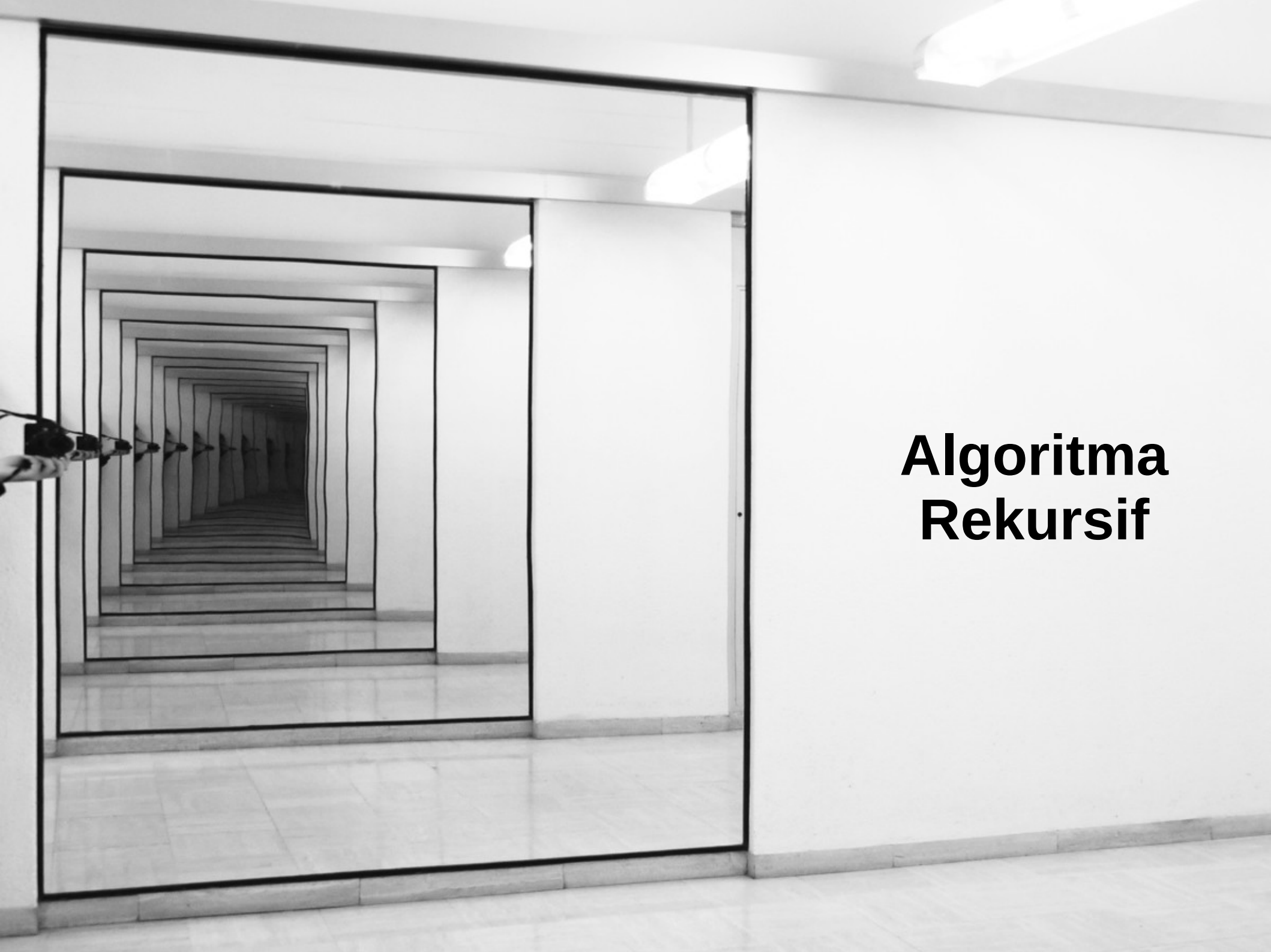


Algoritma Rekursif

Strategi Algoritma

- Garis besar pemecahan masalah
- Contoh pendekatan : Brute Force yaitu memecahkan masalah dengan mencoba seluruh kemungkinan solusi lalu mengevaluasi satu per satu
- Contoh 1 : crack password dengan seluruh karakter yang mungkin (bayangkan mencoba PIN ATM)
- Contoh 2 : menghitung pangkat a^n dengan mengalikan a*a* …. *a hingga n kali
- Contoh 3 : algoritma sorting selection & bubble

Strategi (2)

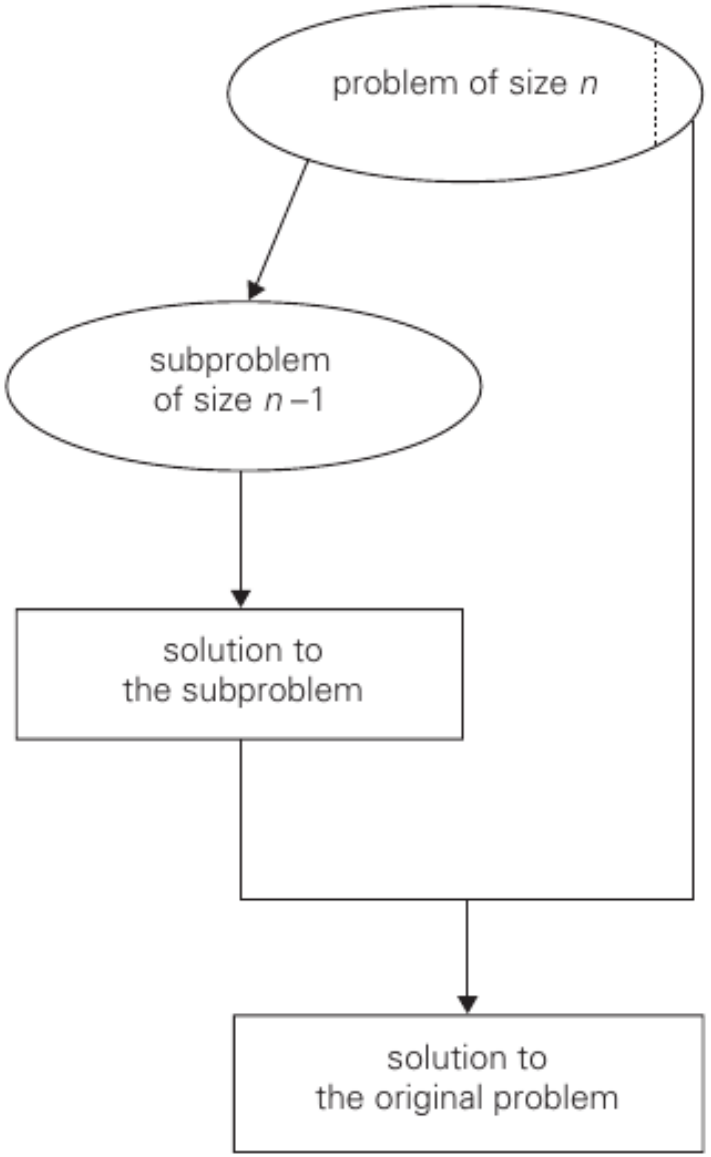
- Decrease and conquer : kurangi dan taklukan
- Mengurangi / mengecilkan ukuran problem / masalah
- Adanya relasi hubungan antar solusi problem & solusi problem yang ukurannya lebih kecil
- Contoh problem : mengolah matriks 100x100
- Problem yang ukurannya lebih kecil: matriks 10x10
- Adanya relasi tersebut memungkinkan kita mulai memecahkan dari masalah yang size nya lebih kecil

Strategi (3)

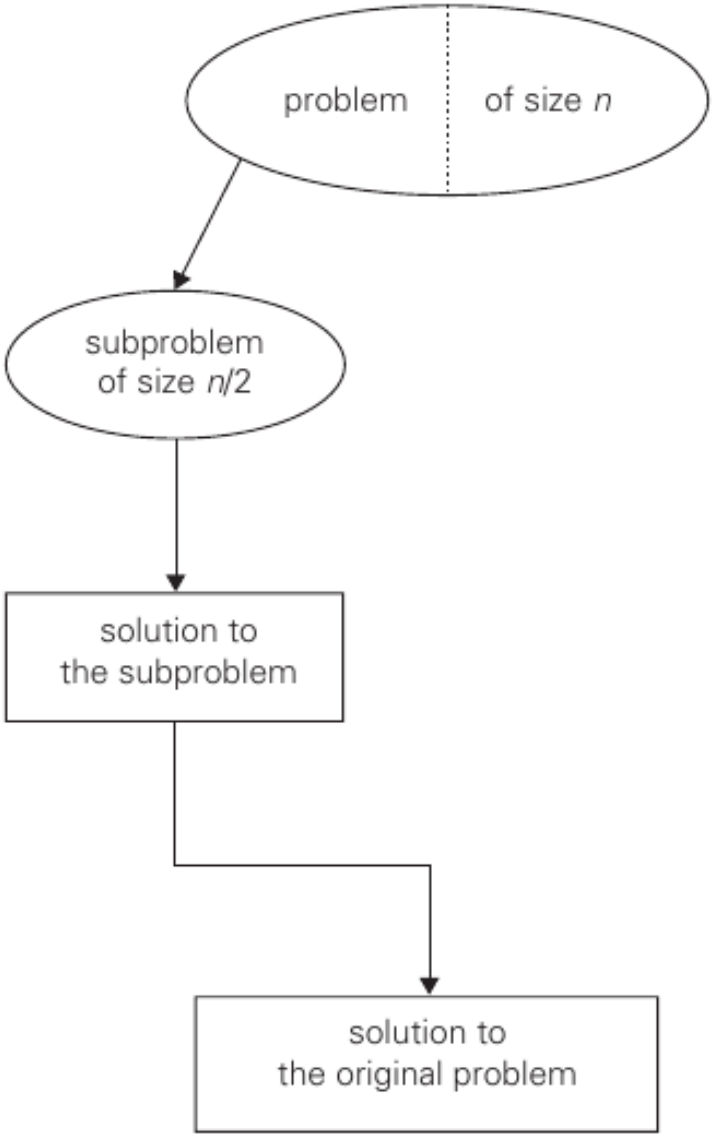
- Decrease & conquer : **top-down** & **bottom Up**
- **Top-down** memecah masalah dari besar diuraikan jadi masalah kecil yang kita selesaikan
- **Bottom up**: memecahkan masalah untuk size besar dimulai dari menyelesaikan size kecil dulu (pendekatan incremental)
- Variasi
  - Decrease by constant ex: n-1, n-2, n-3, n-4, dst
  - Decrease by factor ex: n, n/2, n/4, n/8, n/16 … dst
  - Variable size decrease: perubahan ukuran bervariasi

Strategi Algoritma

Decrease by 1



Decrease by half factor



Contoh

- Menghitung perpangkatan f(n) = a^n
- Dengan pendekatan decrease by one yaitu ukuran problem dikurangi satu per satu :

$$f(n) = \begin{cases} f(n-1) \cdot a & \text{if } n > 0, \\ 1 & \text{if } n = 0, \end{cases}$$

- Dengan pendekatan decrease by factor, ukuran problem dikurangi jadi setengah :

$$a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even and positive,} \\ (a^{(n-1)/2})^2 \cdot a & \text{if } n \text{ is odd,} \\ 1 & \text{if } n = 0. \end{cases}$$

Recurrence Relation

- Recurrence : berulang, relation: hubungan
- Rekursif : algoritma yang memanggil dirinya sendiri
- Syarat algoritma rekursif diterima :
  - Algoritma harus *terminate* (berhenti)
  - Menghasilkan progress komputasi, bekerja dg baik
- Contoh algoritma menghitung nilai faktorial :

  n! = n*(n-1)*(n-2)*….*3*2*1 kita eksplor 2 relasi

  yaitu 1! = 1 dan n! = n*(n-1)! Untuk n > 1

Faktorial (2)

- Faktorial sebagai fungsi matematik :

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

- Contoh : 5! = 5*4*3*2*1 = 120
- Definisi rekursif : definisi fungsi lewat dirinya sendiri

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \geq 1. \end{cases}$$

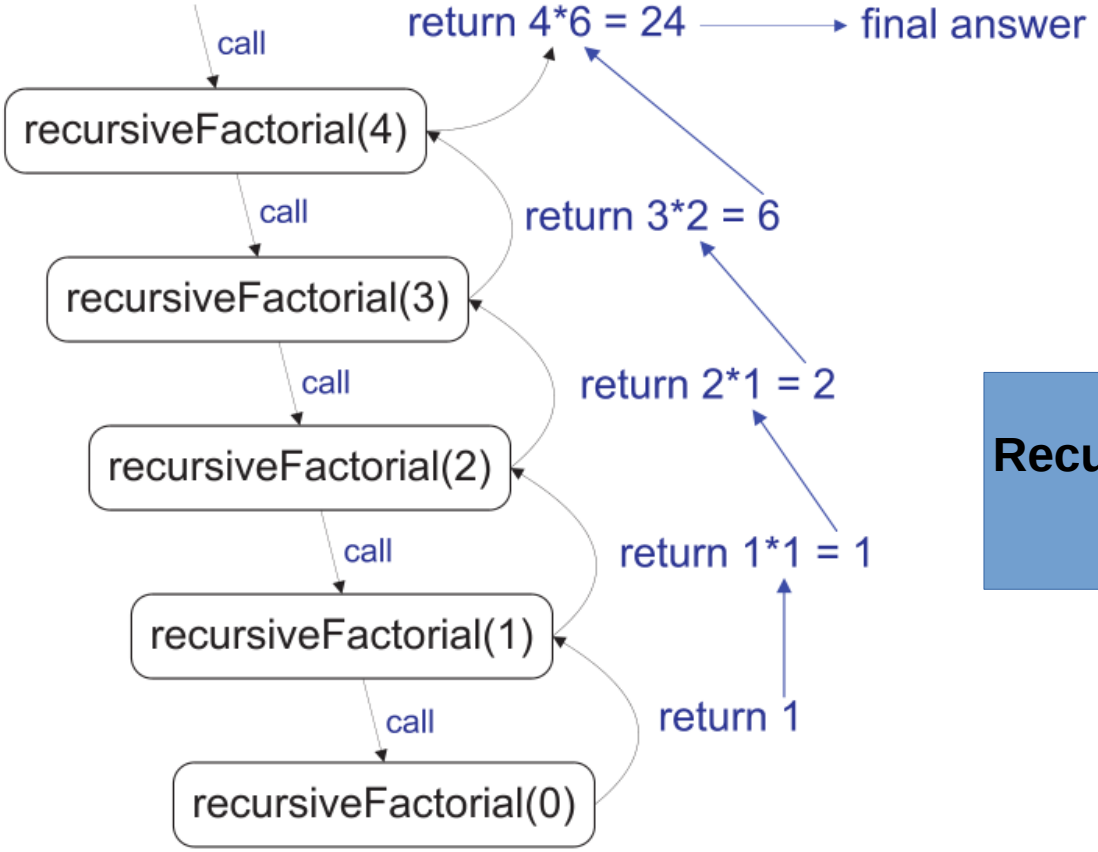
- Kasus basis : yang memberi nilai fix dari fungsi
- Kasus rekursif : membuat definisi fungsi dari dirinya sendiri

Algoritma rekursif

- Desain algoritma yang diimplementasi dalam bentuk fungsi yang didalamnya memanggil fungsi tersebut lagi dengan mengurangi ukuran problem input sehingga menjamin terminasi (berhenti)-nya algoritma tersebut
- Struktur Algoritma rekursif terbagi jadi dua :
  - **Basis** : kondisi awal / akhir solusi pasti diketahui, bagian ini menjamin algoritma rekursif berhenti
  - **Rekursif** : menerima problem dalam ukuran yang mengecil => misal n jadi n-1 atau n/2
- Perhitungan faktorial merupakan contoh rekursif

Implementasi faktorial

```
int recursiveFactorial(int n) {          // recursive factorial function
  if (n == 0) return 1;                  // basis case
  else return n * recursiveFactorial(n-1);   // recursive case
}
```



Recursion trace dari fungsi recursiveFactorial

Contoh rekursif (2)

- Menjumlah isi array tanpa looping
- Dengan rekursif kita akan menjumlah isi array (asumsi array integer/bilangan)

Algorithm LinearSum(A, n):

Input: A integer array A and an integer $n \geq 1$, such that A has at least n elements

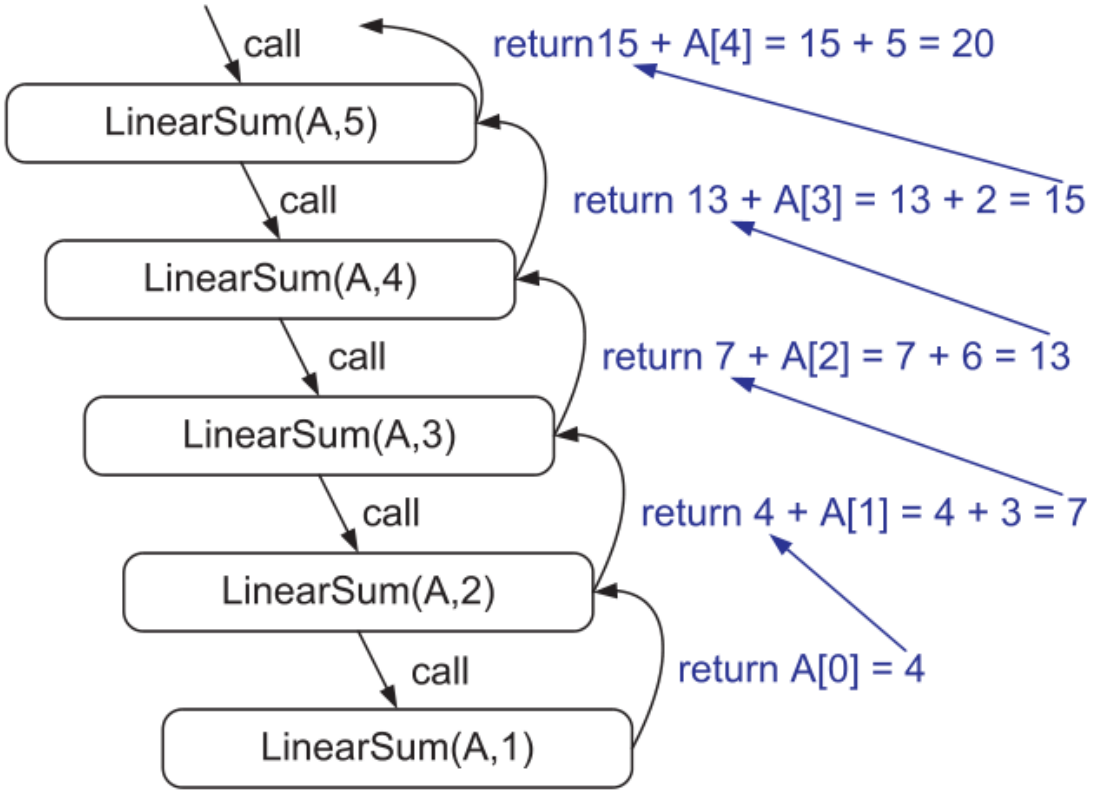
Output: The sum of the first n integers in A

```
if n = 1 then
    return A[0]
else
    return LinearSum(A, n − 1) + A[n − 1]
```

- Tiap kali rekursif ukuran yang dijumlah berkurang

Recursion Trace contoh 2

- Menjumlah isi array dengan rekursif
- Misal array input A={ 4,3,6,2,5 } n=5, berikut recursion trace dari **LinearSum(A, n)**



Contoh rekursif (3)

- Reverse array (membalik isi array): bekerja dengan menukar posisi ujung dengan ujung (0 vs n-1) , lalu (1 vs n-2), dst
- Pseudocode membalik array dengan rekursif

Algorithm ReverseArray(A, i, j):

Input: An array A and nonnegative integer indices i and j

Output: The reversal of the elements in A starting at index i and ending at j

if $i < j$ **then**

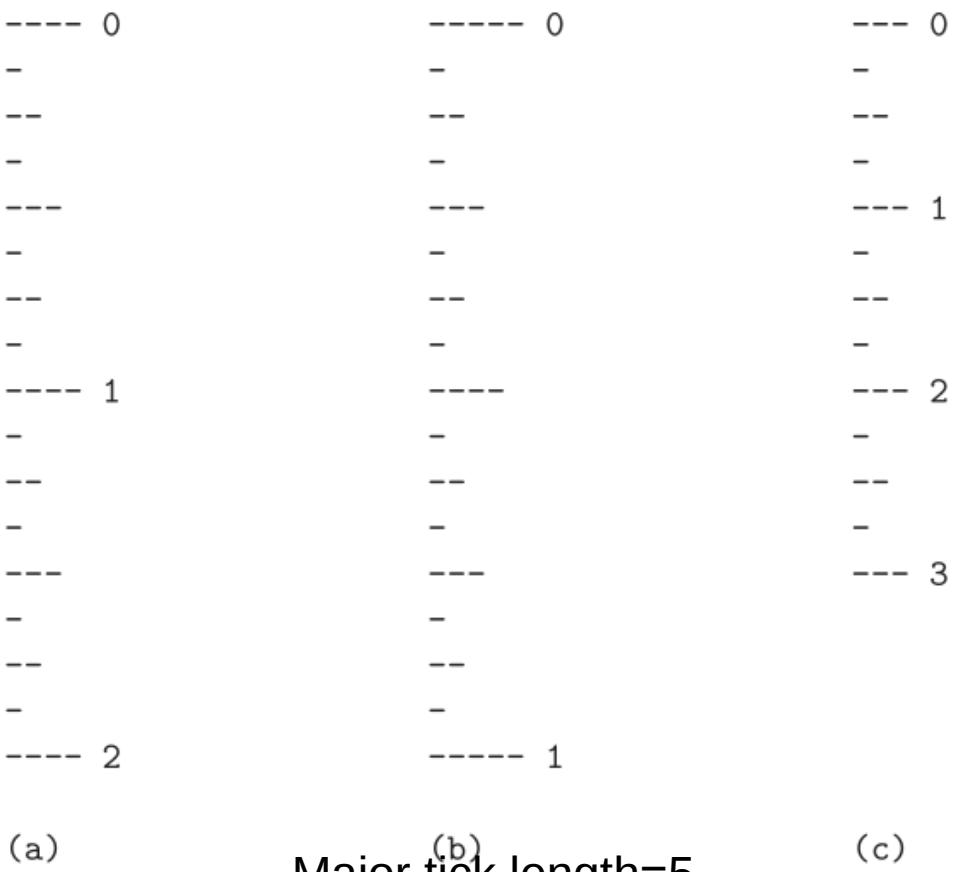
Swap $A[i]$ and $A[j]$

ReverseArray$(A, i+1, j-1)$

return

Contoh problem lain dg sifat rekusif

- Membuat penanda pada penggaris
- Major tick : bagian penanda yang paling panjang



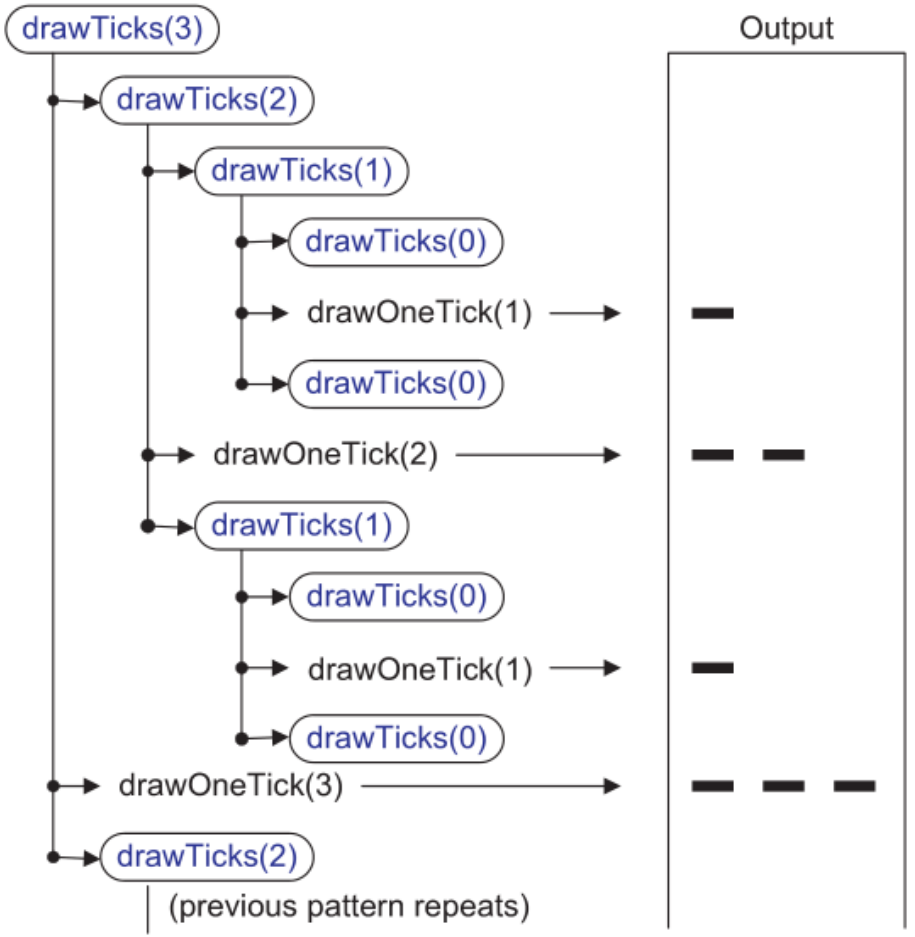
Major tick length=4
Panjang 2

Major tick length=5
Panjang 1

Major tick length=3
Panjang 3

Menggambar penggaris secara rekursif

- drawTicks :skema rekursif, drawOneTick: fungsi menggambar garis strip-nya

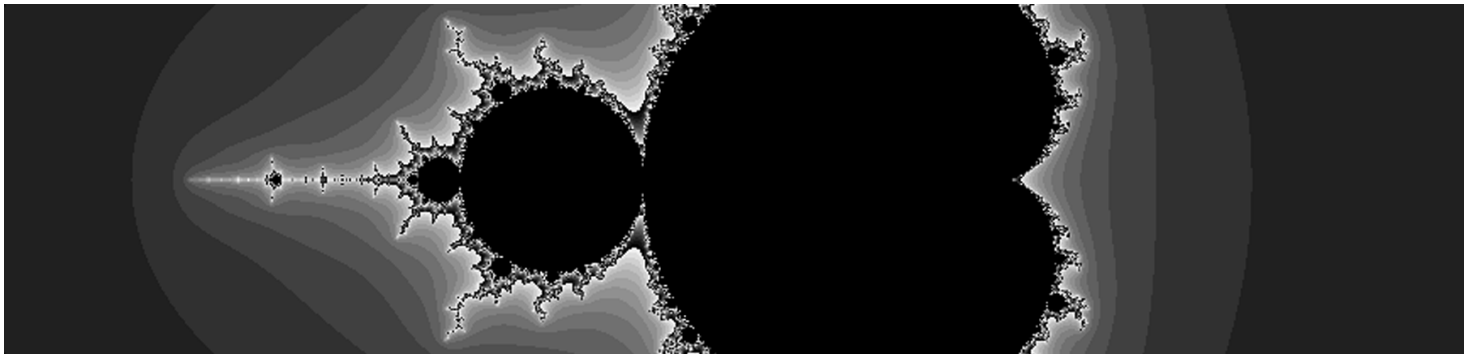


Implementasi rekursif penggaris

```
                                        // one tick with optional label
void drawOneTick(int tickLength, int tickLabel = -1) {
  for (int i = 0; i < tickLength; i++)
    cout << "-";
  if (tickLabel >= 0) cout << " " << tickLabel;
  cout << "\n";
}
void drawTicks(int tickLength) {        // draw ticks of given length
  if (tickLength > 0) {                 // stop when length drops to 0
    drawTicks(tickLength-1);            // recursively draw left ticks
    drawOneTick(tickLength);            // draw center tick
    drawTicks(tickLength-1);            // recursively draw right ticks
  }
}
void drawRuler(int nInches, int majorLength) {// draw the entire ruler
  drawOneTick(majorLength, 0);          // draw tick 0 and its label
  for (int i = 1; i <= nInches; i++) {
    drawTicks(majorLength-1);           // draw ticks for this inch
    drawOneTick(majorLength, i);        // draw tick i and its label
  }
}
```

Rekursif pada grafis

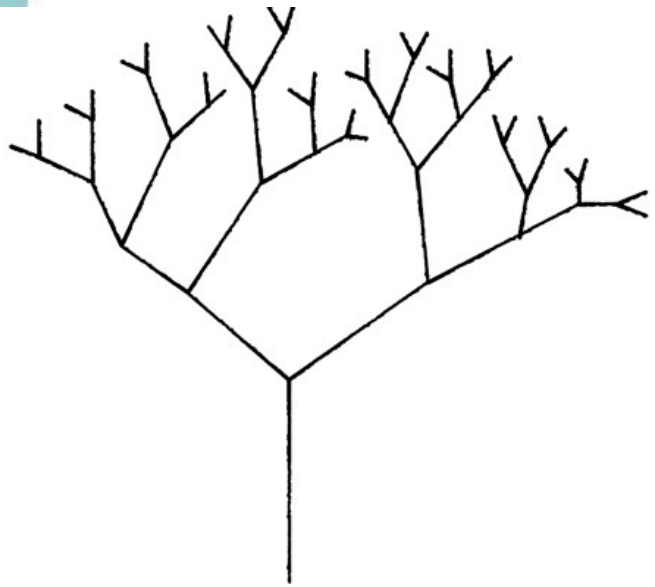
- Tidak hanya menghitung numerik (angka), rekursif banyak digunakan untuk menghasilkan visualisasi grafis
- Cabang matematika yang banyak memakai visualisasi grafis adalah geometri
- Matematikawan berusaha membuat simulasi bentuk-bentuk geometri lewat model (fungsi)

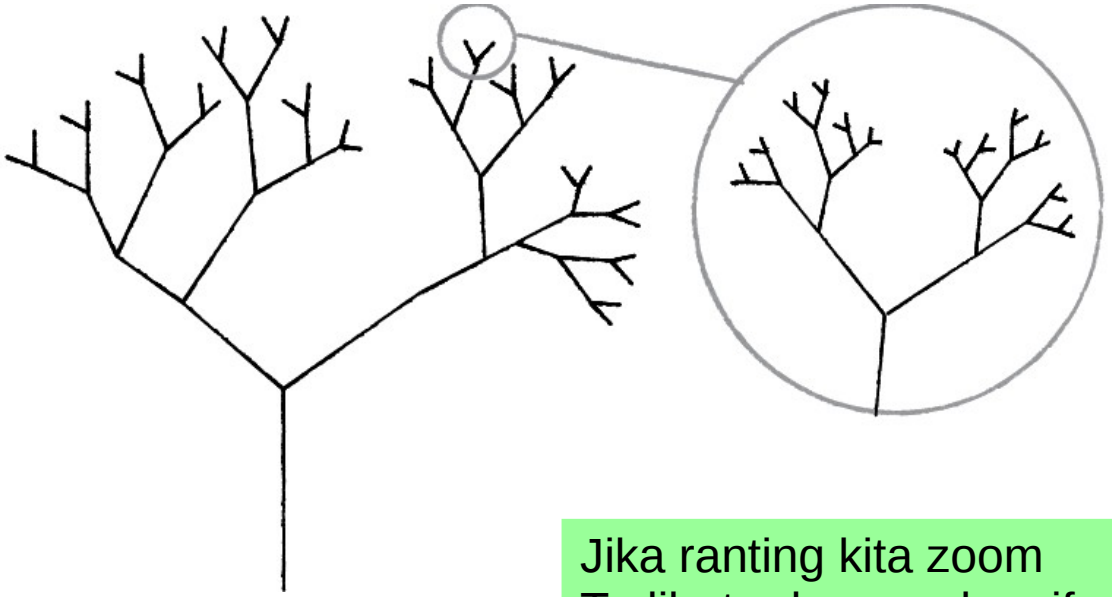


Diantara model Itu ada yang di Sebut Fractals

Contoh Fractal

- Fractal pohon

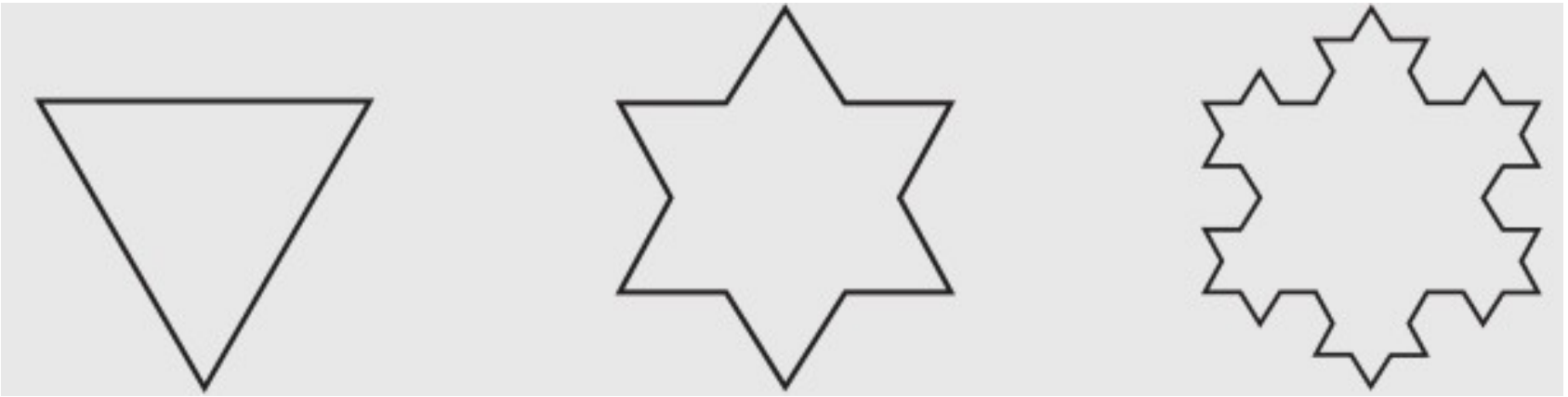




Jika ranting kita zoom
Terlihat adanya rekursif
Mengikuti bentuk dahan utama

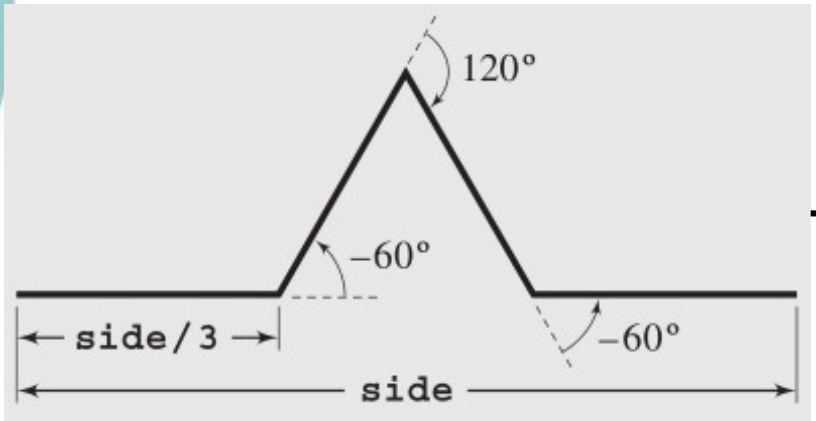
Contoh Fractal(2)

- Kurva Koch, dibuat oleh matematikawan Helge von Koch pada tahun 1904
- Fraktal ini tergolong simpel untuk memodelkan bunga salju (snowflake)



Kurva Koch dalam Recursive

- Algoritma membagi tiap sisi jadi tiga bagian




```
drawFourLines (side, level)
   if (level == 0)
      draw a line;
   else
      drawFourLines(side/3, level-1);
      turn left 60°;
      drawFourLines(side/3, level-1);
      turn right 120°;
      drawFourLines(side/3, level-1);
      turn left 60°;
      drawFourLines(side/3, level-1);
```

Fibonacci Number

- Dikenalkan pada komunitas matematikawan eropa barat pada 1202M oleh Fibonacci dalam bukunya Liber Abaci
- Meski riset menunjukkan hasil orisinil tentang angka ini terlacak jauh dari India 200SM
- Fibonacci dalam relasi recurrence:

$$\mathrm{Fib}(n) = \begin{cases} n & \text{if } n < 2 \\ \mathrm{Fib}(n-2) + \mathrm{Fib}(n-1) & \text{otherwise} \end{cases}$$

- Sequence properties:

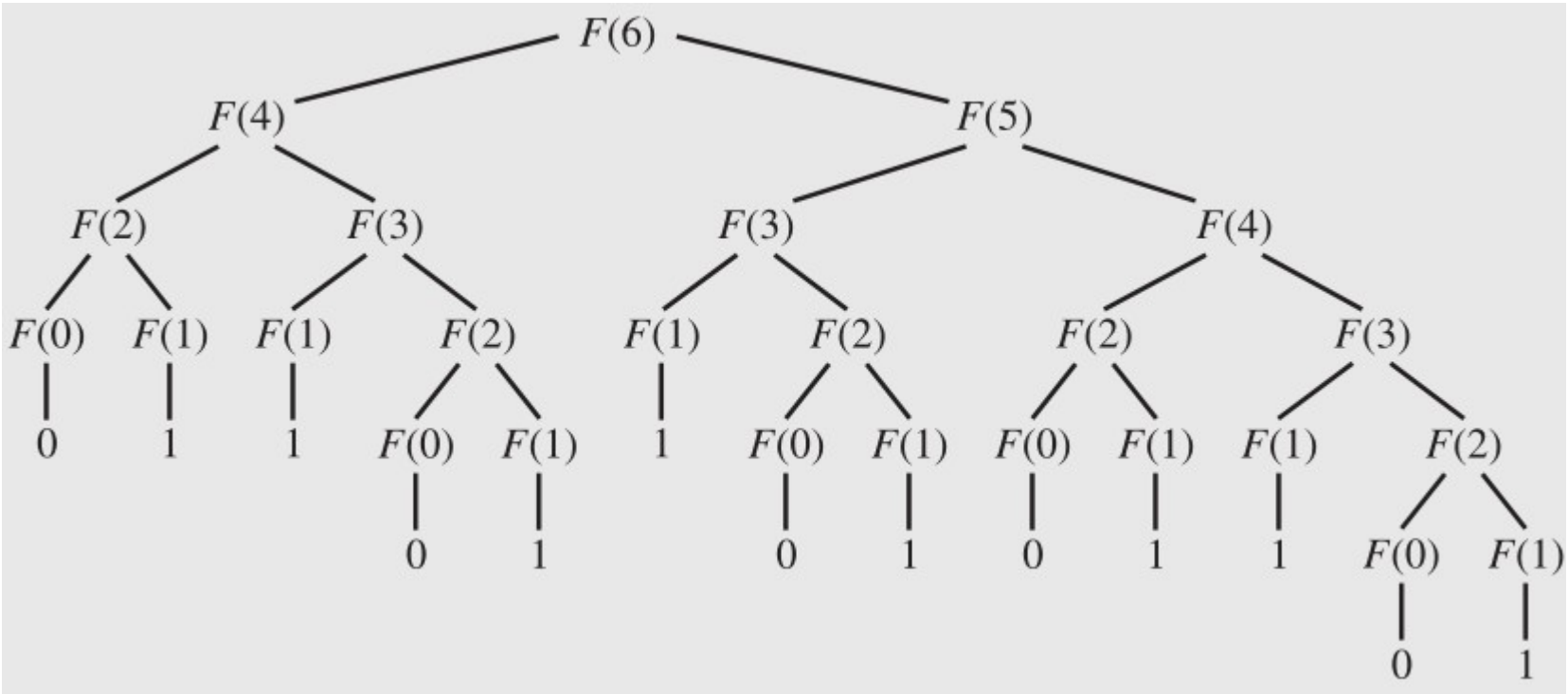
The first 21 Fibonacci numbers F_n are:[2]

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Execution trace of Fibonacci

- banyak terjadi pengulangan hasil komputasi

```
Fib(6) =                  Fib(4)                      + Fib(5)
       =     Fib(2)         +         Fib(3)         + Fib(5)
       = Fib(0)+Fib(1)      +         Fib(3)         + Fib(5)
       =    0    +    1     +         Fib(3)         + Fib(5)
       =         1          + Fib(1)+    Fib(2)     + Fib(5)
       =         1          + Fib(1)+Fib(0)+Fib(1)  + Fib(5)
```



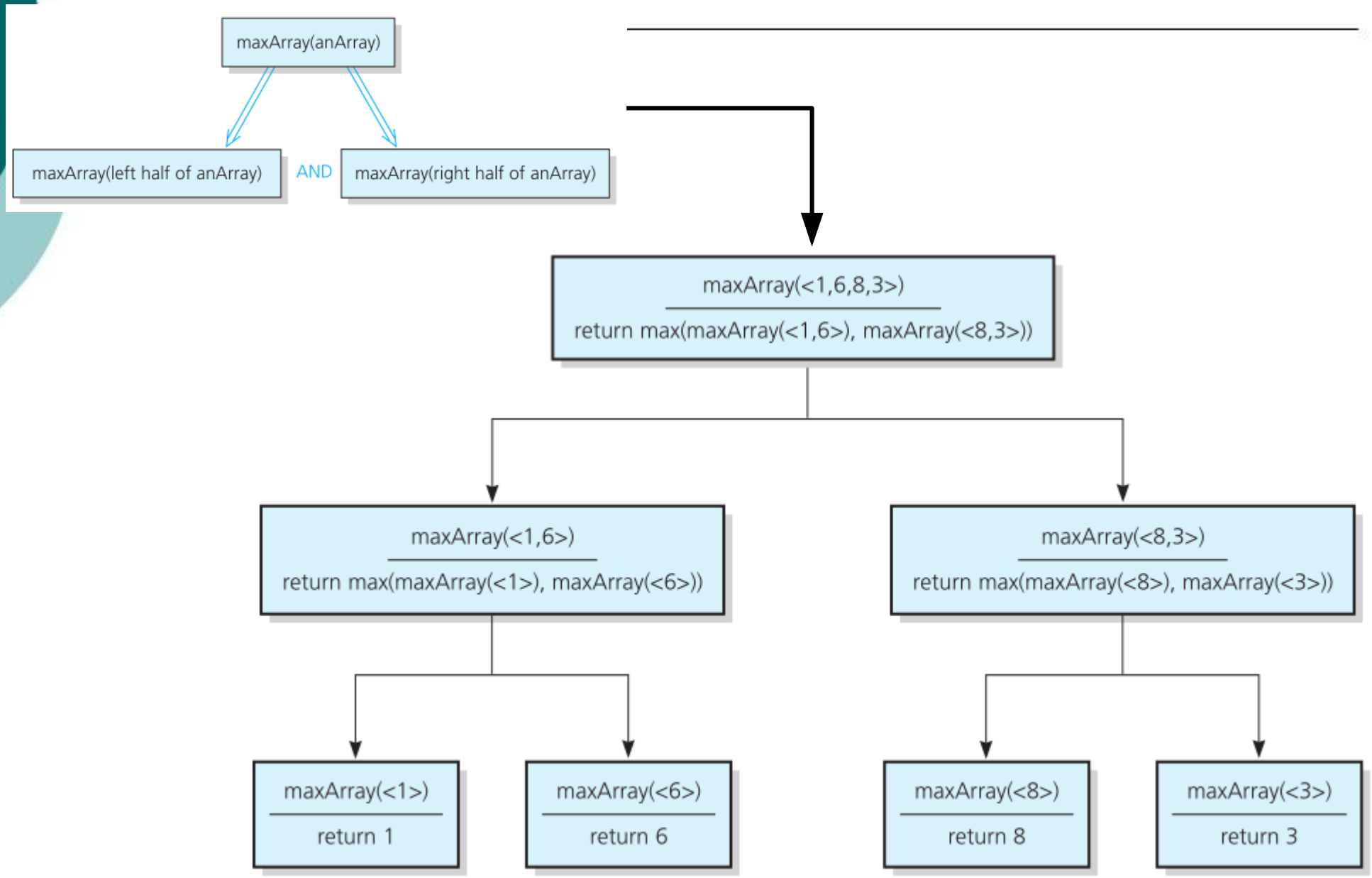
Mencari Elemen Maximum secara rekursif

- Masalah pencarian umumnya diselesaikan dengan iterasi
- Akan tetapi kita bisa membuat formulasi rekursifnya dengan mengurangi ukuran array 1 demi 1 (decrease conquer)
- Array dibagi dua jadi kiri dan kanan

Kondisi basis: hanya ada 1 entry di array

```
if (anArray has only one entry)
    maxArray(anArray) is the entry in anArray
else if (anArray has more than one entry)
    maxArray(anArray) is the maximum of
        maxArray(left half of anArray) and maxArray(right half of anArray)
```

Search Max recursively



Binary Search

- Teknik pencarian dengan syarat kondisi array terurut
- Pencarian dibagi dalam dua bagian array (secara rekursif) dengan mencari titik tengah

```
binarySearch(anArray: ArrayType, target: ValueType)

    if (anArray is of size 1)
       Determine if anArray's value is equal to target
    else
    {
       Find the midpoint of anArray
       Determine which half of anArray contains target
       if (target is in the first half of anArray)
          binarySearch(first half of anArray, target)
       else
          binarySearch(second half of anArray, target)
    }
```

Midpoint=titik tengah

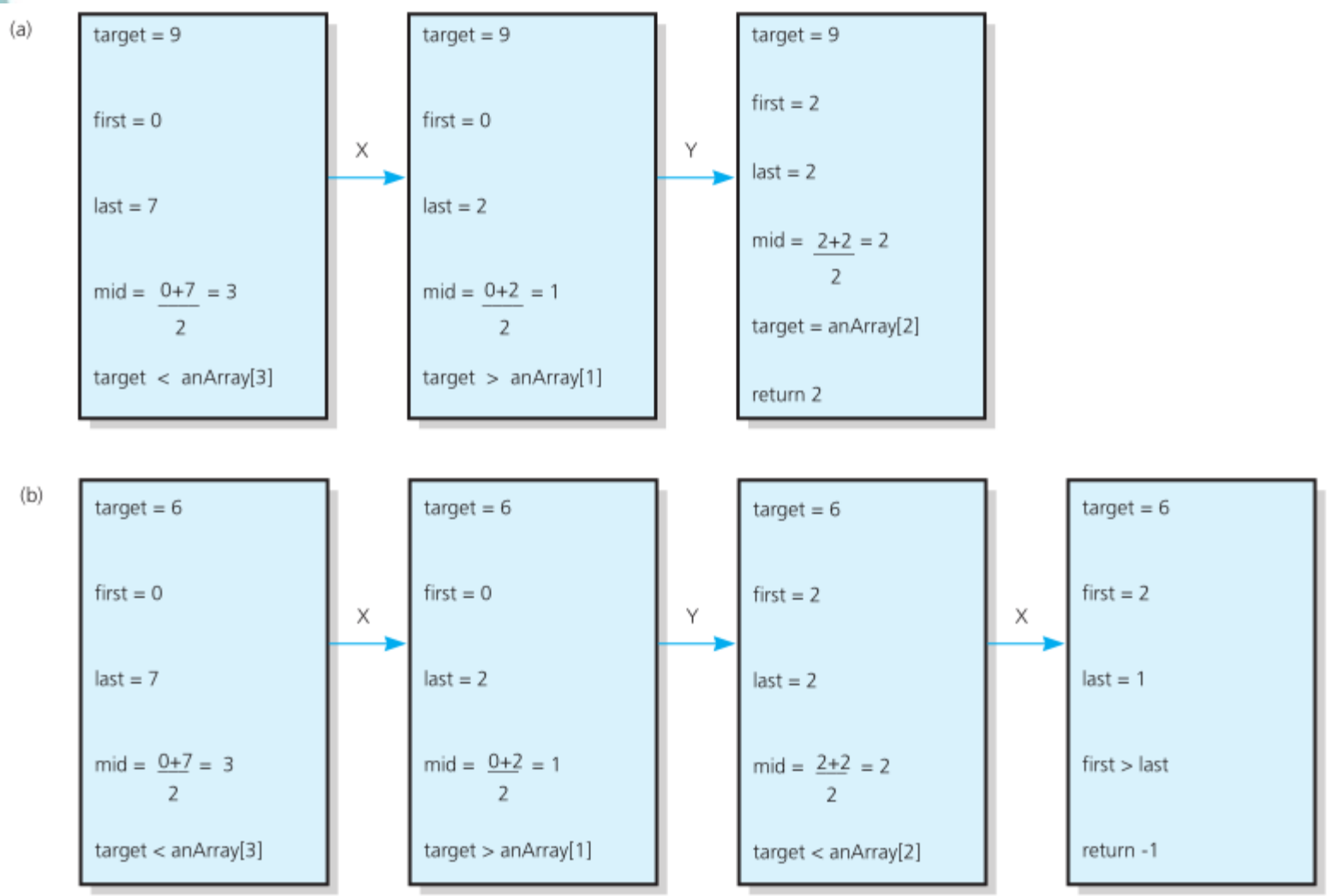
Bagaimana mencari titik tengah?

- Titik tengah = (posisi awal + akhir) / 2
- Test apakah elemen tengah = elemen yg dicari
- Proses pemilihan bagian array yang di search recursive:

  jika elemen yang disearch < elemen tengah : pilih kiri. dan sebaliknya
- Kondisi Basis jika elemen tidak ketemu:
  - Array hasil bagi: ukuran 1 => (n+n)/2 = n
  - Selalu sampai pada kondisi posisi awal > akhir

Contoh binary search

- Array=[1,5,9,12, 15,21, 29, 31]



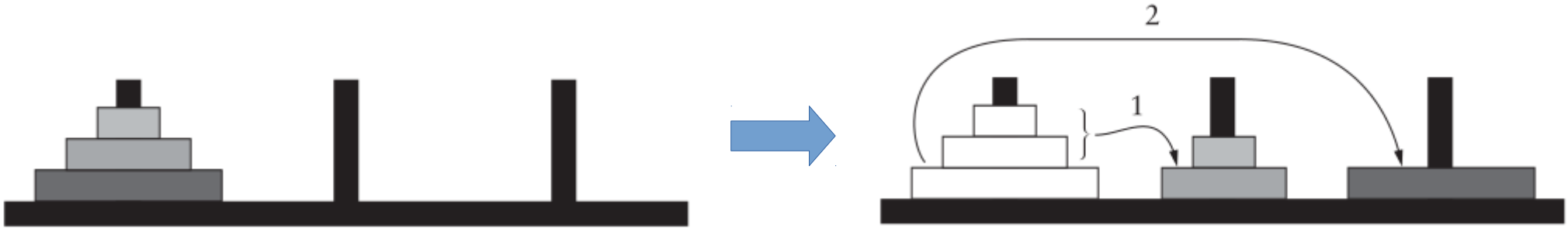
Implementasi Binary Search

```
int binarySearch(const int anArray[], int first, int last, int target)
{
   int index;
   if (first > last)
      index = -1;   // target not in original array
   else
   {
      // If target is in anArray,
      // anArray[first] <= target <= anArray[last]
      int mid = first + (last - first) / 2;
      if (target == anArray[mid])
         index = mid; // target found at anArray[mid]
      else if (target < anArray[mid])
         // Point X
         index = binarySearch(anArray, first, mid - 1, target);
      else
         // Point Y
         index = binarySearch(anArray, mid + 1, last, target);
   }  // end if

   return index;
}  // end binarySearch
```

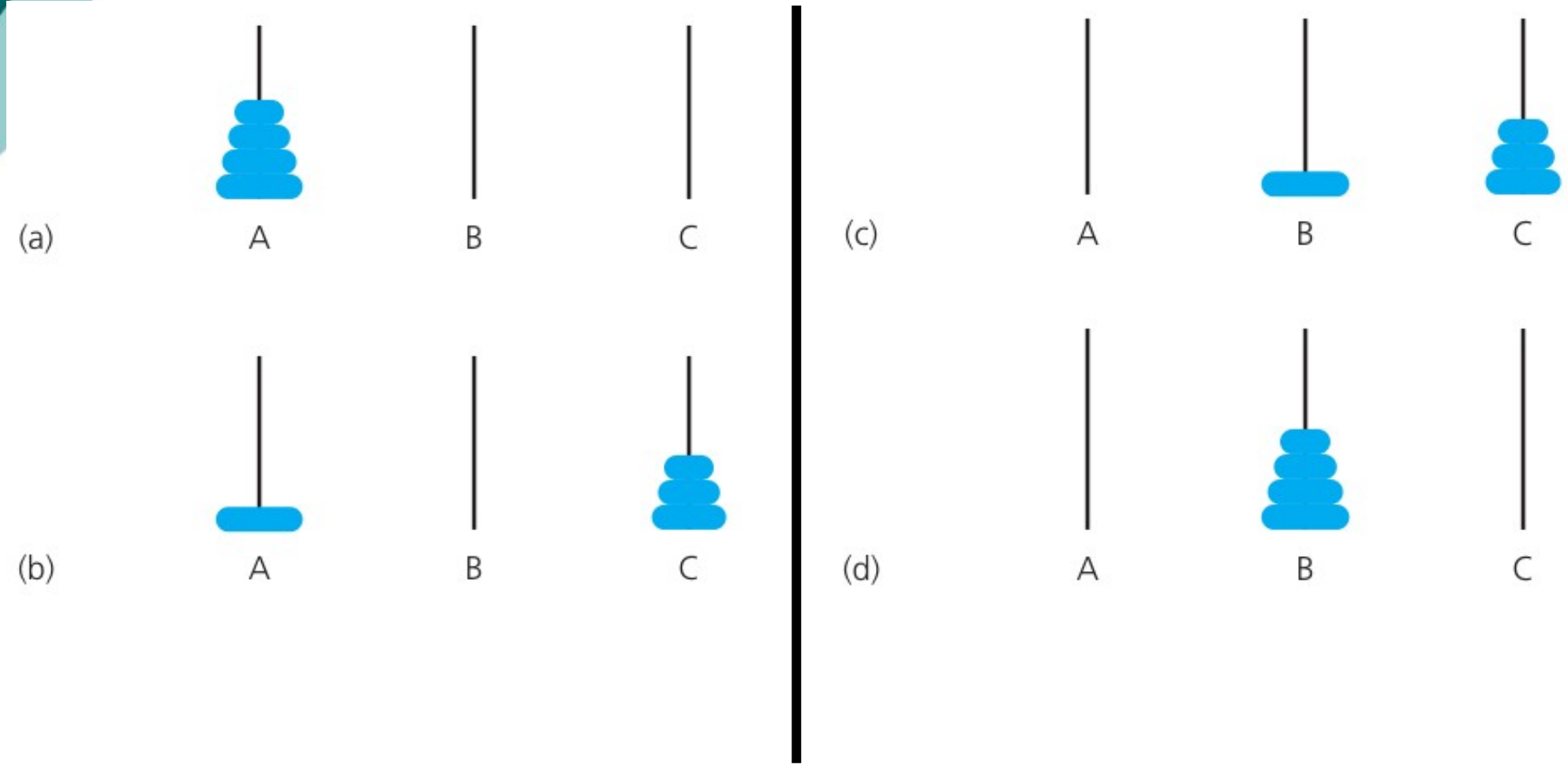
Tower of Hanoi

- Contoh klasik pada Computer Science untuk masalah rekursif
- Legenda pendeta Budha memindahkan cakram emas dari satu menara ke menara lain
- Syarat pemindahan: cakram kecil **tidak boleh ditumpuk dibawah** cakram yang lebih besar
- Hanya ada 1 menara bantu untuk temporary



Problem Tower (Visual)

- Memindah 4 cakram, dimulai dari 3 cakram diatasnya terlebih dulu



Tower of Hanoi

- Pattern:
  - Jika jumlah cakram ada 4: pindahkan dulu 3 cakram paling atas ke temporary, cakram paling bawah taruh ke menara tujuan
  - Jika jumlah cakram ada 3: pindahkan dulu 2 cakram paling atas ke temporary, cakram paling bawah taruh ke menara tujuan
- Masalah ini hanya bisa dipecahkan jika cakram yang lebih kecil (diatasnya) sudah dipindahkan dulu
- Ini berarti problem dengan ukuran lebih kecil

Formulasi rekursif problem menara

- Kita memindahkan cakram-cakram yang lebih kecil dulu ke menara temporary
- Fungsi : towers(n, A, B, C) pindah dari A ke B, C menara bantu (temporary)
- Kondisi awal: seluruh cakram ada di menara A (asal), menara B (tujuan), C (temporary) kosong

  towers(n-1, A, C, B): pindah n-1 cakram ke menara C
- towers(1, A, B, C): pindah cakram terbesar ke B
- Kondisi akhir: cakram terbesar sudah di menara B

  towers(n-1, C, B, A): pindah sisa n-1 ke B dari C

Tower of Hanoi (3)

- Skema rekursif:

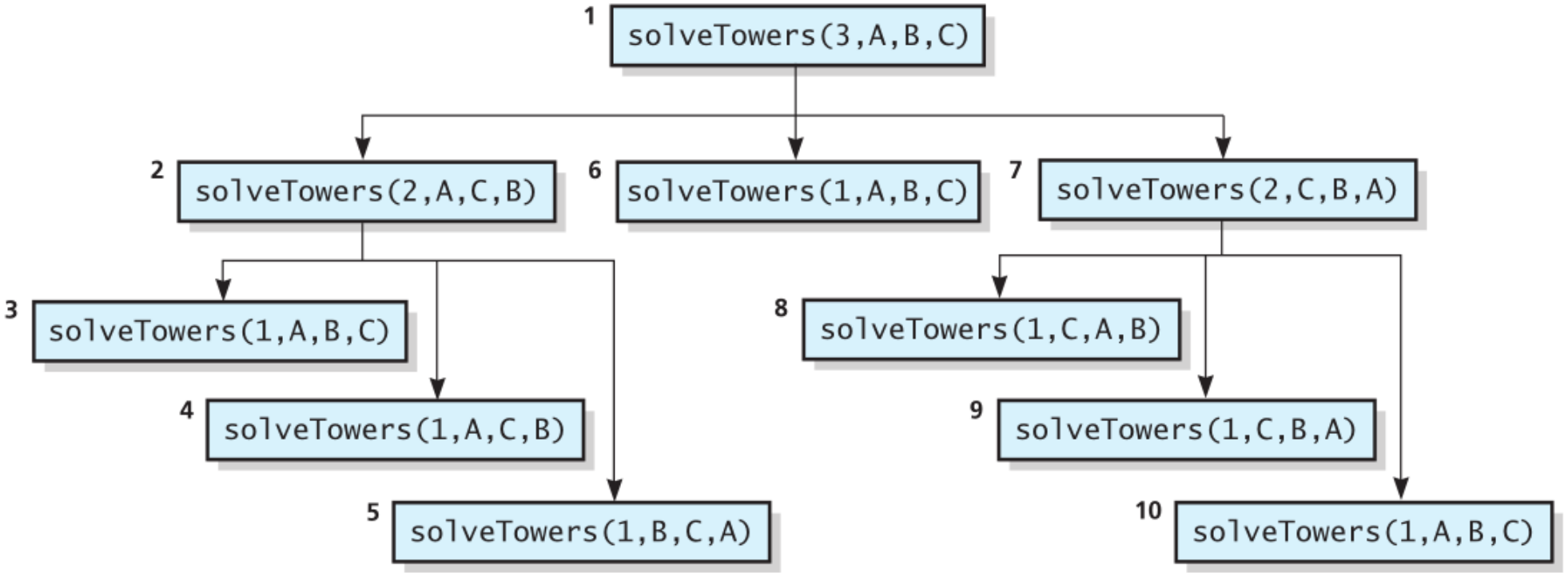
```
solveTowers(count, source, destination, spare)

   if (count is 1)
      Move a disk directly from source to destination
   else
   {
      solveTowers(count - 1, source, spare, destination)
      solveTowers(1, source, destination, spare)
      solveTowers(count - 1, spare, destination, source)
   }
```

- Menyelesaikan masalah tower hanoi = menyelesaikan masalah tower yang jumlah cakramnya lebih sedikit
- Kasus Basis = pemindahan 1 cakram
- Tiap rekursif call kita memastikan problem makin kecil

Contoh masalah tower 3 cakram

- Urutan recursive call solveTowers(3, A, B, C) ditandai angka



Formulasi problem tower

- Ukuran problem N = jumlah cakram
- Jika N=1, simple : ongkos perpindahan = 1
- Moves(1) = 1
- moves(N) = moves(N-1) + moves(1) + moves(N-1)
- Moves(3) = moves(3-1) + moves(1) + moves(3-1) = 2*moves(2) + moves(1) = 2*(2*moves(1)+1)+1=7
- Disebutkan dalam legenda, jika biksu berhasil memindahkan 64 cakram, saat itu bumi telah hancur karena diterpa badai halilintar (Kiamat!!)

Cost memindah cakram dengan recursive

B. Explicit Pattern

Number of Disks	Number of Moves
1	1
2	3
3	7
4	15
5	31

$2^n - 1$

Powers of two help reveal the pattern:

Number of Disks (n)	Number of Moves
1	2^1 - 1 = 2 - 1 = 1
2	2^2 - 1 = 4 - 1 = 3
3	2^3 - 1 = 8 - 1 = 7
4	2^4 - 1 = 16 - 1 = 15
5	2^5 - 1 = 32 - 1 = 31

So the formula for finding the number of steps it takes to transfer n disks from post A to post B is: **2^n - 1**.

From this formula you can see that even if it only takes the monks one second to make each move, it will be **2^64 - 1** seconds before the world will end. This is 590,000,000,000 years (that's 590 billion years) - far, far longer than some scientists estimate the solar system will last. That's a really long time!

Backtracking

- Teknik pencarian solusi jika rekursif menemui stuck maka kembali lagi ke rekursif sebelumnya
- F(n) => F(n-1) => F(n-2) #stuck

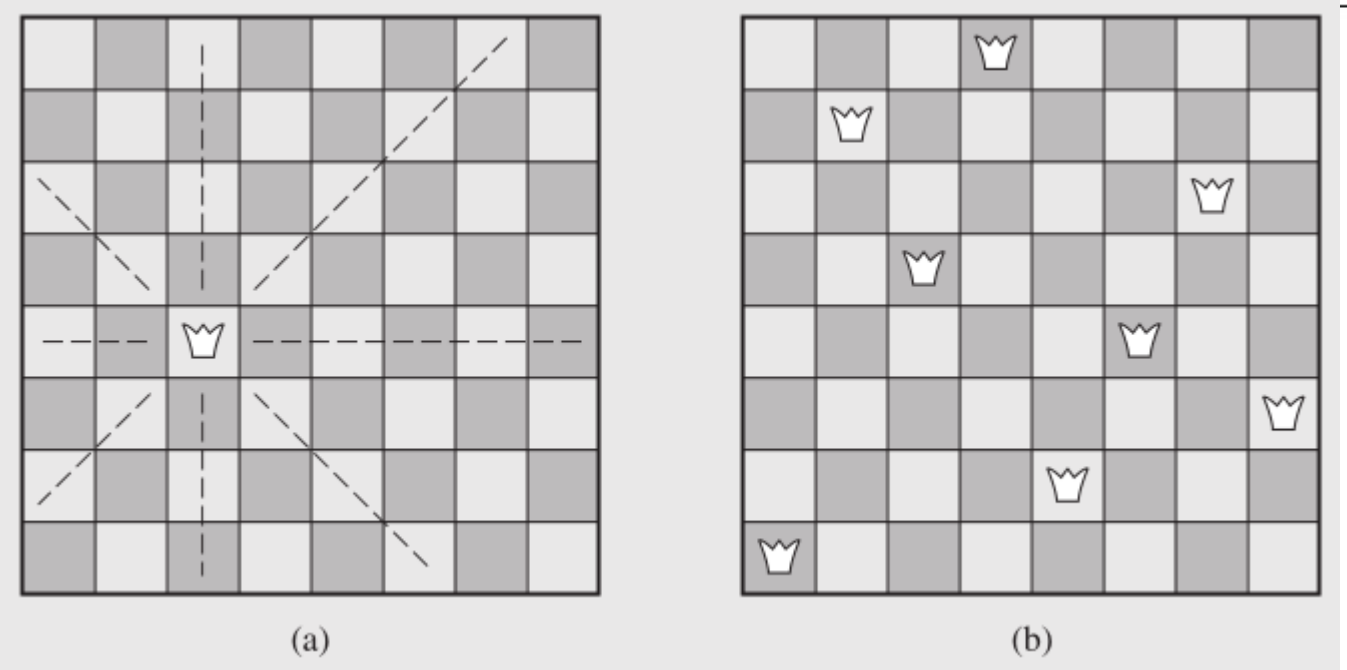
  F(n) <= F(n-1) # stuck lagi

  F(n) evaluasi pilihan opsi lain yang ada dalam kemungkinan solusi
- Rekursif lagi F(n) => F(n-1) => F(n-2)
- Hingga menemui basis (kondisi F(0))

8-Queen problem

- Definisi problem: menempatkan queen (ratu) pada papan catur sebanyak 8-queen dalam posisi semuanya **tidak bisa** menyerang satu sama lain
- Pencarian secara acak menunjukkan: kita harus mencari dan mencoba posisi kombinasi sebanyak 4.426.165.368
- Posisi yang sama dalam baris dan kolom yang sama bisa dieliminasi

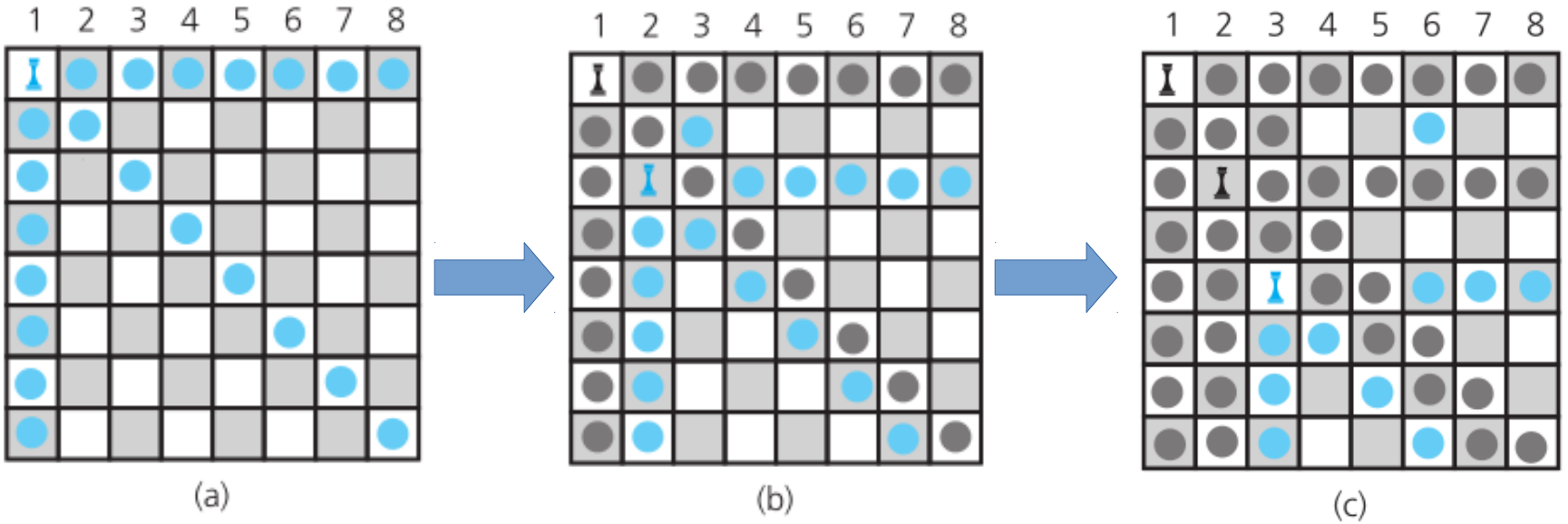
8-Queen problem(2)



- Kita bisa tempatkan queen kolom demi kolom
- Dengan kata lain, kita menyelesaikan problem ini dengan penempatan satu per satu (iteratif)

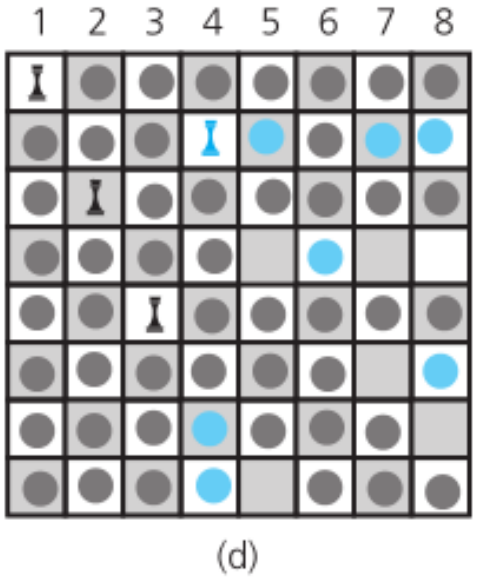
8-Queen problem(3)

- Iterasi dimulai dari menempatkan queen pada kolom pertama 1 , kemudian kolom 2,3...s/d. 8
- Tiapkali penempatan akan menghilangkan posisi cell-cell yang dapat diserang untuk penempatan berikutnya

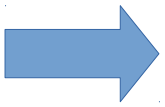


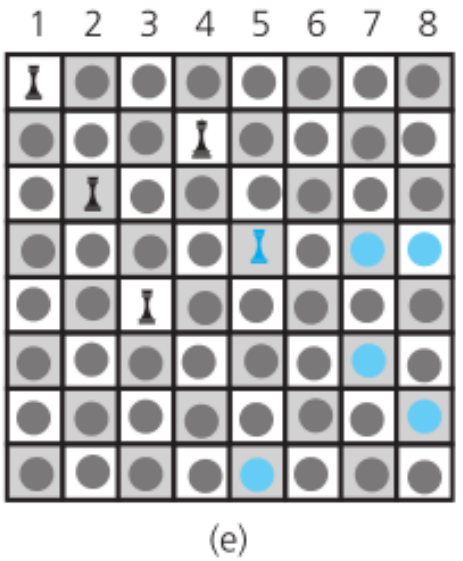
8-Queen problem(3)

- Dimana letak rekursif 8-queen problem?
- Tiap selesai menempatkan queen pada satu kolom, kita mencari posisi pada n-1 kolom berikutnya: 8 → 7 → 6… 1
- Mengapa rekursif ini memerlukan Backtrack?



(d)





(e)

Setelah mengisi Kolom 5, kolom 6 Full terisi dan tidak Mungkin dilanjutkan lagi

Pseudocode Backtrack 8-Queen

```
// Places queens in eight columns.
placeQueens(queen: Queen): void

    if (queen's column is greater than the last column)
        The problem is solved
    else
    {

        while (unconsidered squares exist in queen's column and
                the problem is unsolved)
        {

            Find the next square in queen's column that is
              not under attack by a queen in an earlier column
            if (such a square exists)
            {
                Place a queen in the square

                // Try next column
                placeQueens(new Queen(firstRow, queen's column + 1))
                if (no queen is possible in the next column)
                {

                    Delete the new queen
                    Remove the last queen placed on the board and
                       consider the next square in that column
                }
            }
        }
    }
```

Tail Recursion

- Algoritma rekursif dapat dirancang dengan berbagai desain dan “trik”
- Yang paling sederhana adalah kita memanggil fungsi rekursif di bagian akhir fungsi (akhir = tail)
- Tail recursion = tidak ada rekursif selain bagian akhir

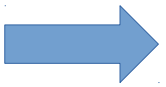
```
void tail(int i) {
  if (i > 0) {
    cout << i << '';
    tail(i-1);
  }
}
```

```
void nonTail(int i) {
  if (i > 0) {
    nonTail(i-1);
    cout << i << '';
    nonTail(i-1);
  }
}
```

Tail (2)

- Karena tail recursive merupakan jenis rekursif paling sederhana, kita dapat mengganti nya sebagai algoritma iteratif (pengulangan biasa)

```
void tail(int i) {
  if (i > 0) {
    cout << i << '';
    tail(i-1);
  }
}
```



```
void iterativeEquivalentOfTail(int i) {
  for ( ; i > 0; i--)
    cout << i << '';
}
```

Pemrograman Dinamis

- Algoritma secara rekursif terlihat sebagai solusi yang natural untuk sejumlah problem
- Membuat implementasi dalam skema rekursif kadang sangat alami untuk situasi tertentu
- Ini menjadikan solusi sangat elegan, simple dan kode program jadi ringkas
- Akan tetapi, ada masalah dengan rekursif
- Pemanggilan fungsi rekursif memakan memori stack

Dinamis(2)

- Sebagai contoh pada fungsi Fibonacci

n	Fib(n+1)	Number of Additions	Number of Calls
6	13	12	25
10	89	88	177
15	987	986	1,973
20	10,946	10,945	21,891
25	121,393	121,392	242,785
30	1,346,269	1,346,268	2,692,537

- Data diatas menunjukkan pemanggilan fungsi rekursif bersifat exponensial terhadap input n

Dinamis (3)

- Pemrograman dinamis berusaha mengurangi pemanggilan berulang-ulang fungsi dimana sebelumnya pernah di eksekusi
- Kita menyimpan hasil perhitungan fungsi tersebut tanpa menghitung ulang kembali
- Misal kita menyimpan hasil F_0 s.d F_{21} dalam bentuk tabel The first 21 Fibonacci numbers F_n are:[2]

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

- Maka menghitung F_{22} kita tinggal memanggil data F_{21} dan F_{20} dari tabel tanpa menghitungnya lewat rekursif

Daftar Pustaka

- Top
- Elliot Koffman
- Walter Savitch
- Michael Goodrich (Java/C++)
- Adam Drozdek
- Narasimha Karumanchi
- Anany Levitin, Design Analysis Algorithm

Terima kasih