

## 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

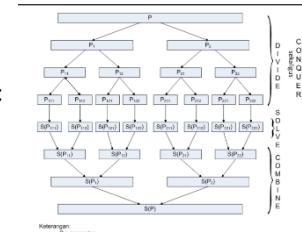
- Skema umum algoritma Divide and Conquer dengan kompleksitas waktu  $T(n)$ :
- $T(n) = \Theta(g(n))$ , jika  $n \leq n_0$
- $T(n) = T(n_1) + T(n_2) + \dots + T(n_r) + \Theta(f(n))$ , jika  $n > n_0$

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

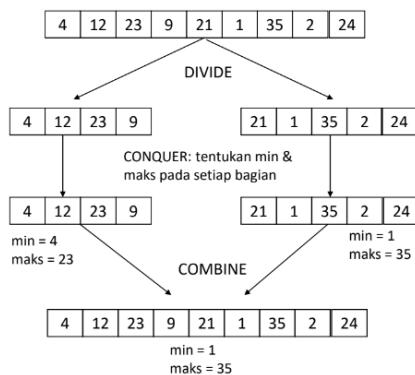
- Penyelesaian dengan algoritma brute force:

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

Algoritma:

- 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 maks

- Untuk kasus  $n > 2$ ,

- DIVIDE: Bagi dua larik A menjadi dua bagian yang sama, A1 dan A2

- CONQUER:

MinMaks(A1,  $n/2$ , min1, maks1)

MinMaks(A2,  $n/2$ , min2, maks2)

- COMBINE:

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

if  $\max1 < \max2$  then  $\max \leftarrow \max2$  else  $\max \leftarrow \max1$

<b>procedure</b> MinMaks2( <b>input</b> A : LarikInteger, i,j : <b>integer</b> , <b>output</b> min, maks : <b>integer</b> )	
{ 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 }	
<b>Deklarasi</b>	
min1, min2, maks1, maks2 : <b>integer</b>	
<b>Algoritma:</b>	
<b>if</b> $i=j$ <b>then</b> { larik berukuran 1 elemen } $\min \leftarrow A[i]$ ; $\max \leftarrow A[i]$	 $i=j$
<b>else</b> <b>if</b> $(i=j-1)$ <b>then</b> { larik berukuran 2 elemen } <b>if</b> $A[i] < A[j]$ <b>then</b> $\min \leftarrow A[i]$ ; $\max \leftarrow A[j]$	 $i \quad j$
<b>else</b> $\min \leftarrow A[j]$ ; $\max \leftarrow A[i]$	
<b>endif</b>	
<b>else</b> { larik berukuran lebih dari 2 elemen } $k \leftarrow (i+j) \text{ div } 2$ { bagidua larik pada posisi k } MinMaks2(A, i, k, min1, maks1) MinMaks2(A, k+1, j, min2, maks2) <b>if</b> $\min1 < \min2$ <b>then</b> $\min \leftarrow \min1$ <b>else</b> $\min \leftarrow \min2$ <b>endif</b> <b>if</b> $\max1 < \max2$ <b>then</b> $\max \leftarrow \max2$ <b>else</b> $\max \leftarrow \max1$ <b>endif</b>	 $i \quad k \quad k+1 \quad j$
<b>endif</b>	
<b>endif</b>	

### 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 }

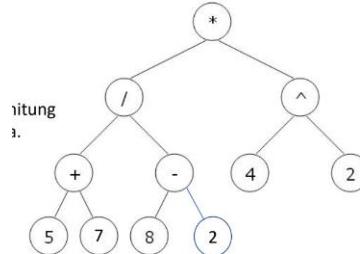
Deklarasi
  x : real

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

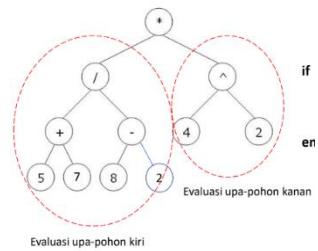
endif

- 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

```

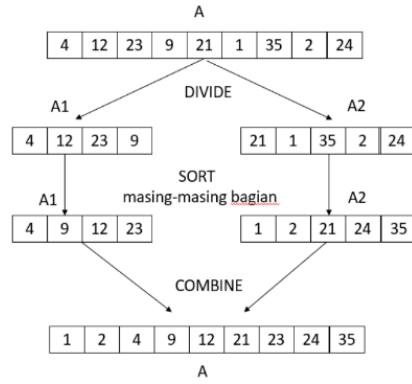
## Pengurutan Secara Divide and Conquer

- 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:

- urut-gabung (Merge Sort)
- 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:

- Larik dibagi sampai ukurannya ( $n$ )  
tinggal berapa elemen?
- Bagaimana menggabungkan  
dua larik terurut menjadi satu  
larik terurut?

Jawaban:

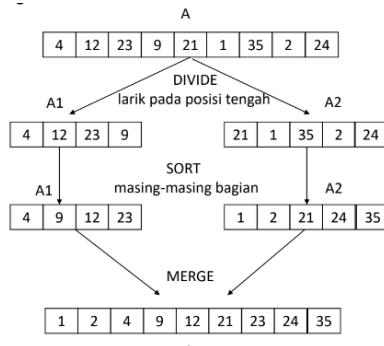
- Sampai  $n = 1$
- Gunakan algoritma merge

- Algoritma Merge Sort ( $A, n$ ):

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

- Jika  $n > 1$ , maka

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



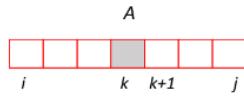
- Pseudocode:

```

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

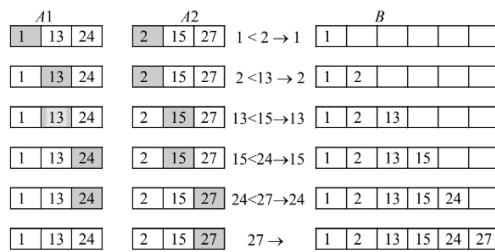
Algoritma:
  if i < j then
    k ← (i + j) div 2           { ukuran(A) > 1 }
    MergeSort(A, i, k)          { bagi A pada posisi pertengahan }
    MergeSort(A, k + 1, j)      { urut upalarik A[i..k] }
    Merge(A, i, k, j)          { urut upalarik A[k+1..j] }
    Merge(A, i, k, j)          { gabung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

```



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

- Contoh:

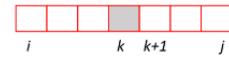


- Pseudocode:

```

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 }

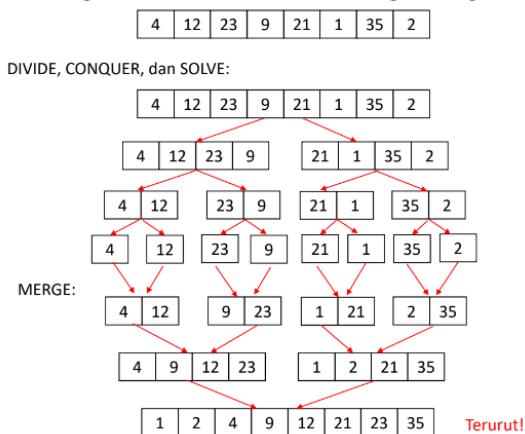
**for** r ← i **to** j **do**

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

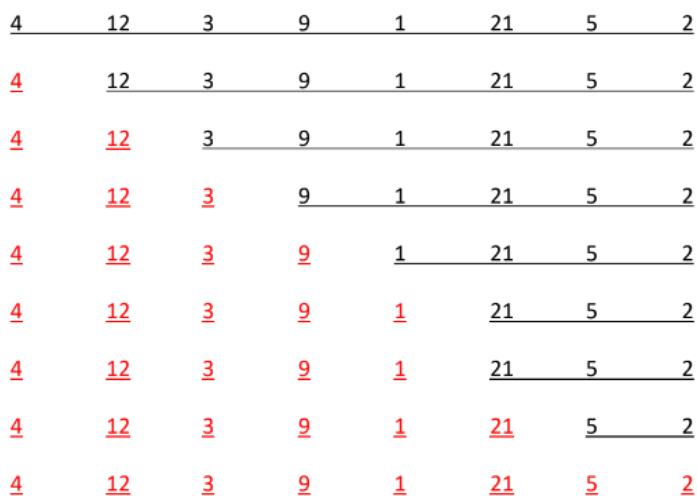
Algoritma:
  if i < j then
    k ← i
    InsertionSort(A, k + 1, j)
    Merge(A, i, k, j)
  end if
  { ukuran(A) > 1 }
  { bagi A pada posisi elemen pertama }
  { urut upalarik A[k+1..j] }
  { gabung hasil pengurutan A[i..k] dan A[k+1..j] menjadi A[i..j] }

```

**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 ]

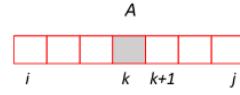
*Combine:* A [ 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ]

- 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**

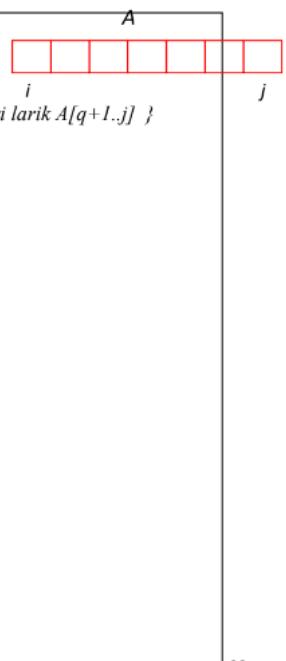
swap( $A[p]$ ,  $A[q]$ ) {pertukarkan  $A[p]$  dengan  $A[q]$  }

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 upalirk A[i+1..j] dengan Selection Sort }
endif
```



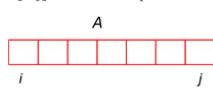
```
procedure Partisi3(input/output A : LarikInteger, input i,j : integer)
{ Mempartisi larik A[i..j] dengan cara mencari elemen minimum di dalam A[i..j], dan menempatkan elemen terkecil sebagai elemen pertama larik.
  Masukan: A[i..j] sudah terdefinisi elemen-elemennya
  Luaran: A[i..j] dengan A[i]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