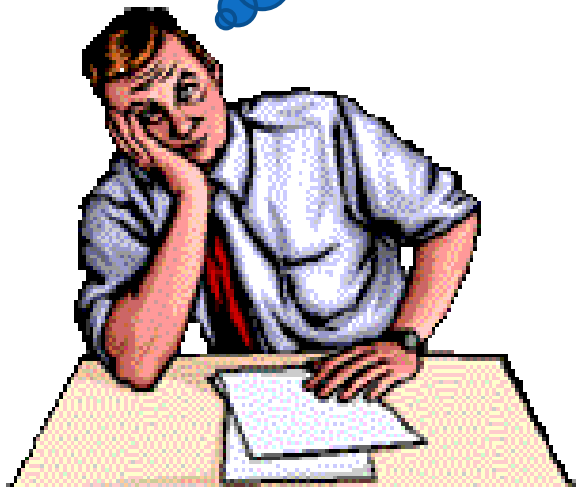
Kompleksitas Algoritma Rekursi

Sukmawati Nur Endah

Departemen Informatika UNDIP

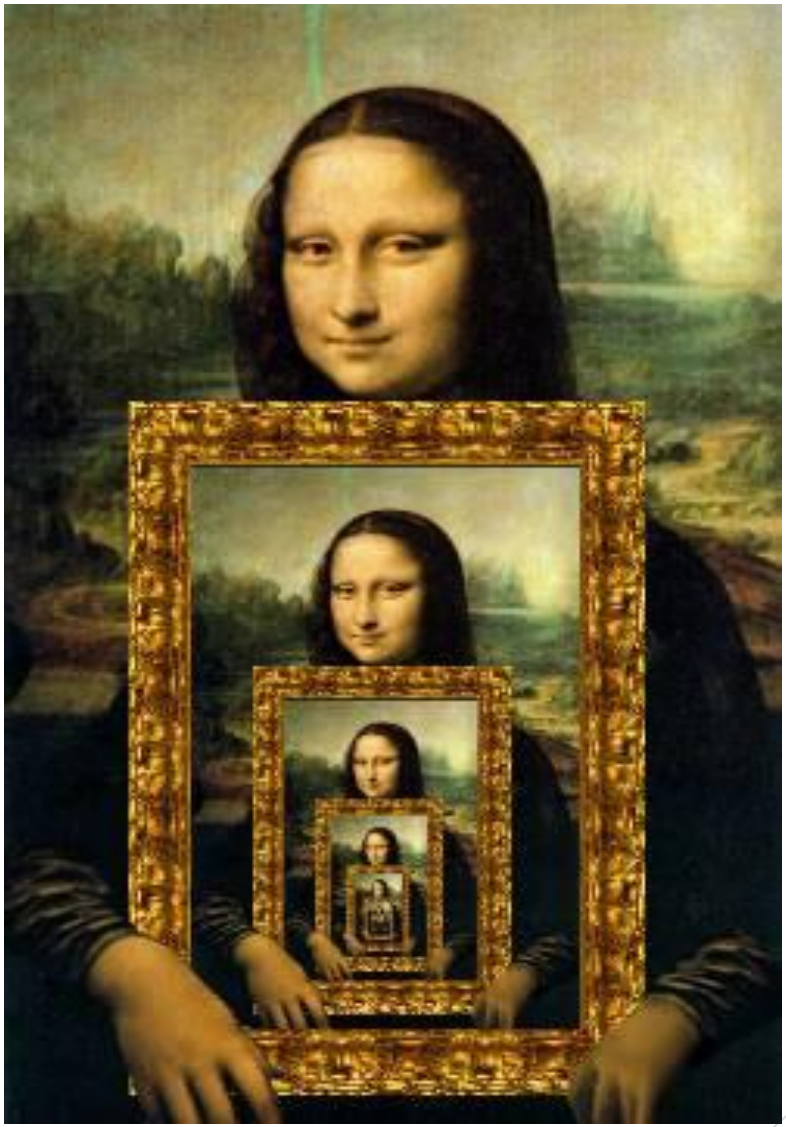
Sumber : Rekursi, Materi Rinaldi Munir

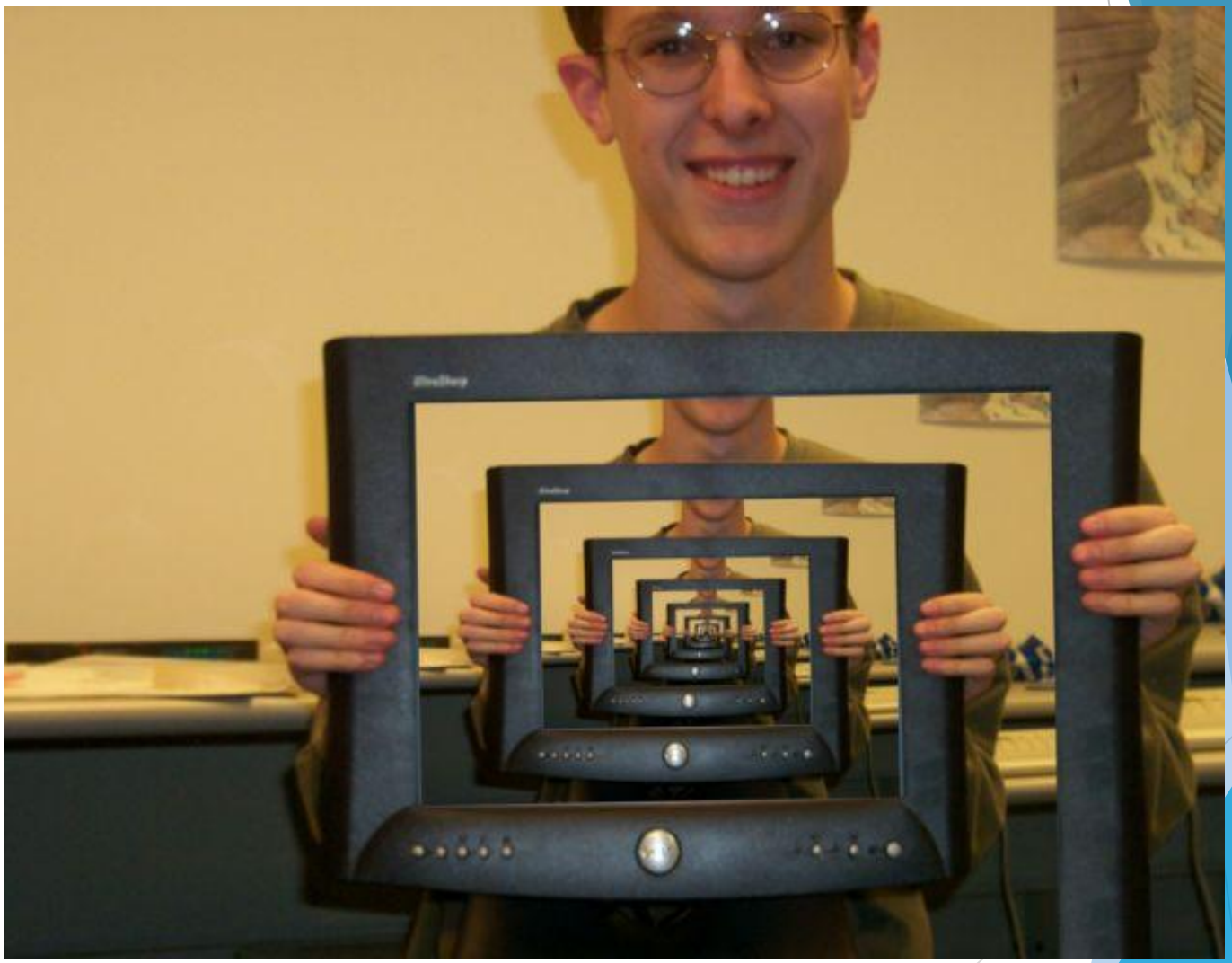
Apakah rekursi itu??



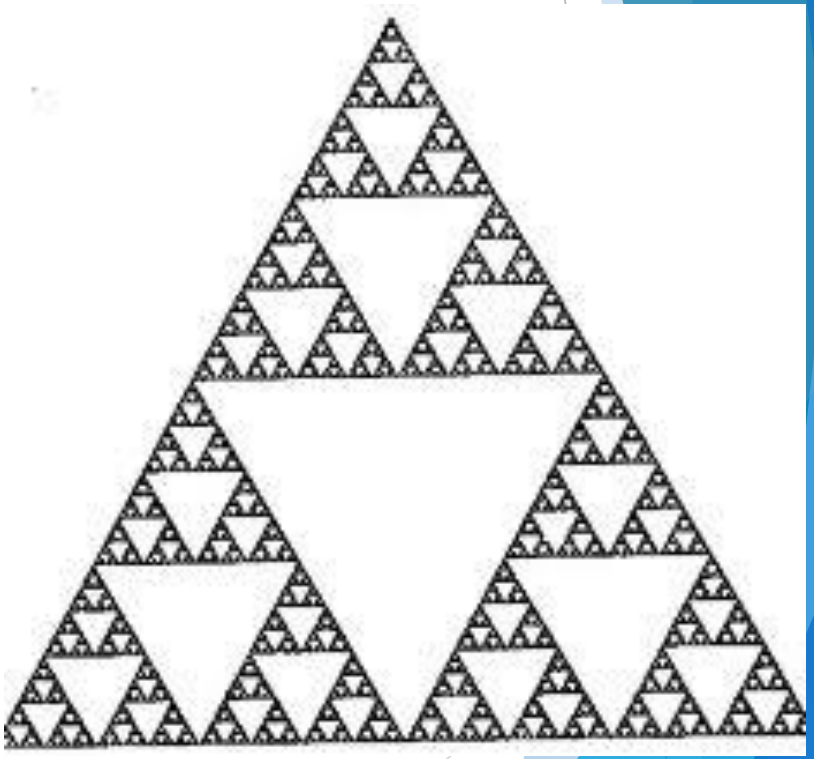
Rekursi

- Sebuah objek dikatakan **rekursif** (*recursive*) jika ia didefinisikan dalam terminologi dirinya sendiri.
- Proses mendefinisikan objek dalam terminologi dirinya sendiri disebut **rekursi** (*recursion*).
- Dengan kata lain bentuk rekursif:
  - sebuah fungsi/prosedur/subrutin yang memanggil dirinya sendiri.





- Objek fraktal adalah contoh bentuk rekursif.



Fraktal di alam

Fungsi Rekursif

- Fungsi rekursif didefinisikan oleh dua bagian:

(i) *Basis*

- Bagian yang berisi nilai fungsi yang terdefinisi secara eksplisit.
- Bagian ini juga sekaligus menghentikan rekursif (dan memberikan sebuah nilai yang terdefinisi pada fungsi rekursif).

(ii) *Rekurens*

- Bagian ini mendefinisikan fungsi dalam terminologi dirinya sendiri.
- Berisi kaidah untuk menemukan nilai fungsi pada suatu input dari nilai-nilai lainnya pada input yang lebih kecil.

- **Contoh 1:** Misalkan *f* didefinsikan secara rekusif sbb

$$f(n) = \begin{cases} 3 & , n = 0 \quad \textit{basis} \\ 2f(n-1) + 4 & , n > 0 \quad \textit{rekurens} \end{cases}$$

Tentukan nilai f(4)!

Solusi: f(4) = 2f(3) + 4

= 2(2f(2) + 4) + 4

= 2(2(2f(1) + 4) + 4) + 4

= 2(2(2(2f(0) + 4) + 4) + 4) + 4

= 2(2(2(2·3 + 4) + 4) + 4) + 4

= 2(2(2(10) + 4) + 4) + 4

= 2(2(24) + 4) + 4

= 2(52) + 4

= 108

Cara lain menghitungnya:

$f(0) = 3$

$f(1) = 2f(0) + 4 = 2 \cdot 3 + 4 = 10$

$f(2) = 2f(1) + 4 = 2 \cdot 10 + 4 = 24$

$f(3) = 2f(2) + 4 = 2 \cdot 24 + 4 = 52$

$f(4) = 2f(3) + 4 = 2 \cdot 52 + 4 = 108$

Jadi, $f(3) = 108$.

- **Contoh 2**: Nyatakan *n*! dalam definisi rekursif

Solusi: $n! = \underbrace{1 \times 2 \times 3 \times \ldots \times (n-1)}_{(n-1)!} \times n = (n-1)! \times n$

Misalkan f(n) = n!, maka

$$n! = \begin{cases} 1 & , n = 0 \\ n \cdot (n-1)! & , n > 0 \end{cases}$$

Menghitung 5! secara rekursif adalah:

$5! = 5 \cdot 4!$

$= 5 \cdot 4 \cdot 3!$

$= 5 \cdot 4 \cdot 3 \cdot 2!$

$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1!$

$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0!$

$= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 120$

Latihan

- Nyatakan secara rekursif deret Barisan Fibonacci 0, 1, 1, 2, 3, 5, 8, 13, 21, ….

- Definisikan a^n secara rekursif , yang dalam hal ini a adalah bilangan riil tidak-nol dan *n* adalah bilangan bulat tidak-negatif.

$$a^n = \underbrace{a \cdot a \cdot a \cdot \ldots a}_{n \text{ kali}} = a \cdot \underbrace{a \cdot a \cdot a \cdot \ldots a}_{n-1 \text{ kali}} = a \cdot a^{n-1}$$

Relasi Rekurens

- Barisan (*sequence*) $a_0, a_1, a_2, \ldots, a_n$ dilambangkan dengan $\{a_n\}$

- Elemen barisan ke-n, yaitu a_n, dapat ditentukan dari suatu persamaan.

- Bila persamaan yang mengekspresikan a_n dinyatakan secara rekursif dalam satu atau lebih *term* elemen sebelumnya, yaitu $a_0, a_1, a_2, \ldots, a_{n-1}$, maka persamaan tersebut dinamakan **relasi rekurens**.

  Contoh: $a_n = 2a_{n-1} + 1$

  $a_n = a_{n-1} + 2a_{n-2}$

  $a_n = 2a_{n-1} - a_{n-2}$

Kompleksitas Alg Rekursif

- Untuk bentuk rekursif, digunakan teknik perhitungan kompleksitas dengan relasi rekurens

- Perhatikan contoh alg faktorial berikut ini

Alg Faktorial

function *Faktorial* (**input** *n* :**integer**)→ **integer**

{ mengembalikan nilai n!;

basis : jika n = 0, maka 0! = 1

rekurens: jika n > 0, maka n! = n × (n-1)!

}

ALGORITMA:

```
if n = 0 then
    return 1                    { basis }
else
    return  n * Faktorial(n – 1)   { rekurens }
end
```

Kompleksitas Faktorial

- Operasi dasar : perkalian
- Kompleksitas waktu :
  - untuk kasus basis, tidak ada operasi perkalian → (0)
  - untuk kasus rekurens, kompleksitas waktu diukur dari jumlah perkalian (1) ditambah kompleksitas waktu untuk faktorial (n-1)

Kompleksitas Faktorial

- Relasi Rekurens :

  $T(n) = 0 \quad , n = 0$

  $T(n) = T(n-1)+1 \quad , n > 0$

- Kompleksitas Waktu :

  $T(n) = T(n-1) + 1$ substitusi $T(n-1) = T(n-2) + 1$

  $= [T(n-2) + 1] + 1 = T(n-2) + 2$ substitusi $T(n-2) = T(n-3) + 1$

  $= [T(n-3) + 2] + 1 = T(n-3) + 3$

  $= .....$

  $= T(0) + n$

  $= 0 + n$

  Jadi, $T(n) = n$

  $T(n) \in O(n)$

Algoritma Menghitung Pangkat

```
Function pangkat(input x,n:integer):integer
   if n = 0 then
      pangkat ← 1
   else
      pangkat ← pangkat(x,n-1)*x
   endif
endfunction
```

Kompleksitas Perpangkatan

- Relasi Rekurens :

  $T(n) = 0 \quad , n = 0$

  $T(n) = T(n-1)+1 \quad , n > 0$

- Kompleksitas Waktu :

  $T(n) = T(n-1) + 1$ substitusi $T(n-1) = T(n-2) + 1$

  $= [T(n-2) + 1] + 1 = T(n-2) + 2$ substitusi $T(n-2) = T(n-3) + 1$

  $= [T(n-3) + 2] + 1 = T(n-3) + 3$

  $= .....$

  $= T(0) + n$

  $= 0 + n$

  Jadi, $T(n) = n$

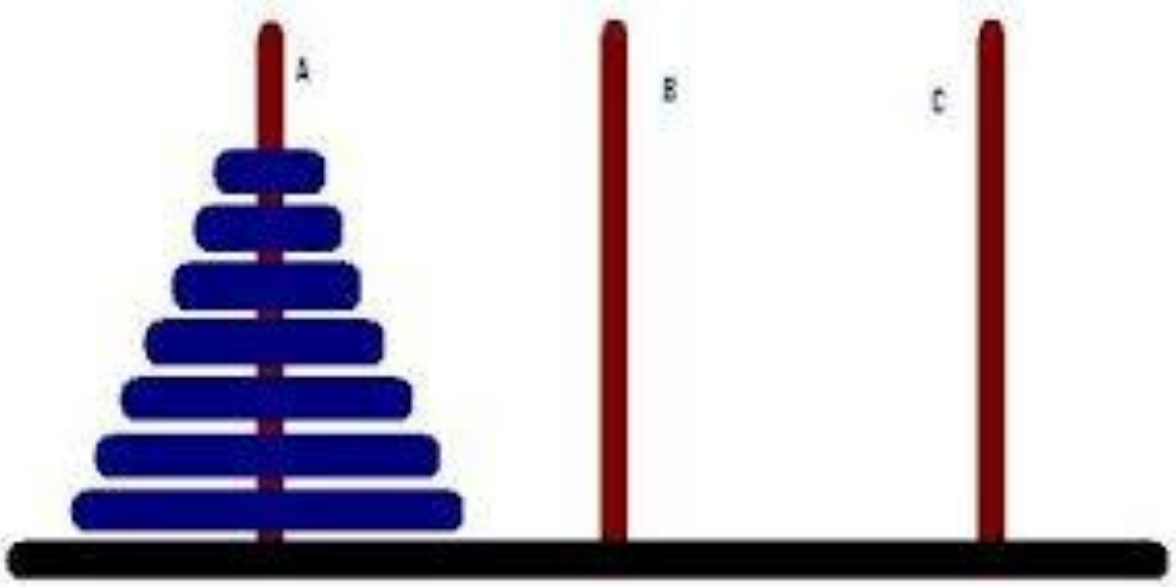
  $T(n) \in O(n)$

Menara Hanoi (*The Tower of Hanoi*)

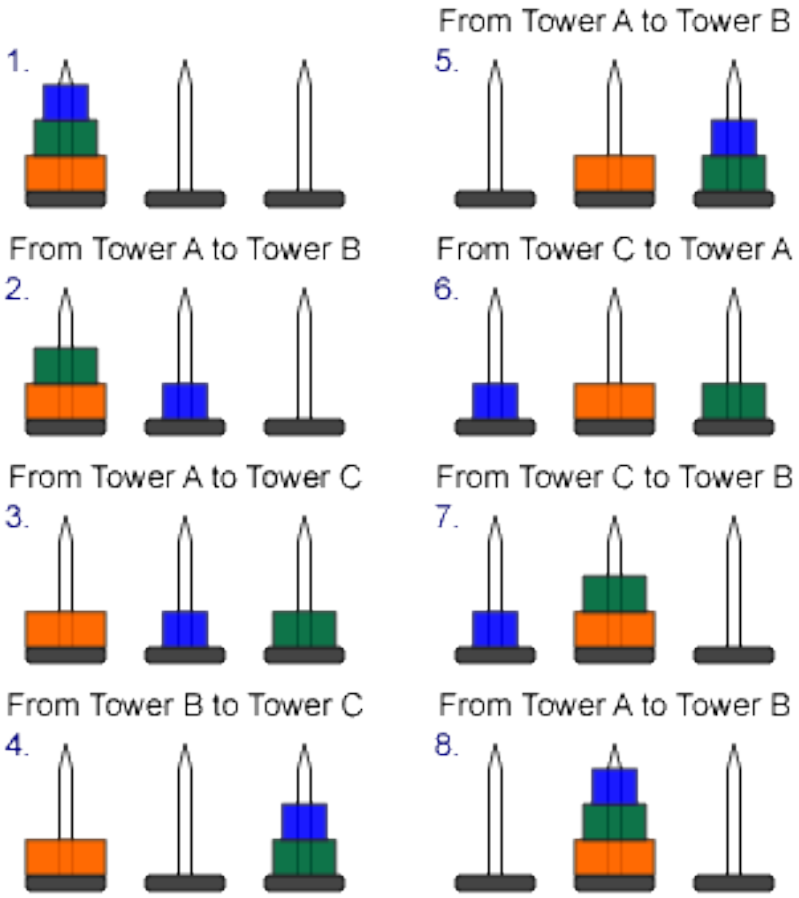
Menara Hanoi adalah sebuah *puzzle* yang terkenal pada akhir abad 19. Puzzle ini ditemukan oleh matematikawan Perancis, Edouard Lucas.

Dikisahkan bahwa di kota Hanoi, Vietnam, terdapat tiga buah tiang tegak setinggi 5 meter dan 64 buah piringan (*disk*) dari berbagai ukuran. Tiap piringan mempunyai lubang di tengahnya yang memungkinkannya untuk dimasukkan ke dalam tiang. Pada mulanya piringan tersebut tersusun pada sebuah tiang sedemikian rupa sehingga piringan yang di bawah mempunyai ukuran lebih besar daripada ukuran piringan di atasnya. Pendeta Budha memberi pertanyaan kepada murid-muridnyanya: bagaimana memindahkan seluruh piringan tersebut ke sebuah tiang yang lain; setiap kali hanya satu piringan yang boleh dipindahkan, tetapi tidak boleh ada piringan besar di atas piringan kecil. Tiang yang satu lagi dapat dipakai sebagai tempat peralihan dengan tetap memegang aturan yang telah disebutkan. Menurut legenda pendeta Budha, bila pemindahan seluruh piringan itu berhasil dilakukan, maka dunia akan kiamat!

Pemodelan:



- Kasus untuk *n* = 3 piringan



- Secara umum, untuk *n* piringan, penyelesaian dengan cara berpikir rekursif adalah sebagai berikut:

  Kita harus memindahkan piringan paling bawah terlebih dahulu ke tiang *B* sebagai alas bagi piringan yang lain. Untuk mencapai maksud demikian, berpikirlah secara rekursif: pindahkan *n* – 1 piringan teratas dari *A* ke *C*, lalu pindahkan piringan paling bawah dari *A* ke *B*, lalu pindahkan *n* – 1 piringan dari *C* ke *B*.

  pindahkan n – 1 piringan dari A ke C

  pindahkan 1 piringan terbawah dari A ke B

  pindahkan n – 1 piringan dari C ke B

  Selanjutnya dengan tetap berpikir rekursif-pekerjaan memindahkan *n* – 1 piringan dari sebuah tiang ke tiang lain dapat dibayangkan sebagai memindahkan *n* – 2 piringan antara kedua tiang tersebut, lalu memindahkan piringan terbawah dari sebuah tiang ke tiang lain, begitu seterusnya.

- Misalkan H_n menyatakan jumlah perpindahan piringan yang dibutuhkan untuk memecahkan teka-teki Menara Hanoi.

pindahkan n – 1 piringan dari A ke C → H_{n-1} *kali*

pindahkan 1 piringan terbawah dari A ke B → *1 kali*

pindahkan n – 1 piringan dari C ke B → H_{n-1} *kali*

Maka jumlah perpindahan yang terjadi adalah:

$$H_n = 2H_{n-1} + 1$$

dengan kondisi awal $H_1 = 1$

Algoritma Menara Hanoi

```
Procedure hanoi(input n, A, B, C : integer)
Algoritma
  if n = 1 then
    write (‘Pindahkan piringan dari’,A,’ke’,B)
  else
    hanoi(n-1,A,C,B)
    write(‘Pindahkan piringan dari’,A,’ke’,B)
    hanoi(n-1,C,B,A)
  endif
endfunction
```

Kompleksitas waktu

- **Relasi Rekurens :**

  $T(n) = \quad 1 \quad , n = 1$

  $T(n) = 2T(n-1)+1 \quad , n > 1$

- **Kompleksitas Waktu :**

  $T(n) = 1 + 2T(n-1)$

  $= 1 + 2(1 + 2T(n - 2)) = 1 + 2 + 2^2T(n - 2)$

  $= 1 + 2 + 2^2 (1 + 2T(n - 3)) = 1 + 2 + 2^2 + 2^3T(n - 3)$

  $= .....$

  $= (1 + 2 + 2^2 + ... + 2^{n-2}) + 2^{n-1}T(1)$

  $= 1 + 2 + 2^2 + ... + 2^{n-2}.1$

  $= 2^n - 1$

  Jadi, $T(n) = 2^n - 1$

  $T(n) \in O(2^n)$

Kompleksitas waktu

- Untuk n = 64 piringan, jumlah perpindahan piringan yang terjadi adalah

  $T(64) = 2^{64} - 1 = 18.446.744.073.709.551.615$

- Jika satu kali pemindahan piringan membutuhkan waktu 1 detik, maka waktu yang diperlukan adalah

  18.446.744.073.709.551.615 detik

  atau setara dengan 584.942.417.355 tahun atau sekitar 584 milyar tahun

  (Sudah kiamatkah dunia sesuai legenda?)

Latihan

- Buatlah algoritma Fibbonanci!
- Carilah kompleksitas dari Algoritma Fibbonanci!

Solusi : Alg Fibonanci

function Fibonacci (input n : integer) → integer;

Algoritma :

```
if (n=0) then
        Fibonacci ← 0
else if  (n = 1) then
            Fibonacci ← 1
      else
            Fibonacci ← Fibonacci  (n-1) + Fibonacci  (n-2)
       endif
endif
endfunction
```

Latihan (Persoalan Minimum dan Maksimum)

procedureMinMaks2(inputA : TabelInt, i, j : integer,

outputmin, 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

Persoalan Minimum & Maksimum

```
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
```

Persoalan Minimum & Maksimum

```
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
```

Latihan (Persoalan Minimum dan Maksimum)

- Buatlah relasi rekurens nya!
- Berapakah Big O nya?