Week 1 - Introduction to Algorithm

- **Algorithm :** A sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in finite amount of time.

- Algorithm described by natural language or pseudocode.
- Algorithm benar kalau untuk setiap input berhenti dengan output benar.
- Range input untuk setiap algoritma harus dispesifikasi carefully
- Algoritma di masalah sama can be based on very different ideas and can solve with dramatically different steps

1. Euclid's Algorithm

- Greatest common divisor / GCD / FPB dari 2 nonnegative integers m and n
- Rumus : $\boxed{gcd(m,n) = gcd(n, m \bmod n)}$ until $m \bmod n = 0$

- Cara :
  1) If $n=0$, return value m as the answer and stop; otherwise proceed to 2)
  2) Divide m by n and assign the value of remainder r
  3) Assign the value of n to m and the value of r to n. Go to 1)

- while $n \neq 0$ do
  - $r \leftarrow m \bmod n$
  - $m \leftarrow n$
  - $n \leftarrow r$

  return m

- e.g. : $\gcd(60,24) = \gcd(24, 60 \bmod 24)$
  $= \gcd(24,12)$
  $= \gcd(12, 24 \bmod 12)$
  $= \gcd(12,0)$
  $= 12$

2. Consecutive Integer Checking

- Cara : 1) Assign the value of $\min(m,n)$ to t
  2) Divide m by t. If the remainder of this division is 0, go to 3). Otherwise, go to 4)
  3) Divide n by t. If the remainder of this division is 0, return the value of t as the answer and stop. Otherwise, proceed to 4)
  4) Decrease the value of t by 1. Go to 2)

- e.g. : $\gcd(60,24) \Rightarrow t = \min(60,24) = 24$
  $r = 60 \bmod 24 = 12$; as $r \neq 0$, $t = t-1 = 24-1 = 23$
  $r = 60 \bmod 23 = 14$; as $r \neq 0$, $t = t-1 = 23-1 = 22$
  $r = 60 \bmod 22 = 16$; as $r \neq 0$, $t = t-1 = 22-1 = 21$
  $r = 60 \bmod 21 = 18$; as $r \neq 0$, $t = t-1 = 21-1 = 20$
  $r = 60 \bmod 20 = 0$; as $r = 0$, $r = n \bmod t$
  $r = 24 \bmod 20 = 4$; as $r \neq 0$, $t = t-1 = 20-1 = 19$
  $r = 60 \bmod 19 = 3$; as $r \neq 0$, $t = t-1 = 19-1 = 18$
  $r = 60 \bmod 18 = 6$; as $r \neq 0$, $t = t-1 = 18-1 = 17$
  $r = 60 \bmod 17 = 9$; as $r \neq 0$, $t = t-1 = 17-1 = 16$
  $r = 60 \bmod 16 = 12$; as $r \neq 0$, $t = t-1 = 16-1 = 15$
  $r = 60 \bmod 15 = 0$; as $r = 0$, $r = n \bmod t$
  $r = 24 \bmod 15 = 9$; as $r \neq 0$, $t = t-1 = 15-1 = 14$
  $r = 60 \bmod 14 = 0$; as $r \neq 0$, $t = t-1 = 14-1 = 13$
  $r = 60 \bmod 13 = 0$; as $r \neq 0$, $t = t-1 = 13-1 = 12$
  $r = 60 \bmod 12 = 0$; as $r = 0$, $r = n \bmod t$
  $r = 24 \bmod 12 = 0$, $t = 12$

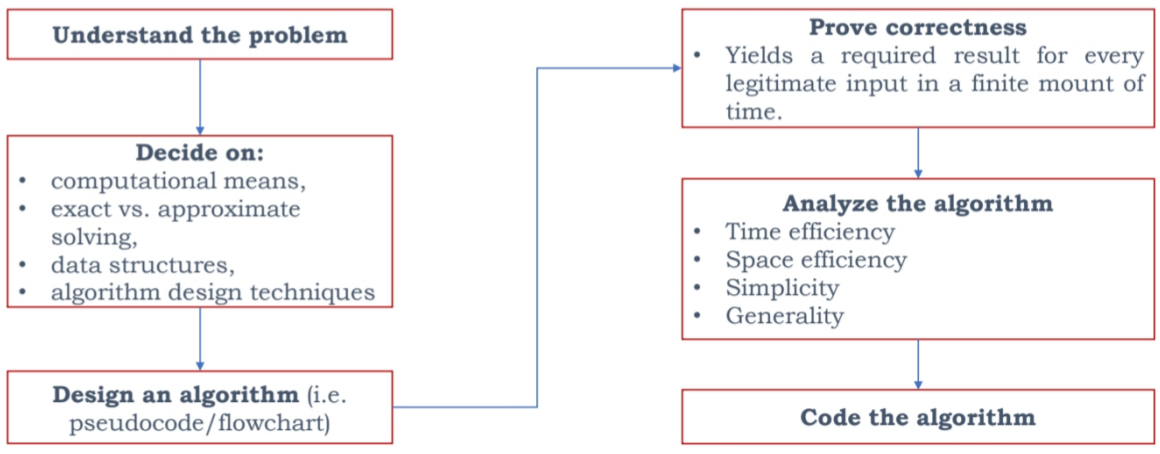
  so $\gcd(60,24) = t = 12$

3. Middle-School Procedure

- Cara :
  1) Find the prime factors of m
  2) Find the prime factors of n
  3) Identify all the common factors in the 2 prime expansions found in 1) & 2). If p is a common factor occuring p_m & p_n times in m and n, respectively, it should be repeated $\min(p_m, p_n)$ times.
  4) Compute the product of all the common factors and return it as the gcd of the numbers given.

- Ego :
  gcd (60, 24)
  $60 = 2 \cdot 2 \cdot 3 \cdot 5$
  $24 = 2 \cdot 2 \cdot 2 \cdot 3$
  } Jangan jadiin pangkat
  ∴ $\gcd(60, 24) = 2 \cdot 2 \cdot 3 = 12$

Algorithm Design & Analysis Process :



Important problem types :

1. Sorting
   ↳ Rearrange the items
   ↳ Selection sort, bubble sort, etc

2. Searching
   ↳ Finding a given value (Search key)
   ↳ Sequential search, binary search, etc

3. String Processing
   ↳ Paling famous

4. Graphs Problem
   ↳ Traveling Salesman Problem
   ↳ Graph-coloring Problem

5. Geometric Problems
   ↳ Deal with geometric objects (Points, lines, & polygons)
   ↳ Closest-pair Problem
   ↳ Convex-hull Problem

6. Numerical Problems
   ↳ Involve mathematical objects of continuous nature : Solving equations, integrals, etc

c) Fundamental Data Structures :

1. Linear Data Structures
   ↳ Array
   ↳ Linked List

2. Graphs
   ↳ Undirected graph
   ↳ Directed graph / digraph
   ↳ Weighted graph / digraph

3. Trees
   ↳ Free Trees
   ↳ Rooted Trees
   ↳ Ordered Trees

4. Sets & Dictionaries
   ↳ Set
   ↳ Elements
   ↳ Dictionary

Week 2 – Analysis of Algorithm Efficiency

- Analysis of Algorithms : An investigation of an algorithm's efficiency (Running time & memory space)
- Algorithm is core of tech & perfomance is currency of computing
- Computing time is bounded resource, so as space in memory
- Time efficiency : How fast an algorithm in question runs
- Space efficiency : Space the algorithm requires
- Analysis framework :

1. Measuring an Input's Size

   - Algorithm efficiency = A function of some parameter n (Input size)
     e.g. : - Sorting & searching algorithm : Size of list
     - A polynomial $p(x) = a_n x^n + \cdots + a_0$: Polynomial's degree

2. Measuring Running Time

   - To count the number of times each the algorithm's operations is executed

   - Identify the most important operation
     ↳ The most time-consuming operation
     ↳ e.g. : - Sorting algorithm → Comparison
     - Matrix multiplication → Multiplication & addition

Estimated RT ← $T(n) = c_{op}\ C(n)$ → Number of times this operation needs to be executed

c_{op} ↓

Execution time of an algo's basic operation

3. Orders of Growth

- Simplifying abstractions
- For large values of n, it is the function's order growth that counts
- e.g.:
  - $an^2 \longrightarrow n^2$
  - $n^3 + n + 5 \rightarrow n^3$

Efficiency cases:

1. Worst Case (Usually)
   - ↳ $C_{worst}(n)$
   - ↳ T(n) = Maximum time of algorithm on any input of size n

2. Average Case (Sometimes)
   - ↳ $C_{avg}(n)$
   - ↳ T(n) = Expected time of algorithm over all inputs size of n

3. Best Case (Bogus)
   - ↳ $C_{best}(n)$
   - ↳ Cheat with slow algorithm that works fast on some input

Sequential / Linear Search

- SequentialSearch(A[0...n-1], key)

```
i ← 0
while i < n and A[i] != key do
      i ← i + 1
if i < n return i
else return -1
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]
20	44	11	3	65	52	38

Cavg

Basic operation?

Key = 20, needs 1 comparison. Cbest

Key = 65, needs 5 comparisons. Cavg

Key = 38, needs 7 comparisons. Cworst

- $T(n) = \frac{1+2+3+4+\dots+n}{n}$

$= \frac{n}{2}(n+1) \Big/ n$

$= \frac{n+1}{2} \longrightarrow O(n) = n$

b) Binary Search

- BinarySearch $(A[0 \dots n-1], key)$

low ← 0, high ← n-1, mid ← 0

while low <= high do

mid ← ⌊(low+ high)/2⌋

if A[mid] == key return mid

else if A[mid] < key then low ← mid+1

else if A[mid] > key then high ← mid-1

return -1

- *Key* = 11 → sorting first.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	11	20	38	44	52	65

low H mid high

- *n* items

	#comparisons
1st comparison	n/2
2nd comparison	n/4
3rd comparison	n/8
…	$n/2^i$

$C_{worst}(n)$?

$\frac{n}{2^i} = 1$

$i = \log n$

$C_{best}(n)$? 1

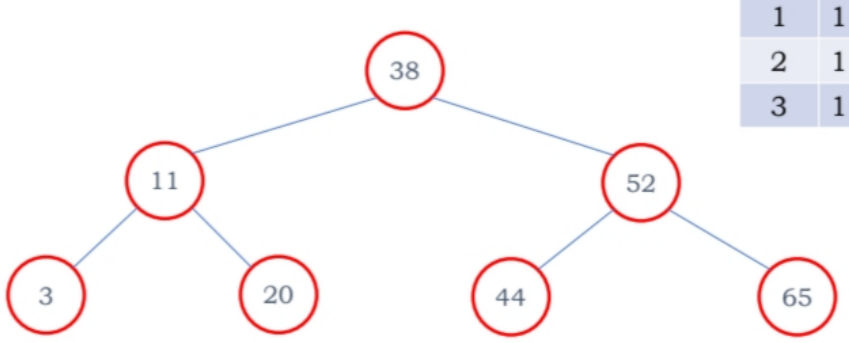
- min comparison = Best case

max comparison = Worst case

- 1st comparison = $\frac{0+6}{2} = 3$

2nd comparison = $\frac{0+2}{2} = 1$

- *n* items



l	n	d	c
1	1	0	1
2	1 + 2 = 3	1	2
3	1 + 2 + 4 = 7	2	3

- $C_{avg}(n) \Rightarrow n = 2^{(d+1)} - 1$

$$n+1 = 2^{(d+1)}$$
$$^2\log(n+1) = ^2\log 2^{(d+1)}$$
$$^2\log(n+1) = d+1$$
$$c = ^2\log(n+1)$$

$\therefore T(n) = ^2\log(n+1)$

Bubble Sort

- BubbleSort $(A[0 \ldots n-1], n)$
  for i = 0 downto n-1 do
    for j = n-1 downto i+1 do
      if A[j] < A[j-1] then Swap (A[j], A[j-1])

- Ada 7 comparison $\Rightarrow 6+5+4+3+2+1 = 21$

- $(n-1)+(n-2)+\ldots+3+2+1 \rightarrow T(n) = \frac{n(n-1)}{2}$

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

Insertion Sort

- InsertionSort $(A[1 \ldots n], n)$
  for j ← 2 to n do
    key ← A[j]
    i ← j-1
    while i > 0 and A[i] > key do
      A[i+1] ← A[i]
      i ← i-1
    A[i+1] = key

- $(n-1)+(n-2)+\dots+3+2+1 \rightarrow T(n)=\frac{n(n-1)}{2}$

$O(n)=n^2=C_{avg}(n)$

- $C_{best}=n$

a)

Class	Name	Sample algorithm type
O(1)	Constant	Algorithm ignores input (i.e., can't even scan input)
O(lgn)	Logarithmic	Cuts problem size by constant fraction on each iteration
O(n)	Linear	Algorithm scans its input (at least)
O(nlgn)	"n-log-n"	Some divide and conquer
$O(n^2)$	Quadratic	Loop inside loop = "nested loop"
$O(n^3)$	Cubic	Loop inside nested loop
$O(2^n)$	Exponential	Algorithm generates all subsets of n-element set of binary values
O(n!)	Factorial	Algorithm generates all permutations of n-element set

fast ↓ slow

a) Growth rates vs. n size:

n	O(1)	O(log n)	O(n)	O(n log n)	$O(n^2)$	$O(2^n)$	O(n!)
1	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec
10	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
100	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	40170 trillion years	> vigintillion years
1,000	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	> vigintillion years	> centillion years
10,000	< 1 sec	< 1 sec	< 1 sec	< 1 sec	2 min	> centillion years	> centillion years
100,000	< 1 sec	< 1 sec	< 1 sec	1 sec	3 hours	> centillion years	> centillion years
1,000,000	< 1 sec	< 1 sec	1 sec	20 sec	12 days	> centillion years	> centillion years


```
int hitungAvg (int n){
int i, k=0, av;
for(i=1; i<=n; i++){
    k=k+1;
}
av=k/n;
return av
}
```

a) assignment = $2n+2$

a) op. aritmetika = $2n+1$

a) $T(n)=(2n+2)a+(2n+1)b$ detik

(a, b: tergantung komputer (konstanta kecepatan))

$O(n)=n$

- maks = bilangan[1] (1)
  counter = 2 (1)
  while counter <= max_array do
    if bilangan[counter] > maks then
      maks = bilangan[counter] (n)
    endif
    k = k+1 (n)
  endwhile (n)

- assignment = 2n + 2

- op. aritmetika = n
  (+, -, *, /)

- Algoritma itu efisien dan optimal jika kebutuhan waktu untuk dan ruang ketika menjalankannya kecil.

- Cara perhitungan di atas kurang optimal karena waktu eksekusi tidak diketahui dengan pasti dan arsitektur komputer berbeda-beda, sehingga pakailah besaran T(n) V M(n)

T(n) = n - 1

- input (data_cari)
  i ← 1
  while (nama[i] ≠ data_cari) and (i < max_array) do
    i ← i+1
  endwhile

$T_{avg} = \frac{1+2+3+\dots+n}{n}$

$T_{maks} = n$
$T_{min} = 1$
(krn lgsg ktmu)

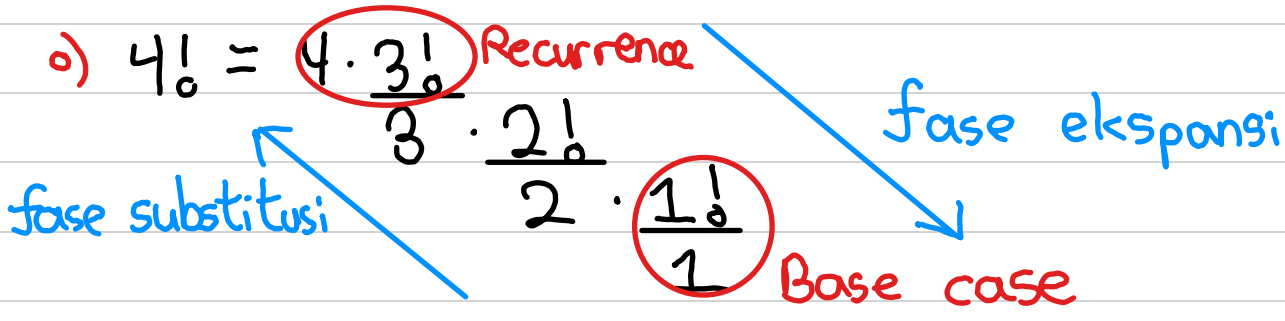
Week 3 - Analysis of Recursive Algorithm

Recursive Algorithm

- Bentuk
  - Suatu subrutin / fungsi / prosedur yang memanggil diri sendiri
  - Pemanggilan subrutin terdapat dalam body subrutin
  - Program lebih mudah dilihat
- Tujuan
  - Menyederhanakan penulisan program
  - Menggantikan bentuk iterasi
- Syarat
  - Ada kondisi terminal (Base case)
  - Subroutine call yang melibatkan parameter yang nilainya menuju kondisi terminal (Recurrence)
- Solved with cara biasa or rekursif

Permasalahan yang dapat diselesaikan oleh fungsi rekursif memiliki sifat:

- Memiliki kasus sederhana yang dapat langsung diselesaikan (Base Case), seperti $0! = 1$
- Kasus kompleks dapat diuraikan menjadi kasus yang identik dengan ukuran yang lebih kecil (Recursive cases), seperti $n! = n * (n-1)!$ atau $n! \rightarrow (n-1)! \rightarrow (n-2)! \rightarrow \dots \rightarrow 0!$



- Fungsi rekursif :
  1. Recurrence → Parameternya berkurang menuju nilai n di base case
  2. Base case → Menghasilkan nilai int

- factorial(4) = 4 * factorial (3)
  = 4 * (3 * factorial (2))
  = 4 * (3 * (2 * factorial (1)))
  = 4 * (3 * (2 * (1 * factorial (0))))

  Expansion phase
  ↳ Mendekati base case

  = 4 * (3 * (2 * (1 * 1)))
  = 4 * (3 * (2 * 1))
  = 4 * (3 * 2)
  = 4 * 6
  = 24

  Substitution phase
  ↳ Mulai dari base case

- faktorial (n) {
  if n=1 return 1 → Base case
  else return n * faktorial(n-1) → Recurrence
  }

- Dalam fungsi mate :

$$faktorial(n) \begin{cases} faktorial(n-1) * n & ; n>1 \\ 1 & ; n=1 \end{cases}$$

- Dalam time complexity :

$$T(n)=M(n)=\begin{cases} M(n-1)+1 & ; n>1 \\ 0 & ; n=1 \end{cases}$$

Karena tidak ada n

- $M(n) = M(n-1) + 1$
  $= (M(n-2)+1)+1$
  $= M(n-2)+2$
  $= (M(n-3)+1)+2$
  $= M(n-3)+3$
  ⋮
  $= M(n-i)+i$
  $= M(1)+n-1$
  $= 0+n-1$
  $= n-1$

- $n-i=1$
  $i=n-1$ → Agar bisa buat dapetin $M(1)=0$

- $T(n)=M(n)=n-1$
  $O(n)=n$

- if n=1 return 1
  else S(n-1) + n + n + n

  - $T(n) = M(n) = M(n-1) + 2$
    $= (M(n-2)+2) + 2$
    $= M(n-2) + 4$
    $= M(n-3) + 6$
    $= M(n-i) + 2i$
  - $M(1) = 0$
  - $n - 1 = i$
    $i = n - 1$
  - $M(n) = M(n-i) + 2i$
    $= M(1) + 2n - 2$
  - $O(n) = n$

- $S = 1 + 2 + 3 + 4 + \dots + n$

  - $$S(n) = \begin{cases} S(n-1) + n, & n > 0 \\ 0, & n = 0 \end{cases}$$
  - $S(3) = 3 + S(2)$
    $2 + S(1)$
    $1 + S(0)$
    0

  - $$T(n) = A(n) = \begin{cases} A(n-1) + 1, & n > 0 \\ 0, & n = 0 \end{cases}$$
  - $n - i = 0$
    $n = i$

  - $T(n) = A(n) = A(n-1) + 1$
    $= A(n-2) + 2$
    $= A(n-3) + 3$
    $= A(n-i) + i$
  - $T(n) = A(n) = A(0) + n$
    $= n$
  - $O(n) = n$

- $S = 2 + 4 + 6 + 8 + \dots + 2n$

  - $$S(n) = \begin{cases} S(n-1) + 2n, & n > 0 \\ 0, & n = 0 \end{cases}$$
  - $S(3) = 6 + S(2)$
    $4 + S(1)$
    $2 + S(0)$
    0

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

- Algo : funct S(n){
  if n=0 return 1
  else return S(n-1)+n+n
  }

- $T(n) = A(n) = A(n-1) + 2$
  $= (A(n-2)+2)+2$
  $= A(n-2)+4$
  $= A(n-3)+6$
  $= A(n-i)+2i$

- $n-i=0$
  $n=i$
- $T(n) = A(n) = A(0) + 2n$
  $= 2n$
- $O(n) = n$

> Hitung 2^n dengan cara rekursif menggunakan rumus $2^n = 2^{n-1} + 2^{n-1}$
a. Buat Algonya
b. Hitung time complexity-nya

- a. function algo{
  if (n=0) retum 1
  else retum Power(n-1) + Power(n-1)
  }

b. $T(n) = A(n) = \begin{cases} A(n-1)+A(n-1)+1 ; & n \neq 0 \\ 0 ; & n=0 \end{cases}$

$T(n) = A(n) = 2A(n-1) + 1$
$= 2(2A(n-1)+1)+1$
$= 4A(n-2)+3$
$= 4(2A(n-3)+1)+3$
$= 8(A(n-3)+7$
$= 2^i A(n-i) + 2^i - 1$

$n-i=0$
$n=i$

$T(n) = A(n)$
$= 2^i A(n-i) + 2^i - 1$
$= 2^i A(0) + 2^n - 1$
$= 2^n - 1$
$O(n) = 2^n$

9) fibonacci {
```
    if n <= 1
        return n
    else
        return fibonacci (n-1) + fibonacci(n-2)
}
```

- $T(0) = T(1) = 1$
- $T(n) = T(n-1) + T(n-2) + 1$

- $T(n) \approx T(n-1) \approx T(n-2)$

- $T(n) = 2T(n-2)+1$
  $= 2(2T(n-4)+1)+1$
  $= 4T(n-4)+3$
  $= 4(2T(n-6)+1)+3$
  $= 8T(n-6)+7$
  $= 2^i T(n-2i) + 2^i - 1$

- $n - 2i = 0$
  $n = 2i$
  $i = \frac{n}{2}$

- $T(n) = 2^{\frac{n}{2}} T(0) + 2^{\frac{n}{2}} - 1$
  $= 2^{\frac{n}{2}} + 2^{\frac{n}{2}} - 1$
  $= 2^n - 1$

- $T(n) = 2T(n-1)+1$
  $= 2(2T(n-2)+1)+1$
  $= 4T(n-2)+3$
  $= 4(2T(n-3)+1)+3$
  $= 8T(n-3)+7$
  $= 2^i T(n-i) + 2^i - 1$

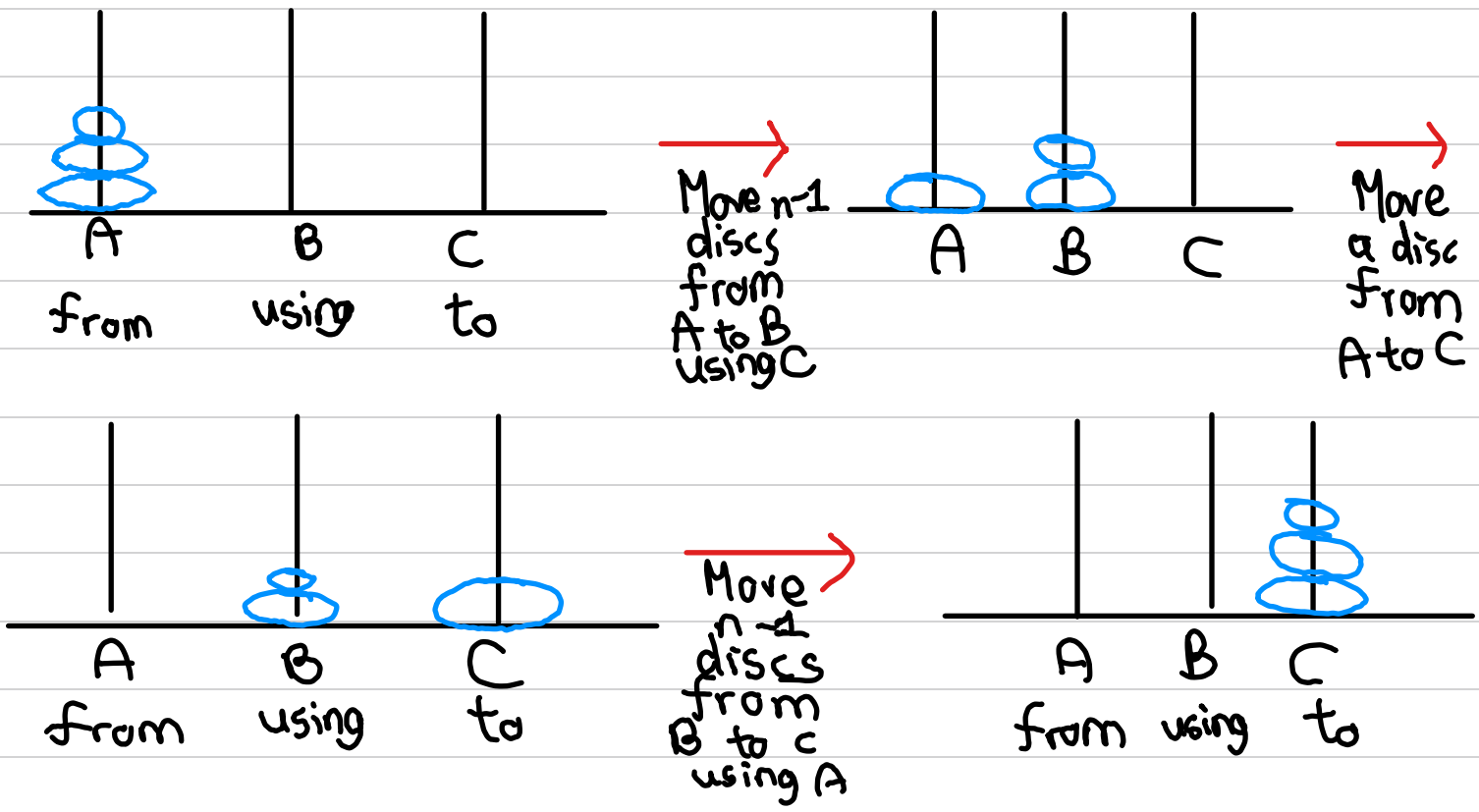
- $n - i = 0$
  $n = i$

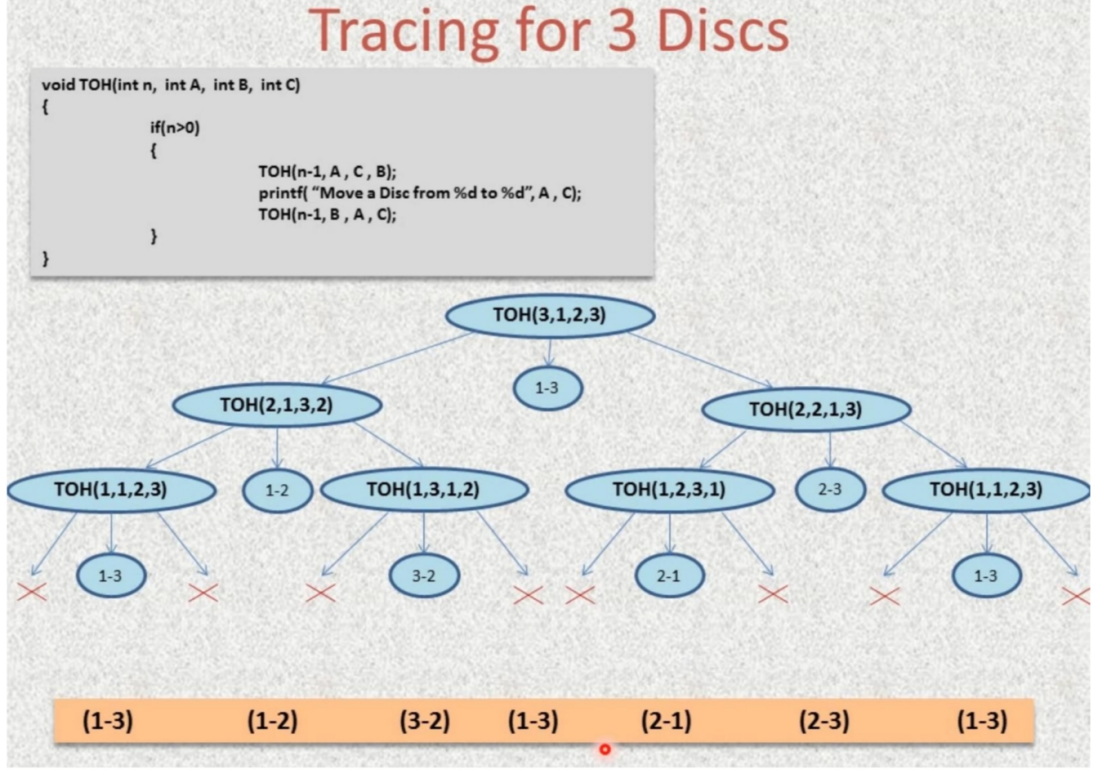
- $T(n) = 2^n T(0) + 2^n - 1$
  $= 2^n + 2^n - 1$
  $= 2^{n+1} - 1$

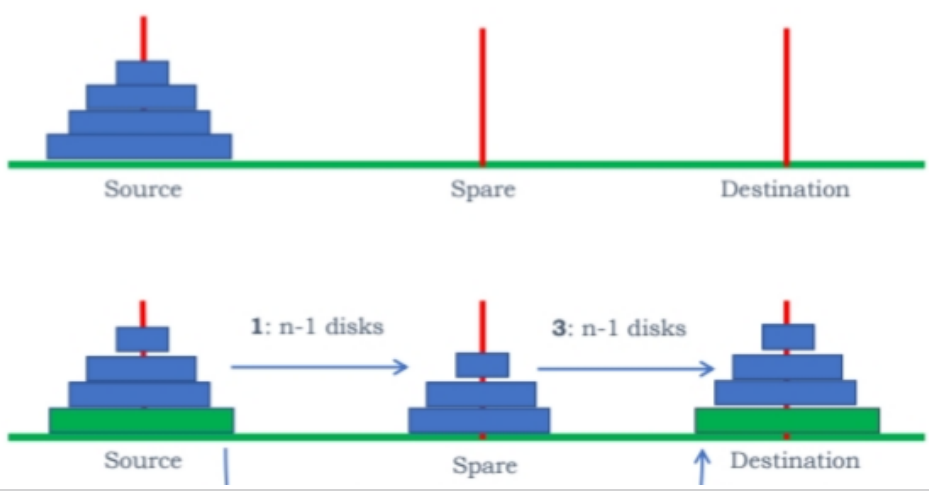
- $T(n) \propto 2^n - 1 \longrightarrow$ Upper bound
- $T(n) \propto 2^{n+1} - 1 \longrightarrow$ Lower bound

- $O(n) = 2^n$

Tower of Hanoi







↗ Move (Basic operation)

- $M(n) = 2M(n-1) + 1$
  $= 2(2M(n-2)+1)+1$
  $= 4M(n-2)+3$
  $= 4(2M(n-3)+1)+3$
  $= 8M(n-3)+7$
  $= 2^i M(n-i) + 2^i - 1$
- $M(1) = 1$

- $M(n) = 2^i M(n-i) + 2^i - 1$
  $= 2^{n-1} M(1) + 2^{n-1} - 1$
  $= 2^{n-1} + 2^{n-1} - 1$
  $= 2^n - 1$
- $O(n) = 2^n$

- $n - i = 1$
  $i = n - 1$

a) Algoritma Min1 (A[0 ... n-1]){
  if (n=1) return A[0]
  else temp ← Min1 (A[0 ... n-2])
    if temp <= A[n-1] return temp
    else return A[n-1]

a. Algo ini mengerjakan apa? → Mencari bilangan terkecil

b. Buat recurrence relationshipnya

→
$$T(n) = C(n) = \begin{cases} C(n-1) + 1 & ; n > 1 \\ 0 & ; n = 1 \end{cases}$$

↓ Comparison (Basic operation)

c. T(n)

→ $T(n) = C(n) = C(n-1) + 1$
$= C(n-2) + 2$
$= C(n-3) + 3$
$= C(n-i) + i$
$= n - 1$

$n - i = 1$
$i = n - 1$

$O(n) = n$

a) Fokus saja di basic operationnya untuk tambah angka

a) if
else if → Kalau ada sampai di sini saja
else

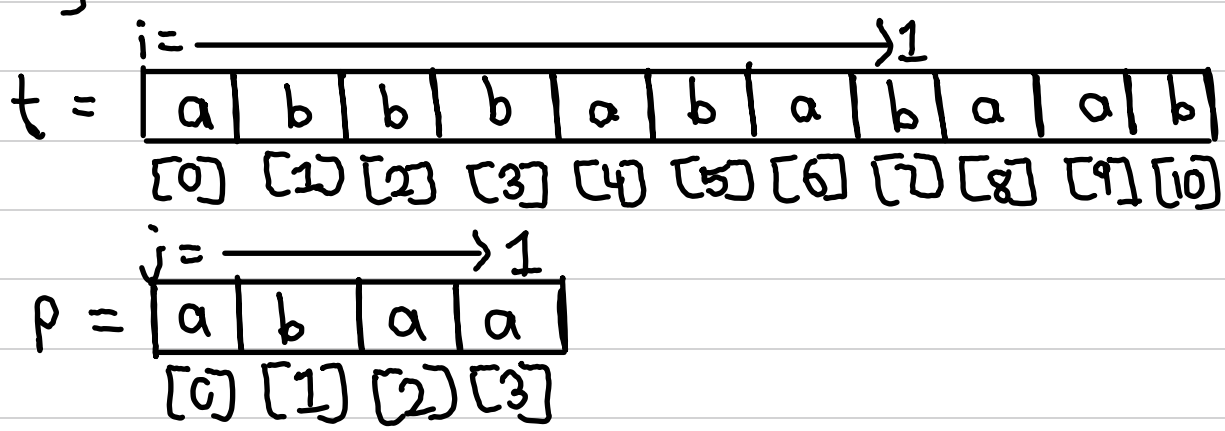
Week 4 - Brute Force 1

- Brute Force = A straightforward approach to solving a problem (Sort → Selection, Bubble & Search → Sequen, BFSM)
  = naive / simple

- Use brute force because applicable to very wide variety of problems, no limitation on instance size, acceptable speed, serve an important theoretical / educational purpose.

- algoritma Bruteforce (t[0...n-1], P[0...m]){

```
m = strlen(P)
n = strlen(t)
for(i=0; i<=n-m; i++){
    for (j=0; j<m; j++){
        if (t[i+j] ≠ P[j])
            break;
    }
    if (m==j){
        return i
    }
}
return -1
}
```



i = 0
- j = 0 t[0+0] = P[0]
- j = 1 t[0+1] = P[1]
- j = 2 t[0+2] ≠ P[2] break
- j = 3

i = 1
- j = 0 t[1+0] ≠ P[0] break

⋮

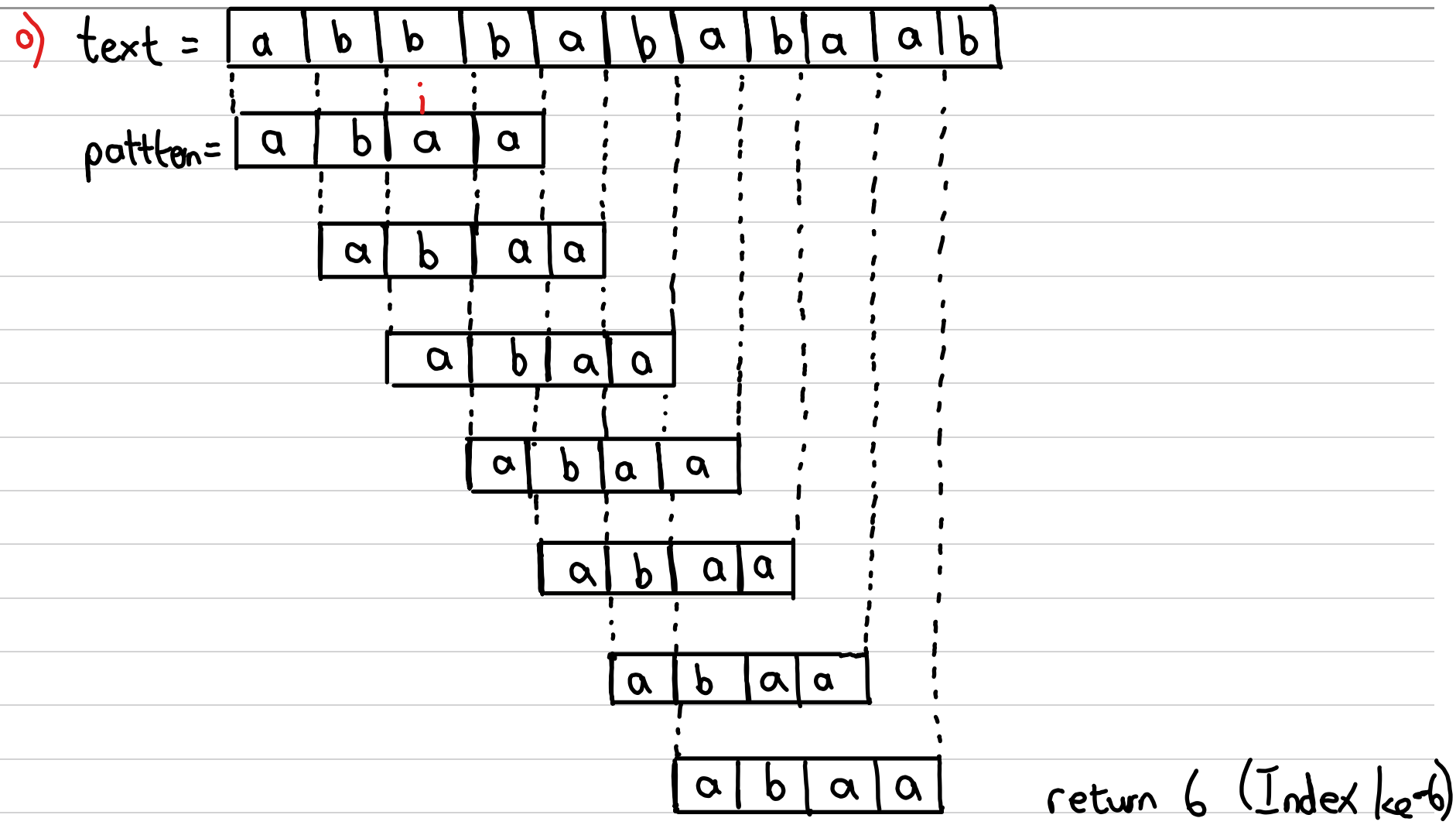
i = 6
- j = 0 t[6+0] ≠ P[0]
- j = 1 t[6+1] ≠ P[1]
- j = 2 t[6+2] ≠ P[2]
- j = 3 t[6+3] ≠ P[3]

return 6

- BFSM = Brute Force String Matching seperti di atas
  - $C_{worst}(n)$ = When a shift is not made until the m^{th} comparison
    = The complexity is $O(nm)$, where n = length text and m = length pattern
  - $C_{avg}(n)$ = Typically shift is made early
    = The complexity is $O(n+m) \rightarrow O(n)$ for $m < n$

- Other SM algo : Knuth-Morris-Pratt & Boyer-Mocre

- Pattern Matching = BFSM
  - text $[0 \ldots n-1]$
  - pattern $[0 \ldots m-1]$



return 6 (Index ke-6)

◦ LPS Table / Prefix Table / Failure Function

- Mengimplementasikan cara KMP algorithm untuk string matching
- Mencocokan sebuah string dari kiri ke kanan. Sebuah teks dicocokkan dengan pattern tertentu untuk menentukan apakah dalam teks yang dicocokkan terdapat pattern tersebut.
- Perbedaan KMP dan BFSM, tdk smw string dicocokan seperti BFSM. Ketika ada text dan pattern yang terjadi mismatch, maka pattern akan mencocokan text meloncat menurut nilai pada tabel array.
- String "welcome"
  - ↳ Prefix = {$, w, we, wel, welc, welco, welcom, welcome}
  - ↳ Suffix = {$, e, me, ome, come, lcome, elcome, welcome}
  - ↳ Proper Prefix = {$, w, we, wel, welc, welco, welcom}
  - ↳ Proper Suffix = {$, e, me, ome, come, lcome, elcome}

- Pattern =

a	b	c	d	a	b	e	a	b	f
0	1	2	3	4	5	6	7	8	9

Failure =

0	0	0	0	1	2	0	1	2	0

i=0, w='a'
↳ pp = {$}
↳ ps = {$}

i=1, w='ab'
↳ pp = {$, a}
↳ ps = {$, b}

i=2, w='abc'
↳ pp = {$, a, ab}
↳ ps = {$, c, bc}

i=3, w='abcd'
↳ pp = {$, a, ab, abc}
↳ ps = {$, d, cd, bcd}

i=4, w='abcda'
↳ pp = {$, a, ab, abc, abcd}
↳ ps = {$, a, da, cda, bcda}

i=5, w='abcdab'
↳ pp = {$, a, ab, abc, abcd, abcda}
↳ ps = {$, b, ab, dab, cdab, bcdab}

i=6, w='abcdabe'
↳ pp = {$, a, ab, abc, abcd, abcda, abcdab}
↳ ps = {$, e, be, abe, dabe, cdabe, bcdabe}

i=7, w='abcdabea'
↳ pp = {$, a, ab, abc, abcd, abcda, abcdab, abcdabe}
↳ ps = {$, a, ea, bea, abea, dabea, cdabea, bcdabea}

i=8, w='abcdabeab'
↳ pp = {$, a, ab, abc, abcd, abcda, abcdab, abcdabe, abcdabea}
↳ ps = {$, b, ab, eab, beab, abeab, dabeab, cdabeab, bcdabeab}

i=9, w='abcdabeabf'
↳ pp = {$, a, ab, abc, abcd, abcda, abcdab, abcdabe, abcdabea, abcdabeab}
↳ ps = {$, f, bf, abf, eabf, beabf, abeabf, dabeabf, cdabeabf, bcdabeabf}

- Algoritma failure function (P){

```
F[0] ← 0
i ← 1
j ← 0
while (i < m) {
    if (P[i] == P[j]) {
        F[i] ← j+1
        i ← i+1
        j ← j+1
    }
    else if (j > 0) {
        j ← F[j-1]
    }
    else {
        F[i] ← 0
        i ← i+1
    }
}
```

}

b) Knuth-Morris-Pratt (KMP)

- Ada F/LPS Table baru bisa jalankan
- Algorithm KMP (t[0 ... n-1], p[0 ... m-1])

```
F ← Failure (P)
i ← 0, j ← 0
while (i < n)
    if (t[i] == P[j])
        if (j == m-1) then
            return i-j
        else i ← i+1
             j ← j+1
    else if j > 0 then
             j ← F[j-1]
         else i ← i+1
```

- t =

b	a	c	b	a	b	a	b	a	b	a	c	a	c	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

P =

a	b	a	b	a	c	a
0	1	2	3	4	5	6

F =

0	0	1	2	3	0	1

return 6

Week 5 - Brute Force 2

Boyer-Moore String Matching

- Mencocokan sebuah string dari kanan ke kiri. Sebuah teks dicocokkan dengan pattern tertentu untuk menentukan apakah dalam teks yang dicocokkan terdapat pattern tersebut.

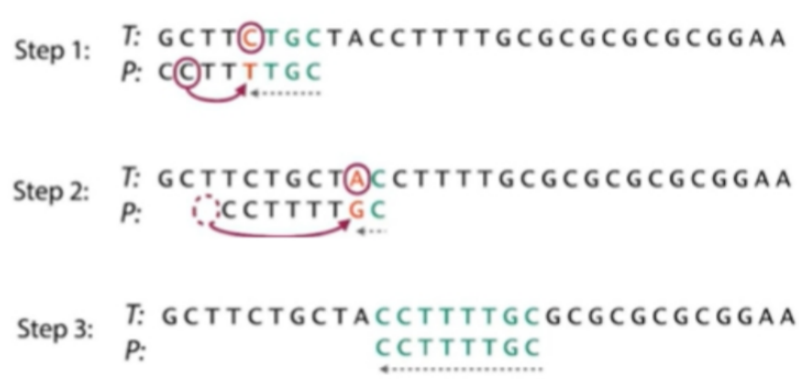
- Perbedaan String Boyer-Moore dengan BFSM tdk smw string dicocokan seperti pada cara BFSM. Ketika ada text dan pattern yang terjadi mismatch, maka pattern akan mencocokan text meloncat menurut nilai pada tabel delta atau sebanyak jumlah karakter yang telah dicocokan. Tergantung nilai maksimum yang terdapat dari keduanya.

- 2 rules in Boyer-Moore :

1) Bad Character Rule

- Upon mismatch, skip alignments until (a) mismatch becomes a match, or (b) P moves past mismatched character.

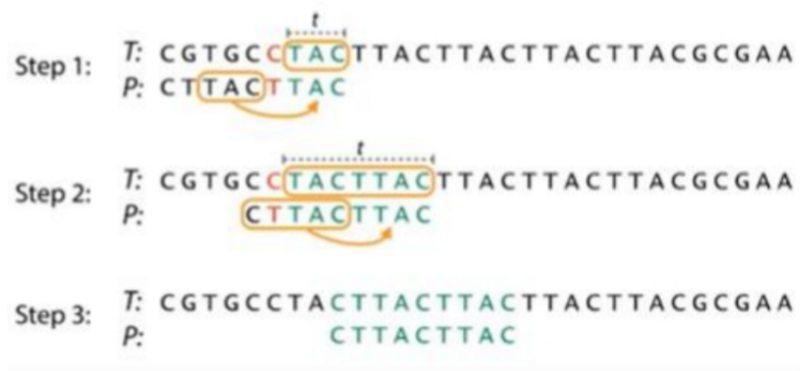
-



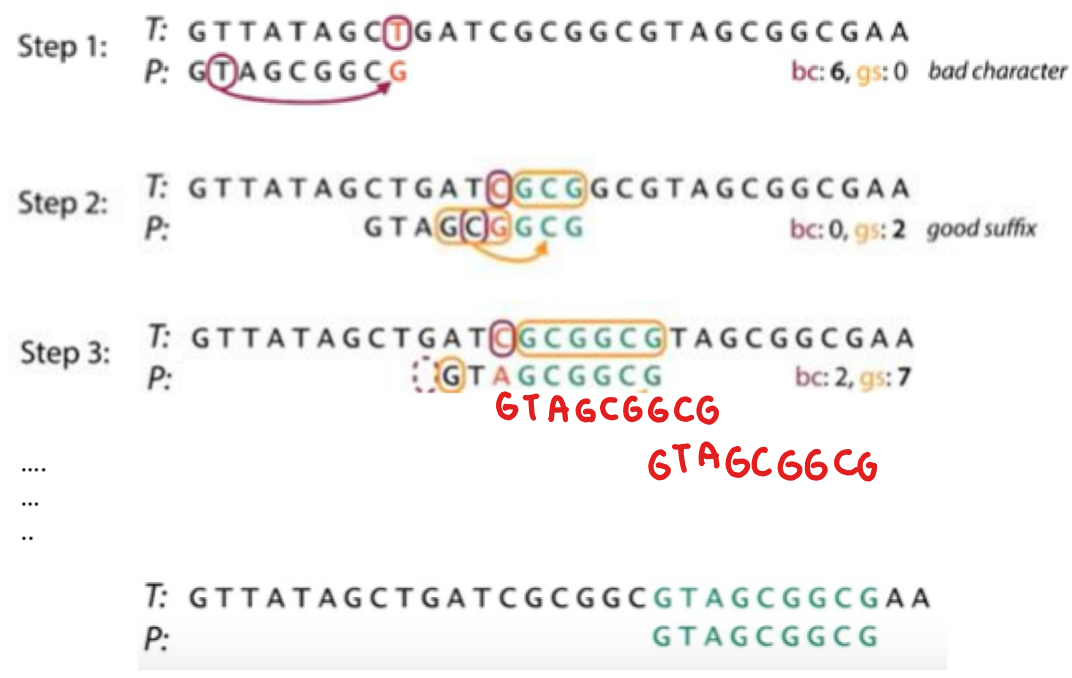
2) Good Suffix Rule

- Let t = substring match by inner loop; skip until (a) there are no mismatches between P or t or (b) P moves past t

-



-



a) Boyer - Moore - Horspool

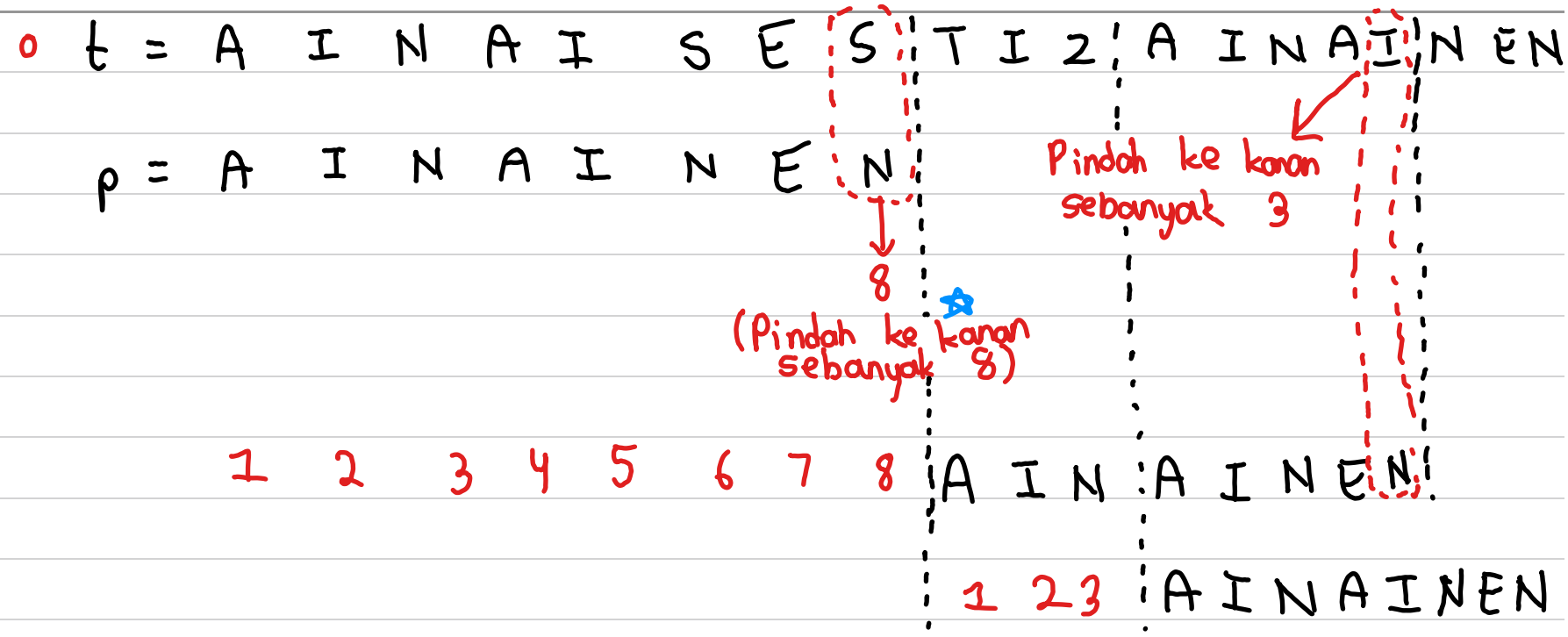
- t = AINAISESTIZAINAINEN n = 19

P = AINAINEN m = 8
0 1 2 3 4 5 6 7

- Bad match table :
(Lihat yang di P saja)

A	I	E	N	★
4	3	1	8	8

E = 8 - 6 - 1 = 1
I = 8 - 4 - 1 = 3
A = 8 - 3 - 1 = 4



Closest-Pair Problem

- Dalam Brute Force, a straight forward approach to 2 well-known problems dealing a finite set of points in the plane is closest-pair problem

Gotau ada keluar / tidak (Closest-pair Problem)

1. Initialize a variable diff as infinite (Diff is used to store the difference between pair and x. We need to find the minimum diff.

2. Initialize two index variables l and r in the given sorted array.
   (a) Initialize first to the leftmost index in ar1: l = 0
   (b) Initialize second the rightmost index in ar2: r = n-1

3. Loop while l = 0
   (a) If abs(ar1[l] + ar2[r] - sum) < diff then update diff and result
   (b) Else if(ar1[l] + ar2[r] < sum) then l++
   (c) Else r-- 4)

4. Print the result.

sample1

Input Ar1[ ] = { 1,4,5,7 }
Input Ar2[ ] = { 10,20,30,40 }
x = 38
Output = 7 and 30

sample2

Input Ar1[ ] = { 1,4,5,7 }
Input Ar2[ ] = { 10,20,30,40 }
x = 50
Output = 7 and 40

Explaination

1
Ar1[0] + ar2[0]
t = 1 + 10 = 11
diff = t-x
= 11 – 38 = -27

2
Ar1[0] + ar2[1]
t = 1 + 20 = 21
diff = t-x
= 21 – 38 = -17

.....

n
Ar1[3] + ar2[2]
t = 7 + 30 = 37
Diff = t-x
= 37 – 38 = -1

- *ClosestPair* (*P*)

```
//Input: A list P of n points (n ≥ 2). P_1 = (x_1, y_1) ...
//Output: Indices 1 and 2 of the closest pair of points.
dmin ← ∞
for i ← 1 to n-1 do
    for j ← i+1 to n-1 do
        d ← sqrt((x_i-x_j)^2 + (y_i-y_j)^2)
        if d < dmin do
            dmin ← d, index1 ← i, index2 ← j
return index1, index2
```

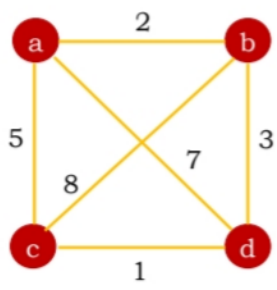
$C(n) \rightarrow \Theta(n^2)$

9) Exhaustive Search :

- Banyak masalah diselesaikan dengan permutasi, kombinasi, atau subsets of a set = Use brute force approach to combinatorial problems

- Contohnya TSP :
  - Find shortest tour through a given set of n cities
  - Must visits each city exactly once before returning to the city where it started

-



Tour	Length
a → b → c → d → a	L = 2 + 8 + 1 + 7 = 18
a → b → d → c → a	L = 2 + 3 + 1 + 5 = 11
a → c → b → d → a	L = 5 + 8 + 3 + 7 = 23
a → c → d → b → a	L = 5 + 1 + 3 + 2 = 11
a → d → b → c → a	L = 7 + 3 + 8 + 5 = 23
a → d → c → b → a	L = 7 + 1 + 8 + 2 = 18

Efficiency: $\Theta((n-1)!)$

-

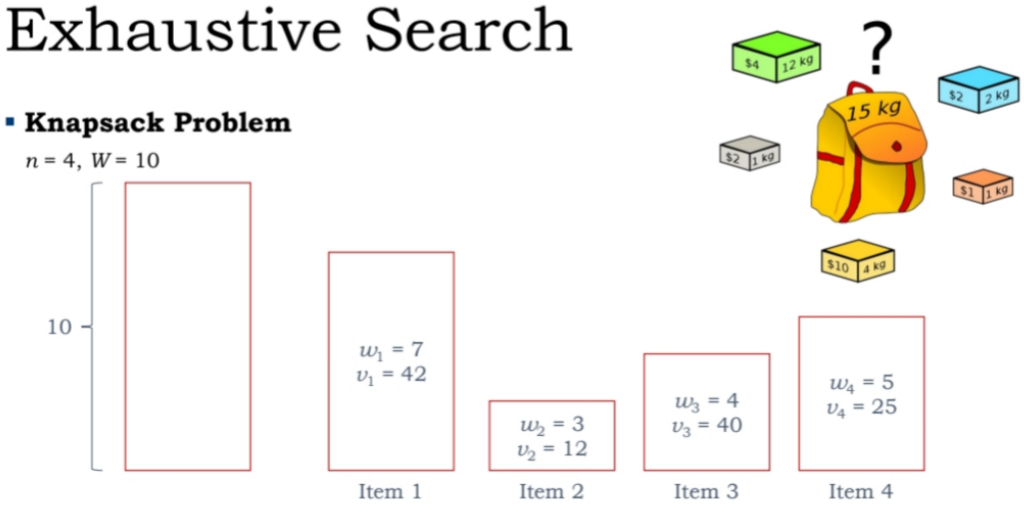
```
C=0
cost=0
visits=0
e=1 (*e=pointer of the visited city)

for r=1 to n-1 do
    choose of pointer j with
    minimum=c(e,j)=min{c(e,k);visits(k)=0 and k=1,..,n}
    cost=cost+minimum
    e=j
    C(r)=j
end r-loop

C(n)=1
cost=cost+c(e,1)
```

- Contohnya Knapsack Problem :- Given n items of known weights $w_1, ..., w_n$ and values $v_1, ..., v_n$ and a knapsack of capacity W
  - Find the most valuable subset of the items that fit into the knapsack

Exhaustive Search



Knapsack Pseudocode

```
val = array(60, 100, 120);
wt = array(10, 20, 30);
W = 50;
n = count(val);
print knapSack(W, wt, val, n);
```

```
function knapSack(W, wt, val, n)
{
  if (n == 0 || W == 0)
    return 0;

  if (wt[n - 1] > W)
    return knapSack(W, wt, val, n - 1);

  else
    return max(val[n - 1] +
        knapSack(W - wt[n - 1],
        wt, val, n - 1),
        knapSack(W, wt, val, n-1));
}
```

- Rumus Knapsack :

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ F(i-1,\ j-w_i)+v_i), & j-w_i \geq 0 \\ F(i-1,j), & j-w_i < 0 \end{cases}$$

- Capacity

	0	j-w_i	j	W
0	0	0	0	
i-1	0	F(i-1, j-w_i)	F(i-1, j)	
i	0	F(i, j-w_i)	F(i, j)	
n	0			Goal

- Contoh Soal :

item	Weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

$w_1 = 2,\ v_1 = 12$
$w_2 = 1,\ v_2 = 10$
$w_3 = 3,\ v_3 = 20$
$w_4 = 2,\ v_4 = 15$

i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Goal → Max yg dpt di-tampung oleh tas

Bandingkan Goal dengan di atasnya :

F(4,5) > F(3,5)
37 > 32 ✓ Item ke-4 dipilih (2kg)

$F(3, 5-2_{j-w_i}) = F(3,3) > F(2,3)$
22 > 22 ✗

F(2, 3) > F(1,3)
22 > 12 ✓ Item ke-2 dipilih (1 kg)

F(1, 3-1) = F(1,2) > F(0,2)
2 > 0 ✓ Item ke-1 dipilih (2 kg)

Rumus LCM

$$LCM(x,y) = \frac{x \cdot y}{GCD(x,y)}$$

Assignment Problem

1. Given n people who need to be assigned to execute n jobs, one person per job.
2. The cost that would accrue if the i-th person is assigned to the j-th job is a known quantity C[i,j].
3. Find an assignment with the minimum total cost.

- **Assignment Problem**

$n = 4$

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

<1, 2, 3, 4> Cost: 9 + 4 + 1 + 4 = 18 → means: P_1 given J_1, P_2 given J_2, P_3 given J_3, etc.

- **Assignment Problem**

First iterations:

Assignment	Cost
<1, 2, 3, 4>	9 + 4 + 1 + 4 = 18
<1, 2, 4, 3>	9 + 4 + 8 + 9 = 30
<1, 3, 2, 4>	9 + 3 + 8 + 4 = 24
<1, 3, 4, 2>	9 + 3 + 8 + 6 = 26
<1, 4, 2, 3>	9 + 7 + 8 + 9 = 33
<1, 4, 3, 2>	9 + 7 + 1 + 6 = 23

Efficiency: $\Theta(n^3)$

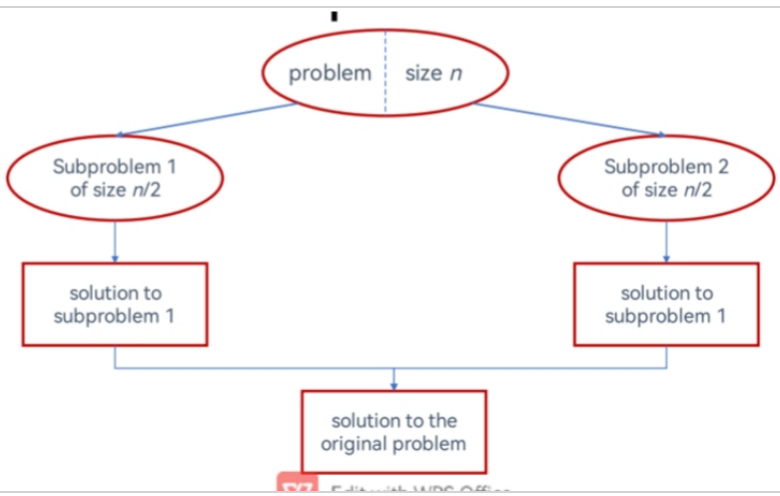
<1, 2, 3, 4> Cost: 9 + 4 + 1 + 4 = 18 → means: P_1 given J_1, P_2 given J_2, P_3 given J_3, etc.

Week 6 - Divide and Conquer

- **Divide :** Breaking it into subproblems that are themselves smaller instances of the same type of problem. Recursively solving the subproblems

- **Conquer :** Appropriately combining their answers

-



- More generally, an instance of size n can be divided into b instances of size n/b, with a of them needing to be solved.

- Recurrence for the running time T(n):

$$T(n) = aT(n/b) + f(n)$$

General divide-and-conquer recurrence

Function that accounts for the time spent on dividing the problem and combining their solutions.

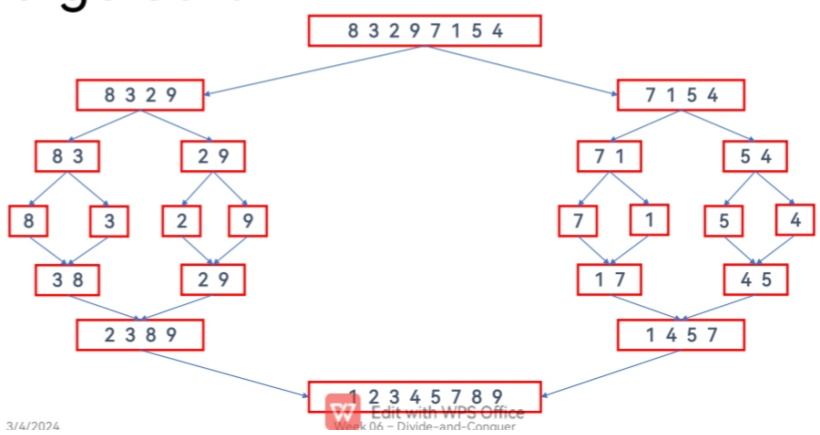
Merge Sort :

-

```
MergeSort (A[0..n-1])
    if n > 1
        copy A[0..⌊n/2⌋-1] to B[0..⌊n/2⌋-1]
        copy A[⌊n/2⌋..n-1] to C[0..⌊n/2⌋-1]
        MergeSort(B[0..⌊n/2⌋-1])
        MergeSort(C[0..⌊n/2⌋-1])
        Merge(B, C, A)
```

```
Merge (B[0..p-1], C[0..q-1], A[0..p+q-1])
    i ← 0; j ← 0; k ← 0
    while i < p and j < q do
        if B[i] ≤ C[j]
            A[k] ← B[i]; i ← i+1
        else A[k] ← C[j]; j ← j+1
        k ← k+1;
    if i = p
        copy C[j..q-1] to A[i..p + q − 1]
    else copy B[i..p-1] to A[k..p + q − 1]
```

Merge Sort



- Merge analysis:
  - **Case 1**: where all of the elements of list B are smaller than the first element of list C?
  - **Case 2**: what if the first element of A is greater than the first element of B, but all of the elements of A are smaller than the second element of B?
  - Which is the worst-case? Best-case?
  - $T(n)$?

Merge Sort

- How efficient? Assume $n = 2^k$

$T(n) = aT(n/b) + cn$

1. $T(n) = 2T(n/2) + cn$
2. $T(n) = 2[2T(n/4) + cn/2] + cn$
   $= 4T(n/4) + 2cn$
3. $T(n) = 4[2T(n/8) + cn/4] + 2cn$
   $= 8T(n/8) + 3cn$

$T(n) = 2^k T(n/2^k) + k\,cn$

a. $T(n) = 4T(n/2) + n,\ T(1) = 1$
b. $T(n) = 4T(n/2) + n^2,\ T(1) = 1$
c. $T(n) = 4T(n/2) + n^3,\ T(1) = 1$

subtitute
$T(1) = 0$
$n = 2^k \rightarrow k = \log n$

to

$T(n) = 2^k\, T(n/2^k) + k\,cn$

$T(n) = 2^{\log n}\, T(1) + cn\ \log n$

$T(n) = cn\ \log n$

$\mathbf{O}(n) = n\ \log n$

9) Quick Sort :

- Unlike mergesort subarrays for sorting and merging are formed dynamically, depending on the input.

- Pick pivot in each step and re-arrange the array:
  - Elements ≤ pivot appear to the left of pivot
  - Elements ≥ pivot appear to the right of pivot

```
QuickSort (A[l..r])
    if l < r
        s ← Partition(A[l..r]) // choose a pivot, s = split position
        QuickSort (A[l..s-1])
        QuickSort (A[s+1..r])
```

- Pivot analysis:
  - **Case 1**: when pivot happened to be the largest or smallest element of the array.
  - **Case 2**: when pivot list creates two parts that are the same size.

- Which is the worst-case? Best-case?

- $T(n)$?

- **Best-case:**
  - When the pivot happens to divide the array into 2 exactly equal parts.
  - Size of an array, $k = n/2$. Size of another array = $n - k = n/2$.

$$T(n) = T(k) + T(n - k) + cn$$
$$T(n) = T(1) + T(n - 1) + cn$$

- **Best-case:** Assume $n = 2^k$

$T(n) = aT(n/b) + cn$

1. $T(n) = 2T(n/2) + cn$
2. $T(n) = 2[2T(n/4) + cn/2] + cn$
   $= 4T(n/4) + 2cn$
3. $T(n) = 4[2T(n/8) + cn/4] + 2cn$
   $= 8T(n/8) + 3cn$

$T(n) = 2^k T(n/2^k) + k\,cn$

subtitute

$T(1) = 0$

$n = 2^k \rightarrow k = \log n$

to

$T(n) = 2^k T(n/2^k) + k\,cn$

$T(n) = 2^{\log n} T(1) + cn \log n$

$T(n) = cn \log n$

$\mathbf{O}(n) = n \log n$

- **Worst-case:**
  - When pivot happened to be the largest or smallest element of the array.
  - Size of an array, $k = 1$. Size of another array = $n - k = n - 1$.

$$T(n) = T(k) + T(n - k) + cn$$
$$T(n) = T(1) + T(n - 1) + cn$$

- **Worst-case:**
  1. $T(n) = T(n-1) + T(1) + cn$
  2. $T(n) = [T(n-2) + T(1) + c(n-1)] + T(1) + cn$
     $= T(n-2) + 2T(1) + c(n-1+n)$
  3. $T(n) = [T(n-3) + T(1) + c(n-2)] + 2T(1) + c(n-1+n)$
     $= T(n-3) + 3T(1) + c(n-2+n-1+n)$

  $T(n) = T(n-i) + iT(1) + c(n-i+1+\ldots+n-1+n)$
  $= T(n-i) + iT(1) + c\sum_{j=0}^{i-1} n-j$

- **Worst-case:**

  $T(n) = T(n-i) + iT(1) + c\sum_{j=0}^{i-1} n-j$

  As recurrence an only go until $i = n-1$, subtitute $i = n-1$

  $T(n) = T(n-(n-1)) + (n-1)T(1) + c\sum_{j=0}^{n-2} n-j$
  $= T(1) + (n-1)T(1) + c\sum_{j=0}^{n-2} n-j$
  $= (n-1)T(1) + c\left(n(n-2) - \frac{(n-2)(n-1)}{2}\right)$

  $O(n) = n^2$

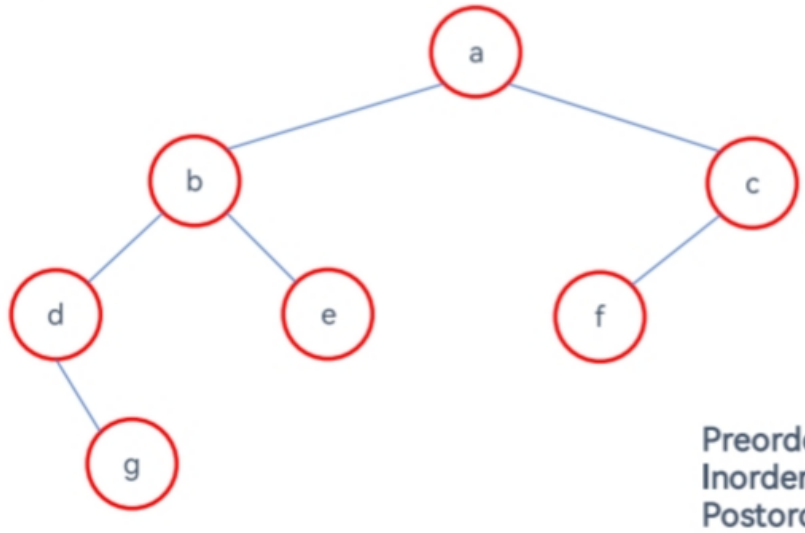
- **Avoiding worst-case:**
  - Not using quick sort on sorted data.
  - Pick a pivot randomly each time.
  - Find the exact median of three rule as a pivot.
    - When sor tng A[6..20], examine A[6], A[13] → ((6+20)/2), and A[20. Select the element with median key.
    - A[6] = 30, A[13] = 2, A[20] = 10 → A[20] as a pivot.

9) Binary Tree Traversal :

1. Preorder Traversal : The root is visited before left and right subtrees are visited

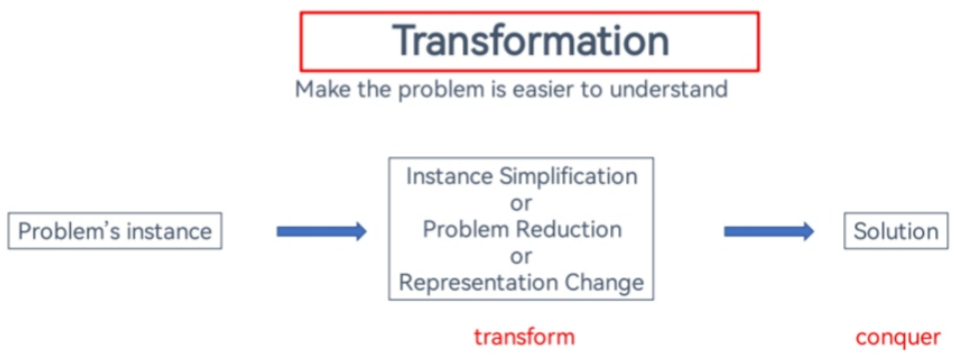
2. Inorder Traversal : The root is visited after visiting its left subtree, but before visiting the right subtree

3. Postorder Traversal : The root is visited after the left and right subtrees



Preorder : a, b, d, g, e, c, f
Inorder : d, g, b, e, a, f, c
Postorder : g, d, e, b, f, c, a

Week 7 – Transform and Conquer



- Presorting = An old idea, sort the data and that follows to more easily compute some answer.

- In presorting, many problems involving lists are easier when list is sorted
  1. Element Uniqueness
  2. Computing the mode of n numbers

- Given a list A of n orderable elements, determine if there are any duplicates of any elements.

Brute Force:	Presorting:
for each x ∈ A	Sort A
for each y ∈ {A-x}	for i←1 to n-1
if x=y return not unique	if A[i] = A[i+1]
return unique	return not unique
Run Time?	return unique

- **Brute force algorithm**: $\Theta(n^2)$

- **Presorting-based algorithm**:
  - Use mergesort (optimal): Θ(n log n)
  - Scan array to find repeated adjacent elements: Θ(n)

  } Θ(n log n)

- **Conclusion**: Presorting yields significant improvement.

Brute Force:

```
max ← max(A)
freq[1..max] ← 0
for each x ∈ A
    freq[x] += 1
mode ← freq[1]
for i ← 2 to max
    if freq[i] > freq[mode] mode ← i
return mode
```

Modus by BruteForce

Presorting:

```
Sort A
i ← 0
modeFrequency ← 0
while i ≤ n-1
    runlength ← 1; runvalue ← A[i]
    while i+runlength ≤ n-1 and A[i+runlength] = runvalue
        runlength=runlength+1
    if runlength > modefrequency
        modefrequency←runlength; modevalue←runvalue
    i←i+runlength
return modevalue
```


Modus by Presorting

- **Brute Force:** scan the list and compute the frequencies of all its distinct values. (assumes all values > 0; what if they aren't?) → $\Theta(n^2)$
- **Presorting:** sort the input first. Then all equal values will be adjacent to each other. To compute the mode, just find the longest run → $\Theta(n \log n)$
- **Conclusion**: Presorting yields significant improvement.

c) Eliminasi Gauss & Gauss - Jordan seperti di linier algebra.