Divide and Conquer

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

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

Definisi

- *Divide*: membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),

- *Conquer*: memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif), dan

- *Combine:* mengabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

- Obyek permasalahan yang dibagi :

  masukan (*input*) atau *instances* yang berukuran *n* seperti:

  - tabel (larik),

  - matriks,

  - eksponen,

  - dll, bergantung pada masalahnya.

- Tiap-tiap upa-masalah mempunyai karakteristik yang sama (*the same type*) dengan karakteristik masalah asal, sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

Skema Umum Algoritma *Divide and Conquer*

```
procedure DIVIDE_and_CONQUER(input n : integer)
{ Menyelesaikan masalah dengan algoritma D-and-C.
  Masukan: masukan yang berukuran n
  Keluaran: solusi dari masalah semula
}
Deklarasi
     r, k : integer
Algoritma
  if n ≤ n0 then  {ukuran masalah sudah cukup kecil }
     SOLVE upa-masalah yang berukuran n ini
  else
     Bagi menjadi r upa-masalah, masing-masing berukuran n/k
     for masing-masing dari r upa-masalah do
        DIVIDE_and_CONQUER(n/k)
     endfor
     COMBINE solusi dari r upa-masalah menjadi solusi masalah semula }
  endif
```

Contoh Masalah

Mencari Nilai Minimum dan Maksimum (MinMaks)

Persoalan: Misalkan diberikan tabel *A* yang berukuran *n* elemen dan sudah berisi nilai *integer*.

Carilah nilai minimum dan nilai maksimum di dalam tabel tersebut.

Penyelesaian dengan *Algoritma Brute Force*

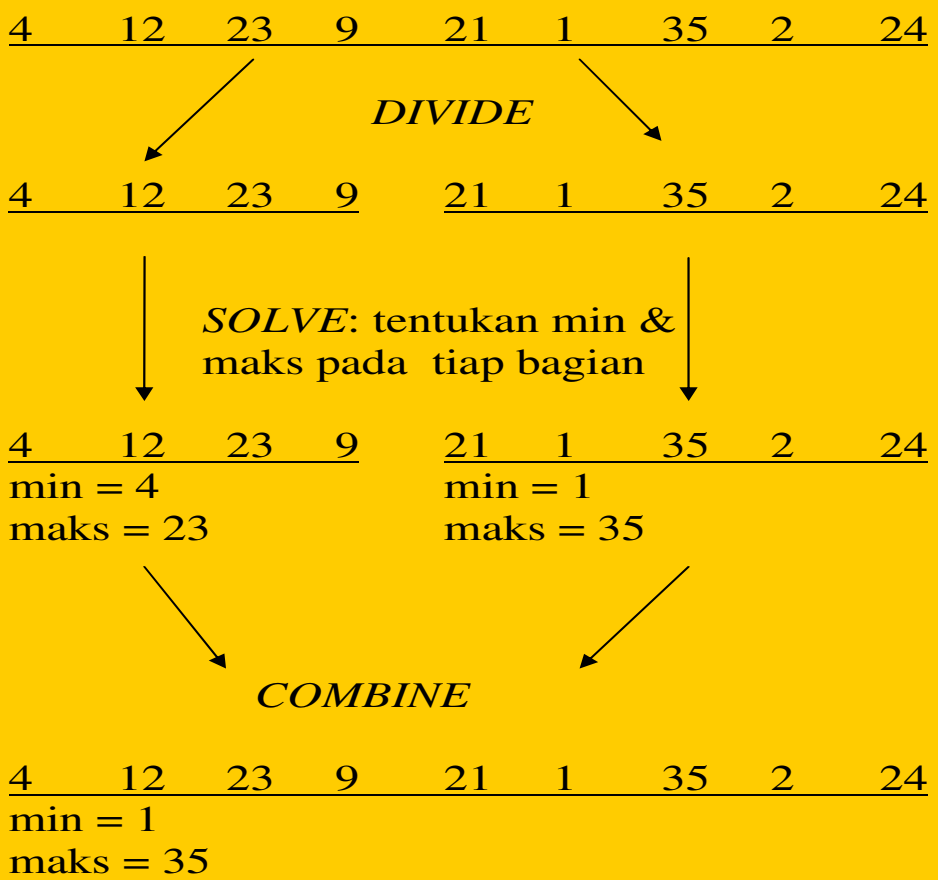
```
procedure MinMaks1(input A : TabelInt, n : integer,
                   output min, maks : integer)
{ Mencari nilai minimum dan maksimum di dalam tabel A yang berukuran n
elemen, secara brute force.
Masukan: tabel A yang sudah terdefinisi elemen-elemennya
Keluaran: nilai maksimum dan nilai minimum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

Algoritma:
    min← A1 { inisialisasi nilai minimum}
    maks←A1 { inisialisasi nilai maksimum }
    for i←2 to n do

      if Ai < min then
        min←Ai
      endif

      if Ai > maks then
        maks←Ai
      endif

   endfor
```

Penyelesaian dengan *Divide and Conquer*

Contoh 4.1. Misalkan tabel *A* berisi elemen-elemen sebagai berikut:

4 12 23 9 21 1 35 2 24

Ide dasar algoritma secara *Divide and Conquer*:



- Ukuran tabel hasil pembagian dapat dibuat cukup kecil sehingga mencari minimum dan maksimum dapat diselesaikan (SOLVE) secara lebih mudah.

- Dalam hal ini, ukuran kecil yang dipilih adalah 1 elemen atau 2 elemen.

MinMaks(A, n, min, maks)

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 tabel *A* menjadi dua bagian yang sama, A1 dan A2

   (b) CONQUER:

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

   MInMaks(A2, n/2, min2, maks2)

   (c) COMBINE:

   if min1 <min2 then min <- min1 else min <- min2

   if maks1 <maks2 then maks <- maks2 else maks <- maks1

Contoh 4.2. Tinjau kembali Contoh 4.1 di atas.

DIVIDE dan CONQUER:

4 12 23 9 | 21 1 35 2 24

4 12 | 23 9 | 21 1 | 35 2 24

4 12 | 23 9 | 21 1 | 35 | 2 24

SOLVE dan COMBINE:

4 12	23 9	21 1	35	2 24
min = 4	min = 9	min = 1	min = 35	min = 2
maks = 12	maks = 23	maks = 21	maks =35	maks = 24

4 12 23 9	21 1	35 2 24
min = 4	min = 1	min = 2
maks = 23	maks = 21	maks = 35

4 12 23 9	21 1 35 2 24
min = 4	min = 1
maks = 23	maks = 35

4 12 23 9 21 1 5 2 24
min = 1
maks = 35

```
procedure MinMaks2(input A : TabelInt, i, j : integer,
                   output min, maks : integer)
{ Mencari nilai maksimum dan minimum di dalam tabel A yang berukuran n
elemen secara Divide and Conquer.
Masukan: tabel A yang sudah terdefinisi elemen-elemennya
Keluaran: nilai maksimum dan nilai minimum tabel
}
Deklarasi
      min1, min2, maks1, maks2 : integer

Algoritma:
      if i=j then                        { 1 elemen  }
        min←Ai
        maks←Ai
      else
        if (i = j-1) then                { 2 elemen  }
           if Ai < Aj then
               maks←Aj
            min←Ai
           else
             maks←Ai
             min←Aj
           endif
        else                             { lebih dari 2 elemen }
            k←(i+j) div 2                { bagidua tabel pada posisi k }
            MinMaks2(A, i, k, min1, maks1)
            MinMaks2(A, k+1, j, min2, maks2)
            if min1 < min2 then
               min←min1
            else
               min←min2
            endif

            if maks1<maks2 then
               maks←maks2
            else
               maks←maks2
          endif
```

Kompleksitas waktu asimptotik:

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

Penyelesaian:

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

$$\begin{aligned} T(n) &= 2\mathrm{T}(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 = 4T(\mathrm{n}/4) + 4 + 2 \\ &= 4T(2\mathrm{T}(n/8) + 2) + 4 + 2 = 8T(n/8) + 8 + 4 + 2 \\ &= \ldots \\ &= 2^{k-1}\, T(2) + \sum_{i=1}^{k-1} 2^i \end{aligned}$$

$$\begin{aligned} &= 2^{k-1} \cdot 1 + 2^k - 2 \\ &= n/2 + n - 2 \\ &= 3n/2 \ - 2 \\ &= O(n) \end{aligned}$$

- MinMaks1 secara *brute force* :

$$T(n) = 2n - 2$$

- MinMaks2 secara *divide and conquer*:

$$T(n) = 3n/2 - 2$$

- Perhatikan: $3n/2 - 2 < 2n - 2$, $n \geq 2$.

- Kesimpulan: algoritma MinMaks lebih mangkus dengan metode *Divide and Conquer*.

Perpangkatan a^n

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

$a^n = a \times a \times \ldots \times a$ (n kali), jika $n > 0$

$= 1$, jika $n = 0$

Penyelesaian dengan Algoritma Brute Force

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

Algoritma:
  hasil←1
  for k←1 to  n do
     hasil←hasil * a
  endfor

  return hasil
```

Kompleksitas waktu algoritma:

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

Penyelesaian dengan Divide and Conquer

Algoritma menghitung a^n:

1. Untuk kasus $n = 0$, maka $a^n = 1$.

2. Untuk kasus $n > 0$, bedakan menjadi dua kasus lagi:

   (i) jika n genap, maka $a^n = a^{n/2} \cdot a^{n/2}$

   (ii) jika n ganjil, maka $a^n = a^{n/2} \cdot a^{n/2} \cdot a$

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

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

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

Algoritma:
   if n = 0 then
      return 1
   else
      x←Exp2(a, n div 2)
      if odd(n) then        { fungsi odd memberikan true jika n ganjil }
       return x * x * a
      else
       return x * x
      endif
  endif
```

Kompleksitas algoritma:

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

Penyelesaian:

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

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

$$(n/2^k) = 1 \rightarrow \quad \log (n/2^k) = \log 1$$
$$\log n - \log 2^k = 0$$
$$\log n - k \log 2 = 0$$
$$\log n = k \log 2$$
$$k = \log n / \log 2 = {}^2\log n$$

sehingga

$$T(n) = \lfloor {}^2\log n \rfloor + T(1)$$
$$= \lfloor {}^2\log n \rfloor + 1 + T(0)$$
$$= \lfloor {}^2\log n \rfloor + 1 + 0$$
$$= \lfloor {}^2\log n \rfloor + 1$$
$$= O\,({}^2\log n)$$

Merge Sort

Algoritma:

1. Untuk kasus $n = 1$, maka tabel *A* sudah terurut dengan sendirinya (langkah SOLVE).

2. Untuk kasus $n > 1$, maka
   (a) DIVIDE: bagi tabel *A* menjadi dua bagian, bagian kiri dan bagian kanan, masing-masing bagian berukuran $n/2$ elemen.
   (b) CONQUER: Secara rekursif, terapkan algoritma *D-and-C* pada masing-masing bagian.
   (c) MERGE: gabung hasil pengurutan kedua bagian sehingga diperoleh tabel *A* yang terurut.

Contoh 4.3. Misalkan tabel *A* berisi elemen-elemen berikut:

4 12 23 9 21 1 5 2

DIVIDE, CONQUER, dan SOLVE:

4 12 23 9 21 1 5 2

4 12 23 9 21 1 5 2

4 12 23 9 21 1 5 2

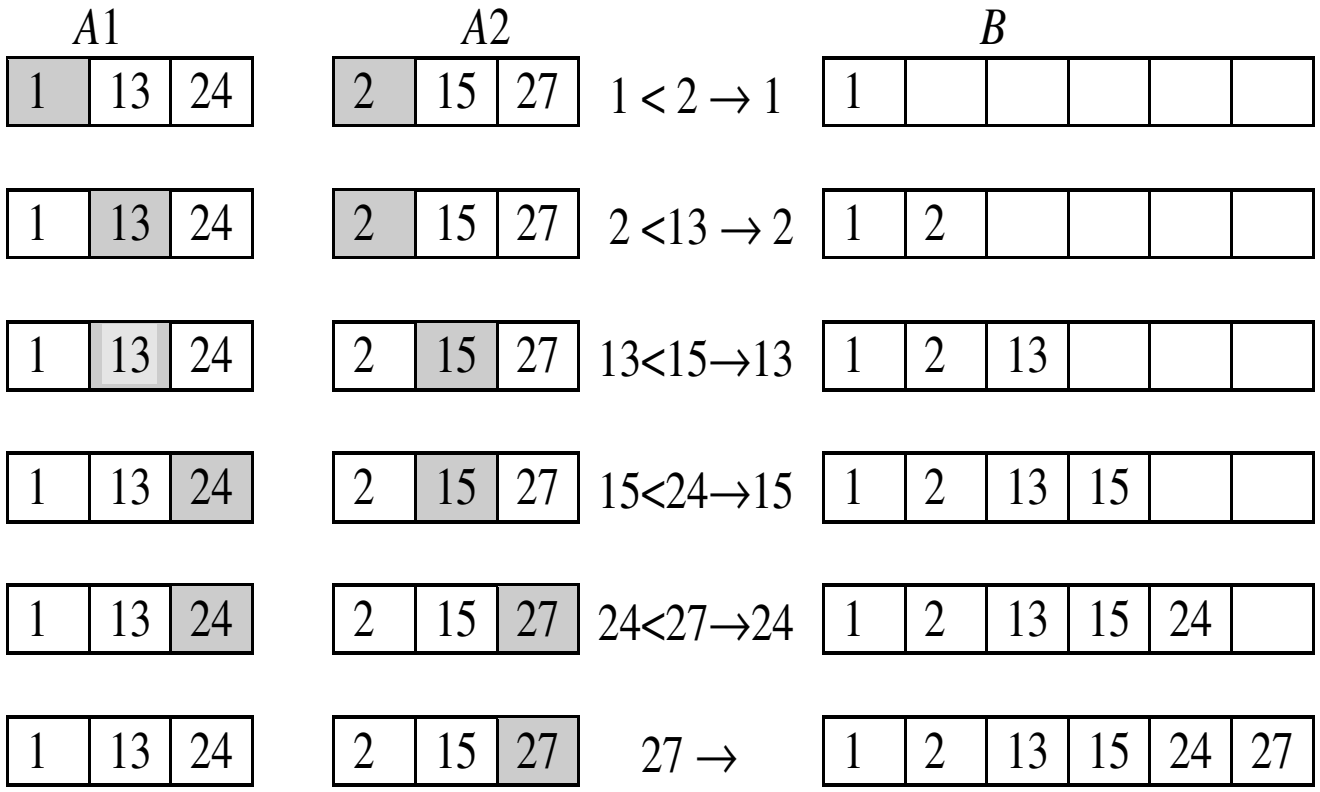
4 12 23 9 21 1 5 2

MERGE: 4 12 9 23 1 21 2 5

4 9 12 23 1 2 5 21

1 2 4 5 9 12 21 23

Contoh *Merge*:



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

{ Mengurutkan tabel A[i..j] dengan algoritma Merge Sort
  Masukan: Tabel A dengan n elemen
  Keluaran: Tabel A yang terurut
}
Deklarasi:
   k : integer

Algoritma:
   if i < j then          { Ukuran(A)> 1}
     k←(i+j) div 2
     MergeSort(A, i, k)
     MergeSort(A, k+1, j)
     Merge(A, i, k, j)
   endif
```

Prosedur *Merge*:

```
procedure Merge(input/output A : TabelInt, input kiri,tengah,kanan :
   integer)
{ Menggabung tabel A[kiri..tengah] dan tabel A[tengah+1..kanan]
  menjadi tabel A[kiri..kanan] yang terurut menaik.
  Masukan: A[kiri..tengah] dan tabel A[tengah+1..kanan] yang sudah
   terurut menaik.
  Keluaran: A[kiri..kanan] yang terurut menaik.
}
Deklarasi
   B : TabelInt
   i, kidal1, kidal2 : integer

Algoritma:
   kidal1←kiri          { A[kiri .. tengah] }
   kidal2←tengah + 1    { A[tengah+1 .. kanan] }
   i←kiri
   while (kidal1 ≤ tengah) and (kidal2 ≤ kanan) do
       if A_kidal1 ≤ A_kidal2 then
          B_i←A_kidal1
          kidal1←kidal1 + 1
       else
          B_i←A_kidal2
          kidal2←kidal2 + 1
       endif
       i←i + 1
    endwhile
   { kidal1 > tengah or kidal2 > kanan }

   { salin sisa A bagian kiri ke B, jika ada }
   while (kidal1 ≤ tengah) do
       B_i←A_kidal1
       kidal1←kidal1 + 1
       i←i + 1
    endwhile
   { kidal1 > tengah }

   { salin sisa A bagian kanan ke B, jika ada }
   while (kidal2 ≤ kanan) do
       B_i←A_kidal2
       kidal2←kidal2 + 1
       i←i + 1
   endwhile
   { kidal2 > kanan }

   { salin kembali elemen-elemen tabel B ke A }
   for i←kiri to kanan do
       A_i←B_i
    endfor
   { diperoleh tabel A yang terurut membesar }
```

- Kompleksitas waktu:

  Asumsi: $n = 2^k$

  $T(n)$ = jumlah perbandingan pada pengurutan dua buah upatabel + jumlah perbandingan pada prosedur *Merge*

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

Penyelesaian:

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

Berhenti jika ukuran tabel terkecil, $n = 1$:

$$n/2^k = 1 \rightarrow \mathrm{k} = {}^2\log n$$

sehingga

$$
\begin{aligned}
T(n) &= nT(1) + cn \, {}^2\log n \\
&= na + cn \ {}^2\log n \\
&= O(n \, {}^2\log n)
\end{aligned}
$$