
- Skema umum algoritma Divide and Conquer dengan kompleksitas waktu $T(n)$:
- $T(n) = \Theta(g(n))$, jika $n \leq n_0$
- $T(n) = T(n_1) + T(n_2) + \dots + T(n_r) + \Theta(f(n))$, jika $n > n_0$

Penjelasan:

$g(n)$: Kompleksitas waktu untuk menyelesaikan persoalan jika n sudah kecil

$T(n_1) + T(n_2) + \dots + T(n_r)$: Kompleksitas waktu untuk memproses setiap upa-persoalan

$f(n)$: Kompleksitas waktu untuk menggabungkan solusi upa-persoalan



Beberapa persoalan yang diselesaikan dengan D&C

- Persoalan MinMaks (mencari nilai minimum dan nilai maksimum)
- Menghitung perpangkatan
- Persoalan pengurutan (sorting) – Mergesort dan Quicksort
- Mencari sepasang titik terdekat (closest pair problem)
- Convex Hull
- Perkalian matriks
- Perkalian bilangan bulat besar
- Perkalian dua buah polinom

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

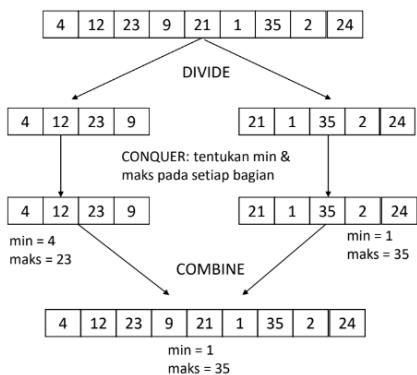
- Penyelesaian dengan algoritma brute force:

```
procedure MinMaks1(input A : TabellInteger, 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
```

- Penyelesaian dengan algoritma 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” yang didefinisikan apabila larik hanya berisi 1 elemen atau 2 elemen.

Proseduri M

- Algoritma:

 1. Untuk kasus $n = 1$ atau $n = 2$,
SOLVE : Jika $n = 1$, maka min = maks = $A[n]$
Jika $n = 2$, maka bandingkan kedua elemen untuk menentukan min dan maks
 2. Untuk kasus $n > 2$,
 - a. DIVIDE: Bagi dua larik A menjadi dua bagian yang sama, A_1 dan A_2
 - b. CONQUER:
 $\text{MinMaks}(A_1, n/2, \text{min1}, \text{maks1})$
 $\text{MinMaks}(A_2, n/2, \text{min2}, \text{maks2})$
 - c. COMBINE:
if $\text{min1} < \text{min2}$ then $\text{min} \leftarrow \text{min1}$ else $\text{min} \leftarrow \text{min2}$
if $\text{maks1} < \text{maks2}$ then $\text{maks} \leftarrow \text{maks2}$ else $\text{maks} \leftarrow \text{maks1}$

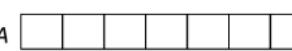
```

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 ← A[i]; maks ← A[i]
  else
    if (i = j - 1) then { larik berukuran 2 elemen }
      if A[i] < A[j] then
        min ← A[i]; maks ← A[j]
      else
        min ← A[j]; maks ← A[i]
      endif
    else
      k ← (i + j) 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 ← min1 else min ← min2 endif
      if maks1 < maks2 then maks ← maks2 else maks ← maks1 endif
    endif
  endif
}

```

A 

$i \quad j$

A 

$i \quad k \quad k+1 \quad j$

Perpangkatan a^n

- Misalkan a anggota \mathbb{R} dan n adalah bilangan bulat tidak negatif, maka perpangkatan a^n didefinisikan sebagai berikut:

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

- Penyelesaian dengan bruteforce:

```

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

```

- Penyelesaian dengan algoritma Divide and Conquer:

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

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

Algoritma divide and conquer untuk menghitung a^n :

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

```

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

```

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

Deklarasi
  x : real

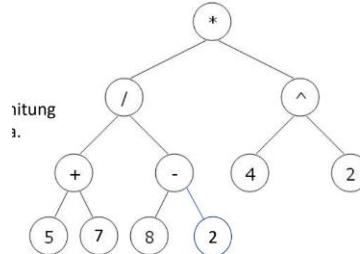
Algoritma:
  if n = 0 then
    return 1
  else
    x ← Exp3(a, n div 2)
    if odd(n) then { kasus n ganjil }
      return x * x * a
    else { kasus n genap }
      return x * x
    endif
  endif

```

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

- Algoritma Divide and conquer:

if pohon tidak kosong then

nilai1 \leftarrow Evaluasi(upa-pohon kiri)

nilai2 \leftarrow 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

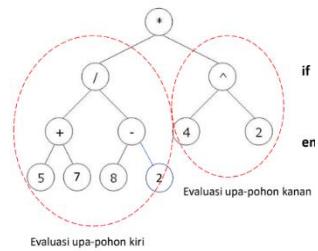
endif

- Pseudocode prosedur:

```

procedure EvaluasiPohon(input T : Pohon, output nilai : real)
{ Mengevaluasi pohon ekspresi T
  Masukan: Pohon ekspresi T, asumsik T tidak kosong
  Luaran: nilai berisi hasil evaluasi ekspresi
}
Deklarasi
  nilai1, nilai2 : real
Algoritma:
  if left(T) = NIL and right(T) = NIL { simpul daun }
    nilai  $\leftarrow$  info(T)
  else { simpul dalam }
    EvaluasiPohon(left(T), nilai1);
    EvaluasiPohon(right(T), nilai2);
    case info(T) of
      "+" : nilai  $\leftarrow$  nilai1 + nilai2
      "-" : nilai  $\leftarrow$  nilai1 - nilai2
      "*" : nilai  $\leftarrow$  nilai1 * nilai2
      "/" : nilai  $\leftarrow$  nilai1 / nilai2 {dengan syarat nilai2 ≠ 0}
      "^" : nilai  $\leftarrow$  nilai1 ^ nilai2 {dengan syarat nilai1 ≠ 0 dan nilai2 ≠ 0}
    end
  end

```



- Pseudocode fungsi:

```

function EvaluasiPohon(T : Pohon) → real
{ mengevaluasi pohon ekspresi T }
Deklarasi
    nilai1, nilai2 : real
Algoritma:
    if left(T) = NIL and right(T) = 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 ≠ 0}
            "^" : return EvaluasiPohon(left(T), nilai1) ^ EvaluasiPohon(right(T), nilai2); {nilai1 ≠ 0 dan nilai2 ≠ 0}
        end
    end

```

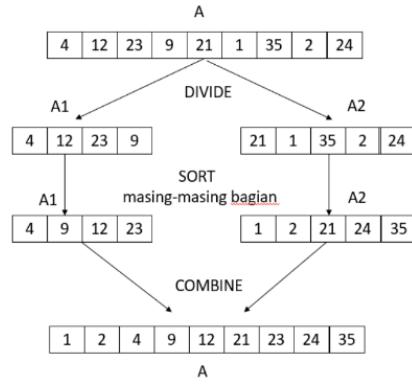
Pengurutan Secara Divide and Conquer

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

```

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, A1 dan A2, masing-masing berukuran n1 dan n2 (n = n1 + n2)
    Sort(A1, n1) { urut larik bagian kiri yang berukuran n1 elemen }
    Sort(A2, n2) { urut larik bagian kanan yang berukuran n2 elemen }
    Combine(A1, A2, A) { gabung hasil pengurutan bagian kiri dan bagian kanan }
end

```



- Terdapat dua pendekatan melakukan pengurutan dengan divide and conquer:
 - 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)
 - 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

Contoh: Misalkan larik A adalah sebagai berikut:

<i>A</i>	8	1	4	6	9	3	5	7
----------	---	---	---	---	---	---	---	---

Dua pendekatan (approach) pengurutan:

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

Tabel A dibagi dua berdasarkan posisi elemen:

<i>Divide:</i>	<i>A1</i>	8	1	4	6	<i>A2</i>	9	3	5	7
----------------	-----------	---	---	---	---	-----------	---	---	---	---

<i>Sort:</i>	<i>A1</i>	1	4	6	8	<i>A2</i>	3	5	7	9
--------------	-----------	---	---	---	---	-----------	---	---	---	---

<i>Combine:</i>	<i>A1</i>	1	3	4	5	6	7	8	9
-----------------	-----------	---	---	---	---	---	---	---	---

Algoritma pengurutan yang termasuk jenis ini:

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

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

Tabel A dibagi dua berdasarkan nilai elemennya. Misalkan elemen-elemen A1 \leq elemen-elemen A2.

<i>A</i>	8	1	4	6	9	3	5	7
----------	---	---	---	---	---	---	---	---

<i>Divide:</i>	<i>A1</i>	5	1	4	3	<i>A2</i>	9	6	8	7
----------------	-----------	---	---	---	---	-----------	---	---	---	---

<i>Sort:</i>	<i>A1</i>	1	3	4	5	<i>A2</i>	6	7	8	9
--------------	-----------	---	---	---	---	-----------	---	---	---	---

<i>Combine:</i>	<i>A</i>	1	3	4	5	6	7	8	9
-----------------	----------	---	---	---	---	---	---	---	---

A. 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:

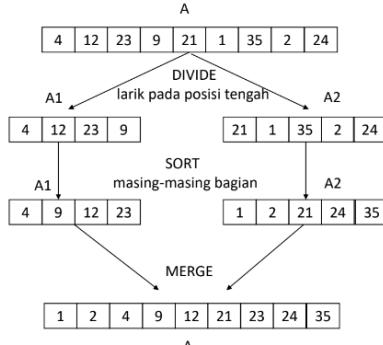
1. Sampai $n = 1$
2. Gunakan algoritma merge

- Algoritma Merge Sort (A, n):

1. Jika $n = 1$, maka larik A sudah terurut dengan sendirinya (langkah SOLVE).

2. Jika $n > 1$, maka

- a. DIVIDE: bagi larik A menjadi dua bagian pada posisi pertengahan, masing-masing bagian berukuran $n/2$ elemen.
- b. CONQUER: secara rekursif, terapkan Merge Sort pada masing-masing bagian.
- c. MERGE: gabung hasil pengurutan kedua bagian sehingga diperoleh larik A yang terurut.



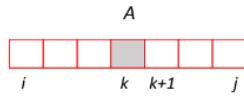
- Pseudocode:

```

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

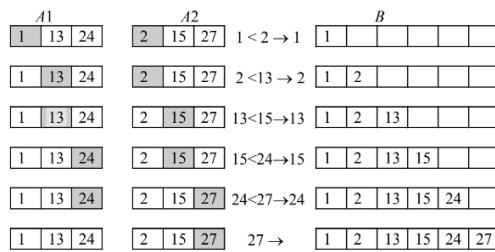
Algoritma:
  if i < j then
    k ← (i + j) div 2           { ukuran(A) > 1 }
    MergeSort(A, i, k)          { bagi A pada posisi pertengahan }
    MergeSort(A, k + 1, j)      { urut upalarik A[i..k] }
    Merge(A, i, k, j)          { urut upalarik A[k+1..j] }
    Merge(A, i, k, j)          { gabung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

```



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

- Contoh:



- Pseudocode:

```

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[i..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

```



```

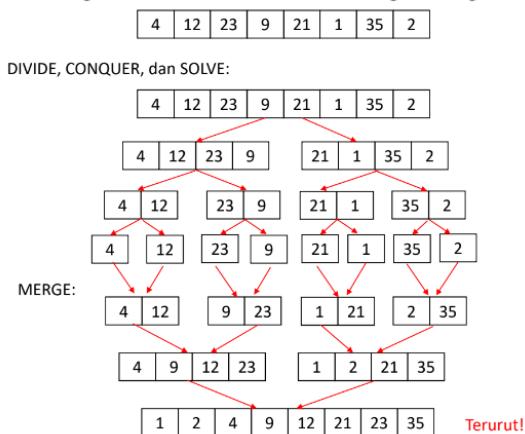
{ salin sisa larik A bagian kiri ke larik B, jika masih ada }
while (p ≤ k) do
  B[r] ← A[p]
  p ← p + 1
  r ← r + 1
endwhile
{ p > k }

{ salin sisa larik A bagian kanan ke larik B, jika masih ada }
while (q ≤ j) do
  B[r] ← A[q]
  q ← q + 1
  r ← r + 1
endwhile
{ q > j }

{ salin kembali elemen-elemen larik B ke dalam A }
for r ← i to j do
  A[r] ← B[r]
endfor
{ diperoleh larik A yang terurut membesar }

```

- Contoh: Pengurutan larik A di bawah ini dengan Merge Sort



B. INSERTION SORT

- Insertion Sort adalah pengurutan easy split/hard join dengan cara membagi larik menjadi dua buah upalarik yang tidak sama ukurannya,
 - yaitu, upalarik pertama hanya satu elemen, sedangkan upalarik kedua berukuran $n - 1$ elemen.



- Insertion Sort dapat dipandang sebagai kasus khusus dari Merge Sort dengan hasil pembagian terdiri dari 1 elemen dan $n - 1$ elemen.
 - Pseudocode:

```

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

Deklarasi:
  k : integer

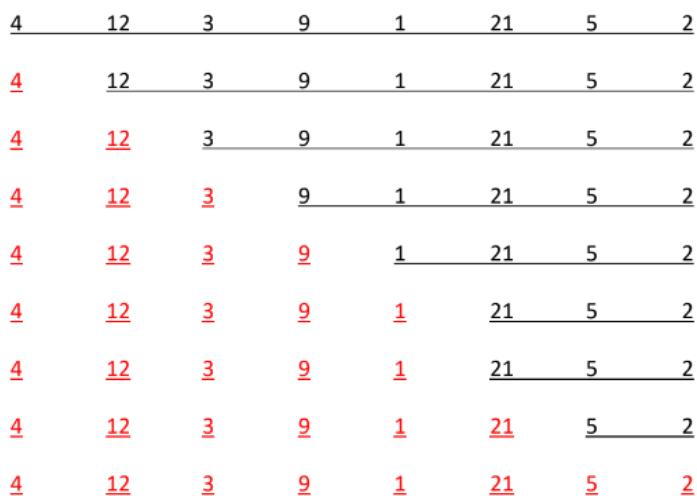
Algoritma:
  if i < j then
    k ← i
    InsertionSort(A, k + 1, j)
    Merge(A, i, k, j)
  end if
  { ukuran(A) > 1 }
  { bagi A pada posisi elemen pertama }
  { urut upalarik A[k+1..j] }
  { gabung hasil pengurutan A[i..k] dan A[k+1..j] menjadi A[i..j] }

```

Contoh 6 (Insertion Sort): Misalkan larik A berisi elemen-elemen berikut:

4 12 23 9 21 1 5 2

DIVIDE, CONQUER, dan SOLVE:



C. QUICK SORT

- Algoritma pengurutan Quicksort merupakan algoritma pengurutan yang terkenal dan tercepat (sesuai namanya).
- Quicksort ditemukan oleh Tony Hoare tahun 1959 dan dipublikasikan tahun 1962.
- Quicksort merupakan algoritma pengurutan secara divide and conquer, dan termasuk ke dalam pendekatan sulit membagi, mudah menggabung (hard split/easy join)
- Di dalam Quicksort, larik A dibagidua (istilahnya: dipartisi) menjadi dua buah upalarik, A1 dan A2, sedemikian sehingga:
semua elemen di A1 \leq semua elemen di 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]

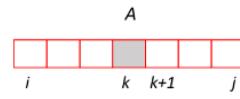
Combine: A [1 | 3 | 4 | 5 | 6 | 7 | 8 | 9]

- Pseudocode:

```
procedure QuickSort(input/output A : LarikInteger, input i,j : integer)
{ Mengurutkan larik A[i..j] dengan algoritma Quicksort.
  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 }
 Partisi(A, i, j, k) { Larik 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 : LarikInteger, input i, j : integer, output q : integer)
{ Membagi larik A[i..j] menjadi upalarik A[i..q] dan A[q+1..j]
  Masukan: Larik A[i..j] udah terdefinisi harganya.
  Luaran: upalarik A[i..q] dan upalarik A[q+1..j] sedemikian sehingga A[i..q] lebih kecil dari larik A[q+1..j] }
```

Deklarasi

pivot, temp : integer

Algoritma:

pivot \leftarrow pilih sembarang elemen larik sebagai pivot, misalkan pivot = elemen tengah

p \leftarrow i {awal pemindai dari kiri}

q \leftarrow j { awal pemindai dari kanan }

repeat

while $A[p] < \text{pivot}$ **do**

p \leftarrow p + 1

endwhile

{ $A[p] \geq \text{pivot}$ }

while $A[q] > \text{pivot}$ **do**

q \leftarrow q - 1

endwhile

{ $A[q] \leq \text{pivot}$ }

if p < q **then**

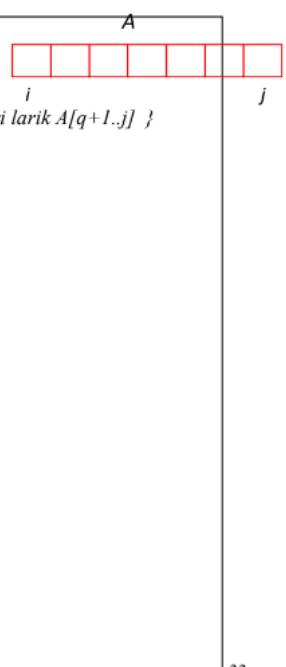
swap($A[p]$, $A[q]$) {pertukarkan $A[p]$ dengan $A[q]$ }

p \leftarrow p + 1 {awal pemindai berikutnya dari kiri}

q \leftarrow q - 1 {awal pemindai berikutnya dari kanan}

endif

until p \geq q



- Cara pemilihan pivot (khusus pada Quicksort versi 1):
 - Pivot = elemen pertama/element terakhir/element tengah larik
 - Pivot dipilih secara acak dari salah satu elemen larik.
 - Pivot = elemen median larik

D. SELECTION SORT

- Pseudocode:

```
procedure SelectionSort(input/output A : LarikInteger, input i,j : integer)
{ Mengurutkan larik A[i..j] dengan algoritma Selection 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 }
  Partisi3(A, i,j) { Partisi menjadi 1 elemen dan n - 1 elemen } i i+1
  SelectionSort(A, i+1,j) { Urut hanya upalarik A[i+1..j] dengan Selection Sort }
endif
```



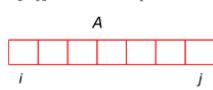
```
procedure Partisi3(input/output A : LarikInteger, input i,j : integer)
{ Mempartisi larik A[i..j] dengan cara mencari elemen minimum di dalam A[i..j], dan menempatkan elemen terkecil sebagai elemen pertama larik.
  Masukan: A[i..j] sudah terdefinisi elemen-elemennya
  Luaran: A[i..j] dengan A[i] adalah elemen minimum.
}
```

Deklarasi

idxmin, k : integer

Algoritma:

```
idxmin ← i
for k ← i+1 to j do
  if A[k] < A[idxmin] then
    idxmin ← k
  endif
endfor
swap(A[i], A[idxmin]) { pertukarkan A[i] dengan A[idxmin] }
```



PERKALIAN MATRIKS

- Penyelesaian secara brute force:

```
function KaliMatriks(A, B : Matriks, n : integer) → Matriks
{ Mengalikan matriks A dan B yang berukuran n × n, menghasilkan matriks C yang juga berukuran n × n }
```

Deklarasi

i, j, k : integer

Algoritma:

```
for i←1 to n do
  for j←1 to n do
    C[i,j]←0 { inisialisasi penjumlahan }
    for k ← 1 to n do
      C[i,j]←C[i,j] + A[i,k]*B[k,j]
    endfor
  endfor
endfor
return C
```

- Penyelesaian secara divide and conquer:

```
function KaliMatriks2(A, B : Matriks, n : integer) → Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran n × n. }
Deklarasi
i, j, k : integer
A11, A12, A21, A22, B11, B12, B21, B22, C11, C12, C21, C22 : Matriks
```

Algoritma:

```
if n = 1 then { matriks berukuran 1 x 1 atau sebagai scalar }
  return A * B { perkalian dua buah scalar biasa }
else
  Bagi A menjadi A11, A12, A21, dan A22 yang masing-masing berukuran n/2 x n/2
  Bagi B menjadi B11, B12, B21, dan B22 yang masing-masing berukuran n/2 x 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, C21, C22 }
endif
```

- Perkalian matriks stassen

```

function KaliMatriksStrassen(A, B : Matriks, n : integer) → Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran  $n \times n$ . }
Deklarasi
i,j, k : integer
A11, A12, A21, A22, B11, B12, B21, B22, C11, C12, C21, C22, M1, M2, M3, M4, M5, M6, M7 : Matriks

Algoritma:
if n = 1 then { matriks berukuran 1 x 1 atau sebagai scalar }
    return A * B { perkalian dua buah scalar 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  $n/2 \times n/2$ 
    M1 ← KaliMatriksStrassen(A12 – A22, B21 + B22, n/2)
    M2 ← KaliMatriksStrassen(A11 + A22, B11 + B22, n/2)
    M3 ← KaliMatriksStrassen(A11 – A21, B11 + B12, n/2)
    M4 ← KaliMatriksStrassen(A11 + A12, B22, n/2)
    M5 ← KaliMatriksStrassen(A11, B12 – B22, n/2)
    M6 ← KaliMatriksStrassen(A22, B21 – B11, n/2)
    M7 ← KaliMatriksStrassen(A21 + A22, B11, n/2)
    C11 ← M1 + M2 – M4 + M6
    C12 ← M4 + M5
    C21 ← M6 + M7
    C22 ← M2 – M3 + M5 – M7
    return C { C adalah gabungan C11, C12, C21, C22 }
endif

```

Perkalian Bilangan Bulat yang Besar

- Penyelesaian secara brute force

```

function Kali1(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma brute force. }

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 ← xj * yi
        temp ← temp + AngkaPuluhan
        AngkaSatuan ← temp mod 10
        AngkaPuluhan ← temp div 10
        write(AngkaSatuan)
    endfor
endfor
Z ← Jumlahkan semua hasil perkalian dari atas ke bawah
return Z

```

- Penyelesaian secara divide and conquer

```

function Kali2(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma divide and conquer. }

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

Algoritma:
if n = 1 then
    return X * Y { perkalian skalar 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
    return Kali2(a, c, s)*102s + Kali2(b, c, s)*10s + Kali2(a, d, s)*10s + Kali2(b, d, s)
endif

```

- Algoritma Perkalian Karatsuba

```

function Kali3(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma Karatsuba. }

Deklarasi
a, b, c, d, p, q, r : LongInteger
s : integer

Algoritma:
if n = 1 then
    return X * Y { perkalian skalar 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

```

23

PERKALIAN POLINOM

- Pseudocode:

```

function KaliPolinom2(A, B : Polinom, n : integer) → Polinom
{ Memberikan hasil perkalian polinom A(x) dan B(x), masing-masing berderajat n dengan algoritma divide and conquer. }

Deklarasi
A0, A1, B0, B1, U, Y, Z : Polinom
s : integer

Algoritma:
if n = 0 then
    return A * B { perkalian skalar biasa }
else
    s ← n div 2 { bagidua suku-suku polinom pada posisi s }
    A0 ← a0 + a1x + a2x2 + ... + as-1xs-1
    A1 ← as + as+1x + as+2x2 + ... + anxn-s
    B0 ← b0 + b1x + b2x2 + ... + bs-1xs-1
    B1 ← bs + bs+1x + bs+2x2 + ... + bnxn-s
    Y ← KaliPolinom2(A0 + A1, B0 + B1, s)
    U ← KaliPolinom2(A0, B0, s)
    Z ← KaliPolinom2(A1, B1, s)
    return U + (Y - U - Z) * xs + Z * x2s
endif

```

35