

# Design & Analysis of Algorithms

Email: KhalidKhan@gmail.com

## Pre Requisites :

- (i) Data Structures
  - (ii) Programming (Constructs)
  - (iii) Discrete Maths (Combinatorics)
    - + Recurrences
    - + Logarithms
    - + Functions

GATE + TIFR + BARC + ISRO  
+ State Exams +  
Placements +  
Campus || in P.B.C

# (Competitive Programming)

Read Books :

1) Introduction to Algorithms

- Cormen (CLR)

2) Fundamentals of Computer Algorithms

By - Horowitz Sahni

$\checkmark$  BFS + DFS

Search :  $\checkmark$   $W \cdot C$  +  $B \cdot C$

## Syllabus :

### 1) Analysis :

- Concept of Algorithm & Lifecycle steps
- Need for Analysis
- Methodology of Analysis
- Types of Analysis
- Asymptotic Notations
- Framework for Recursive & Non Recursive Algo's
- Loop complexities

→ Mathematic Background

### 2) Design Strategies

- Divide & Conquer:
- Greedy Method:
- Dynamic Programming:

### 3) Graph Techniques:

- Traversals (DFS + BFS)
- Components

### 4) Heap Algorithms:

### 5) Sets & operations:





6) Sorting Algorithms:

7) Hashing:

Concept of Algorithm: Consists of finite set of steps to solve a problem.

Computer II

Statements

<CS>

Algorithm:

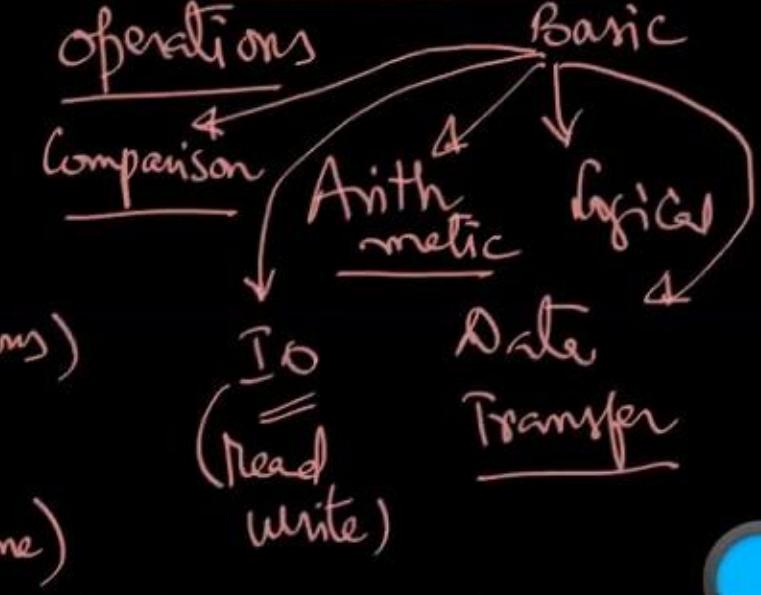
Md. Al Khwarizmi (Musa)

$$a = b + c *$$

Ex:

1.  $x \leftarrow y + z$ ; Read
2. ~~Read (x);~~ Scanf ("%d", &x);
3. [for  $i \leftarrow 1$  to  $n$   
 $a \leftarrow b + c$  ]  
• • •

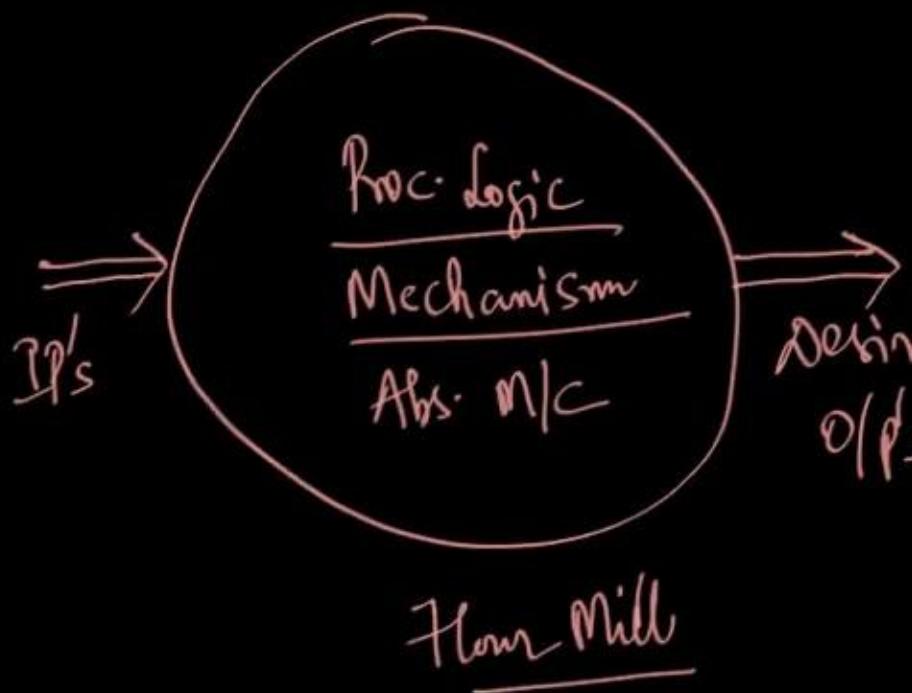
each Statement / Step  
involves one (more) fundamental



Opn:

- ⇒ definiteness (clear unambiguous)
- ⇒ effective (finite time)

→ Algorithm may accept or ignore I/P's, but must produce at least one O/P;



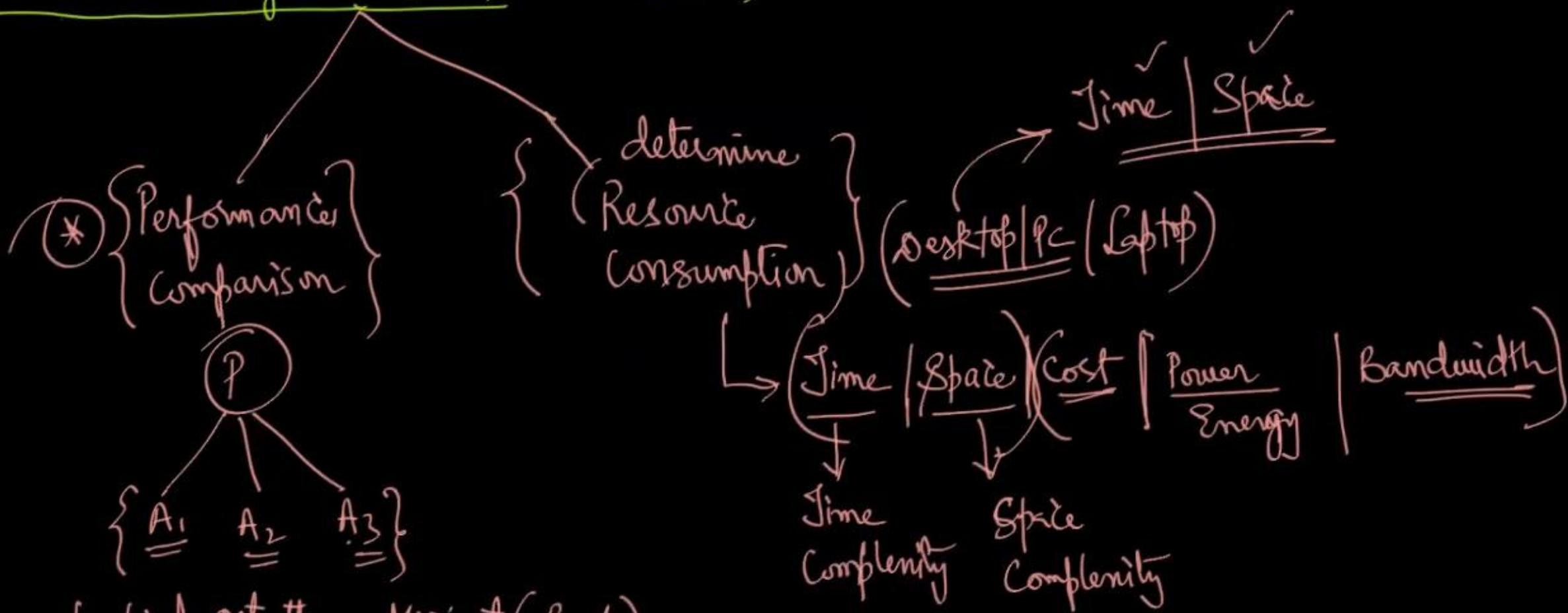
## Lifecycle Steps

- Problem Statement
  - Constraints / Requirements (SRS)
  - ~~Design~~ | Logic | Mechanism \*
  - Develop Algo | Flowchart
  - Validation (correctness)
  - Analysis (why) (how) \*
  - Implementation
  - Testing & Debugging

DAA

CS Problem  
 $\downarrow$   
 $(S \sqsubseteq M) = \text{False}$

Need for Analysis : (why) : + (what)

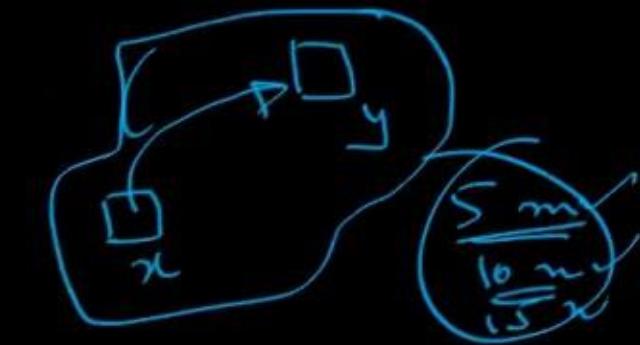


to find out the efficient (Best)  
optimal Soln | Algo.

# Methodology of Analysis : (How to Analyze)?

$$1. [x \leftarrow y + z]; \quad \begin{cases} H/w : (\underline{\text{CPU}} + \underline{\text{Mem}} + \underline{\text{IO}} \text{ services}) \\ S/w : (\underline{\text{Compiler}} + \underline{\text{OS}}) \end{cases}$$

→ ① Aposteriori Analysis (Platform Dependent)



## Advantages

→ { Exact value of Resources in real units }

## Disadvantages

→ difficult to carry out  
→ diff. Times (units): difficult to make perf. comparison  
(Non-Uniformity) → Not inclusive of all



## 2) Apriori Analysis : (Platform Independent Analysis)

↳ (bf Implementation)

### Analytic Framework:

$$\left\{ \begin{array}{l} x \leftarrow y + z \\ \hline \end{array} \right.$$

- Take into account all possible IP's ✓
- Allows us to evaluate the relative efficiency of two Algo's in a way that is independent from the Platform.
- Can be carried out by studying the high level description of Algorithm without actual Implementation

## Components of Analytic Framework to Carryout Apriori Analysis:

## 1. A language for describing algorithm (Sparks)

② A Computational Model that the Algorithm executes within it RAM

## RAM Mode

$$x \leftarrow y + 3$$

for  $i \leftarrow 1$  to  $n$

fn i:=1 to n

for ( i=1; i < n; ++i )

Memory | .. . . . .

fund. \$80

3. A metric for measuring  
Algo. running time

↳ An approach to characterize the Running time. (ASN)

fund- opn (+, -, \*, /, =, !, <, >, =, read, -)

(CPY) ↳ takes 1 unit of time =  $T(n)$

(10 units  
1000 units  
'n' - n units) } Step-Count = Time  
opn-Count

Limitation of A priori Analysis

→ Approximate values of time & Space

→ Uniformity in Analysis & hence facilitate Perf. Comparison ;



Algorithm Test

1.  $x \leftarrow y + z;$  ✓2. for  $i \leftarrow 1$  to  $n$   
{ $x \leftarrow y + z;$ }3. for  $i \leftarrow 1$  to  $n$   
{for  $j \leftarrow 1$  to  $n$   
( $c \leftarrow a + b;$ )}

} }

 $n = \text{size of Input}$  $i=1$  $i \leq n$ { $1 \leq 2$ }{ $2 \leq 2$ }{ $3 \leq 2$ }

linear

quadratic

cubic

(Polynomial)

Time

(exponential)

 $(2^n, 4^n, n^n)$ Time Complexity :  $T(n) = \overline{1+1} + \left( \overline{1+ \underline{(n+1)}} + \underline{n} + \underline{n} \right)$  $+ \underline{n}$ for { $\overline{1+ \underline{(n+1)}} + \underline{n} + n(n+1)$  $+ n \cdot n + n \cdot n$ 

$$= \underline{6} + \underline{8n} + \underline{3n^2}$$

$$\underline{T(n)} = \underline{\left( 3n^2 + 7n + 6 \right)}$$

expressing the  
time w.r.t to input size)

=

## Algorithm Test

```

1.  $x \leftarrow y + z;$  1
2. {for  $i \leftarrow 1$  to  $n$ 
     $x \leftarrow y + z;$ }  $n$ 
3. for  $i \leftarrow 1$  to  $n$ 
    {for  $j \leftarrow 1$  to  $n$ 
         $c \leftarrow a + b;$ }  $n^2$ 
    }
}

```

Determining the Time Complexity of the Algorithm based on Order of Magnitude:

→ order of Magnitude of a statement refers to the frequency (no. of times) of the fund. operation (metric) involved in the statement;

$$T(n) = \underbrace{(1 + n + n^2)}_{\text{Polynomial of degree '2'}}$$

$$= \underline{\underline{O(n^2)}} \checkmark$$

→ The objective of Apriori Analysis is to represent the time complexity of the Algorithm by means of a mathematical fn in terms of 'IP' size (say  $n$ ) .

→ These fns can be in the form of either Polynomials or Exponentials;

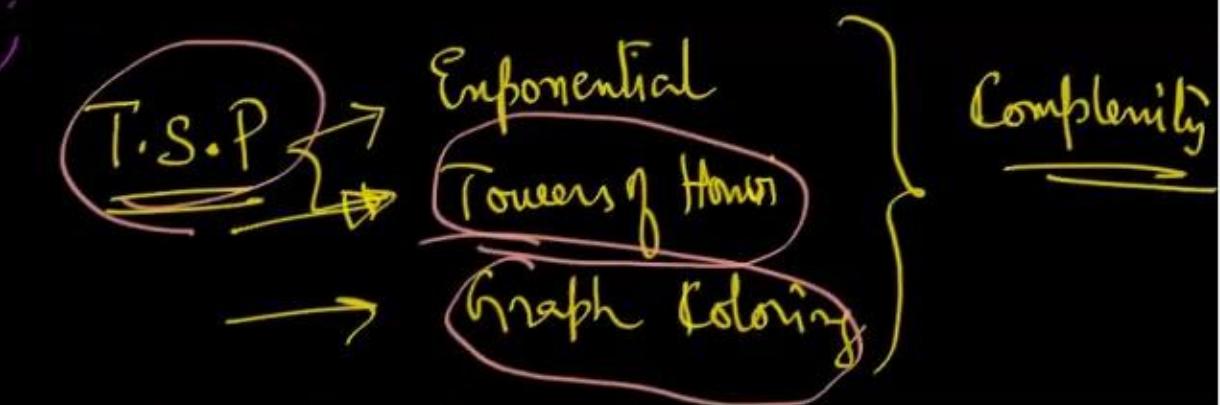
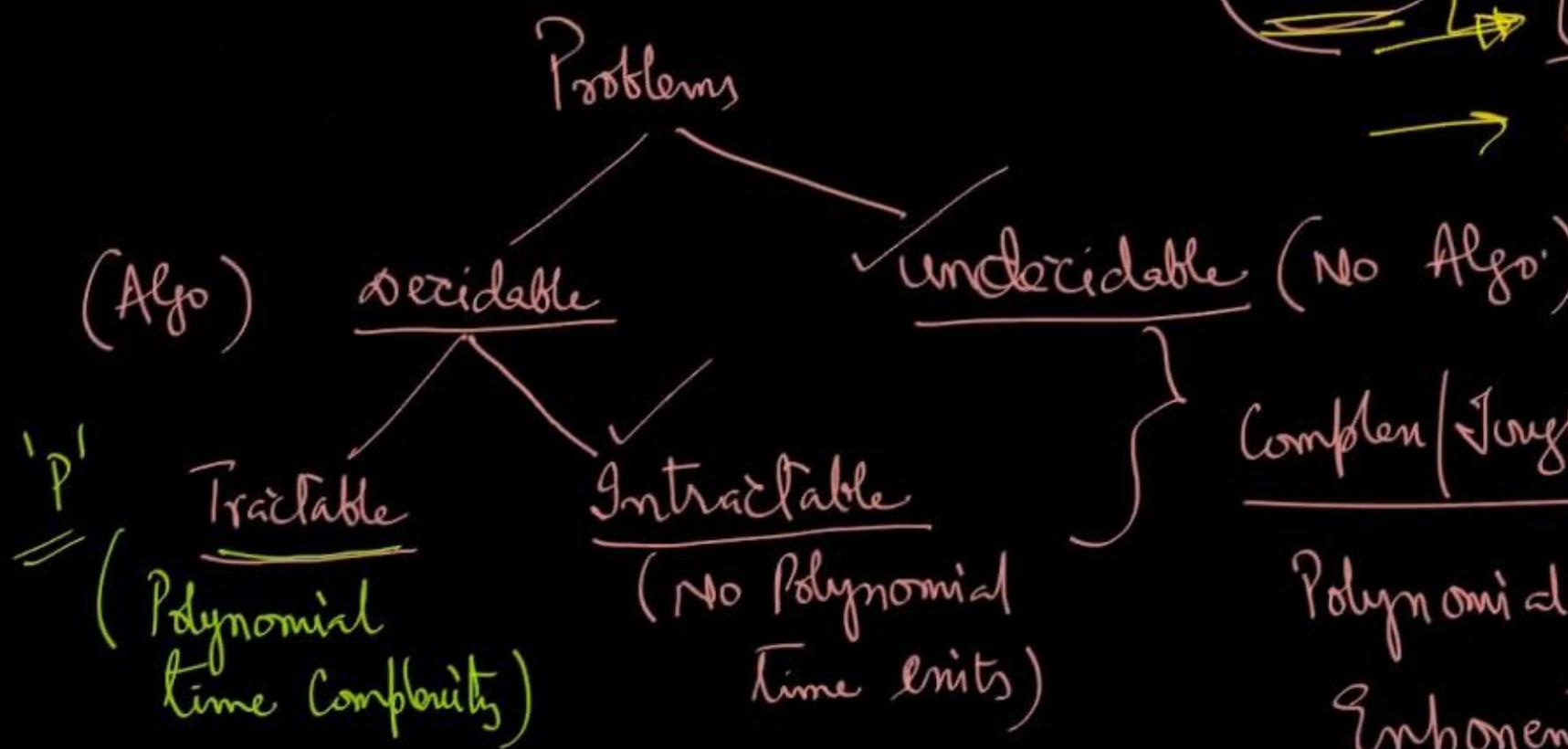
→ Polynomials has lesser rate of growth than exponentials.

$$T_1(n) = \underline{\underline{n^2}} \quad T_2(n) = \underline{\underline{2^n}}$$

| $n$ | $n^2$ | $2^n$ |
|-----|-------|-------|
| 4   | 16    | 16    |
| 5   | 25    | 32    |
| 6   | 36    | 64    |
| 7   | 49    | 128   |

$(n^2 < 2^n)$

→ It is always desirable to develop Algorithms for Problems that has Polynomial Complexity;

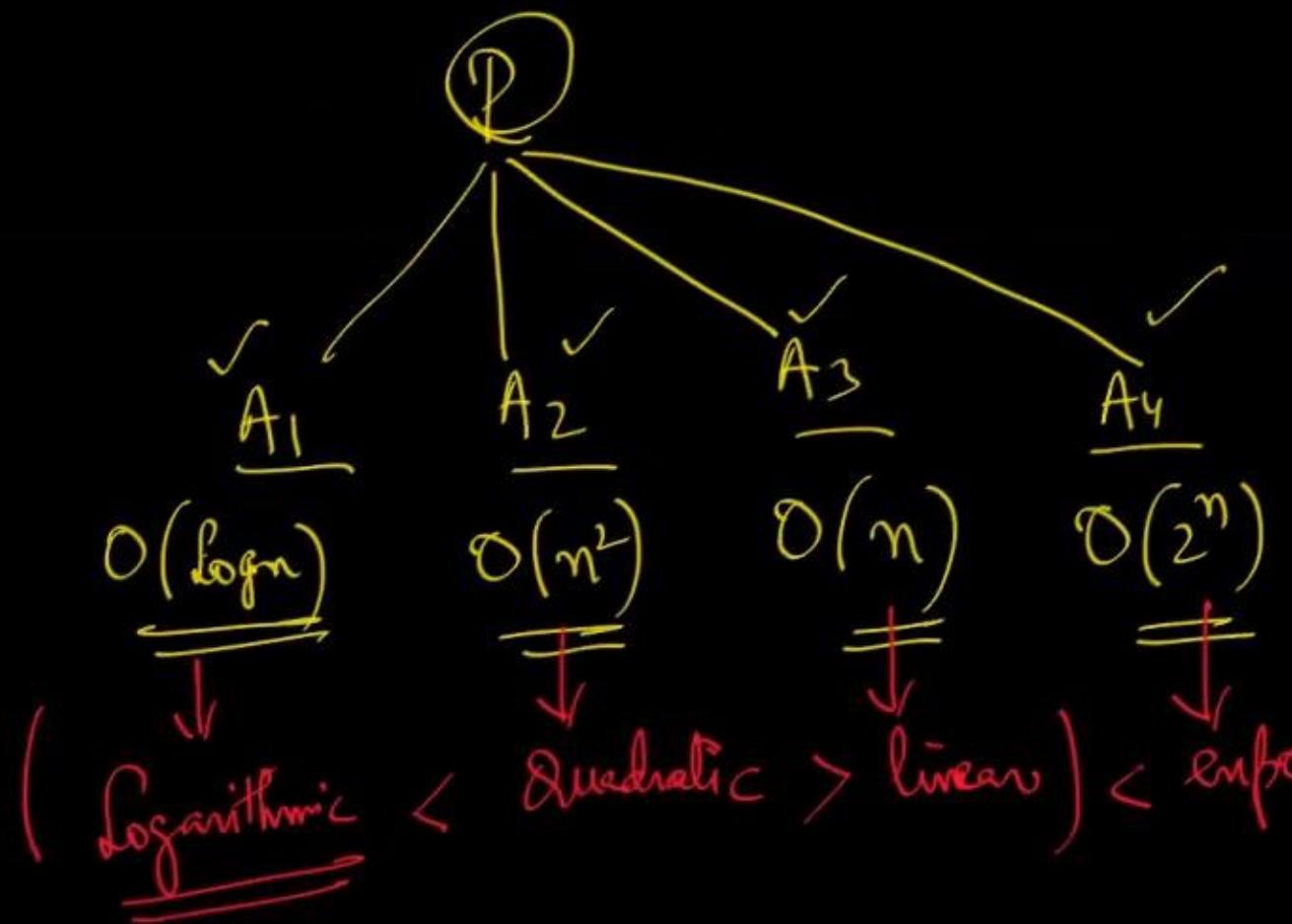


Complexity

Complex (Tough Problems)

Polynomial fn:  $n^x$  ( $x > 0$ )

Exponential fn:  $a^n$  ( $a > 1$ )



| <u>n</u> | <u>log n</u> |
|----------|--------------|
| 4        | 2            |
| 8        | 3            |

$$\left( \log n < \underline{n} < \underline{n^2} \right) < 2^n$$



$$\log n \rightarrow \sqrt{n} \xrightarrow{(n \rightarrow \infty)} (n > K \rightarrow \infty)$$

$f_1 < f_2$

Apply Log. on both  $f_{n1}$

$$\log_2 \log n$$

$$\log_2 \sqrt{n}$$

$$\log_2 n^{1/2}$$

$$\log_2 \log_2 n < \frac{1}{2} \log_2 n$$

$\frac{1}{2}$

$$\begin{cases} n = 16 \\ n = 64 \end{cases}$$

$$f(n) = n$$

$$g(n) = 2^{\log_2 n} = n^{\log_2 2} = a^{\log_b c} = b^{\log_a c}$$

$$\frac{q > 8}{\frac{n^2}{16} < \frac{2^n}{16}}$$

$n, \dots$

$$\log_2 n^x = x \cdot \log n$$



$$\underline{\underline{(n)^{\frac{1}{\log_2}}}} < \log_2 n$$

$$n^{\frac{\log_2}{2}}$$

$$2^{\log n}$$

$$\underline{\underline{\alpha}} < \log n$$

$$c < \log_2 n$$

### Types of Analysis | Behaviour of Algorithm:

- (i) Worst Case
- (ii) Best Case
- (iii) Avg. Case

→ For a fixed input of size ' $n$ ', one can have various arrangements of input (Input classes)  $(I_1, \dots, I_K)$



Worst Case: The IP class for which the Algo. does max. work & hence take max. time is known as W-C input & the corresponding time is known as W-C time.

Ex: Linear Search: element is found at place:  $O(n)$

Binary Search:  $O(\log n)$ : leaf level

Quick Sort: Sorted order:  $O(n^2)$

Best Case: The IP class for which the Algo. does minimum work & take least time.

Ex: linear Search: element (key) is found @ first place:  $O(1)$

Quick Sort: Unsorted input :  $O(n \log n)$

Average Case :

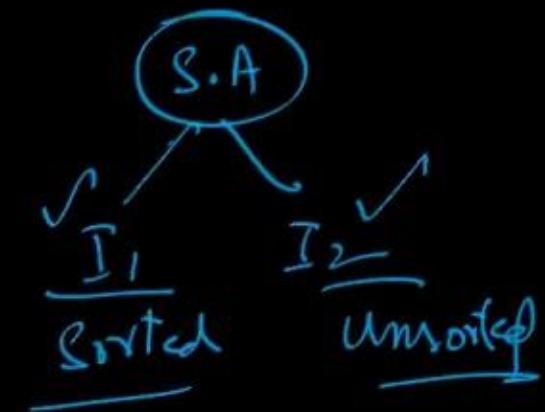
Avg. Case is defined in 3 steps

Step 1: define All input classes ( $I_1, I_2, \dots, I_k$ )

Step 2: determine the time for each IP class ( $t_1, t_2, \dots, t_k$ )

Step 3: Associate with each IP class ( $p_1, p_2, \dots, p_k$ )  
a prob. func.

$$A(n) = \sum_{i=1}^k p_i * t_i$$



If  $B(n) = \text{Best Case Time}$   
 $w(n) = \text{Worst Case Time}$   
 $A(n) = \text{Avg. Case Time}$

$\Rightarrow \boxed{B(n) \leq A(n) \leq w(n)}$  - ①

for of IL-size = n

(IV)  $\left\{ \begin{array}{l} B(n) < A(n) < w(n) \\ \log n < n < n^2 \end{array} \right.$  ???

i.  $B(n) = A(n) = w(n)$  :  $\frac{n \log n}{\text{MergeSort}} + \frac{n \log n}{\text{HeapSort}} + \frac{n^2}{\text{SS}}$ .

ii.  $B(n) < [A(n) = w(n)]$  :  $\frac{\text{Linear Search}}{n} + \frac{\text{Binary Search}}{\log n} + \text{IS}$

iii.  $[B(n) = A(n)] < w(n)$  :  $\frac{BS}{n \log n} + \frac{n^2}{\log n}$

www.KhaleefKhan.info

↳ Resources

↳ Algo Handout

→ Need for Analysis

→ Methodology of Analysis

(Aposteriori Analysis  
Platform dependent)

Apriori Analysis (Platform Independent)

$$\hookrightarrow \text{Time} = (\sum \text{Step-Count})$$

$$\text{Time} \propto f(n)$$

# 1) Apriori Analysis:

→ Rate of growth of time by a  $f_n(n)$  : ( $n$  increases)

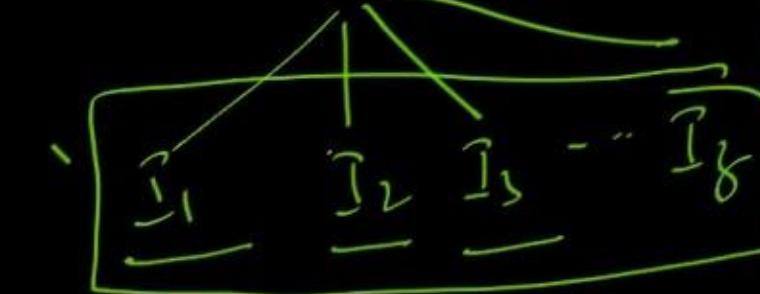
W.C. → Behavior of Algo  
(Types of analysis)

A.C.

B.C.

$$B(n) \leq A(n) \leq W(n)$$

For a fixed IP of size  $n$



$\Sigma$  list

Poly

Slow

(Fast) sum

Expo :  $2^n; 4^n; n^n$

Fast ( $a^n$ )

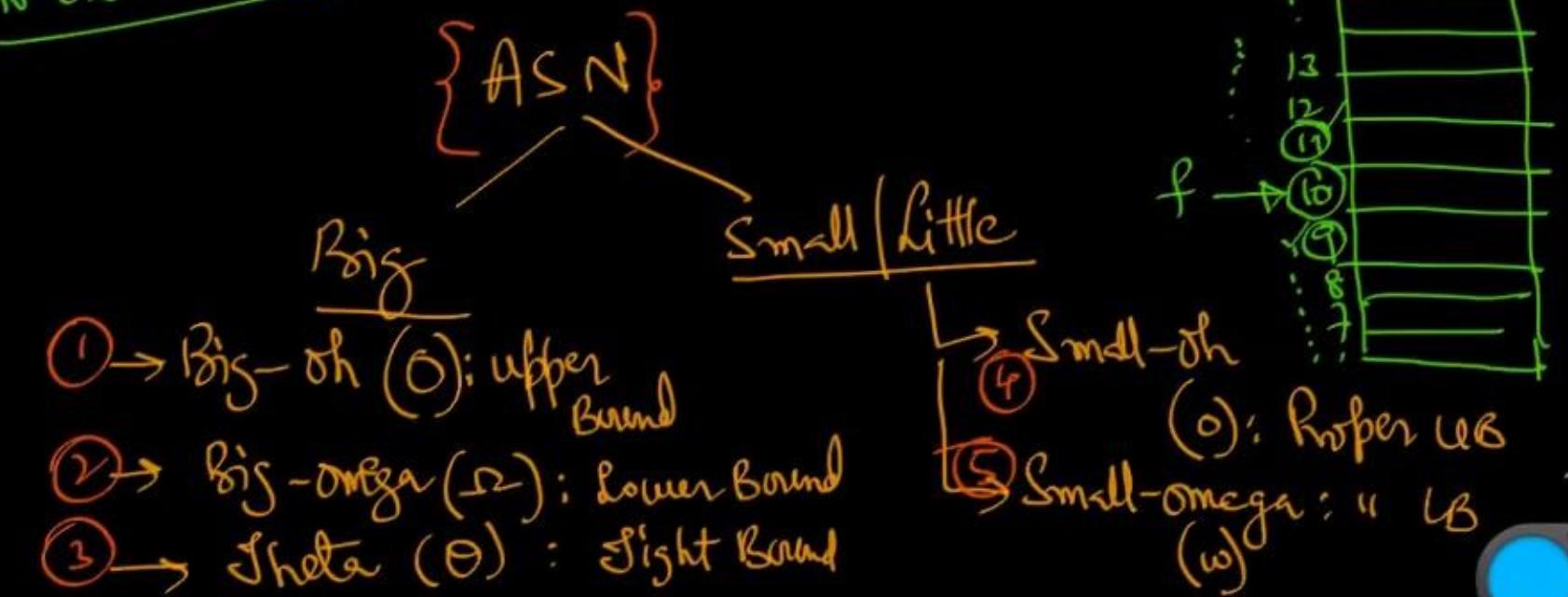
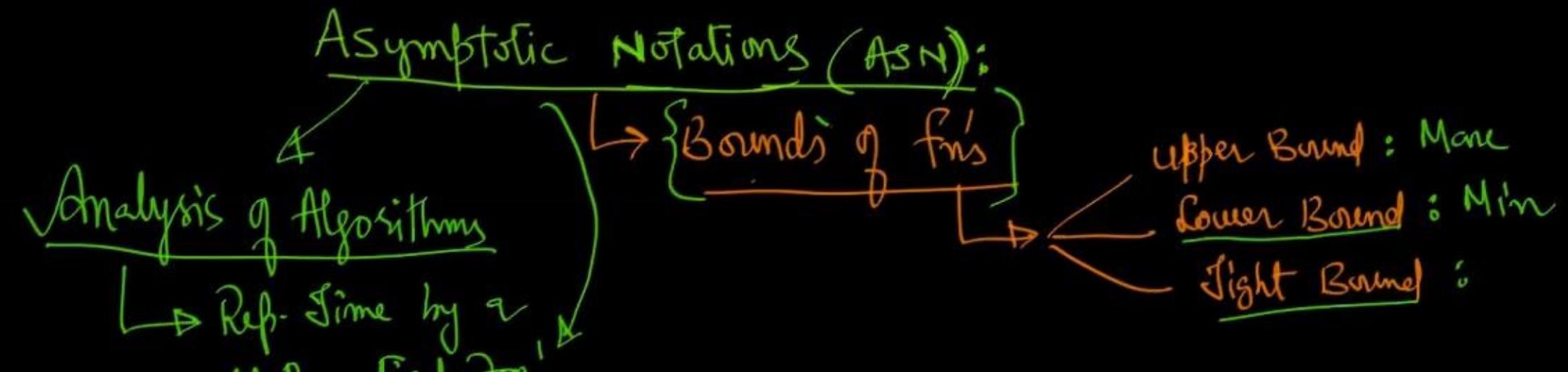
Slow ( $a>1$ )

Linear :  $n$

Quadratic :  $n^2$

Cubic :  $n^3$

Logarithmic :  $\log n$



Definitions: Let ' $f$ ' & ' $g$ ' be functions from set of integers or reals to reals;

### ① Big-Oh(0): Upper Bound:

$f(n)$  is  $O(\underline{g(n)})$  [ $f(n)$  is order of  $\underline{g(n)}$ ], iff  $\underline{\underline{f(n)}} \leq c \cdot \underline{\underline{g(n)}}$  ✓  
for some  $c > 0$ , & whenever  $n \geq k$  (for some  $k > 0$ ) [ $n > k \dots \infty$ ]

$$\exists n: f(n) = \underline{(1+n+n^2)} = O(\underline{n^2})$$

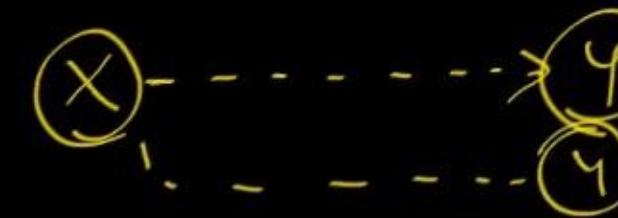
$$1+n+n^2 < 2 \cdot n^3, n \geq 1$$

$c \uparrow g$

$$\begin{aligned} 1+n+n^2 &< n^2+n^2+n^2, \quad n \geq 1 \\ 1+n+n^2 &< 3n^2 \quad \checkmark \\ f(n) &\uparrow \quad \quad \quad c \uparrow g \end{aligned}$$

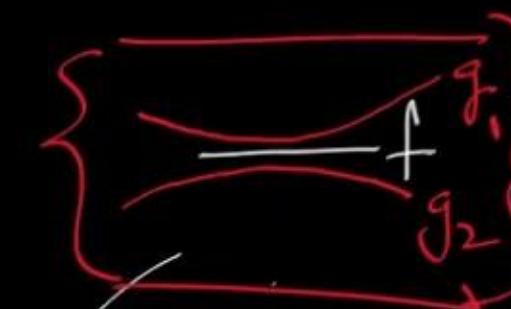
$$\left\{ \begin{array}{l} f(n) \text{ is } O(n^2) : 11 \\ f(n) \text{ is } O(n^3) : 12 \\ O(n^4) \quad \checkmark \end{array} \right.$$





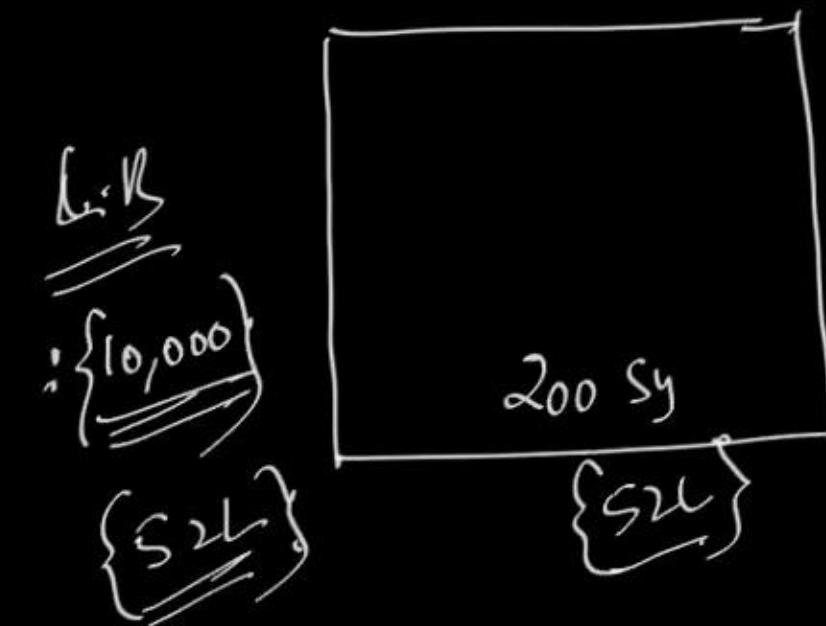
$$\begin{aligned} & : 15m \checkmark \\ & : 1m \times \\ & : 1(\text{one day}) \times \\ & : (1 \text{ year}) \times \end{aligned}$$

$\longrightarrow g$



$$\frac{g(n)}{f(n)} \sim \frac{g(n)}{f(n)}$$

Max : Upper Bound  
 $\therefore \underline{\underline{(100L)}}$



## 2) Big-Omega ( $\Omega$ ); Lower Bound:

$f(n)$  is  $\Omega(g(n))$  iff  $f(n) \geq c \cdot g(n)$  for some  $c > 0$   
 and whenever  $n > K$  ( $K > 0$ )

---

$$f(n) = 1 + n + n^2 \leq O(n^2)$$

$$\Omega(n^2) \left\{ \begin{array}{l} \Omega(1) \quad [1 + n + n^2 > \frac{1}{c}g, n > 1] \\ \Omega(n) \quad [1 + n + n^2 > \frac{n}{c}g, n > 1] \\ \Omega(n^2) \quad [f \quad [1 + n + n^2 > \frac{n^2}{c}g, n > 1]] \end{array} \right.$$

### 3) Theta( $\theta$ ): Tight Bound

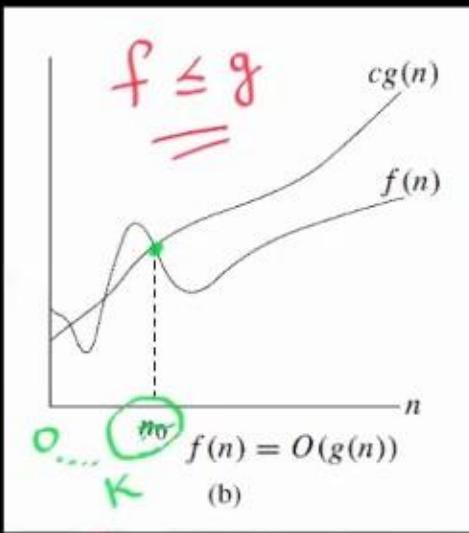
$f(n)$  is  $\Theta(g(n))$  iff  $f(n)$  is  $\underline{\mathcal{O}(g(n))}$  and  
 $f(n)$  is  $\overline{\Omega(g(n))}$ ;

$$\boxed{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$$

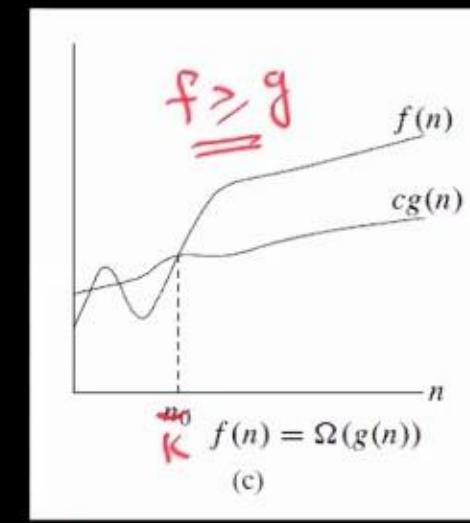
$$\boxed{1 \cdot n^2 \leq 1 + n + n^2 \leq 3 \cdot n^2} \quad n > 1$$

for some  $(c_1, c_2) > 0$   
 whenever  $n > k$

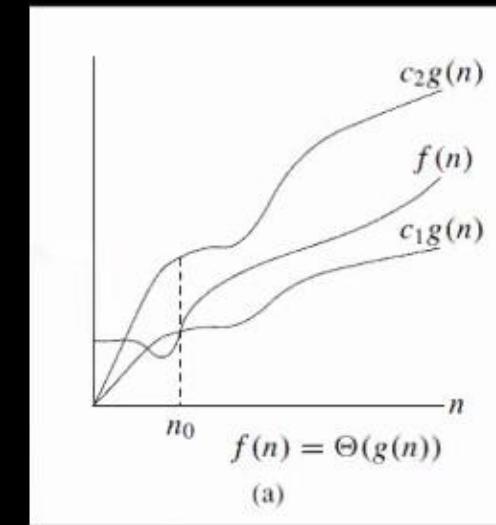
$$\therefore f(n) \in \Theta(n^2)$$



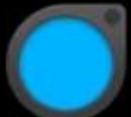
(Big-Oh)



(Big-Omega)



(Theta)



$$2) f(n) = \underline{2^{100}} \begin{cases} O(1) \\ \Omega(1) \end{cases} \Rightarrow \boxed{\frac{2^{100}}{c} < 3^{\underline{100}} \cdot 1} \therefore f \in O(1)$$

$$3) f(n) = \underline{n} \begin{cases} O(n) \\ \Omega(n) \end{cases} \therefore \Theta(n) \quad \frac{n < 2n, n > 1}{f} \quad \frac{2^{100}}{c} > \frac{100 \cdot 1}{g}$$

$$4) f(n) = \frac{n + \log n}{\therefore \Theta(n)} \begin{cases} O(n) \\ \Omega(\log n) \\ \Omega(n) \end{cases} \quad \frac{n + \log n}{c} < n + n < \frac{2n}{g}$$

$$5) f(n) = n + n^2 \log n + n \sqrt{n} \quad \left[ \begin{array}{l} O(n^2 \log n) \\ \Omega(n^2 \log n) \end{array} \right] \quad \Theta(n^2 \log n) \quad \frac{(n + \log n)}{c} > \frac{n}{g}$$

$$6) f(n) = \frac{n + 2^{100}}{\begin{cases} O(n) \\ \Omega(n) \end{cases}} \therefore \Theta(n)$$

$$f(n) = n + \underline{n^2 \log n} + n \sqrt{n} \leq n^2 \log n + n^2 \log n + n^2 \log n = \textcircled{3.0} n^2 \log n, n > 0$$

$\Omega(n^2 \log n)$

$$n + \underline{n^2 \log n} + n \sqrt{n} > \underline{\frac{n^2 \log n}{2}} \geq n^2 \log n = \mathcal{O}(n^2 \log n)$$

$$f(n) \leq c \cdot g(n)$$

$\cancel{c}$

for some  $c > 0$

$$n + \underline{n^2 \log n} + n \sqrt{n} < \textcircled{1.0} n^2 \log n, n > 1$$

$$4 + \underline{16 \cdot 2 + 4 \cdot 2} < 16 \cdot 2 \quad n=4$$

$$1) f(n) = 2^{\sqrt{n} \log n} \underset{\sim}{\sim} O(n^{\sqrt{n}}) \underset{\sim}{\sim} \Theta(n^{\sqrt{n}}) 2^{\sqrt{n} \cdot \log_2 n} = 2^{\log_2 n^{\sqrt{n}}} = 2^{\sqrt{n} \log_2 n}$$

$$\frac{a \log_b c}{a \cdot \log_c b} = \frac{\log_a c}{\log_b a}$$

$$2) f(n) = (n)^{\frac{1}{\log_2 n}} \underset{\sim}{\sim} O(1) \underset{\sim}{\sim} \Theta(1)$$

$$(n)^{\frac{1}{\log_2 n}} = n^{\frac{1}{\log_2 n}} = 2^{\frac{\log n}{\log_2 n}} = 2^{\log n} = 2^n \geq 2$$

$$3) f(n) = n! \underset{\sim}{\sim} O(?)$$



$n!$  =  $O(n^n)$ ? (weak vs forse)

$$n! = n(n-1)(n-2) \dots 1 < \underbrace{n \cdot n \cdot n \cdots n}_{n} = n^n$$

$$n! < 1 \cdot n^n, n > 1$$

$\therefore f(n) = n!$  is  $O(n^n)$

$= n!$  is  $\text{Not } \Omega(n^n)$

$= n!$  is  $\Omega(n)$  ✓

$= n!$  is  $\Omega(n^2)$  ✓

$= n!$  is  $\Omega(2^n)$  ✓

Is  $n!$   $\Omega(n^n)$ ?

$$\overline{n(n-1)(n-2) \dots \overline{(n \cdot n \cdot n \cdots n)}} \times$$

Forse Lower ~~Bound~~  
Borend



$$f(n) = \log n! \text{ is } O(n \cdot \log n)$$

$$n! < n^n$$

$$\begin{aligned} \log n! &< \log n^n \\ &< n \cdot \log n \end{aligned}$$

$$4) f(n) = \sum_{i=1}^n i^3 = \left[ \frac{n(n+1)}{2} \right]^2 = O(n^4)$$

$$5) f(n) = \sum_{i=1}^n 1 = O(1)$$

$$6) f(n) = \sum_{i=1}^n i = O(n^2)$$

①  $f(n) = \sum_{i=1}^n 1 = O(n) \therefore O(n)$

$$= \frac{1+1+\dots+1}{n} = n$$

②  $f(n) = \sum_{i=1}^n i = O(n^2)$

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

③  $f(n) = \sum_{i=1}^n i^2 = O(n^3)$

$$= \frac{n(n+1)(2n+1)}{6}$$

## Mathematical Background Needed for Analysis of Algo's:

### 1) Monotonicity:

A function  $f(m)$  is monotonically increasing iff  $m \leq n$

Implies  $\underline{f(m) \leq f(n)}$

$\rightarrow$  Strictly Mono. inc :  $m < n \Rightarrow f(m) < f(n)$

### 2) Floors & Ceils:

Floor :  $\lfloor x \rfloor$  = greatest integer less than or equal to  $x$

Ceil :  $\lceil x \rceil$  = smallest " greater " " " "  $x$

for all real  $x$ ,  $(x-1) < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$

Exponentials:for all real  $a > 0, m, n$ :

$$\hat{a} = 1$$

$$a^1 = a$$

$$\bar{a}^1 = 1/a$$

$$(a^m)^n = a^{mn}$$

$$(a^m)^n = (a^n)^m$$

$$a^m \cdot a^n = a^{m+n}$$

Logarithms:

$$\log n = \lg n = \log_2 n \text{ (Binary)}$$

$$\ln n = \log_e n = (\text{Natural Log})$$

$$\log^k n = (\log n)^k = \text{Exponentiation}$$

$$\log \log n = \log(\log n) = [\text{Composition}]$$

for all real  $a > 0, b > 0, c > 0$  and  $n$

$$a = b^{\log_b a} \quad \text{--- (1)}$$

$$\log_c ab = \log_c a + \log_c b \quad \text{--- (2)}$$

$$\log_b a^n = n \cdot \log_b a \quad \text{--- (3)}$$

$$\log_b a = \frac{\log_c a}{\log_c b} \quad \text{--- (4)}$$

$$\log_b \frac{1}{a} = -\log_b a \quad \text{--- (5)}$$

$$\log_b a = \frac{1}{\log_a b} \quad \text{--- (6)}$$

$$a^{\log_c b} = b^{\log_c a} \quad \text{--- (7)}$$

Iterated Logarithm:

$\log_n^{(i)} = \log$  applied ' $i$ ' times in succession starting on argument ' $n$ '

$$\log_4^{(2)} 4 \stackrel{?}{=} 1 \quad \log \log_2 4 = \log 2 \stackrel{?}{=} 1$$

Constants:  $\pi$ ,  $e$ ,  $\sqrt{2}$ ,  $\sqrt{3}$ ,  $\log_2^2$ ,  $\log_{10}e$ ,  $\log_e^2$ , Eulers Const.,  
 $\sqrt{e}$ ,  $\sqrt{\pi}$ ,  $\sqrt{ }$

→ Stirling's Approximation

$$\hookrightarrow \left[ n \log_2^n - n + \frac{1}{2} \log_2^n + O(1) \right]$$

$$\hookrightarrow O(n \log n)$$

$$\rightarrow \phi = \frac{\sqrt{5} + 1}{2} = 1.618 \text{ (Golden ratio)}$$

→ The ratio of consecutive Fibonacci no.s approaches Golden ratio;

$$\frac{F_{i+1}}{F_i}$$

→ Binomial Formulas

$$(i) \sum_{m=0}^n nC_m = 2^n ; \quad (ii) nC_m < n^m$$

$$\rightarrow \sqrt[n]{a/b} = \sqrt[n]{a} / \sqrt[n]{b}$$

→ Gamma Functions

→ Series of Const's

→ Taylor Series

→ Prob. distribution

→ Prob. Theory

→ Matrix Manipulations

→ Splitting Summations

→ Approximation Integrals

→ Differentiation

→  $\sum f(n)$

$\hat{n} = m$

$$\sim \sum_{n=1}^m f(n) \cdot dn \leq \sum f(n) \leq \int_m^{m+1} f(n) \cdot dn$$



## Commonly occurring Asymptotic orders:

$$f(n) = \sum_{i=1}^n i^d = \Theta(n^{d+1})$$

$$\sum_{k=0}^n x^k = \frac{x^{n+1}-1}{x-1} \quad x \neq 1$$

$$f(n) = \sum_{i=1}^n n^i = \Theta(n^n)$$

$$\ln|x| < 1 = \frac{1}{1-x}$$

=====

$$f(n) = \sum_{i=1}^n \log i = \Theta(n \log n)$$

$$f(n) = \sum_{i=1}^n i^d \cdot \log i = \Theta(n^{d+1} \cdot \log n)$$

$$\text{Q1) } \sum_{i=1}^n i^3 = x \quad \text{MSQ} = \left( \frac{n(n+1)}{2} \right)^2 \in O(n^4) \subset \Omega(n^4) \therefore \Theta(n^4)$$

what are the correct  
choices for ' $x$ '

I.  $\Theta(n^4)$  ✓

II.  $\Theta(n^5)$  ✗

III.  $\Theta(n^5)$  ✓

IV.  $\Sigma(n^3)$  ✓

$\langle \text{I}, \text{III}, \text{IV} \rangle$

$$f(n) = n \sum_{i=1}^n \Theta(i) \stackrel{?}{=} \Theta(n^2)$$

$$f \leq cg \quad \begin{cases} n < 5 \cdot n, n > 1 \\ n < 2 \cdot \frac{n^2}{2}, n > 1 \end{cases}$$

Q2)  $f(n) = \sum_{i=1}^n i^{1/2} \sim \int_1^n \sqrt{i} \cdot di = O(n\sqrt{n})$

Prove it

Hint: use Integration

$$f(n) = \sum_{i=1}^n 2^i = \underline{\underline{2^{n+1}-2}} = O(2^n) \checkmark$$

$S_n = \frac{a(r^n - 1)}{r-1} = \frac{2(2^n - 1)}{2-1} = \underline{\underline{2^{n+1}-2}}$

$$f(n) = \sum_{i=1}^n i \cdot 2^i = \underline{\underline{(n-1) \cdot 2^{n+1} + 2}} = O(n \cdot 2^n)$$

$$f(n) = \sum_{i=1}^n 1/2^i = \left(1 - \frac{1}{2^n}\right) = O(1) \checkmark$$

$$f(n) = \sum_{x=1}^n 1/x \approx \int_1^n 1/x \cdot dx = \left[\log x\right]_1^n = \underline{\underline{\log n}} = O(\log n) \checkmark$$

## Little / Small Notations:

→ The bounds provided by Asymptotically Tight;  
 $f(n) \leq g(n)$

Big Notations, may or may not be

$$f(n) = \underline{\underline{O(n)}} : \text{Tight Bound}$$

$$\underline{\underline{O(n^2)}} : \text{Loose Bound}$$

NOT tight

→ The Bounds Provided by the Small Notations is always NOT tight (Loose Bound)

4) Small-oh ( $\circ$ ): Proper U.B:

$f(n) \in \circ(g(n))$ , iff  $\begin{cases} f(n) < c \cdot g(n) \\ \text{whenever } n \geq K \quad (K > 0) \end{cases}$ , for all  $c > 0$

$$f(n) = n \underset{\substack{\circ(n) \\ \times}}{\overset{\circ(n)}{\longrightarrow}} \circ(n^2) \circ(n \log n)$$

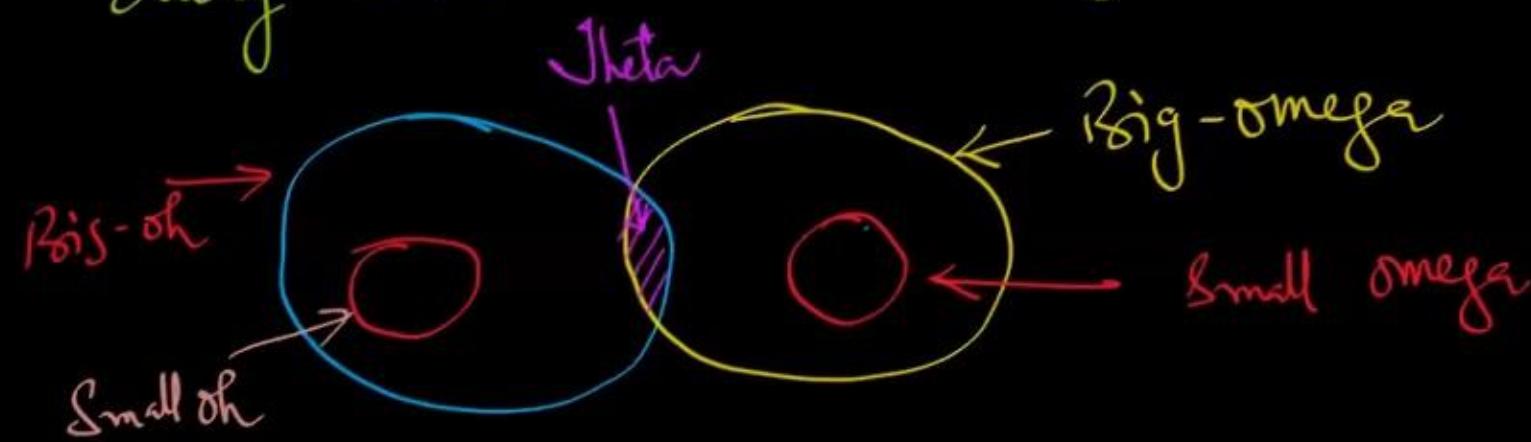
$$n < n \cdot \log n, \quad n > 1$$

5) Small-omega ( $\omega$ ): Proper L.B.

$f(n)$  is  $\omega(g(n))$  iff  $f(n) > c \cdot g(n)$ , for all  $c > 0$   
whenever  $n > k$

St1: Every big- $\Theta$  is also small- $\Theta$   $\times$   $(f \leq g) \mid (f \leq g)$

$\text{St}_2$ : Every Small-oh is also Big-oh ✓



## Analogy between ASN and Real No's:

Let  $f, g$ : functions ;  $a, b$ : real no's

$$\begin{array}{c} \text{Analogies} \\ \text{between} \\ \text{ASN} \text{ and} \\ \text{Real No's:} \end{array}$$

$$f = g$$

- 1)  $f(n) \in O(g(n)) \iff a \leq b$
- 2)  $f(n) \in \Omega(g(n)) \iff a \geq b$
- 3)  $f(n) \in \Theta(g(n)) \iff a = b$
- 4)  $f(n) \in o(g(n)) \iff a < b$
- 5)  $f(n) \in \omega(g(n)) \iff a > b$

$$\underline{n \cdot \log n}; \underline{n \sqrt{n}}$$

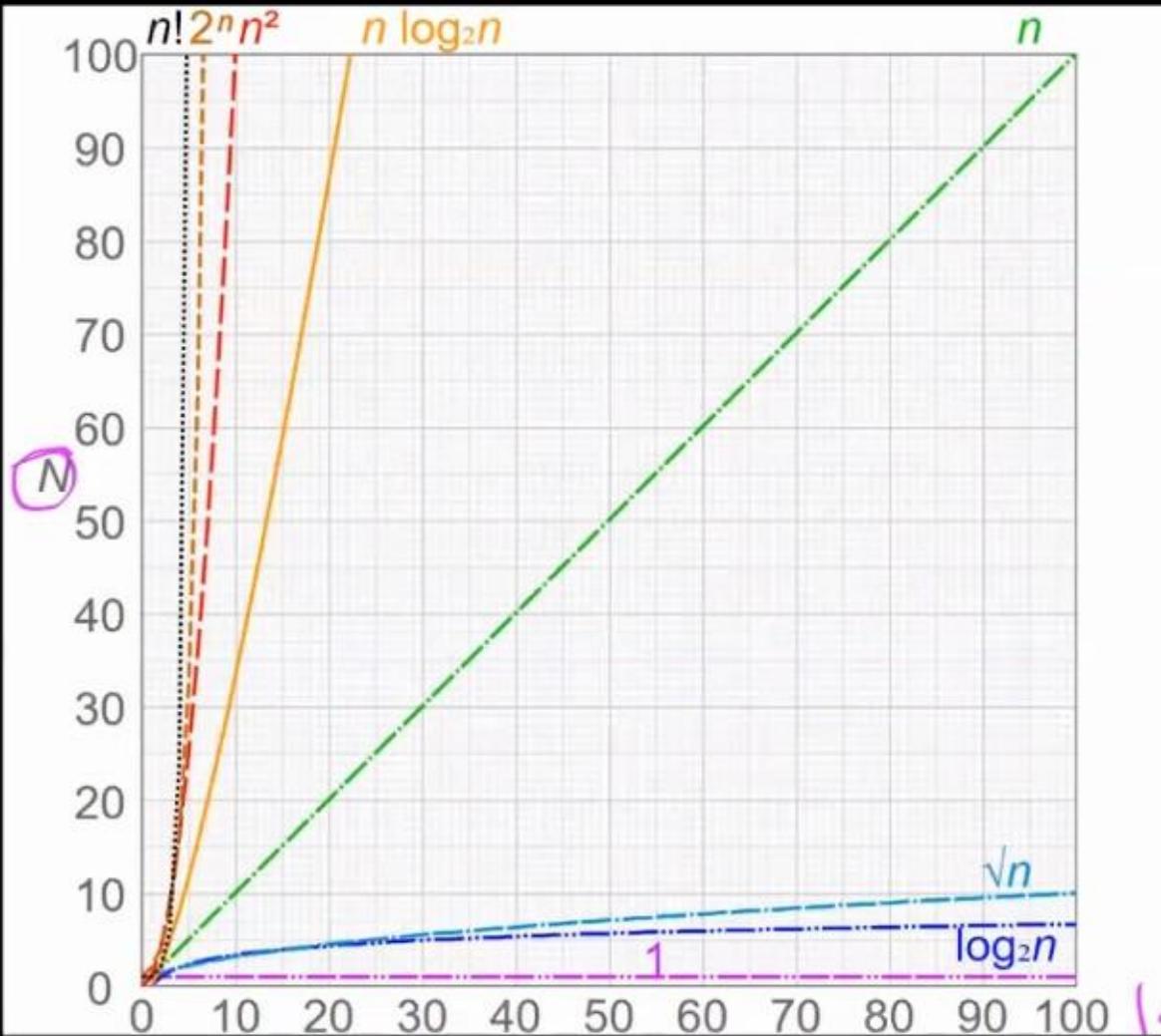
$$\underline{\log n} \leq \underline{\sum n}$$

$$\left\{ \frac{1}{n} < \log_2 n \right\}$$

$$n^2 < 2^n \rightarrow n^2 < 2^n, \quad \underline{n \geq k=4}$$

| $n$ | $n^2$ | $2^n$            |
|-----|-------|------------------|
| 1   | 1     | $\frac{2}{4} <$  |
| 2   | 4     | $=$              |
| 3   | 9     | $> \frac{8}{16}$ |
| 4   | 16    | 16               |
| 5   | 25    | 32               |
| 6   | 36    | 64               |
| 7   | 49    | 128              |





Wiki

$$\text{Cons} = O(\log)$$

$$\log s = O(\text{Poly})$$

$$\text{Poly} = O(\log n)$$

$$\underline{1 < \log n < \sqrt{n} < n < n \log n < n^2 < 2^n < n!}$$

## Dominant Relation

(Constants < Logarithm < Polynomial  
< Exponentials)

AS N:

Big      Small

$$\rightarrow \left. \begin{array}{c} O : \text{UB} \\ \Omega : \text{LB} \end{array} \right\} \rightarrow \left. \begin{array}{c} o : \text{UB} \\ \omega : \text{LB} \end{array} \right\} \left. \begin{array}{c} f(x) < c \cdot g(x) : o \\ > c \cdot g(x) : \omega \end{array} \right\}$$

— Theta : JB

$$(f(x)) \leq c(g(x)) : 'o'$$

f is O(g)

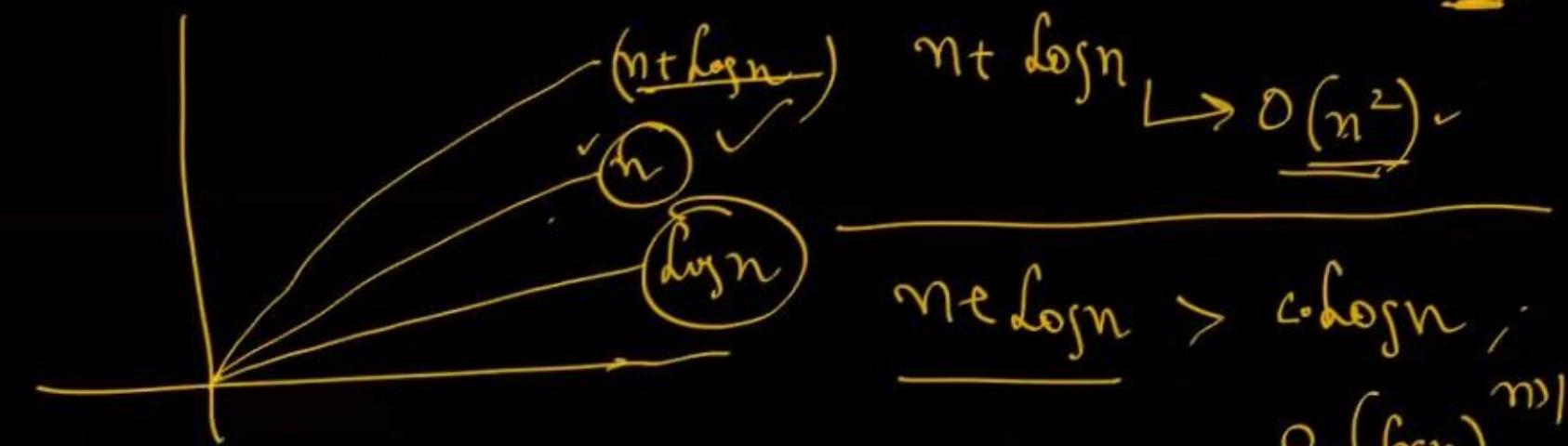
$$\begin{aligned} f: O(\delta) : a \leq b \\ f: \Omega(\delta) : a \geq b \\ f: \Theta(\delta) : a \approx b \\ f: o(\delta) : a < b \\ f: \omega(\delta) : a > b \end{aligned}$$



$$f(n) = n + \log n \rightarrow O(n)$$

$$n + \log n < n + n, \quad n > 1$$

$$\frac{n + \log n}{2^n} < \frac{2n}{2^n} \rightarrow O(n)$$

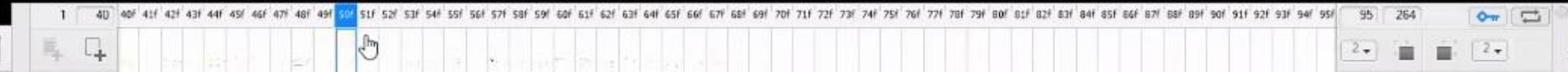


$$\underline{n + \log n} > \underline{\log n},$$

$$\Omega(\log n)^{n!}$$

$$n + \log n > n, \quad n > 1$$

$$\Omega(n)$$





## Properties of Asymptotic Notations:

### 1. General:

1. If  $f(n)$  is  $\mathcal{O}(g(n))$  then  $a \cdot f(n)$  is  $\mathcal{O}(g(n))$   $\quad [a > 0]$

$$\begin{aligned} \text{Ex: } f(n) &= n = \mathcal{O}(n) \\ &= 100n + \mathcal{O}(n) \quad [100n < \underline{200n}] \end{aligned}$$

2. If  $f(n)$  is  $\mathcal{O}(g(n))$  and  $d(n)$  is  $\mathcal{O}(e(n))$  then

$$\left. \begin{aligned} f &= n^2 - \mathcal{O}(n^2) \\ d &= n^3 - \mathcal{O}(n^3) \\ n^2 + n^3 &= \mathcal{O}(n^3) \end{aligned} \right\}$$

- a)  $f(n) + d(n) = \mathcal{O}(\max(g(n), e(n)))$
- b)  $f(n) * d(n) = \mathcal{O}(g(n) * e(n))$ ;



$$3) \log_2 n^x = x \cdot \log n = O(\log n) \quad (x > 0)$$

↑

$\log^x n = O(n^y) \checkmark \quad (x > 0, y > 0) \quad [x \cdot \log \underline{\log n} ; y \cdot \log \underline{n}]$

$n^x = O(a^n) \quad (x > 0; a > 1) \quad [x \cdot \log \underline{a} ; n \cdot \log \underline{2}]$

5

↳ Logarithmic function is in  $O(\text{Poly})$

↳ Poly are in  $O(\text{Exp})$

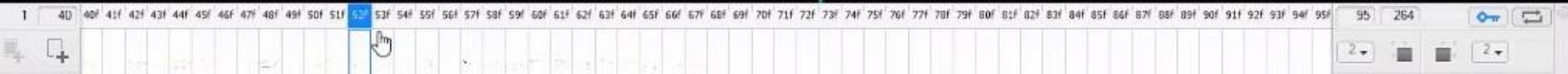
Dominance Relation:

Const < Log < Poly < Exp

$C < \log n < n \log n < n^2 < n^3 < 2^n < n^n$

$$f: (100)^{1000} \quad \text{Logn: g}$$

$$\sqrt{n}$$



Discrete Properties :

Notations

| Property              | $O$  | $\Omega$ | $\Theta$ | $o$   | $\omega$ |
|-----------------------|--|----------|----------|---|----------|
| 1) Reflexive          | ✓  | ✓        | ✓        | ✗   | ✗        |
| 2) Symmetric          | ✗  | ✗        | ✓        | ✗   | ✗        |
| 3) Transitive         | ✓  | ✓        | ✓        | ✓   | ✓        |
| 4) Transpose Symmetry | $f(n) \in O(g(n))$<br>then $g(n) \in \Omega(f(n))$ |          |          | $f(n) \in o(g(n))$<br>then<br>$g(n) \in \omega(f(n))$ |          |

$$a \leq b$$

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

$$n^2 \text{ is NOT } O(n)$$

$$\underline{n = O(n)}$$

$a \leq b$   
 $\Rightarrow b \geq a$

Transpose  
Symmetry



Incidence Property: (applies to real no's)

$a, b : \text{real}$

$\begin{cases} a < b \\ a > b \\ a = b \end{cases}$

always  
True

If  $f, g : \text{functions}$ :

$f < g : O, o$

$f > g : \Omega, \omega$

$f \asymp g : \Theta$

$\left\{ \begin{array}{l} f \text{ is } O(\delta) ; o(\delta) \\ f \text{ is } \Omega(\delta) : \omega(\delta) \\ f \text{ is } \Theta(\delta); \end{array} \right.$

1)  $f(n) = n; g(n) = n^2 \Rightarrow f \text{ is } O(g) \checkmark$   
 $f \text{ is } o(g) \checkmark$

2)  $f(n) = \log n; g(n) = 1/n$   
 $g(n) \text{ is } O(f(n)) \checkmark$

3)  $f(n) = n; g(n) = n^{1+\sin(n)}$



$$f(n) = \underline{n}; \quad g(n) = \overline{n^{1+\sin(n)}}$$

$$= n \quad \quad \quad = \overline{n^0} = 1$$

 $n > 1$ 

$$= n < \underline{n} = \overline{n}$$

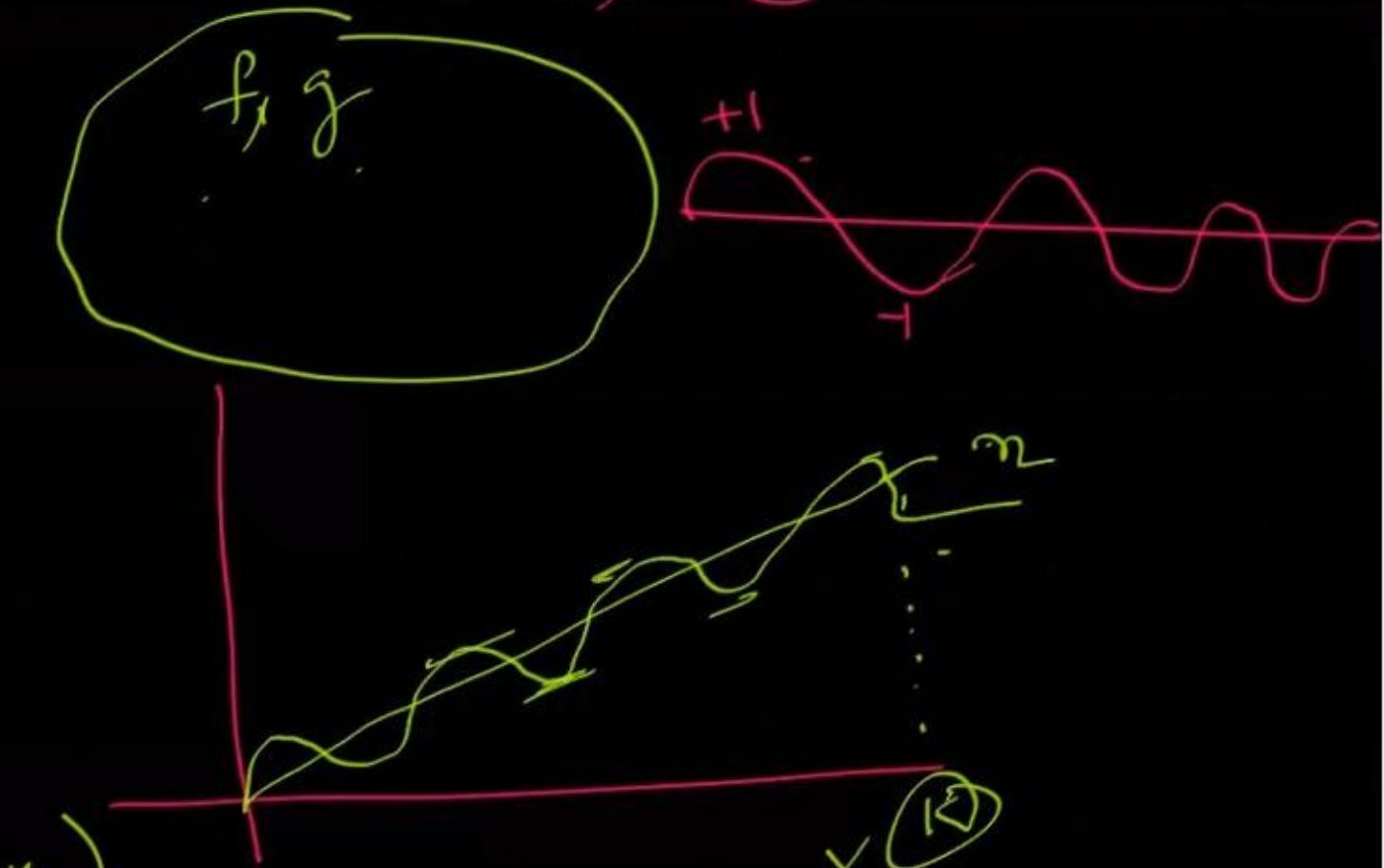
$$f > g$$

$$f < g$$

(ASNs  
may not  
guarantee  
Trichotomy  
Property always)

$$\min(\sin) : (-1)$$

$$\max(\sin) : (+1)$$



(www.KhaleelKhamb.info) → Resources



→ In general, Smaller functions are in the order of Bigger fns  
 → " " , Bigger " " " " Omega of Smaller fns  
 $f_1 \in \Theta(g)$  ;  $f_2 \in \Omega(g)$

---

$$\begin{aligned}
 n^2 &\leq n^3 \\
 (2 \cdot \log n) &\quad (3 \cdot \log n) \\
 \downarrow &\quad \downarrow \\
 O(1) &= O(1) \\
 \therefore n^2 &= n^3
 \end{aligned}$$

{ (i) Def'n of ASN's,  
 (ii) Properties of ASN's  
 (iii) (Ability to decide  
 which function is  
 large / small than the  
 other)



## 02. Asymptotic Comparisons

01.  $f(n) = n, g(n) = \log n$

02.  $f(n) = n^2 \log n, g(n) = n \cdot \log^{10} n$

03.  $f(n) = n^3, 0 < n \leq 10,000$   
 $= n, n > 10,000$

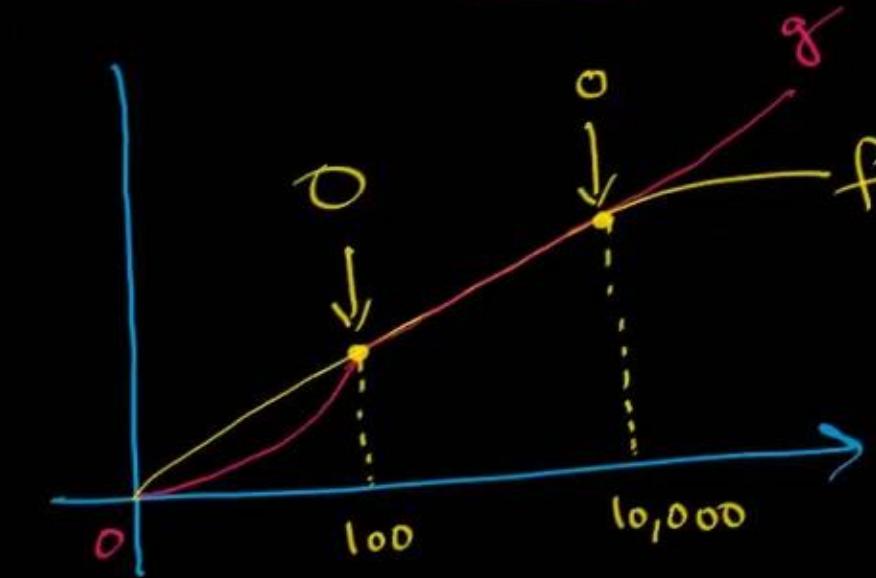
$g(n) = n, 0 < n \leq 100$   
 $= n^3, n > 100$

1)  $f$  is  $\mathcal{O}(g)$ :  $n > K$   
 $\hookrightarrow$  smallest integer 'K'  
 $\frac{10,000}{100}$

2)  $f$  is  $\mathcal{O}(g)$ :  $n > K$   
 $\hookrightarrow 10,000$

$f < g$

$\left\{ \begin{array}{l} f(n) = n^3, \quad , \quad n > 100, \leq 10,000 \\ \quad \quad \quad = n, \quad , \quad n > 10,000 \\ \\ g(n) = n^3, \quad , \quad " \\ \quad \quad \quad = n^3, \quad , \quad n > 1000 \end{array} \right.$



04. Arrange the functions in increasing order of rates of growth.

01.  $n^2; n \log n; n\sqrt{n}; e^n; n; 2^n; (1/n)$

02.  $2^n; n^{3/2}; n \log n; n^{\log n}$

03.  $n^{(1/3)}; e^n; n^{7/4}; n \log^9 n; 1.001^n$

$$\begin{array}{c} n \log n \\ n \\ \hline \log n \cdot \log n \\ \underline{\underline{(Logn)^2 < n}} \\ n \cdot \log 2 \end{array}$$

Q 1)  $\frac{n^2}{P}; \frac{n \log n}{P}; \frac{n\sqrt{n}}{P}; \frac{e^n}{P}; \frac{n}{P}; \frac{2^n}{P}; \frac{1/n}{P}$

$$\left[ \frac{1}{n} < n < n \log n < n\sqrt{n} < n^2 = 2^n < e^n \right] \checkmark$$

Q 2)  $\frac{2^n}{P}; \frac{n^{3/2}}{P}; \frac{n \log n}{P}; \frac{n^{\log n}}{P}$

$$(n \log n < n^{3/2} < n^{\log n} < 2^n) \checkmark$$

$$\begin{aligned} n^{3/2} &= n^{1.5} \\ n &= n \\ &= \underline{\underline{n\sqrt{n}}} \end{aligned}$$

Q 3)  $n^{1/3}; e^n; n^{7/4}; n \log n; 1.001^n$



$$3) n^{1/3}; e^n; n^{7/4}; n \log n; 1.001^n$$

$$a) n \log n < n^{7/4} < n^{1/3} < 1.001^n < e^n$$

$$\checkmark c) n^{1/3} < n \log n < n^{7/4} < 1.001^n < e^n$$

$$b) n^{1/3} < n < n \log n < 1.001^n < e^n$$

d)

$$\log^x n \quad (\log n)^x$$

$$f(n) = n$$

$$g(n) = n^{1/3}$$

$$\sqrt{f(n)} = n^{7/4} \\ = n^{1.75}$$

$$\sqrt{g(n)} = n \sqrt{n} \\ = n^{1.5}$$

$$\checkmark h(n) = n \cdot \log n$$

$$(n \log n < n \sqrt{n} < n^{7/4})$$

$$\underline{n^{1/3} < n \cdot \log n < n^{7/4} < (1.001)^n < e^n}$$

$$\sqrt{n} > \log n$$



03. Two Packages are available for processing a Data Base having  $10^n$  records.

Package A takes  $\alpha$  times of  $10 \cdot n \cdot \log_{10} n$  while package B takes a time of  $0.0001n^2$  for processing  $n$  records. Determine the smallest integer  $\alpha$  for which Package 'A' outperforms Package 'B'.

$$\frac{10 \cdot n \cdot \log_{10} n}{n^2} > 0.0001$$

$$A \rightarrow 10 \cdot n \cdot \log_{10} n = O(n \log n)$$

$$B \rightarrow 0.0001n^2 = O(n^2)$$

$$A: 10 \cdot n \cdot \log_{10} n$$

$$B: 0.0001n^2$$

$$10^\alpha = n$$

$$\alpha = 1$$

$$1) A \rightarrow 10 \cdot 10 \cdot \log_{10} 10 \\ \rightarrow 100 \text{ units}$$

$$2) B \rightarrow 10^{-4} \times 10^2 = 10^{-2} = \frac{1}{100} \text{ units}$$

$$A \rightarrow 10 \cdot 10^2 \cdot \log_{10} 10^2 \\ \rightarrow 2000 \text{ units}$$

$$B \rightarrow 10^{-4} \times (10^2)^2$$

$$\Rightarrow 1 \text{ unit}$$



03. Two Packages are available for processing a Data Base having  $10^x$  records.

Package A takes  $a$  times of  $10 \cdot n \cdot \log_{10} n$  while package B takes a time of  $0.0001n^2$  for processing  $n$  records. Determine the smallest integer  $x$  for which Package 'A' outperforms Package 'B'.

$$\begin{aligned} A &: 10 \cdot n \cdot \log_{10} n \\ B &: 0.0001n^2 \end{aligned}$$

$\Rightarrow$

$$\begin{aligned} n &= 10^x \\ \text{Case 1: } n &= 10 \\ 1) A &\rightarrow 10 \cdot 10 \cdot \log_{10} 10 \\ &\rightarrow 100 \text{ units} \\ 2) B &\rightarrow 10^4 \times 10^2 = 10^6 = \frac{1}{100} \text{ units} \end{aligned}$$

$\Rightarrow 100 \text{ units} < \frac{1}{100} \text{ units}$

$$x = 1$$

$$n = 10^x$$

$$\underline{n = 10}$$

$$\underline{A < B}$$

$$\begin{array}{c} \rightarrow n = 10^x \\ \log_{10} \end{array}$$

$$\underline{n = 10^x}$$

$$10 \cdot n \cdot \log_{10} n < 10^4 \times n^2$$

$$10 \cdot 10^x \cdot \log_{10} 10^x < 10^4 \times 10^2$$

$$10 \cdot 10^x < \frac{10^4}{10^2}$$

$$\boxed{x < \frac{10^4}{10^2}}$$

$$\underline{5 < 10^2}$$

$$6 < 10^2 \checkmark$$

$$\begin{array}{c} \circlearrowleft \\ x = 6 \\ \text{Smallest} \end{array}$$

$$6 < \frac{10^6}{10^2}$$



Q) Let  $W(n)$  &  $A(n)$  denote respectively, the Worst Case and Avg. Case Running Time of an Algorithm on IP size ' $n$ '. Which of the following is always True?

a)  $A(n) = o(W(n))$

b)  $A(n) = \Theta(W(n))$

c)  $A(n) = \Omega(W(n))$  something

d)  $\underline{A(n) = O(W(n))}$ : Always True

$\checkmark \quad \boxed{B(n) \leq A(n) \leq W(n)}$   $\Rightarrow$

$\underline{A(n) = \omega(W(n))}$  is Always False

$\boxed{A(n) > W(n)}$

$\checkmark \quad \left\{ \begin{array}{l} B(n) \in O(A(n)) \\ A(n) \in O(W(n)) \end{array} \right\} \quad \left\{ \begin{array}{l} W(n) \in \Omega(A(n)) \\ A(n) \in \Omega(B(n)) \end{array} \right\} \quad \checkmark$



# Framework for Analysis (Time) of Non-Recursive & Recursive Algorithms:

## I. General Problem Solving :

07. An element in an Array is called Leader if it is greater than all elements to the right of it. The Time Complexity of the most efficient Algorithm to print all Leaders of the given Array of size 'n' is  $O(n)$

|                                     |      |    |    |    |    |   |    |    |   |   |    |
|-------------------------------------|------|----|----|----|----|---|----|----|---|---|----|
| $\{ \underline{\underline{L-R}} \}$ | A: [ | 1  | 2  | 3  | 4  | 5 | 6  | 7  | 8 | 9 | 10 |
|                                     |      | 20 | 25 | 18 | 20 | 9 | 15 | 12 | 7 | 8 | 5  |
|                                     |      | L  | L  | L  | L  | L | L  | L  | L | L | L  |

Algo Leader (A, n)

```

for i ← 1 to (n-1)
    for j ← (i+1) to n
        if (A[i] < A[j])
            break;
    }
}
  
```

if ( $j = (n+1)$ ) then  
print ("A[i]");

} \*

metric c: (Comparison)

- a)  $n \log n$
- b)  $n$
- c)  $n^2$
- d)  $\log n$

## Brute-force (enumeration)

$$\begin{cases} L=1 & i=2 \\ \dots & \dots \end{cases}$$

$$\begin{cases} (n-1) & + (n-2) + (n-3) + \\ & \dots + 1 \end{cases}$$

$$(1+2+3+\dots+n-1)$$

$$= \frac{n(n-1)}{2} = O(n^2)$$

A:  $\begin{bmatrix} 1 & 2 & 3 & 7 & 5 & 6 & 3 & 8 & 9 & 10 \\ 20 & 25 & 18 & 20 & 9 & 15 & 12 & 7 & 8 & 5 \end{bmatrix}$

Perf. Linear Search from R-L

Algo Leader<sub>2</sub> (A, n)

: Time Complexity =  $O(n)$



```

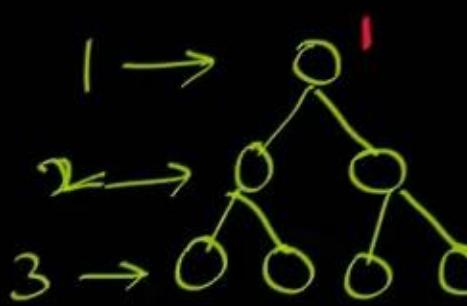
1. L <- A[n]; : l : O(1)
2. for i <- (n-1) down to 1 (n-1)
   {
      if (A[i] > L)
      {
         print(A[i]);
         L = A[i];
      }
   }
}

```

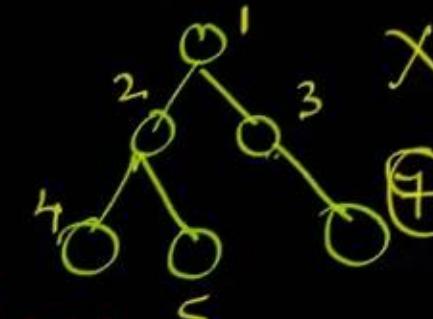


Binary Tree:

Full BT

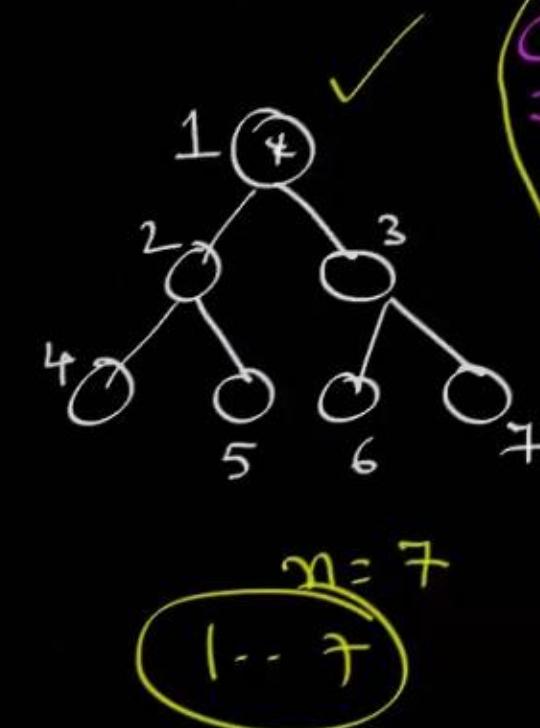


Complete BT

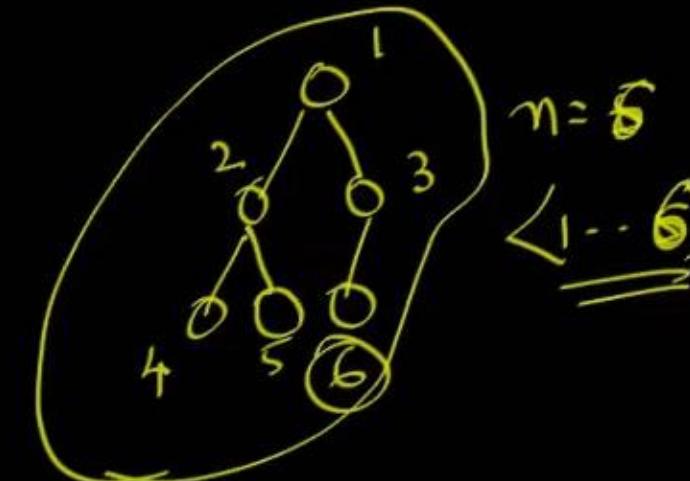


Level order Indexing:

Repr. of tree in an array

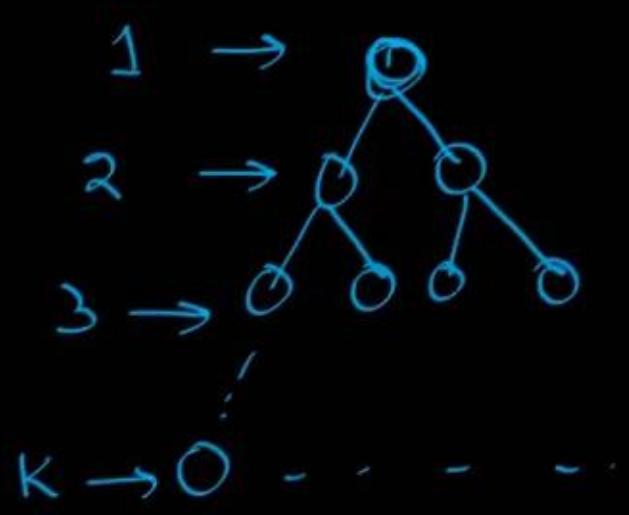


CBT: A BT is Complete iff the level order indices of the nodes should lie in the interval  $1 \dots n$  where  $n =$  no. of nodes in the tree



→ Max. No. of Nodes  
at level 'i' =  $2^{i-1}$

→ Height of Full / Compl. BT having 'n' elements :



$$n = \sum_{i=1}^k 2^{i-1} = \frac{1}{2} \sum_{i=1}^k 2^i$$

$$= \frac{1}{2} [2^{k+1} - 2]$$

$$\boxed{n = 2^k - 1}$$

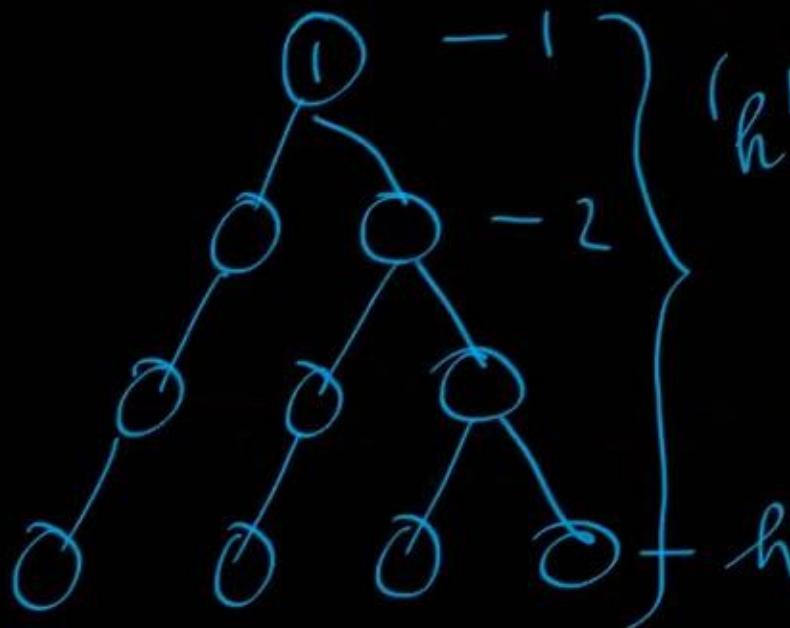
$$\left\{ \begin{array}{l} k = O(\log n) \\ \hline \end{array} \right.$$

$$\therefore (n+1) = 2^k$$

$$\therefore k = \log_2(n+1)$$



GQ → Every level 'i' has 'i' nodes, starting root @ level 1



$n = \text{no. of nodes}$

$$n \geq \sum_{i=1}^h i = \frac{h(h+1)}{2}$$

(Asym Bound)

$$\left\{ \begin{array}{l} 2n = h^2 + h \\ \hline \end{array} \right\}$$

$$h^2 \sim n$$

$$h \sim \sqrt{n}$$

$\therefore h \in O(\sqrt{n})$

$$\frac{n^2 + n + 1}{2} \in O(n^2)$$

## 11. Algorithm What(n)

```
{
  if (n = 1) return;
  else
  {
    What (n - 1);
    B(n);
  }
}
```

Let  $T(n)$  repr the Time Complexity of  $\text{What}(n)$ ;

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

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

$$\boxed{T(n) = T(n-1) + d} \quad n > 1$$

$$d = a+b$$

$$(i) \underline{\underline{B(n)}} : O(1)$$

$$\cancel{\cancel{O(n)}}$$

Substitution Method:

$$T(n) = T(n-1) + d \quad -\textcircled{1}$$

$$T(n-1) = T(n-2) + d \quad -\textcircled{2}$$

$$T(n) = T(n-2) + 2d \quad -\textcircled{3}$$

$$= T(n-3) + 3d \quad -\textcircled{4}$$

$$= T(n-4) + 4d$$

$$= T(n-k) + k \cdot d \quad -\textcircled{5}$$

Time Comp- g Recursive Algo's is obtained by defining a Mathematical Recurrence Equation;

$$n-k=1$$

$$\Rightarrow k = n-1$$

∴

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

$$\boxed{T(n) = (C + dn - d)}$$

$$\therefore \underline{\underline{\Theta(n)}} \quad \underline{\underline{\Sigma(n)}}$$

$$\therefore T(n) \in \Theta(n) \quad \checkmark$$

## 11. Algorithm What(n)

```
{
  if (n = 1) return;
  else
  {
    What (n - 1);
    B(n);
  }
}
```

$$1) T(n) = O(n) \Rightarrow B(n) : \underline{\underline{O(1)}}$$

Assignm<sup>t</sup> :

$$\begin{aligned} T(n) &= ? && \text{if } \underline{\underline{B(n)}} = O(n) \quad \checkmark \\ &= ? && \text{if } \underline{\underline{B(n)}} = O(1/n) \quad \checkmark \end{aligned}$$



Algorithms Handout 2021.pdf - Adobe Acrobat Reader DC

File Edit View Window Help Tools Fill & Sign Comment Sign In

# Algorithms

## I. Asymptotic Notations & Apriori Analysis

For Micro Notes by the Student

01. State True / False

- 1.  $100n \cdot \log n = O(n \cdot \log n)$  : ✓
- 2.  $2^{n+1} = O(2^n)$  : ✓
- 3.  $2^{2n} = O(2^n)$  : ✗
- 4.  $O(x < y)$  then  $n^x = O(n^y)$  : ✓
- 5.  $(n+k)^m \neq \Theta(n^m)$  ( $k, m > 0$ )
- 6.  $\sqrt{\log n} = O(\log \log n)$
- 7.  $\log(n)$  is  $\Omega(1/n)$
- 8.  $2^{n^2}$  is  $O(n!)$
- 9.  $n^2$  is  $O(2^{2\log n})$
- 10.  $a^n \neq O(n^x)$ ,  $a > 1$ ,  $x > 0$
- 11.  $2^{\log_2 n^2}$  is  $O(n^2)$

$2^{n+1} = 2 \cdot 2^n = O(2^n)$

$2^n = (2^2)^n = \underbrace{4^n}_{2^n}$

$0 < x < y \Rightarrow n^x \underset{\approx}{=} n^y$  is  $O(n^y)$  ✓

Export PDF  
Adobe ExportPDF  
Convert PDF files to Word or Excel online.  
Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB  
Convert To:  
Microsoft Word (\*.docx)  
Recognize Text in English(U.S.)  
Change  
Convert  
Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

# Algorithms

## I. Asymptotic Notations & Apriori Analysis

For Micro Notes by the Student



### 01. State True / False

- 1.  $100n \cdot \log n = O(n \cdot \log n)$  : ✓
- 2.  $2^{n+1} = O(2^n)$  : ✓
- 3.  $2^{2n} = O(2^n)$  : ✗
- 4.  $O(x) < y$  then  $n^x = O(n^y)$  : ✓
- 5.  $(n+k)^m \neq \Theta(n^m)$  ( $k, m > 0$ ) : ✗
- 6.  $\sqrt{\log n} = O(\log \log n)$
- 7.  $\log(n)$  is  $\Omega(1/n)$
- 8.  $2^{n^2}$  is  $O(n!)$
- 9.  $n^2$  is  $O(2^{\log n})$
- 10.  $a^n \neq O(n^x)$ ,  $a > 1$ ,  $x > 0$
- 11.  $2^{\log_2 n^2}$  is  $O(n^2)$

$$2^{n+1} = 2 \cdot 2^n = O(2^n)$$

$$(n+k)^m = \underbrace{n^m + k n^{m-1} + \dots + k^{m-1} n + k^m}_{\Theta(n^m)} = \Theta(n^m)$$

$$(n+2)^2 = \frac{n^2 + 4n + 4}{n^2} = \Theta(n^2)$$

$$\Theta(n^2) \cdot \Theta(n^2) = \Theta(n^4)$$

### 02. Asymptotic Comparisons

- 01.  $f(n) = n$ ,  $g(n) = \log n$

Export PDF

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
 Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
 Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
 Change

Convert

► Create PDF  
► Edit PDF  
► Combine PDF  
► Send Files  
► Store Files

Algorithms Handout 2021.pdf - Adobe Acrobat Reader DC

File Edit View Window Help Tools Fill & Sign Comment Sign In

# Algorithms

## I. Asymptotic Notations & Apriori Analysis

For Micro Notes by the Student

1. 100n.logn = O(n.logn) : ✓

2.  $2^{n+1} = O(2^n)$  : ✓

3.  $2^{2n} = O(2^n)$  : ✗

4. O<x<y then  $n^x = O(n^y)$  : ✓

5.  $(n+k)^m \neq \theta(n^m)$  ( $k, m > 0$ ) : ✗

6.  $\sqrt{\log n} = O(\log\log n)$  : ✗

7.  $\log(n)$  is  $\Omega(1/n)$  : ✓

8.  $2^{n^2}$  is  $O(n!)$

9.  $n^2$  is  $O(2^{\log n})$

10.  $a^n \neq O(n^x)$ ,  $a > 1$ ,  $x > 0$

11.  $2^{\log_2 n^2}$  is  $O(n^2)$

$2^{n+1} = 2 \cdot 2^n = O(2^n)$

$\frac{1}{2} \log \log n > \log \log \log n$

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

Algorithms Handout 2021.pdf - Adobe Acrobat Reader DC

File Edit View Window Help Tools Fill & Sign Comment Sign In

# Algorithms

## I. Asymptotic Notations & Apriori Analysis

For Micro Notes by the Student

1. State True / False

1.  $100n \cdot \log n = O(n \cdot \log n)$  : ✓
2.  $2^{n+1} = O(2^n)$  : ✓
3.  $2^{2n} = O(2^n)$  : ✗
4.  $O(x < y \text{ then } n^x = O(n^y)}$  : ✓
5.  $(n+k)^m \neq \Theta(n^m)$  ( $k, m > 0$ ) : ✗
6.  $\sqrt{\log n} = O(\log \log n)$  : ✗
7.  $\log(n)$  is  $\Omega(1/n)$  : ✓
8.  $2^{n^2}$  is  $O(n!)$  : ✗
9.  $n^2$  is  $O(2^{2\log n})$  : ✓
10.  $a^n \neq O(n^x)$ ,  $a > 1$ ,  $x > 0$  : ✓
11.  $2^{\log n^2}$  is  $O(n^2)$

$2^{n+1} = 2 \cdot 2^n = O(2^n)$

$2^{2\log n} = 2^{\log n^2} = 2^{\log n^2} = n^2$

$n^2 \cdot \log_2 n < n \cdot \log n$

$n^2 > n \log n$

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

## 02. Asymptotic Comparisons

01.  $f(n) = n, g(n) = \log n \Rightarrow$

$g$  is  $O(f)$  ✓  
 $g$  is  $O(f)$

02.  $f(n) = n^2 \log n, g(n) = n \cdot \log^{10} n$

$$\begin{aligned} n^2 \log n &> n \cdot \log^{10} n \\ (n \cdot \log n) \cdot \underline{n} &= n \cdot \underline{\log^n} \\ \underline{\log n} &> 9 \cdot \underline{\log \log n} \end{aligned}$$

03.  $f(n) = n^3, 0 < n \leq 10,000$

$= n, n > 10,000$

$g(n) = n, 0 < n \leq 100$

$= n^3, n > 100$

03. Two Packages are available for processing a Data Base having  $10^x$  records.

Package A takes a time of  $10 \cdot n \cdot \log n$  while package B takes a time of  $0.0001n^2$  for processing  $B$  records. Determine the smallest integer  $x$  for which Package 'A' outperforms Package 'B'.

Algorithms Handout 2021.pdf Adobe Reader

File Edit View Window Help Tools Fill & Sign Comment Sign In

Open Export PDF

Selected PDF File: Algorithms Handout 2021.pdf 1 file / 537 KB

Convert To: Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change

Convert

Create PDF Edit PDF Combine PDF Send Files Store Files

05. Consider the following functions from positive integers to real numbers:

$10, \sqrt{n}, n, \log_2 n, \frac{100}{n}$ .

The CORRECT arrangement of the above functions in increasing order of asymptotic complexity is:

(a)  $\log_2 n, \frac{100}{n}, 10, \sqrt{n}, n$

(b)  $\frac{100}{n}, 10, \log_2 n, \sqrt{n}, n$

(c)  $10, \frac{100}{n}, \sqrt{n}, \log_2 n, n$

(d)  $\frac{100}{n}, \log_2 n, 10, \sqrt{n}, n$

*Handwritten notes:*

$\left( \frac{100}{n} < 10 < \log_2 n < \sqrt{n} \right)$

*Handwritten notes:*

Const  $10 > \frac{100}{n}$   
(dec. Function)

*Handwritten graph:*

06. Which of the following is TRUE?

f(n) is O(g(n))

g(n) is NOT O(f(n))

g(n) is O(h(n))

h(n) is O(g(n))

(a) f(n) is O(h(n)) (b) f(n) + h(n) is O(g(n)+h(n))

(c) h(n) ≠ O(f(n)) (d) f(n) . g(n) ≠ O(g(n)). h(n))

07. An element in an Array is called Leader if it is greater than all elements to the right of it. The Time Complexity of the most efficient Algorithm to print

Algorithms Handout 2021.pdf Adobe Reader

File Edit View Window Help Tools Fill & Sign Comment Sign In

Open Export PDF

Adobe ExportPDF Convert PDF files to Word or Excel online.

Select PDF File: Algorithms Handout 2021.pdf 1 file / 537 KB

Convert To: Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change

Convert

msq

06. Which of the following is TRUE?

$f(n) \text{ is } O(g(n))$

$g(n) \text{ is NOT } O(f(n))$

$g(n) \text{ is } O(h(n))$

$h(n) \text{ is } O(g(n))$

$f < g \quad g = h$

$f < (g = h)$

(a)  $f(n) \text{ is } O(h(n))$  ✓

(b)  $f(n) + h(n) \text{ is } O(g(n)+h(n))$  ✓

(c)  $h(n) \neq O(f(n))$  ✓

(d)  $f(n) \cdot g(n) \neq O(g(n)). h(n)$  X

$f+h < g+h$

$f \cdot g < g \cdot h$

07. An element in an Array is called Leader if it is greater than all elements to the right of it. The Time Complexity of the most efficient Algorithm to print all Leaders of the given Array of size 'n' is \_\_\_\_\_.



having 'n' nodes is order of \_\_\_\_\_.

09. Consider the following operation along with EnQueue & DeQueue operation on queues where 'k' is a global parameter

MultiQueue (Q)      :  $O(1)$        $\therefore \text{time} = O(m)$

```
{  
    m = k;  
    while ((q is not empty) and (m > 0)) :  
    {  
        DeQueue (q);  
        m = m - 1;  
    }  
}
```

What is the worst case time complexity of a sequence of 'n' Queue operations on an initially empty Queue?  $=$

(a)  $O(n)$       (b)  $O(n+k)$   
(c)  $O(n.k)$       (d)  $O(n^2)$

10. A Queue is implemented using an Array such that EnQueue DeQueue operations are performed efficiently. Which one of the following statements is correct for a Queue of size 'n'.  
(a) Both operations can be performed in  $O(1)$

Sign In

Export PDF

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

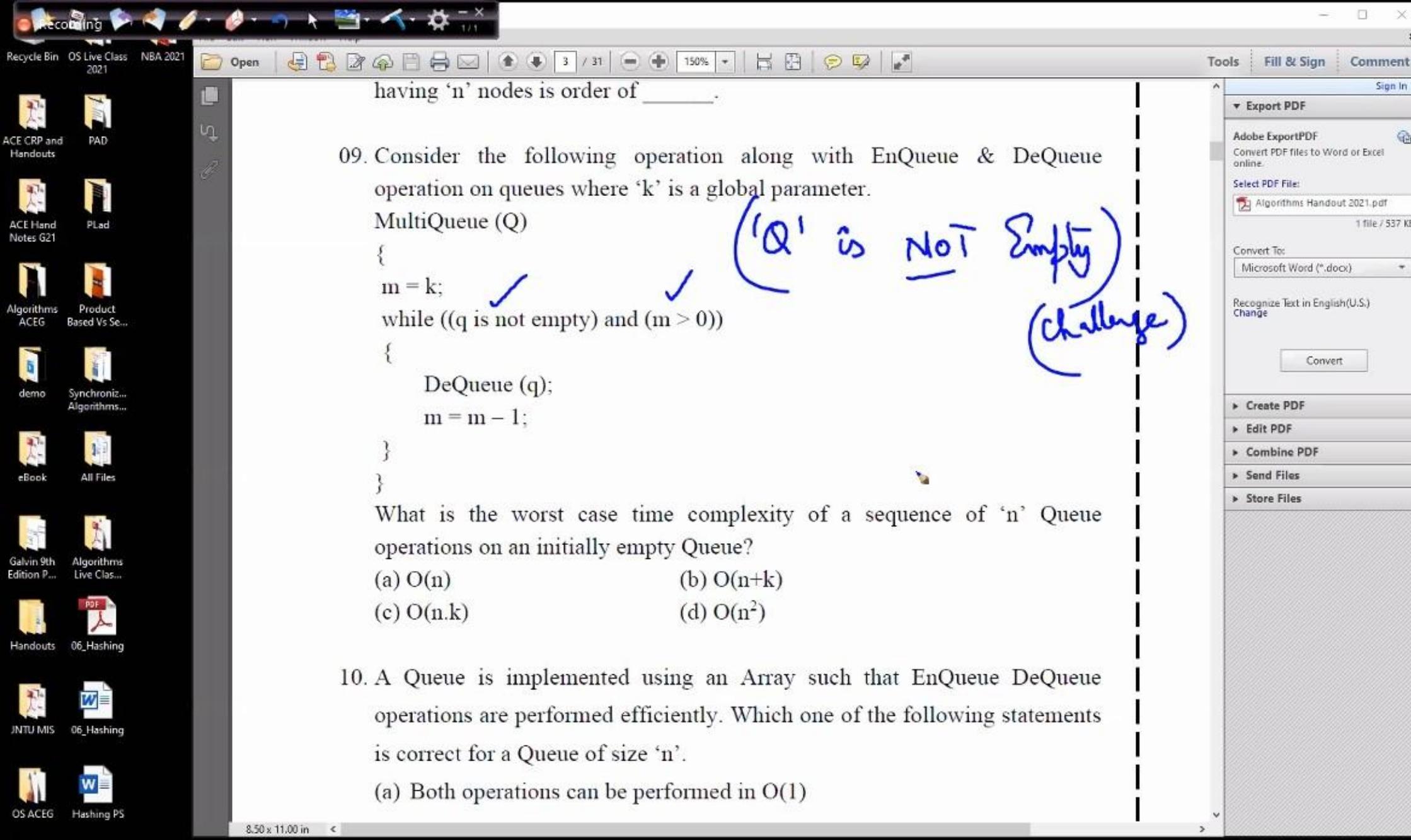
Create PDF

Edit PDF

Combine PDF

Send Files

Store Files



having 'n' nodes is order of \_\_\_\_\_

09. Consider the following operation along with EnQueue & DeQueue operation on queues where 'k' is a global parameter.

## MultiQueue (Q)

{

$$m = k$$

while ((q is not empty) and (m > 0))

{

DeQueue (q)

$$m = m - 1$$

}

}

What is the worst case time complexity of a sequence of 'n' Queue operations on an initially empty Queue?



10. A Queue is implemented using an Array such that EnQueue DeQueue operations are performed efficiently. Which one of the following statement is correct for a Queue of size ‘n’

- (a) Both operations can be performed in  $O($

Recording

Recycle Bin OS Live Class NBA 2021 2021

Open

3 / 31 150%

Tools Fill & Sign Comment

ACE CRP and Handouts

PAD

ACE Hand Notes G21

PLad

Algorithms ACEG

Product Based Vs Se...

demo Synchroniz... Algorithms...

eBook All Files

Galvin 9th Edition P... Algorithms Live Clas...

Handouts 06\_Hashing

JNTU MIS 06\_Hashing

OS ACEG Hashing PS

10. A Queue is implemented using an Array such that EnQueue DeQueue operations are performed efficiently. Which one of the following statements is correct for a Queue of size 'n'.

(a) Both operations can be performed in  $O(1)$

(b) Worst complexity of both operation will be  $\Omega(n)$

(c) Worst case time of both operation will be  $\Omega(\log n)$

(d) Atleast one operation can be performed in  $O(1)$  time and the worst case time for the option will be  $\Omega(n)$ .

ACE Engineering Academy Hyderabad | Delhi | Bhopal | Pune | Bhubaneswar | Lucknow | Patna | Bengaluru | Chennai | Vijayawada | Vizag | Tirupati |

8.50 x 11.00 in

Sign In

Export PDF

Select PDF File:

Algorithms Handout 2021.pdf 1 file / 537 KB

Convert To: Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change

Convert

Create PDF

Edit PDF

Combine PDF

Send Files

Store Files

# Time Complexity Framework for Recursive Algorithms

For M  
Studen

11. Algorithm What (n)

```
    {
        if (n = 1) return;
        else
        {
            What (n - 1);
            B(n);
        }
    }
```

12. Algorithm A(n)

```
    {
        if (n = 2) return;
        else
        return  $(A(\sqrt{n}))$ ;
    }
```

13. Algorithm Do\_It (n)

```
    {
```

$\Theta(B(n)) : O(1) : \text{given}$

Recycle Bin OS Live Class NBA 2021 2021 Open Tools Fill & Sign Comment Sign In Export PDF Adobe ExportPDF Convert PDF files to Word or Excel online. Select PDF File: Algorithms Handout 2021.pdf 1 file / 537 KB Convert To: Microsoft Word (\*.docx) Recognize Text in English(U.S.) Change Convert Create PDF Edit PDF Combine PDF Send Files Store Files

```
11. Algorithm What(n)
{
    if (n = 1) return;
    else
    {
        What (n - 1);
        B(n);
    }
}
```

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

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

$$T(n) = T(n-1) + n + a - ①$$

$$T(n-1) = T(n-2) + (n-1) + a - ②$$

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

$$= T(n-3) + (n-2) + (n-1) + n + 3a - ④$$

$$\underline{B(n) = O(n)}$$

$$\begin{aligned}
 &= T(n-k) + (n-(k-1)) + \dots + n + k \cdot a - ⑤ \\
 &= T(1) + (n-(n-1)) + \dots + k \cdot a \quad n-k=1 \\
 &\qquad\qquad\qquad\qquad\qquad\qquad\qquad k=n-1 \\
 &= (c + 2 + 3 + 4 + \dots + n) + a(n-1) \quad \text{let } c=1
 \end{aligned}$$

$$T(n) = \left[ \frac{n(n+1)}{2} + an - a \right] \checkmark$$

$$\begin{aligned}
 &= \left( \frac{n^2}{2} + \frac{n}{2} + an - a \right) = \mathcal{O}(n^2) \\
 &\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathcal{\Omega}(n^2) \\
 &\qquad\qquad\qquad\qquad\qquad\qquad\qquad = \Theta(n^2)
 \end{aligned}$$

11. Algorithm What(n)

```
{
    if (n = 1) return;
    else
    {
        What (n - 1);
        B(n);
    }
}
```

$$B(n) = O(1/n)$$

$\Rightarrow 6$

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

$\vdots$

$$= T(n-k) + \frac{1}{(n-(k-1))} + \dots + \frac{1}{n} + ka - ⑤$$

$$= T(1) + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} + a(n-1) - ⑥$$

$n-k=1$   
 $k=n-1$

$$= C + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} + an - a$$

$$= \sum_{x=1}^n \frac{1}{x} + an - a$$

Let  $C=1$

$$\downarrow f \quad \downarrow g$$

$$a > \frac{1}{n}$$

$$T(n) = \left[ O(\log n) + an - a \right] \Rightarrow O(n)$$

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

$$= \cancel{0} + T(n-1) + \cancel{\frac{1}{n}}, \quad n>1$$

$$T(n) = T(n-1) + 1 + a - ①$$

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

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

$$T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} + 3a$$

## 12. Algorithm A(n)

```
{
    if (n = 2) return;
    else
        return (A( $\sqrt{n}$ ));
}
```

$$T(n) = C, \quad \underline{n=2}$$

$$= a + T(\sqrt{n}), \quad n > 2$$

$$T(n) = T(\sqrt{n}) + a - \textcircled{1}$$

$$= T(n^{1/2}) + a - \textcircled{1}$$

$$T(n^{1/2}) = \{T(n^{1/4}) + a\} - \textcircled{2}$$

$$T(n) = T(\underline{\underline{n^{1/4}}}) + 2a - \textcircled{3}$$

$$T(n) = T(n^{1/8}) + 3a - \textcircled{4}$$

$$= T(n^{1/16}) + 4a - \textcircled{5}$$

$$= T(\underline{\underline{n^{1/2^k}}}) + k \cdot a - \textcircled{6}$$

$$= T(2) + a \cdot \log \log n$$

$$= (C + a \cdot \log \log n)$$

$$= O(\log \log n) \checkmark$$

$$T(n) = T(n^{1/2}) + a$$

$$T(n^{1/4}) = T(n^{1/8}) + a$$

$$\left\{ \begin{array}{l} \frac{1}{2^K} = 2 \\ n = 2^K \end{array} \right\}$$

$$\frac{1}{2^K} \cdot \log_2 n = \log_2 2^K$$

$$\log_2 n = 2^K$$

$$\log \log_2 n = \log_2 2^K$$

$$\log \log_2 n = K$$

14. Algorithm  $A(n)$ 

```
{
    if (n = 2) return;
    else
        return  $A(\sqrt{n}) + A(\sqrt{n})$ ;
}
```

$$T(n) = c, \quad n=2$$

$$= a + T(\sqrt{n}) + T(\sqrt{n}) + b, \quad n > 2$$

$$T(n) = 2 \cdot T(\sqrt{n}) + d \quad \text{--- (1)}$$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + d \quad \text{--- (1)}$$

$$T(n^{\frac{1}{2}}) = 2 \cdot T\left(n^{\frac{1}{4}}\right) + d \quad \text{--- (2)}$$

$$T(n) = 4 \cdot T\left(n^{\frac{1}{4}}\right) + 3d \quad \text{--- (3)}$$

$$= 2^2 \cdot T\left(n^{\frac{1}{2^2}}\right) + (2^2 - 1)d$$

$$T(n) = 8 \cdot T\left(n^{\frac{1}{8}}\right) + 7d \quad \text{--- (4)}$$

$$= 2^3 \cdot T\left(n^{\frac{1}{2^3}}\right) + (2^3 - 1)d$$

$$= 2^k \cdot T\left(n^{\frac{1}{2^k}}\right) + (2^k - 1)d$$

$$= 2^k \cdot T\left(\frac{n}{2^k}\right) + (2^k - 1)d$$

$$= \log n \cdot T(2) + (\log n - 1) \cdot d$$

$$= c \cdot \log n + d \cdot \log n - d$$

$$= (c \cdot \log n - d) = O(\log n)$$

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

$$2^k = \log_2 n$$



Algorithm RSum ( $A, n$ )       $A[1..n]: \text{integer}$

```
{
    if ( $n = 1$ ) return ( $A[1]$ );
    else
        return ( $RSum(A, n-1) + A[n]$ );
}
```

Value +  $n$  value

$$\begin{aligned}T(n) &= c, \quad n=1 \\&= a + T(n-1) + b \\&= \underline{\underline{T(n-1)}} + d \Rightarrow O(n)\end{aligned}$$

Algorithm abc ( $n$ )       $B(n) = O(n)$

```
{
    if ( $n = 1$ ) return;
    else
        return ( $abc(n-1) + B(n)$ );
}
```

$n$

$$\begin{aligned}T(n) &= c, \quad n=1 \\&= a + T(n-1) + b \Rightarrow O(n) \\&\quad \cancel{+} \cancel{+} \\&\Rightarrow O(n^2)\end{aligned}$$



Algo Test(n)

```
{
    if (n=1) return;
    else
        {
            Test(n-1)
        }
}
```

~~Fact~~

fact(n) = n \* fact(n-1)

fact(0) = 1

int fact(n)

```
{
    if (n=1) return(1);
    else
        return (n * fact(n-1));
    }
```

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

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

O(n)



$$16. T(n) = 2, n = 2$$

$$= \sqrt{n} \cdot T(\sqrt{n}) + n, \quad n > 2$$

$$T(n) = 2, \quad n = 2$$

Value of recurrence:  $\frac{2n}{2}$

$$T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \quad \textcircled{1}$$

$$= n^{1/2} \cdot T(n^{1/2}) + n \quad \textcircled{1}$$

$$T(n^{1/2}) = \underbrace{n^{1/4} \cdot T(n^{1/4}) + n^{1/2}}_{\textcircled{2}} \quad \textcircled{2}$$

$$T(n) = n^{1/2} \cdot \left( n^{1/4} \cdot T(n^{1/4}) + n^{1/2} \right) + n$$

$$= n^{3/4} \cdot T(n^{1/4}) + 2n \quad \textcircled{3}$$

$$= n^{-\frac{1}{2^2}} \cdot T(n^{1/2}) + 2n \quad \textcircled{4}$$

$$T(n) = n^{-\frac{1}{2^3}} \cdot T(n^{1/2^3}) + 3n \quad \textcircled{5}$$

$$= n^{-\frac{1}{2^k}} \cdot T(n^{1/2^k}) + k \cdot n \quad \textcircled{6}$$

$$= \frac{n}{n^{1/2^k}} \cdot T(2) + k \cdot n \quad \textcircled{7}$$

$$= \frac{n}{2} + n \cdot \log \log n$$

$$T(n) = \underline{\underline{n}} + \underline{\underline{n \cdot \log \log n}} \quad O(n \cdot \log \log n)$$

$$\frac{n}{2^k} = \frac{n}{2^{\log \log n}}$$

## 15. Algorithm Recur (n)

```

    {
        if (n = 1) return;
        else
        {
            recur(n/2);
            recur(n/2);
            B(n);
        }
    }
  
```

$$\begin{aligned}
 T(n) &= c, \quad n=1 \\
 &= a + T(n/2) + T(n/2) + c.
 \end{aligned}$$

$$T(n) = 2 \cdot T(n/2) + d, - \textcircled{1}$$

$$T(n/2) = 2 \cdot T(n/4) + d - \textcircled{2}$$

$$\begin{aligned}
 T(n) &= 2 [ 2T(n/4) + d ] + d \\
 &= 4T(n/4) + 3d - \textcircled{3}
 \end{aligned}$$

$$\textcircled{1} B(n) = O(1)$$

$$\textcircled{2} B(n) = \underline{\underline{O(n)}}$$

$$\begin{aligned}
 T(n) &= 4T(n/4) + 3d \\
 &= 2^2 \cdot T(n/2^2) + (2^2 - 1) \cdot d
 \end{aligned}$$

$$= 2^3 \cdot T(n/2^3) + (2^3 - 1) \cdot d$$

$$\vdots$$

$$= 2^K \cdot T(n/2^K) + (2^K - 1) \cdot d$$

$$= n \cdot T(1) + (n-1) \cdot d$$

$$= \frac{c \cdot n + dn - d}{O(n)} \checkmark$$

$$\begin{aligned}
 \frac{n}{2^K} &= 1 \\
 n &= 2^K
 \end{aligned}$$

```
15. Algorithm Recur(n)
{
    if (n = 1) return;
    else
    {
        recur(n/2);
        recur(n/2);
        B(n);
    }
}
```

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

$$= a + 2 \cdot T(n/2) + n \quad \textcircled{1}$$

$$T(n) = 2T(n/2) + n + a \quad \textcircled{1}$$

$$T(n/2) = 2T(n/4) + n/2 + a \quad \textcircled{2}$$

$$\begin{aligned} T(n) &= 2[2T(n/4) + n/2 + a] + n + a \\ &= 4T(n/4) + 2n + 3a \end{aligned}$$

$$B(n) = O(n)$$

$T(n)$ : Time for  
Recur(n)

$$\begin{aligned} T(n) &= 2^2 \cdot T(n/2^2) + 2n + 3a \\ &= 2^3 \cdot T(n/2^3) + 3n + 7a \quad \textcircled{3} \\ &\vdots \\ &= 2^K \cdot T(n/2^K) + K \cdot n + (2^{K-1}) \cdot a \quad \textcircled{4} \end{aligned}$$

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

$$\Rightarrow n = 2^K$$

$$K = \log n$$

$$= n \cdot T(1) + n \cdot \log n + (n-1) \cdot a$$

$$= \underline{(cn + n \log n + an - a)}$$

$$O(n \log n) \checkmark$$

Algorithm Test(n)

```
{
    if (n=1) return
    else
        { Test(n/2);
        }
}
```

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

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

$$\overline{T}(n) = T(n/2) + a - \textcircled{1}$$

$$T(n/2) = T(n/4) + a - \textcircled{2}$$

$$T(n) = T(n/4) + 2a - \textcircled{3}$$

$$= T(n/2^2) + 2a - \textcircled{3}$$

$$= \overline{T}(n/2^3) + 3a$$

$$= \overline{T}(n/2^K) + ka$$

$$= T(1) + ka$$

$$= c + a \cdot \log n$$

$$\underline{\underline{O(\log n)}}$$

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

$$n = 2^K$$

$$k = \log n$$

## 13. Algorithm Do\_It(n)

```
{
    if (n = 1) return;
    else
        return (Do_It (n - 1) + Do_It (n - 1));
}
```

$$\begin{aligned} T(n) &= c, \quad n=1 \\ &= \overbrace{a + 2 \cdot T(n-1) + b}^{\text{recurrence relation}}, \quad n>1 \end{aligned}$$

$$T(n) = 2T(n-1) + d \quad \text{--- (1)}$$

$$T(n-1) = 2T(n-2) + d \quad \text{--- (1)}$$

$$\begin{aligned} T(n) &= 2[2T(n-2) + d] + d \\ &= 4T(n-2) + 3d \quad \text{--- (1)} \\ &= 2^2 T(n-2) + (2^2 - 1) \cdot d \end{aligned}$$

$$\begin{aligned} T(n) &= 2^3 \cdot T(n-3) + (2^3 - 1) \cdot d \\ &= 2^K \cdot T(n-K) + (2^K - 1) \cdot d \\ &= 2^{n-1} \cdot T(1) + (2^{n-1} - 1) \cdot d \\ &= \frac{2^n}{2} \cdot c + \left(\frac{2^n}{2} - 1\right) d \\ &= \underline{\underline{O(2^n)}} \quad \checkmark \end{aligned}$$

$$\begin{aligned} n - K &= 1 \\ \Rightarrow K &= n-1 \end{aligned}$$

$$Q_1) \hat{n} = 2^{2k}, k \geq 0$$

$$\left\{ \begin{array}{l} T(n) = \sqrt{2} \cdot T(n/2) + \sqrt{n}, n > 1 \\ T(1) = ' \end{array} \right.$$

Evaluate: Final value

$$Q_4) T(n) = 4 \cdot T(n/2) + n$$

$$Q_5) T(n) = 3T(n/2) + n$$

(Back Substitution)

$$Q_2) \left. \begin{array}{l} T(n) = T(n/2) + n, \\ T(1) = 1 \end{array} \right\}$$

$$Q_3) \left. \begin{array}{l} T(n) = 2T(n-1) + n, n > 1 \\ T(1) = 1 \end{array} \right\}$$

$$Q_6) \left. \begin{array}{l} T(n) = n * T(n-1), n > 1 \\ T(1) = 1 \end{array} \right\}$$

$$Q_7) T(n) = 2T(n/2) + \log n$$



# Running Time of Program Segments with loops: (Non-Recursive framework)

1) for-loop | while-loop | repeat-until | do-while

2) for (Init; Condition; update) : None

↓      ↓      ↓

①    ②    ③

} Infinite loop

for i = 1 to n

i = 1, {1, 2, 3}

for i ← 1 to n

(iv) 'S' : n

(i) Init : 1  
 (ii) Comparison : (n+1)  
 (iii) update : n

$c = 0$

$$\left. \begin{array}{l} 1. \text{ for } i \leftarrow 1 \text{ to } n \\ \quad \left\{ \begin{array}{l} c = c + 1; \\ \quad \overline{o(1)} \end{array} \right. \end{array} \right\} \rightarrow \begin{array}{l} \text{Time} = O(n \cdot q \cdot 2^k) : O(n) \\ \text{Value of } c : \underline{\underline{n}} : O(n) \end{array}$$

$\frac{i=1 \quad i=2 \quad i=3 \quad \dots \quad i=n}{o(1) \cdot o(1) \quad o(1)} \quad o(1) = O(n)$  : unfolding

$T(n) = \sum_{i=1}^n o(1) = O(n) = \text{for } i \leftarrow 1 \text{ to } n = o(1)$

2.  $\left[ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n/2 \\ \quad (c = c + 1; o(1)) \end{array} \right] \rightarrow \begin{array}{l} \text{Time} = O(n/2) = O(n) \checkmark \\ \text{Value} = \frac{n}{2} = O(n) \end{array}$



3)  $\text{for } (i = n/2; i \leq n; i += n/2)$   
 $c = c + 1;$

$i = n/2$        $i = n$        $i = 3n/2$   
 $c = 1$                $c = 2$

Time  $\rightarrow O(2^{\frac{n}{2}}) = O(1)$   
 $\text{val}(c) = 2 \in O(1)$

$n = 8$   
 $i = (1, 5, 6, 7, 8)$   
 $c = 2, 4, 5$

4)  $\text{for } (i = n/2; i \leq n; i += n/2)$   
 $c = c + 1;$

Time:  $O(n/2) = O(n)$   
 $\text{val}(c): n/2 = O(n)$

5)  $\text{for } i \leftarrow 1 \text{ to } 2^n \Rightarrow \text{Time: } O(2^n)$   
 $c = c + 1;$   
 $\text{val}(c): O(2^n)$

2)  $\left\{ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad \left\{ \begin{array}{l} \text{for } j \leftarrow 1 \text{ to } n \\ \quad \quad c = c + 1; \end{array} \right. \end{array} \right\}$  Time:  $O(n^2)$   
 $\text{value} = \underline{\underline{n^2}}$        $i = 1 \xrightarrow{n} i = 2 \xrightarrow{n} i = 3 \xrightarrow{n} \dots i = n \xrightarrow{n} n = n * n = \underline{\underline{n^2}}$

$\rightarrow \left\{ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad O(n) \end{array} \right\} \sum_{i=1}^n \sum_{j=1}^n O(1) = O(n^2)$

3)  $\left\{ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad \left\{ \begin{array}{l} \text{for } j \leftarrow 1 \text{ to } n/2 \\ \quad \quad c = c + 1 \end{array} \right. \end{array} \right\}$  Time:  $n \cdot \frac{n}{2} = O(n^2)$   
 $\text{value: } \frac{n^2}{2} = O(n^2)$

4)  $\left. \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad \text{break;} \end{array} \right\}$  Time:  $O(1)$

5)  $\left. \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad \left[ \begin{array}{l} \text{for } j \leftarrow 1 \text{ to } n \\ \quad \text{break;} \end{array} \right] \end{array} \right\}$  Time:  $O(n)$

6)  $\left. \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \quad \left[ \begin{array}{l} \text{for } j \leftarrow i \text{ to } n \\ \quad c = c + 1; \end{array} \right] \end{array} \right\}$  Time:  $O(n^2)$

$$\text{Val}(c) : \frac{n(n+1)}{2} = O(n^2)$$

$$\underbrace{i=1}_{(n+0)} + \underbrace{i=2}_{(n-1)} + \underbrace{i=3}_{(n-2)} + \dots + \underbrace{i=n}_{(1)} = \overline{\text{line}} = \frac{n(n+1)}{2}$$

$$c = n + (n-1) + (n-2) + \dots + 1$$

$i = 1;$   
while ( $i \leq n$ )  
     $i = i + 1;$   
 $i = 0$

$\Rightarrow$  Time:  $O(N \cdot \sqrt{i}) = O(n)$

while ( $i \leq n$ )  
     $i = i + 2;$

$\Rightarrow$  Time:  $O\left(\frac{n}{2}\right) = O(n)$

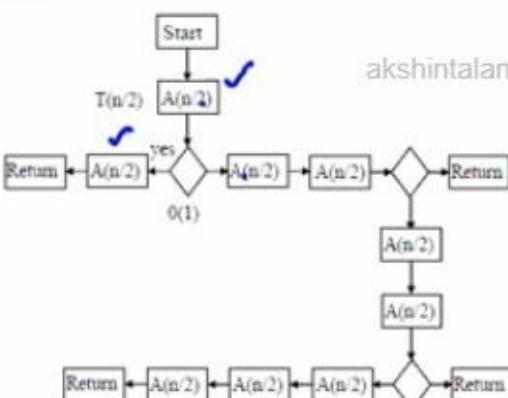
for ( $i=1; i \leq n; ++i$ ): $n$  Time:  $\underline{\mathcal{O}(n \cdot \log n)}$   
for ( $j=1; j \leq n; j*=2$ ): $\log n$  value( $c$ )  $\in \mathbb{N} \cdot \log n$

$c = c + 1;$

$i=1 \quad i=2 \quad i=3 \quad \dots \quad i=n$

$\log n \quad \log n \quad \log n \quad \dots \quad \log n$

17. The given diagram represents the flowchart of recursive algorithm A(n). Assume that all statements except for the recursive calls have order(1) time complexity. Then the best case & worst case time of this algorithm is \_\_\_\_\_.



$$B.C = T(n) = 2 \cdot T(n/2) + O(1) = O(n)$$

$$W.C = T(n) = 8T(n/2) + O(1) \\ = O(n^3)$$



18. void abc(char \*s)  
{  
 if (\*s != '\0')  
 {  
 printf("%c", \*s);  
 abc(s + 1);  
 abc(s + 1);  
 }  
}

For Micro Notes by the  
Student



Tools | Fill & Sign | Comment | Sign In

Export PDF  
Adobe ExportPDF  
Convert PDF files to Word or Excel online.  
Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB  
Convert To:  
Microsoft Word (\*.docx)  
Recognize Text in English(U.S.)  
Change  
Convert  
Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files



Open



18. void abc(char \*s)

{

if (\*s != '\0')

{

printf("%c", \*s);

abc(s + 1);

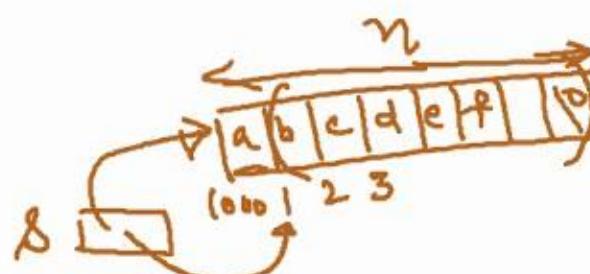
abc(s+1);

}

*S: ptr to a string of length*

*s = "xyz"*

*For Micro Notes by the  
Student  
'n' chars*



Time-Complexity :

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

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

$$T(n) = \underline{2 \cdot T(n-1) + d}, \quad n>1$$

$$\underline{\underline{O(2^n)}}$$

19. for  $i \leftarrow 1$  to  $n$  }  $O(n)$   
 $c = c + 1;$

20. for  $i \leftarrow 1$  to  $n$  }  $O(n^2)$   
 for  $j \leftarrow 1$  to  $n/2$   
 $c = c + 1;$

21. for  $i \leftarrow 1$  to  $n$  }  $O(n^2)$   
 for  $j \leftarrow 1$  to  $n/4$   
 for  $k \leftarrow 1$  to  $n$   
 break;

22. for  $(i=1; i <= n; ++i)$   
 for  $(j=1; j <= n; ++j)$   
 for  $(k=n/2; k <= n; k += n/2)$   
 $c = c + 1;$

23.  $i=2$   
 while ( $i <= n$ ) }  $n=2^K$   
 {  $i=i*2;$  }  $(i+1)$

24.  $i=n;$   
 while ( $i > 0$ )  
 {  
 $i=i/2;$   
 }

$\text{val}(c) = 2n^2$

$2^2 = n$

$i = 1, 2, 4, 8, 16, 32$

$= (2, 2, 2, 2, 2)$

Time :  $O(\text{No. of iterations}) = O(K)$   
 $= O(\log n)$



▼ Export PDF

Adobe ExportPDF  
 Convert PDF files to Word or Excel online.

Selected PDF File:

Algorithms Handout 2021.pdf

1 file / 537 KB

Convert To:

Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
 Change

Convert

► Create PDF

► Edit PDF

► Combine PDF

► Send Files

► Store Files

Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

Export PDF  
Adobe ExportPDF  
Convert PDF files to Word or Excel online.  
Selected PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB  
Convert To:  
Microsoft Word (\*.docx)  
Recognize Text in English(U.S.)  
Change  
Convert

Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

20. for  $i \leftarrow 1$  to  $n$   
    for  $j \leftarrow 1$  to  $n/2$   
         $c = c + 1;$

21. for  $i \leftarrow 1$  to  $n$   
    for  $j \leftarrow 1$  to  $n/4$   
        for  $k \leftarrow 1$  to  $n$   
            break;

22. for ( $i = 1$ ;  $i \leq n$ ;  $++i$ ) :  $n$   
    for ( $j = 1$ ;  $j \leq n$ ;  $++j$ ) :  $n$   
    for ( $k = n/2$ ;  $k \leq n$ ;  $k += n/2$ ) :  $2$   
         $c = c + 1;$

23.  $i = 1;$   
    while ( $i \leq n$ )  
    {  
         $i = i * 2;$   
    }

24.  $i = n;$   
    while ( $i > 0$ )  
    {  
         $i = i/2;$

$c = 0$   
 $\text{for } (i = n/2; i \leq n; i += n/2) \text{ (2)}$   
 $c = c + 1;$

$i = \frac{n}{2} + \frac{n}{2} = n + \frac{n}{2} = \frac{3n}{2}$

$c = 1 + 1$

Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

Export PDF  
Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

19. 

```
for i ← 1 to n
    c = c + 1;
```

20. 

```
for i ← 1 to n
    for j ← 1 to n/2
        c = c + 1;
```

21. 

```
for i ← 1 to n
    for j ← 1 to n/4
        for k ← 1 to n
            break;
```

22. 

```
for (i= 1; i <= n; ++ i)
    for (j = 1; j <= n; ++ j)
        for (k = n/2; k <= n; k += n/2)
            c = c + 1;
```

23. 

```
i = 1;
while (i <= n)
{
    i = i * 2;
}
```

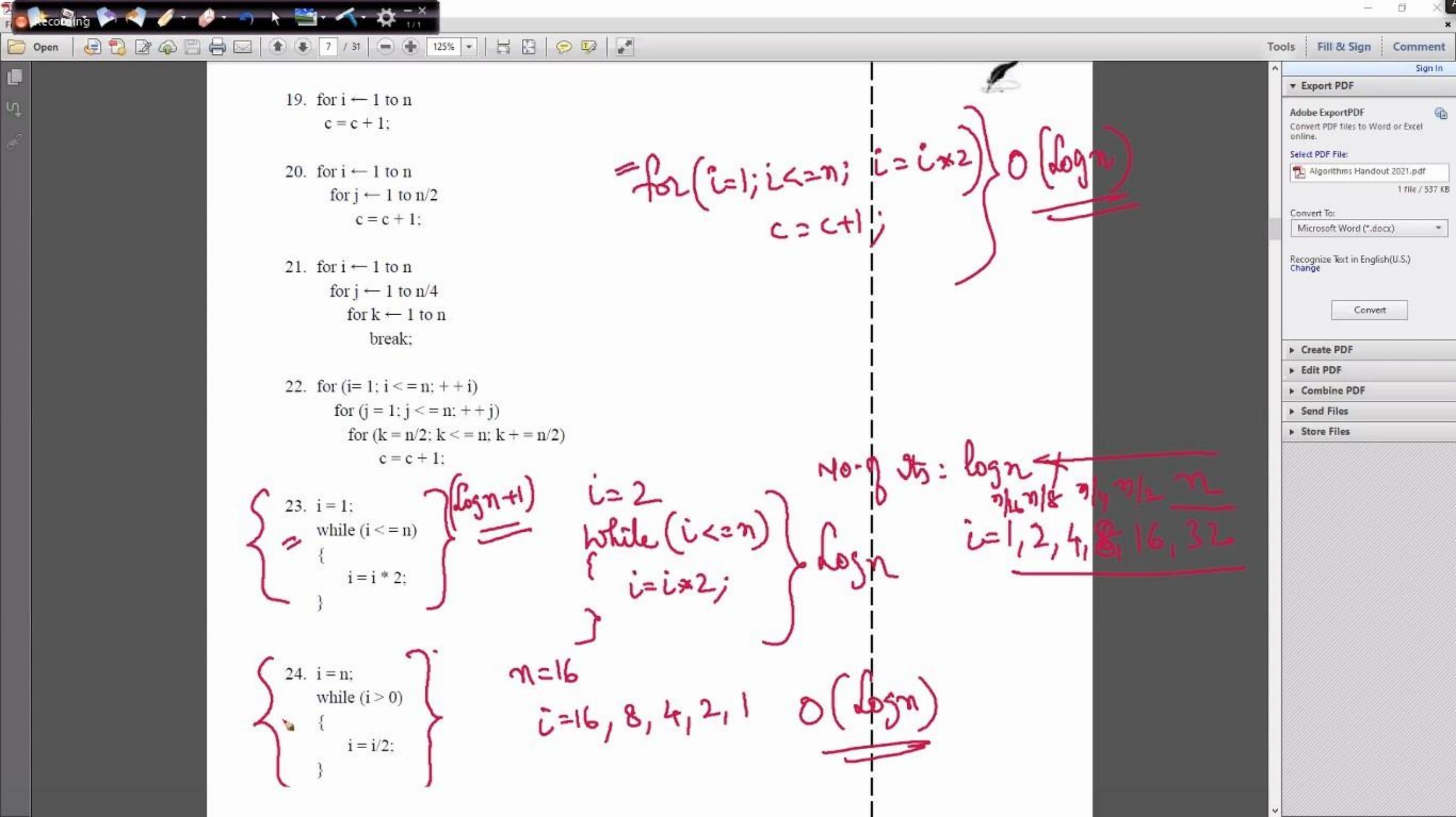
24. 

```
i = n;
while (i > 0)
{
    i = i/2;
}
```

$= \text{for}(i=1; i \leq n; i = i * 2) \quad c = c + 1; \quad O(\log n)$

No. of It's:  $\log n$   
 $n = 16 \rightarrow i = 1, 2, 4, 8, 16, 32$

$n = 16 \rightarrow i = 16, 8, 4, 2, 1 \quad O(\log n)$



Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

Export PDF  
Adobe ExportPDF  
Convert PDF files to Word or Excel online.  
Selected PDF File:  
Algorithms Handout 2021.pdf  
1 file / 537 KB  
Convert To:  
Microsoft Word (\*.docx)  
Recognize Text in English(U.S.)  
Change  
Convert

Tools | Fill & Sign | Comment

8 / 31 125% 2/2

27.  $m = 2^n$   
for ( $i = 1; i \leq n; ++i$ ) :  $n : O(n^2)$   
for ( $j = 1; j \leq m; j = 2 * j$ ) :  $\log m$  } Time:  $n * \log_2 m$   
 $c = c + 1$

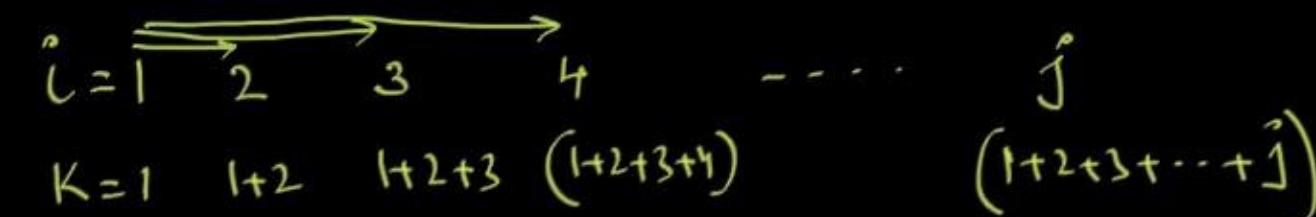
28. for  $i \leftarrow 1$  to  $n$   
for  $j \leftarrow i$  to  $n$ }  $O(n^2)$

G 29.  $f(n) = \sum_{i=1}^n O(n) = O(n^2)$

30.  $i = n;$   
while ( $i > 0$ ):  $\log n$   
{  
     $j = 1;$   
    while ( $j \leq n$ ):  $\log n$  }  $\log n$   $O(\log n)^2 = \log n^2$   
    {  
         $j = 2 * j;$   
    }  
     $i = i/2;$   
}

ACE Engineering Academy ➤ Hyderabad | Delhi | Bhopal | Pune | Bhubaneswar | Lucknow | Patna | Bengaluru | Chennai | Vijayawada | Vizag | Tirupati | Kukatpally | Kolkata

```
25. k = 1; i = 1;
    while (k <= n)
    {
        i++;
        k = k + i
    }
```



$$\begin{aligned} \text{Time: } & O(j) \\ & : O(\sqrt{n}) \end{aligned}$$

$$\begin{aligned} K &= \underbrace{(1+2+3+\dots+j)}_j = n \\ \frac{j(j+1)}{2} &= n \\ \Rightarrow j^2 + j &= 2n \end{aligned}$$

$$\begin{aligned} j^2 &\sim n \\ j &\sim \sqrt{n} \end{aligned}$$

```

31. int fun (int n)
{
    int i, j, p, q = 0;
    for (i = 1; i <= n; ++ i)
    {
        p = 0; v = 0
        for (j = n; j > 1; j = j/2)
            ++ p;
        for (k = 1; k < p; k = k * 2)
            ++ q;
    }
    return (q);
}

```

```

32. for (i = 1; i <= n; ++ i)
{
    a j = 1;
    while (j <= n)
        j = 2 * j;
    c for (k = 1; k <= n; ++ k)
        c = c + 1
}

```

(i)  $\text{Value } (v) = O(n \cdot \log \log n)$  ✓

$$\log n : P = \log n$$

$$\log P : \log \log n$$

$$v = \underline{\log \log n} + \log \log n$$

$$\text{Final}(v) = \underline{\underline{n \cdot \log \log n}}$$

(ii) Time Complexity:  $O(n \cdot \log n)$  ✓

$$a) (n + n \log n + n \cdot \log \log n)$$

(iii)  $\text{Value } (v) = \underline{\underline{\log \log n}}$

3) Time:  $O(n^2)$  ✓

$$\underline{n + n \log n + n^2}$$

```
33. n = 2k
for (i = 1; i <= n; ++ i)
{
j = 2;
while (j <= n)
{
j = j * j;
printf("*");
}
}
```

No. of Times that '\*' gets outputted:

$$n = 2^{2k}$$

$$k = \log \log n$$

'K'  
 $\downarrow$   
 $(k+1)$  '\*' : per- $\sqrt{}$

$$\begin{array}{l} k=1 \\ \hline n=4 \\ j=2, 4, 16 \\ \hline \text{***} \\ \hline n*(2) \end{array}$$

$$\begin{array}{l} k=2 \\ \hline n=16 \\ j=2, 4, 16, 256 \\ \hline (\ast\ast\ast) \\ \hline n*(3) \end{array}$$

$$\begin{array}{l} k=3 \\ \hline n=256 \\ j=(2, 4, 16, 256)(256)^2 \\ \hline \ast\ast\ast\ast \\ \hline n*(4) \end{array}$$

Total Stars:  $n * (k+1)$   
 $\therefore \underline{\underline{n(\log \log n + 1)}}$

Time:  $\underline{\underline{O(n \log \log n)}}$  ✓

34. A: Array [1 .....n] of binary:

```

f(m) = θ(m)
count: integer;
count = 1;
for i = 1 to n
{
    if (A[i] == 1) count++;
    else
    {
        f(count);
    }
}

```

(i)  $A[1..n] = \{1\}$  ✓  
 $: O(n)$  : O(n)

(ii)  $A[1..n] = \{0\}$  ✓  
 $: O(n)$  : O(n)

(i) Best-Case : O(n) : O(n) (v)  $A[i] = \{1\}$  : O(n) : O(n) | 1|0|1|0|1|0  
 Worst-Case : O(n) : O(n) (f(1) + f(2) + f(3) + ... + f(n)/2)  $\downarrow$   
 $f(m) = \theta(m)$  (ii)  $A[i] = 1, i=1, n-1$  | 1|1|1|1|1|0  
 $f(1) = \theta(1)$   $i=n$   
 $f(n) = \theta(n)$   
 $f(\log n) = \theta(\log n)$

(iii)  $A[i] = 1, i=1, n-1$  : O(1)(n-1) + (n-1)+1 : O(n) : O(n)

(iv)  $A[i] = 1, i=1, n/2$  | n/2  
 $= 0, i=(n/2+1), n$  | n/2  
 w.c. ✓

First Half : (n/2) + (n/2+1) + O(1)(n/2-1) = O(n) ✓  
 ↗ Remaining (n/2-1) MDS  
 1st & 2nd Half : (n/2) + (n/2) = O(n^2) ↗ 2nd 1/2  
 $\therefore \frac{n}{2} + \left(\frac{n}{2}\right)\frac{n}{2} = O(n^2)$   $\left(1+2+3+\dots+\frac{n}{2}\right) = O(n^2)$

35. Consider the following three functions

$$f_1 = 10^n, f_2 = n \log n, f_3 = \sqrt{n}^n$$

Which one of the following options arranges the functions in the increasing order of asymptotic growth rate?

- (a)  $f_1, f_2, f_3$       (b)  $f_3, f_2, f_1$   
 (c)  $f_2, f_3, f_1$       (d)  $f_2, f_1, f_3$

36. N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease-key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order:  $\Theta(N)$  delete,  $O(\log N)$  insert,  $O(\log N)$  find, and  $\Theta(N)$  decrease-key. What is the time complexity of all these operations put together?

- (a)  $O(\log^2 N)$       (b)  $O(N)$   
 (c)  $O(N^2)$       (d)  $\Theta(N^2 \log N)$

| <u>DLL: Sorted:</u> |                            | <u>No. of opns</u>       | <u>Time opn</u>                     | <u>Total time</u> |
|---------------------|----------------------------|--------------------------|-------------------------------------|-------------------|
| <u>Delete</u>       | <u>N</u>                   |                          |                                     |                   |
| <u>Insert</u>       | <u><math>\log N</math></u> | <u><math>O(N)</math></u> | <u><math>N \times \log N</math></u> |                   |
| <u>Find</u>         | <u><math>\log N</math></u> | <u><math>O(1)</math></u> | <u><math>N \times \log N</math></u> |                   |
| <u>Dec-Key</u>      | <u>N</u>                   | <u><math>O(N)</math></u> | <u><math>N^2</math></u>             |                   |

$$f_1 = 10^n; f_2 = n \log n; f_3 = \sqrt{n}^n$$

$\downarrow \quad \downarrow \quad \downarrow$

$$n \cdot \log 10 \quad (\log n)^2 \quad \sqrt{n} \log n$$

$$f_2 < f_3 < f_1$$

$$\begin{aligned} & (\sqrt{n})^n \\ & (\frac{n}{2} \log n) \end{aligned}$$

$$\underline{\underline{O(n)}}$$

$$\underline{\underline{O(N^2)}}$$

SPACE COMPLEXITY : of a Program 'P' is the no. of memory words needed for carrying computational steps

- The Space complexity excludes, the space allocated to hold the input.
- It is therefore only the work space (Additional Space) required by the Algorithm/P to carryout its computational steps.

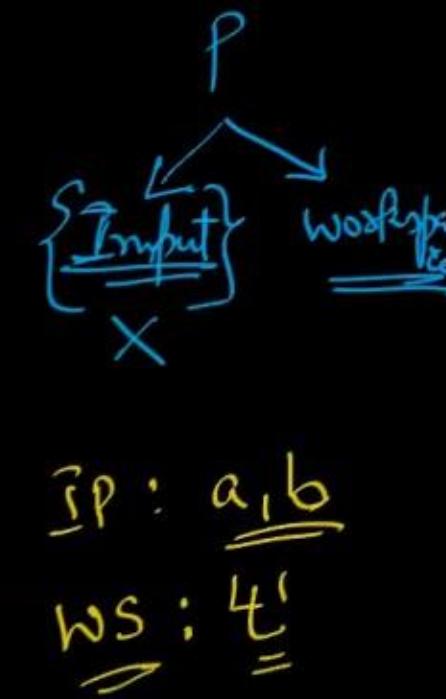
Space-Complexity :  $S(P) = \text{WorkSpace}$

→ All definitions of rates of growth & Asymptotic Bounds pertaining to time complexity equally applies to Space Complexity.

$$S(P) \sim \text{w.r.t Input}(n)$$

```

    void Swap(int a, int b)
    {
        int t;  $\geq O(1)$ 
        t = a;
        a = b;
        b = t;
    }
  
```



→ Workspace cannot exceed the running time of Algorithm, i.e. it requires a constant time to write into memory cell.

If  $T(n)$  &  $S(n)$  repr. respectively Time & Space complexities,  
then  $S(n) = O(T(n))$

→ In Case of Recursive Algorithms, the Stack Space also contributes to workspace.

① Algo Test( $A, n$ ):  
 $\{ \quad S(): O(1)$   
 $T(n): O(n)$   
 $\quad \text{int } A[1..n]; \underline{\text{Sum}} = 0; \underline{i} = 1$   
 $\quad \text{for } i \leftarrow 1 \text{ to } n$   
 $\quad \quad \text{Sum} += A[i];$   
 $\}$

② int  $A[1..n]; S():$    
 $\underline{\text{Algo RSum}(A, n) O(n)}$   
 $\{ \quad \text{if } (n=1) \text{ return}(A[1]);$   
 $\quad \text{else return}(A[n] + \underline{\text{RSum}(A, n-1)})$   
 $\}$

3) Algo Test (n)

```
{ If (n=1) return;
```

else

```
{ Test (n/2);  
Test (n/2);
```

}

Time:  $O(n)$

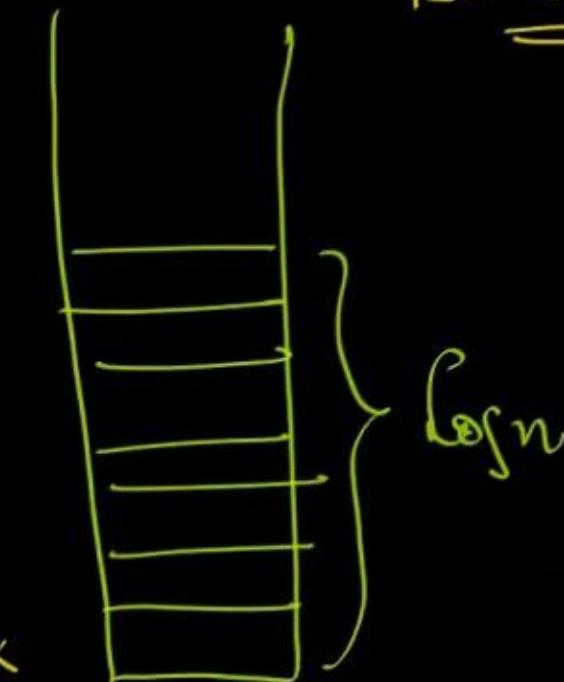
Space:  $O(\log n)$

; Time:  $O(\log n)$

; Space:  $O(\log n)$

$n \downarrow$   
 $n/2$   
 $\downarrow$   
 $n/4$   
 $\downarrow$   
 $n/8$   
 $\vdots$

$n \downarrow$   
 $n/2$   
 $\downarrow$   
 $n/4$   
 $\downarrow$   
 $n/8$   
 $\vdots$



$$n = 2^K$$

$$k = \underline{\log n}$$



Algo  $D0\_iL(n)$

{ if ( $n=1$ ) return;

else

return  $(D0\_iL(n-1) + D0\_iL(n-1))$

}

$S(): O(n)$

Algo  $A(n)$

{ if ( $n=2$ ) return

else return  $(A(\sqrt{n}))$

}

$\textcircled{K}_2 \underline{\log \log n}$

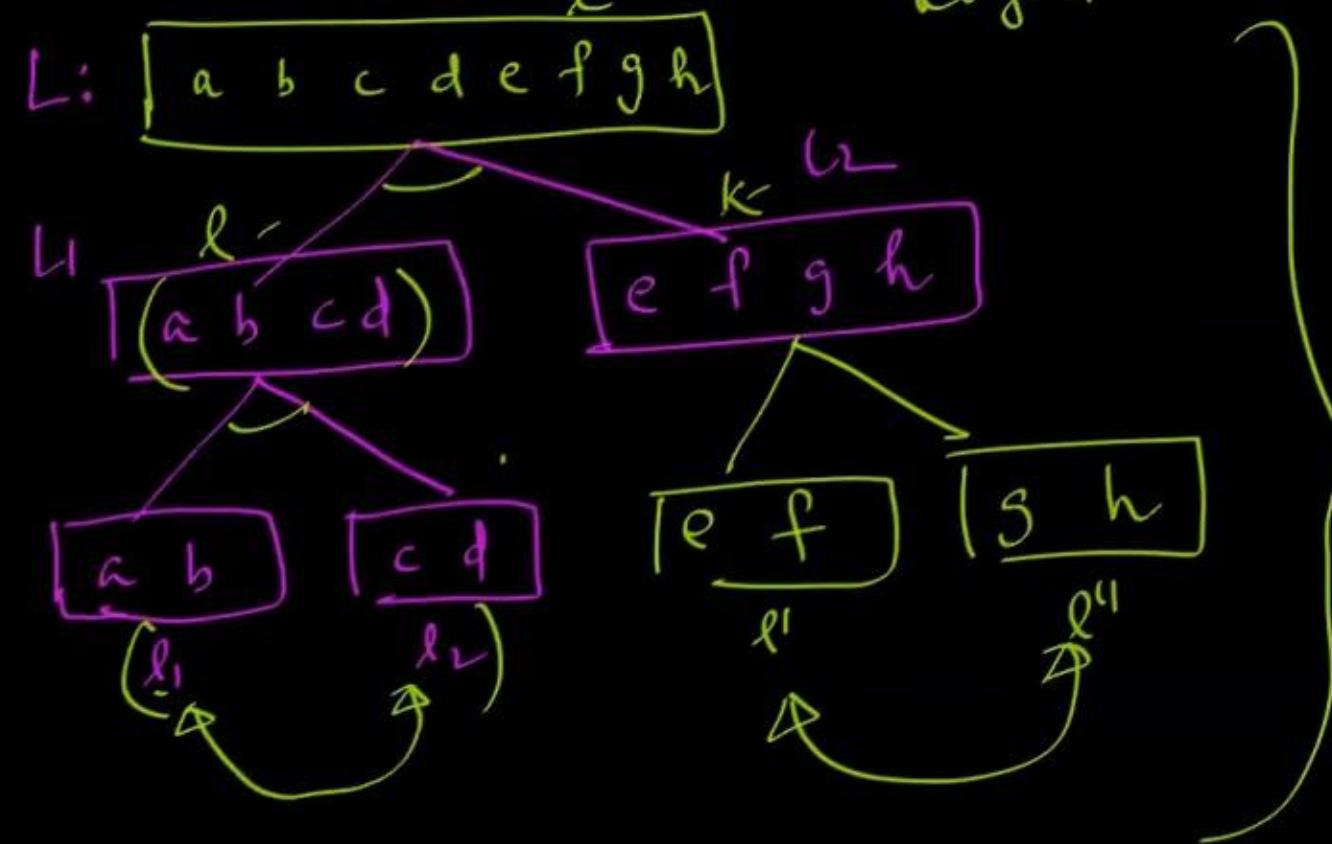
~~$O(\log n)$~~   $S(): O(\log \log n) \times O(\sqrt{n}) \times$

$T(n): O(\log \log n)$

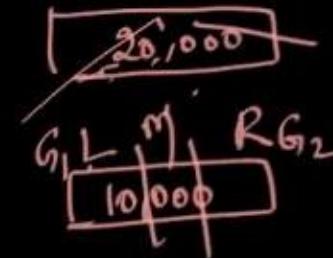


# Design Strategies : $\langle DC + GM + DP \rangle$

1. Divide & Conquer : (D and C)



{Napolean}



(GATE) : P

Easy

Subject

difficult

Manageable

$\frac{NM}{Attent}$

D and C

divide : divide is mandatory

Conquer (Combine) : optional : (~~Binary Search~~) <sup>divide</sup> or ~~QuickSort~~ <sup>divide</sup>

Control Abstraction | General Method

Flow of  
control  
is  
clear

details  
are  
abstracted

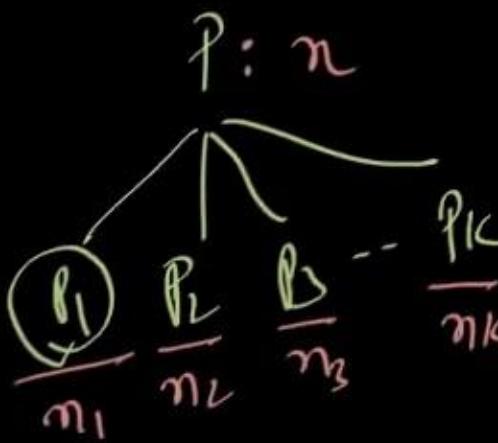


```

1 Algorithm DAndC( $P$ )
2 {
3   if Small( $P$ ) then return  $S(P)$ ;
4   else
5   {
6     divide  $P$  into smaller instances  $P_1, P_2, \dots, P_k$ ,  $k \geq 1$ ;
7     Apply DAndC to each of these subproblems;
8     return Combine(DAndC( $P_1$ ), DAndC( $P_2$ ), ..., DAndC( $P_k$ ));
9   }
10 }

```

Algorithm 3.1 Control abstraction for divide-and-conquer



Time-Complexity of Problems Solvable by  
DandC :

Let  $T(n)$  repr. time complexity of  $\text{DandC}(n)$ ;

$$T(n) = f(n) \quad , 'n' \text{ is } \underline{\text{Small}} ; f(n) = \underline{\text{SMALL}} + \underline{S}$$

$$= T(n_1) + T(n_2) + T(n_3) + \dots + T(n_k) + g(n) \quad , 'n' \text{ is } \underline{\text{large}}$$

$$\left( g(n) = \text{SMALL} + \begin{matrix} \text{divide} \\ \text{Combine} \end{matrix} \right)$$

Algo DandC( A, p, q)

{  
if (small(p, q))  
return S(p, q);

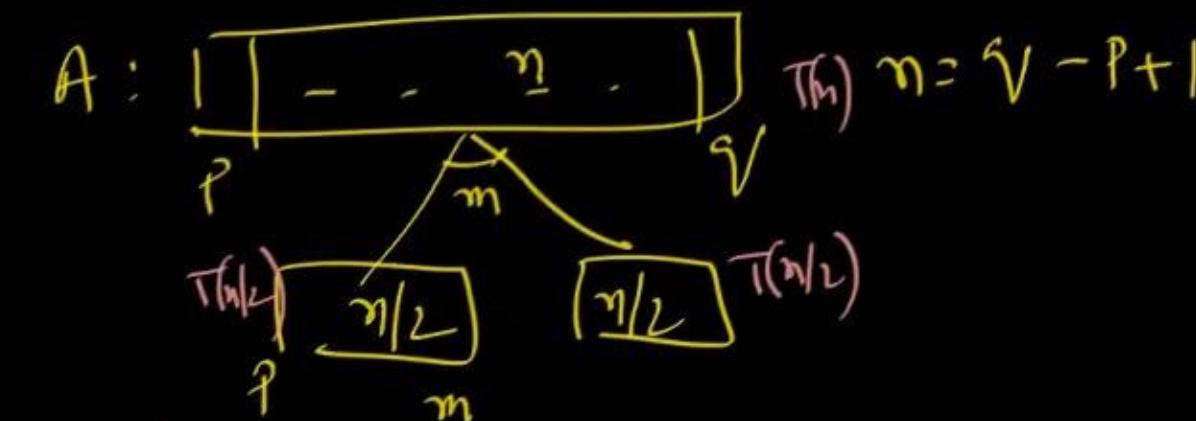
else

{  
m  $\leftarrow \lfloor \frac{p+q}{2} \rfloor$

sd<sub>1</sub>  $\leftarrow$  DandC( A, p, m);

sd<sub>2</sub>  $\leftarrow$  DandC( A, m+1, q);

} COMBINE( sd<sub>1</sub>, sd<sub>2</sub> );



$T(n) = f(n)$ , ' $n$ ' is small  
 $= T(n/2) + T(n/2) + g(n)$ , ' $n$ ' is large

$\boxed{T(n) = 2 \cdot T(n/2) + g(n)}$

DandC Recurrence  
 Size of each Subproblem  
 No. of Subproblems



Generalized Form of the DandC Recurrence:

Symmetric :  
DandC

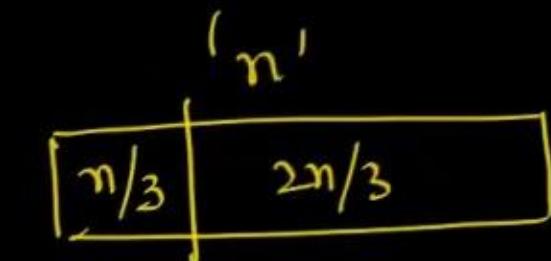
$$T(n) = a \cdot T(n/b) + g(n)$$

No. of Subproblem      Size of each  
Subproblem

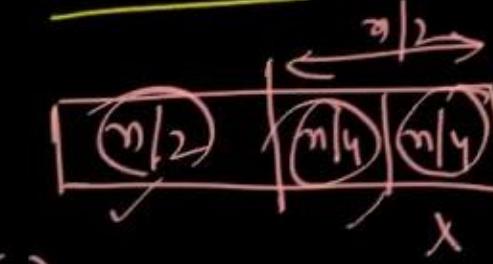
$b > 1$ ;  $a \geq 1$ ;  $g(n)$  is +ve

Asymmetric :  
DandC

$$T(n) = T(n/3) + T(2n/3) + g(n)$$



$$T(n) = T(\alpha n) + T((1-\alpha)n) + g(n), \quad 0 < \alpha < 1$$



Asymmetric :  
Care 2

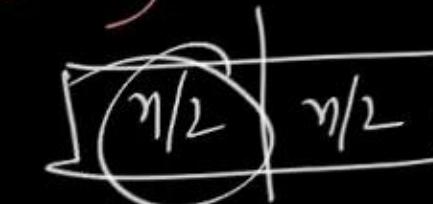
$$T(n) = T(n/2) + T(n/4) + g(n);$$

$$= T(\alpha n) + T(\beta n) + T(\gamma n) + g(n)$$



→ Problem may be divided into multiple (more than 2) Subproblems.  
 (It is not necessary to solve all of them)

$$\text{Ex: } T(n) = T(n/2) + g(n) : \text{(Binary Search)}$$



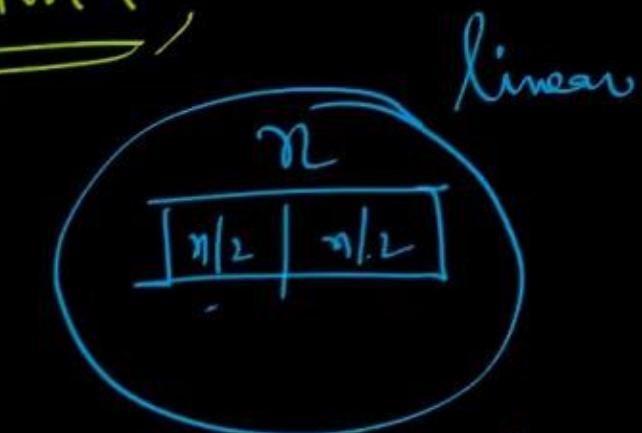
→ Combining the results of the Subproblems is optional:

$$T(n) = 4T(n/2) + g(n) : \checkmark$$

↑  
multiple instance

$$T(n) = 3T(n/2) + g(n) \checkmark$$

multiple instance



## Applications of Domd C :

1) Max-Min : Given an array of size 'n' elements, it is required to find the Max & Min of the array.

a) Non-DC StratMAXMIN : Time:  $O(n)$  X Space:  $O(1)$

Algo STRATMAXMIN( $A, n, \text{max}, \text{min}$ )

```

1. max ← min ← A[1];
2. for i ← 2 to n [n-1]
  { if (A[i] > max)
    else max ← A[i];
    if (A[i] < min)
      min ← A[i];
  }
}

```

: 'Metric' → (Time - depends on)  
 ↳ Comparisons ↳ Element Comparison  
 ↳ Index Comparison  
 → Total Element Comparisons:  $2(n-1)$

(i) Best-Care : Inc/Asc :  $(n-1)$

(ii) worst-care : Dcr/Dsc :  $2(n-1)$

(iii) Avg. Care : first comp. fails for  $\frac{1}{2}$  of given elements;

## Applications of DomdC :

1) Max-Min : Given an array of size 'n' elements, it is required to find the Max & Min of the array.

a) Non-DC [ StraightMAXMIN : Time:  $O(n)$  X Space:  $O(1)$  ]

Algo STRAIGHTMAXMIN(A, n, max, min)

{  
1. max  $\leftarrow$  min  $\leftarrow A[1]$ ;

2. for  $i \leftarrow 2$  to  $n$  [ $n-1$ ]

{ if ( $A[i] > \text{max}$ )

else  $\text{max} \leftarrow A[i]$ ;

if ( $A[i] < \text{min}$ )

$\text{min} \leftarrow A[i]$ ;

}

: 'Metric'  $\rightarrow$  (Time - depends on)  
 ↳ Comparisons ↳ Element Comparison ✓

↳ Index Comparison

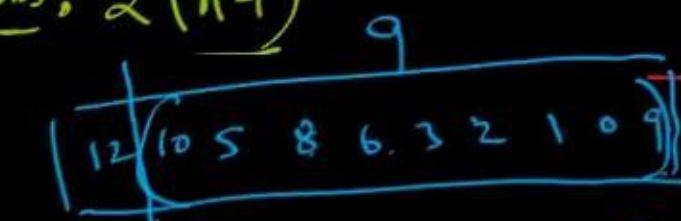
→ Total Element Comparisons:  $2(n-1)$

(i) Best-Case : Inc/Asc :  $(n-1)$

(ii) Worst-Case : Dec/Dsc :  $2(n-1)$

(iii) Avg. Case : First Comp. fails for  $\frac{1}{2}$  of given elements;

$$\therefore (n-1) + \frac{n}{2} = \left(\frac{3n}{2} - 1\right)$$

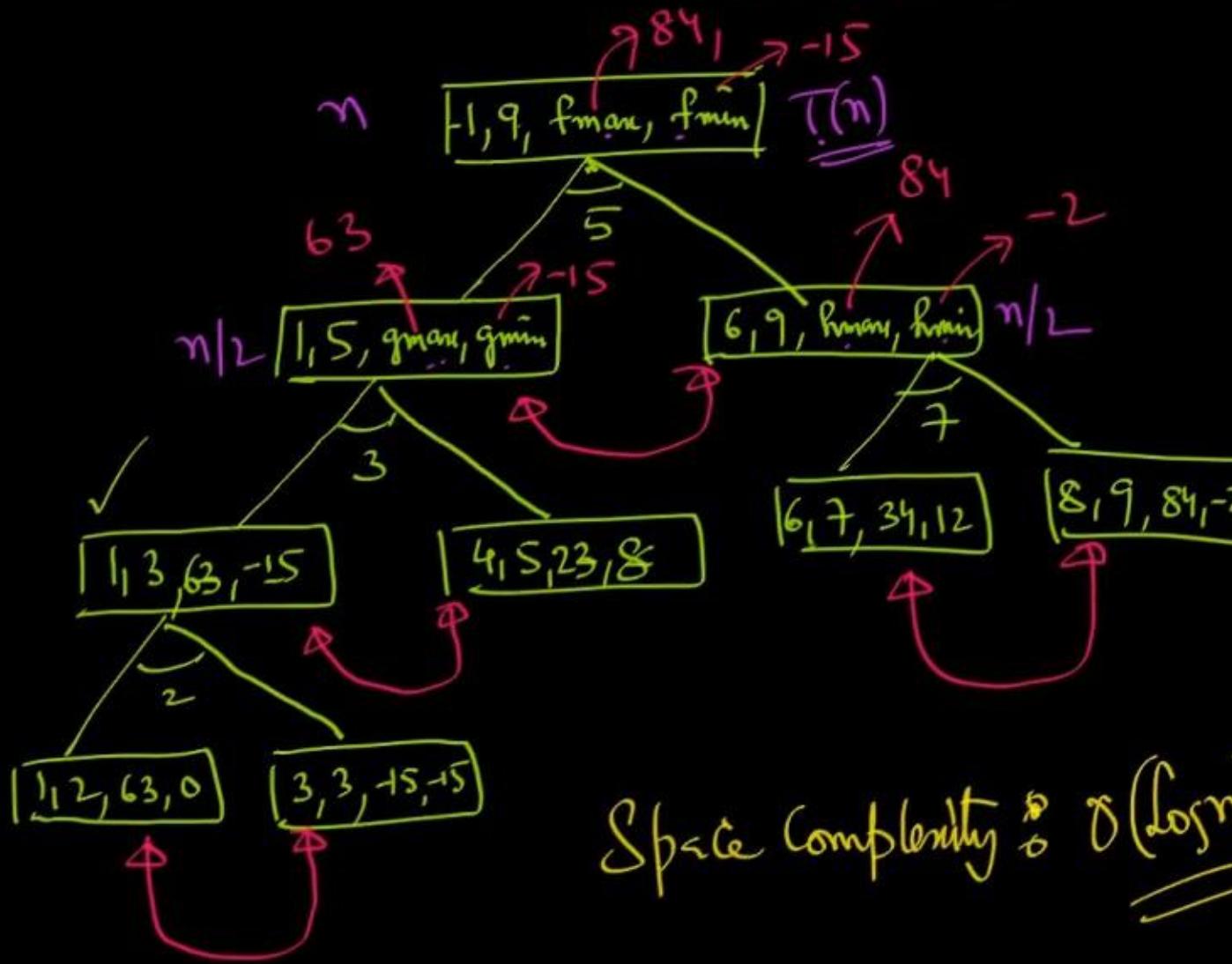


(5)



DandC-MaxMin:

|    |   |     |    |   |    |    |    |    |
|----|---|-----|----|---|----|----|----|----|
| 1  | 2 | 3   | 4  | 5 | 6  | 7  | 8  | 9  |
| 63 | 0 | -15 | 23 | 8 | 12 | 34 | 84 | -2 |



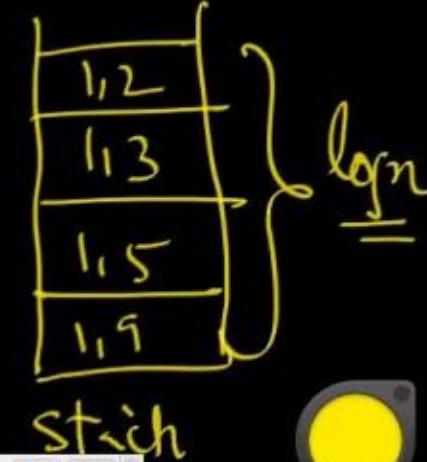
Let  $T(n)$  repr. the No. of Element Comparisons involved in DandC MaxMin ( $n$ )

$$T(n) = 0 \quad , \quad n=1$$

$$= 1 \quad , \quad n=2$$

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

$$T(n) = 2 \cdot T(n/2) + 2$$



$$T(n) = 2 \cdot T(n/2) + 2 \quad \text{---(1)}$$

$$T(n/2) = 2 \cdot T(n/4) + 2 \quad \text{---(2)}$$

$$T(n) = 2 \left[ 2T(n/4) + 2 \right] + 2$$

$$= 4T(n/4) + 4 + 2$$

$$= 2^2 T(n/2^2) + \sum_{i=1}^2 2^i$$

$$= 2^3 T(n/2^3) + \sum_{i=1}^3 2^i$$

$$T(n) = 2^K T(n/2^K) + \sum_{i=1}^K 2^i$$

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

$$= \frac{\log n}{2} + 2 - 2$$

$$= \frac{n}{2} + n - 2$$

$$T(n) = \left( \frac{3n}{2} - 2 \right)$$

No. of Element Comparisons  
(Decreasing) } All-Cases

$$\frac{n}{2^K} = 2$$

$$n = 2^{K+1}$$

$$K+1 = \log n$$

$$K = \underline{\log n - 1}$$

Perf. ComparisonComparisons

1) Inc

$$D_{andC} : \left(\frac{3n}{2} - 2\right)$$

$$\text{Straight Mem Min: } (n-1) \checkmark$$

$$\underline{\mathcal{O}(n)}$$

2) DCY

$$D_{and}: \left(\frac{3n}{2} - 2\right) \checkmark$$

$$\text{Straight Mem Min: } 2(n-1)$$

3) Avg. Gen:

$$D_{andC} : \left(\frac{3n}{2} - 2\right) \checkmark$$

$$\text{Straight Mem Min: } \left(\frac{3n}{2} - 1\right)$$



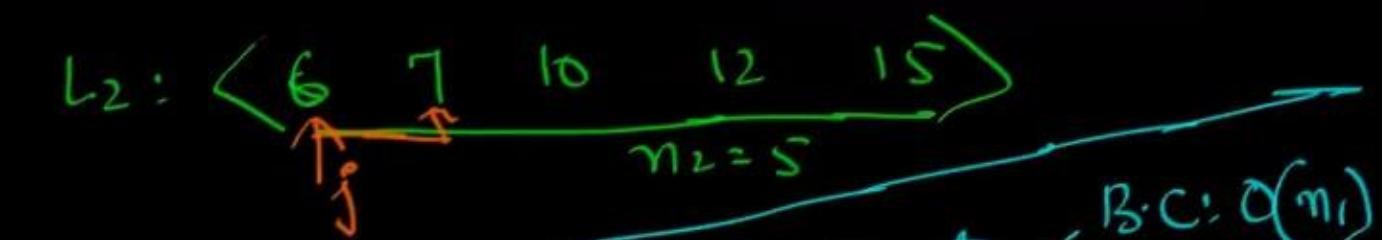
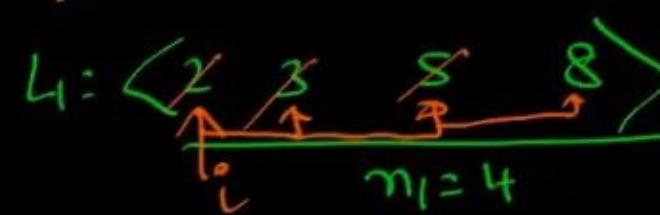
## Divide and Conquer (DandC):

### 2) MergeSort:

↳ Principle of Merging (Conquer)

↳ refers to Combining two sorted lists of  
may be diff. sizes;

\* Given two sorted lists  $L_1$  &  $L_2$ , having  $n_1$  &  $n_2$  elements ( $n_1 \leq n_2$ )  
Then it is required to merge them using 2-way merging to get  
a single sorted list;



$L: \langle 2, 3, 5, 6, \dots, 10, 12, 15 \rangle$

Time Complexity =  $O(n_1 + n_2)$

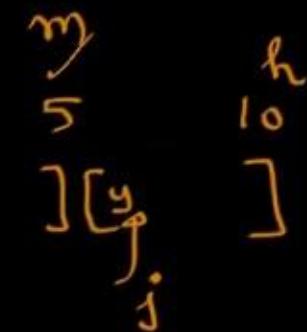
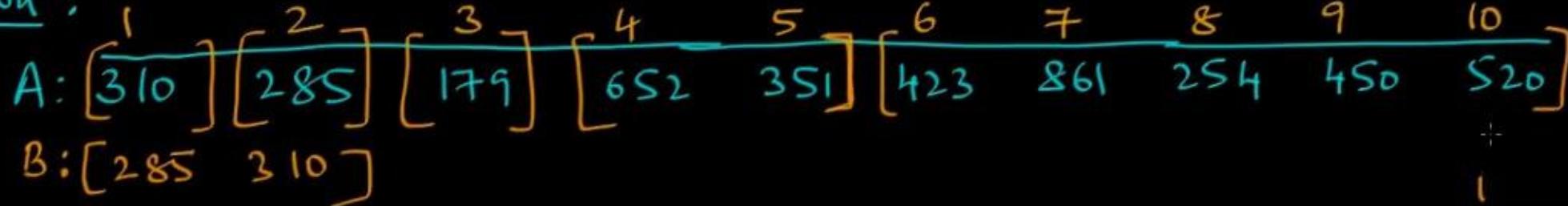
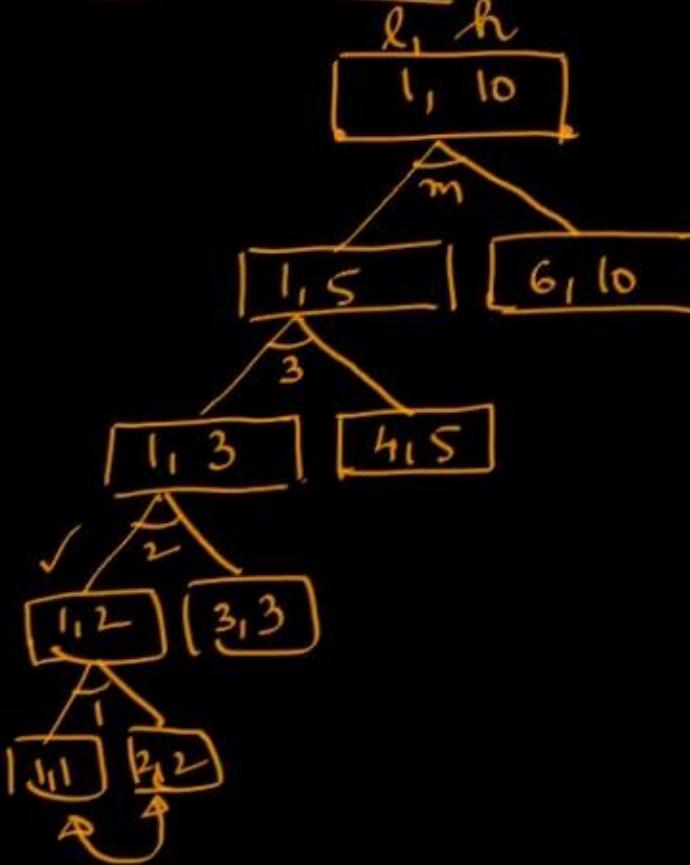
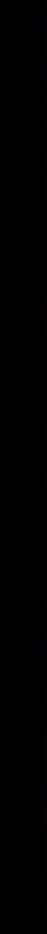
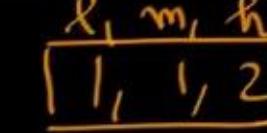
B.C:  $\Theta(n_1)$   
W.C:  $O(n_1 + n_2)$

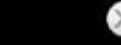
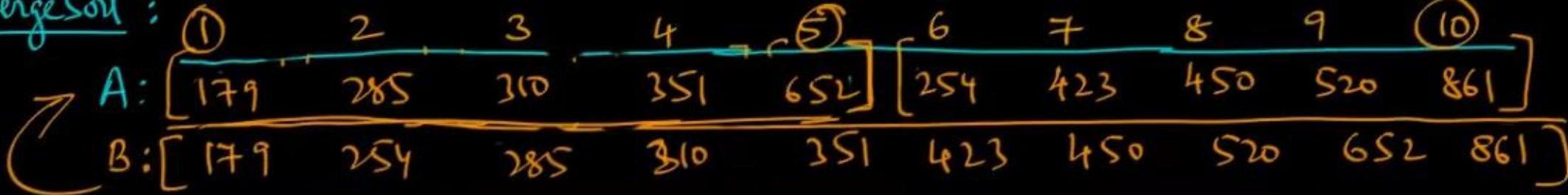
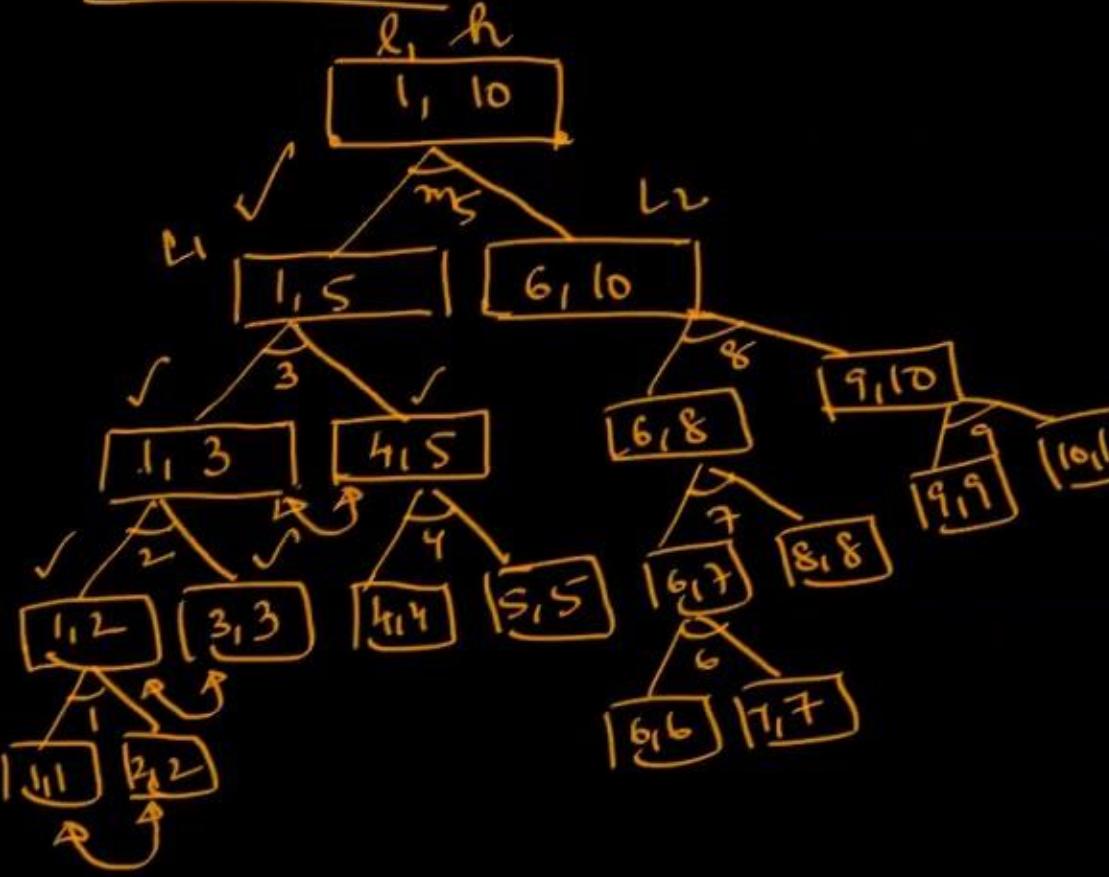
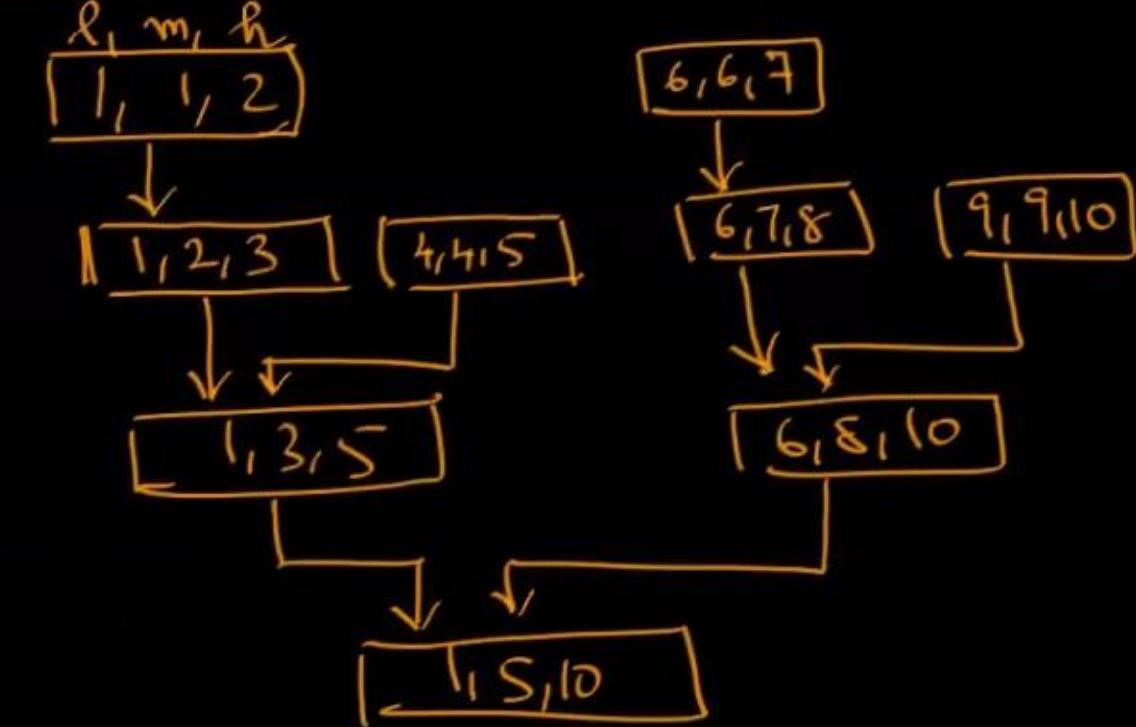
E<sub>n</sub>: B.C      L<sub>1</sub>: < <sup>n<sub>1</sub></sup> 2 3 5 6 >      L<sub>2</sub>: < <sup>n<sub>2</sub></sup> 10, 12, 15, 18, 25 >  
(L<sub>1</sub>) < f<sub>max</sub> | L<sub>2</sub>) : (n<sub>1</sub>) : Min

N.Cn<sub>1</sub>L<sub>1</sub>: < 4 5 8 10 40 >n<sub>2</sub>L<sub>2</sub>: < 12 15 18 22 25 30 32 >(n<sub>1</sub>-1) | L<sub>1</sub> | < f<sub>max</sub> | L<sub>2</sub>)Last |L<sub>1</sub>| > |L<sub>2</sub>)

L: &lt; 4 5 8 10 - - - 40 &gt;

Total Comparisons: (n<sub>1</sub>-1) + n<sub>2</sub> = (n<sub>1</sub>+n<sub>2</sub>-1) : Max

MergeSort:Divide Tree:Merge-Tree:

MergeSort:Divide-Tree:Merge-Tree:

Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

Export PDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Horowitz Sahni.pdf  
1 file / 30.95 MB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files

Algorithm 3.8 Merging two sorted subarrays using auxiliary storage

```
1 Algorithm Merge(low, mid, high)
2 // a[low : high] is a global array containing two sorted
3 // subsets in a[low : mid] and in a[mid + 1 : high]. The goal
4 // is to merge these two sets into a single set residing
5 // in a[low : high]. b[ ] is an auxiliary global array.
6 {
7     h := low; i := low; j := mid + 1;
8     while ((h ≤ mid) and (j ≤ high)) do
9     {
10         if (a[h] ≤ a[j]) then
11             {
12                 b[i] := a[h]; h := h + 1;
13             }
14         else
15             {
16                 b[i] := a[j]; j := j + 1;
17             }
18         i := i + 1;
19     }
20     if (h > mid) then
21         for k := j to high do
22         {
23             b[i] := a[k]; i := i + 1;
24         }
25     else
26         for k := h to mid do
27         {
28             b[i] := a[k]; i := i + 1;
29         }
30     for k := low to high do a[k] := b[k];
31 }
```

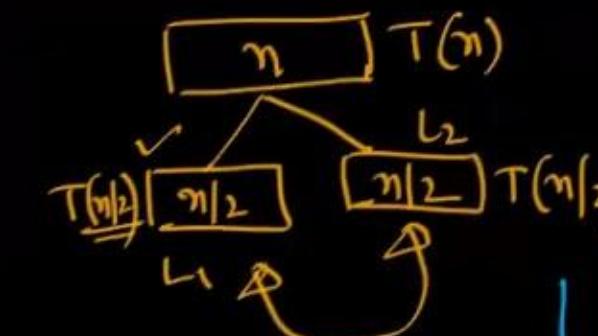
Perf. Analysis of M-S:(i) Time Complexity: Let  $T(n)$  repr. T.C of DandC-MS( $n$ )

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

$$T(n) = 2T(n/2) + bn \quad \text{--- (1)}$$

$$O(n \cdot \log n) \rightsquigarrow (n \log n)$$

$\therefore T(n)$  is  $\Theta(n \log n)$ : In All Cases of If, it is  $n \log n$



$$\xrightarrow{\text{Merging}} \text{B.C.: } \left\{ \frac{n}{2} \right\} : O(n)$$

$$\xrightarrow{\text{W.C.: }} \left\{ (n-1) \right\}$$

Space Complexity:

$$\text{WorkSpace: } (n + \log n)$$

$O(n)$ : NOT-IN PLACE

# 2-way MergeSort / Bottom-up MS:

A: (310 285 179 625 351 423 861 254)

$n=2^k$   
No. of Passes

Min

Max

Time:  $O(n \log n)$

$$\frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 3 + \frac{n}{8} \cdot 7 \dots$$

$$\frac{n}{4} \cdot 2 + \frac{n}{8} \cdot 4 \dots$$

$\log n$

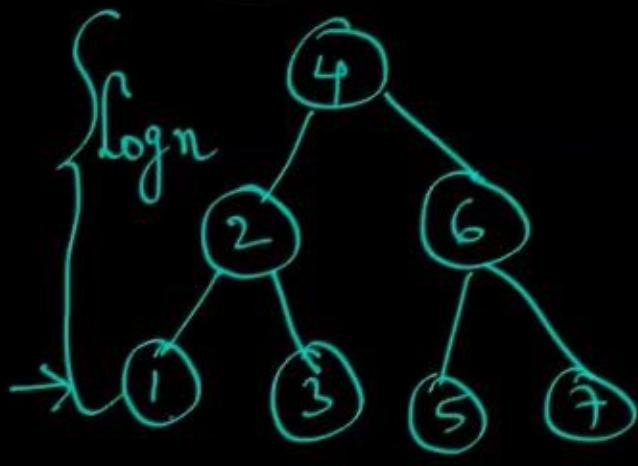
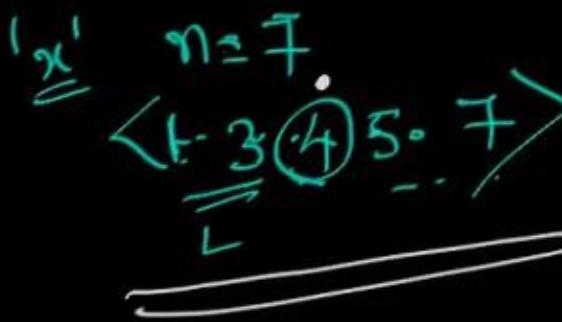
$$\frac{n}{2} \cdot 1 + \frac{n}{4} \cdot 3 + \frac{n}{8} \cdot 7 \dots$$

Chaining: 'Q'.

Evaluate the min & max no. of comparisons involved in 2-way MS with  $n$  elements ( $n=2^k$ )

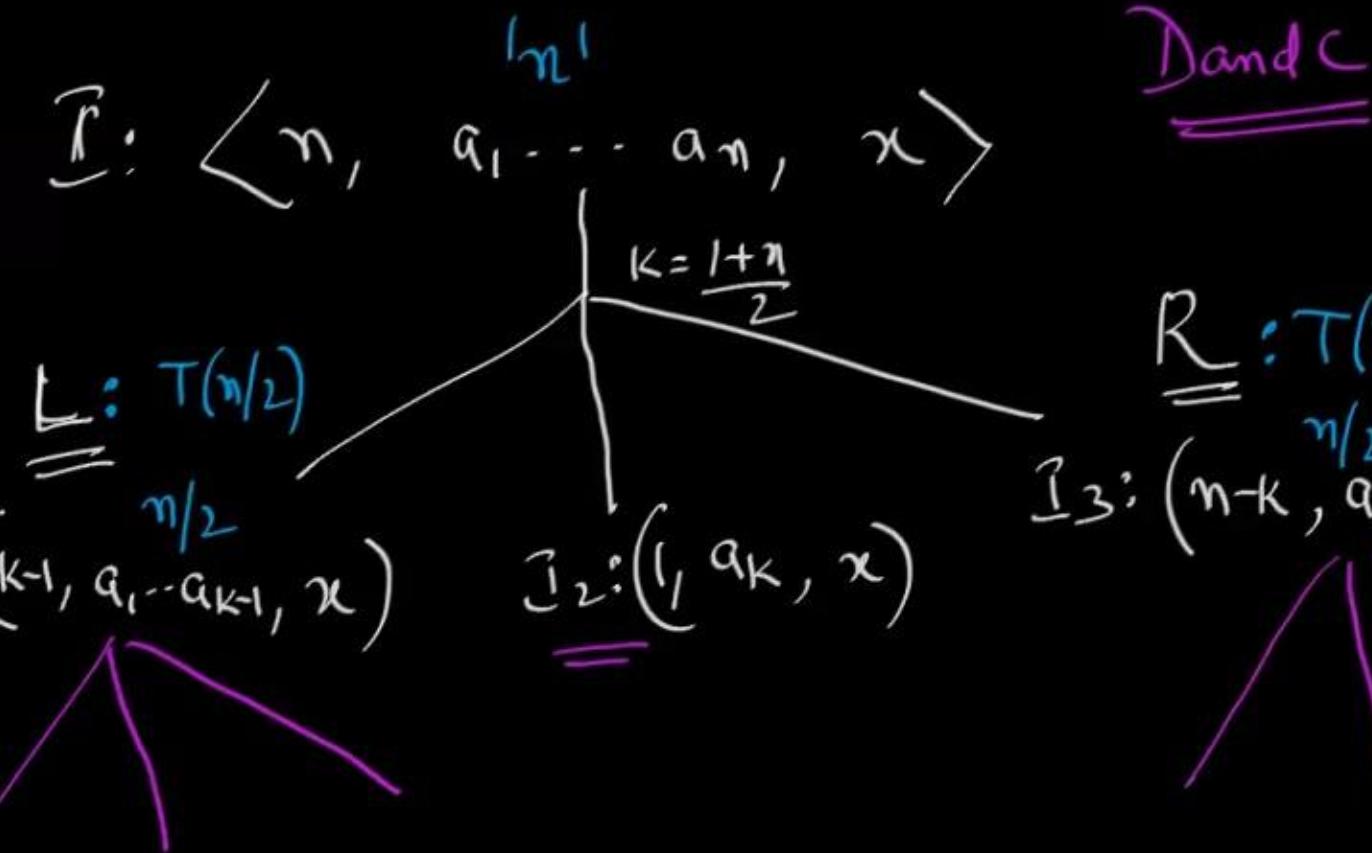


3) Binary Search: Requirement: Elements in the array must be sorted:



BST:  $O(\log n)$

Succ - unsucc



```
1 Algorithm BinSearch(a, n, x)
2 // Given an array a[1 : n] of elements in nondecreasing
3 // order, n ≥ 0, determine whether x is present, and
4 // if so, return j such that x = a[j]; else return 0.
5 {
6     low := 1; high := n;
7     while (low ≤ high) do
8     {
9         mid := ⌊(low + high)/2⌋;
10        if (x < a[mid]) then high := mid - 1; L
11        else if (x > a[mid]) then low := mid + 1; R
12        else return mid;
13    }
14    return 0;
15 }
```

---

**Algorithm 3.3** Iterative binary search

that *low* and *high* are integer variables such that each time through the loop either *x* is found or *low* is increased by at least one or *high* is decreased by at least one. Thus we have two sequences of integers approaching each other and eventually *low* becomes greater than *high* and causes termination in a finite number of steps if *x* is not present.

**Example 3.6** Let us select the 14 entries

-15, -6, 0, 7, 9, 23, 54, 82, 101, 112, 125, 131, 142, 151

place them in *a[1 : 14]*, and simulate the steps that BinSearch goes through as it searches for different values of *x*. Only the variables *low*, *high*, and

A screenshot of a software window titled "Export PDF" from "Adobe ExportPDF". It shows a list of files under "Selected PDF File" with "Horowitz Sahni.pdf" selected. Below it, "Convert To:" is set to "Microsoft Word (\*.docx)". There are buttons for "Convert", "Create PDF", "Edit PDF", "Combine PDF", "Send Files", and "Store Files". A sidebar on the right lists "Recognize Text in English(U.S.)" and a "Change" button.

Q&: Let  $T(n)$  repr. T.C of DandC-BS( $n$ )

$$\begin{aligned} \underline{\text{W-C}}: \quad & \left\{ \begin{array}{l} T(n) = c, \quad n=1 \\ = a + T(n/2), \quad n>1 \end{array} \right. \end{aligned} \quad \left. \begin{array}{c} \text{B-C:} \\ T(n) = O(1) \end{array} \right\} \quad T(n) = O(\log n)$$

Space-Complexity:

(i)  $O(1)$ : Iterative

(ii)  $O(\log n)$ : Recursive

4) Quicksort (Partition-Exchange Sort): T. Hoare;  $P \xrightarrow{QS} O(n^2)$   
 $m \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad P \quad 10(n+1) \xrightarrow{MS} O(n \log n)$  ✓  
 $A: \underline{65} \quad 70 \quad 75 \quad 80 \quad 85 \quad 60 \quad 55 \quad 50 \quad 45 \quad \underline{\infty(9999)}$

$v \leftarrow A[1] = \underline{\underline{65}}$

$v \leftarrow 65 \quad i \quad j$   
 $\underline{1} \quad \underline{10}$   
 $2 \quad 9$

```

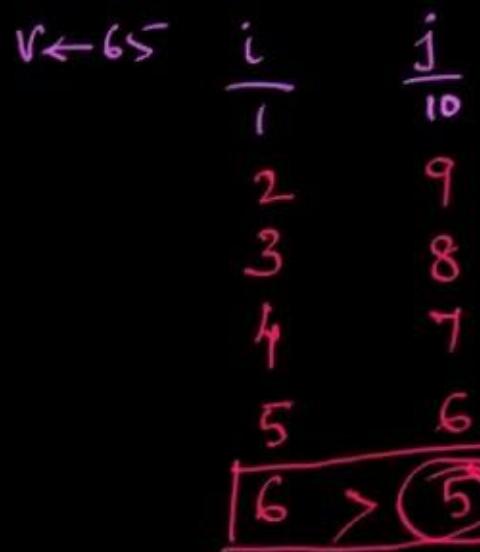
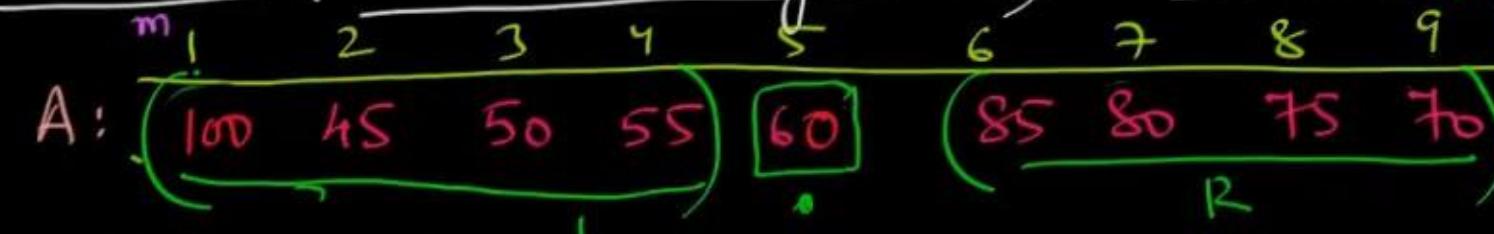
1 Algorithm Partition( $a, m, p$ )
2 // Within  $a[m], a[m + 1], \dots, a[p - 1]$  the elements are
3 // rearranged in such a manner that if initially  $t = a[m]$ ,
4 // then after completion  $a[q] = t$  for some  $q$  between  $m$ 
5 // and  $p - 1$ ,  $a[k] \leq t$  for  $m \leq k < q$ , and  $a[k] \geq t$ 
6 // for  $q < k < p$ .  $q$  is returned. Set  $a[p] = \infty$ .
7 {
8      $v := a[m]; i := m; j := p;$ 
9     repeat
10    {
11        repeat
12             $i := i + 1;$  ) L-R
13        until ( $a[i] \geq v$ );
14        repeat
15             $j := j - 1;$  ) R-L
16        until ( $a[j] \leq v$ );
17        if ( $i < j$ ) then Interchange( $a, i, j$ );
18    } until ( $i \geq j$ );
19     $a[m] := a[j]; a[j] := v; \text{return } j;$ 
20 }
```

```

1 Algorithm Interchange( $a, i, j$ )
2 // Exchange  $a[i]$  with  $a[j]$ .
3 {
4      $p := a[i];$ 
5      $a[i] := a[j]; a[j] := p;$ 
6 }
```

} Swap

# 4) Quicksort (Partition-Exchange Sort): T. Hoare



Note: Impl. of QS needs  
 $(n+1)^{\text{th}}$  element initialized  
 to  $\infty(9999)$ ? • to avoid an Inf. loop

Time Complexity of Partition:  $\underline{\underline{O(n)}}$

```

1 Algorithm QuickSort( $p, q$ )
2 // Sorts the elements  $a[p], \dots, a[q]$  which reside in the global
3 // array  $a[1 : n]$  into ascending order;  $a[n + 1]$  is considered to
4 // be defined and must be  $\geq$  all the elements in  $a[1 : n]$ .
5 {
6     if ( $p < q$ ) then // If there are more than one element
7     {
8         // divide P into two subproblems.
9          $j := \text{Partition}(a, p, q + 1);$ 
10        // j is the position of the partitioning element.
11        // Solve the subproblems.
12        QuickSort( $p, j - 1$ );
13        QuickSort( $j + 1, q$ );
14    } // There is no need for combining solutions.
15 }
16 }
```

P  $\frac{QS}{\infty} \rightarrow O(n^2)$   
 $\infty \frac{MS}{\infty} \rightarrow O(n \log n)$

$v \leftarrow A[1] = 65$

```

1 Algorithm Partition( $a, m, p$ )
2 // Within  $a[m], a[m + 1], \dots, a[p - 1]$  the elements are
3 // rearranged in such a manner that if initially  $t = a[m]$ ,
4 // then after completion  $a[q] = t$  for some  $q$  between  $m$ 
5 // and  $p - 1$ ,  $a[k] \leq t$  for  $m \leq k < q$ , and  $a[k] \geq t$ 
6 // for  $q < k < p$ .  $q$  is returned. Set  $a[p] = \infty$ .
7 {
8      $v := a[m]; i := m; j := p;$ 
9     repeat
10    {
11        repeat
12             $i := i + 1;$ 
13        until ( $a[i] \geq v$ );
14        repeat
15             $j := j - 1;$ 
16        until ( $a[j] \leq v$ );
17        if ( $i < j$ ) then Interchange( $a, i, j$ );
18    } until ( $i \geq j$ );
19     $a[m] := a[j]; a[j] := v; \text{return } j;$ 
20 }
```

```

1 Algorithm Interchange( $a, i, j$ )
2 // Exchange  $a[i]$  with  $a[j]$ .
3 {
4      $p := a[i];$ 
5      $a[i] := a[j]; a[j] := p;$ 
6 }
```

} Swap

$A[10] \cdot A[15]$

Analysis of & S:Space:  $\Theta(\log n)$ Best Case  
unsorted $T(n) \stackrel{?}{=} \langle a_1 \ a_2 \ a_3 \ \dots \ a_n \rangle$ Avg. Case (AS):  $\Theta(n \log n)$ Space:  $\Theta(n)$ Partition:  $\Theta(n)$ Worst-CaseSlow

Case II

 $i_1$  $\left\langle \left( \frac{n}{2} \right) \boxed{a_1} \left( \frac{n}{2} \right) \right\rangle$ Work faster

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

$= \Theta(n) + 2T(n/2) : \Theta(n \log n)$

Case I  
Sorted $i_2$  $\left\langle \left( \frac{n-1}{2} \right) \boxed{a_1} \left( \frac{n-1}{2} \right) \right\rangle$  $(n-2) \boxed{a_1} \boxed{a_2}$  $i_3$  $\left\langle \boxed{a_1} \left( \frac{n-1}{2} \right) \right\rangle$  $\boxed{a_1} \boxed{a_2} \left( \frac{n-1}{2} \right)$ 

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

$= \Theta(n) + \underline{\underline{T(n-1)}} : \Theta(n^2)$

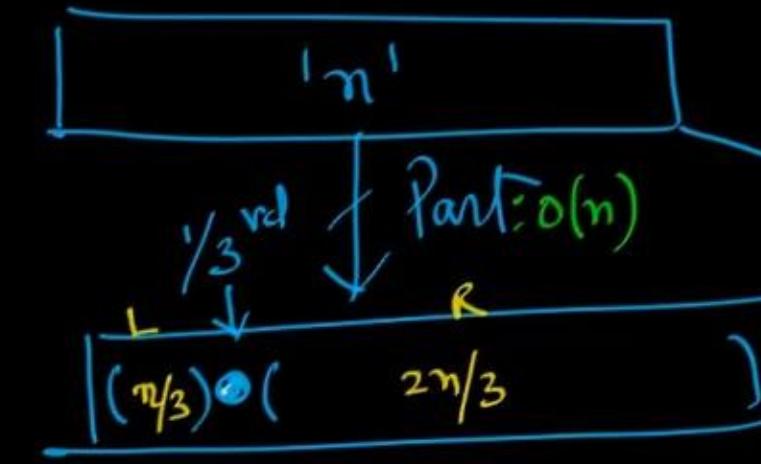
\* Best case arises in QS, whenever Pivot gets pivoted in the middle of the list thereby creating two partitions;

$n \rightarrow \text{V.L. Value}$   
100,000

$$T(n) = O(n) + T(\alpha n) + T((1-\alpha)n)$$

$$0 < \alpha < 1$$

$$O(n \log n)$$



$$T(n) = O(n) + T(n/3) + T(2n/3)$$

$$1/5^n : O(n)$$



$$T(n) = T(n/5) + T(4n/5) + O(n)$$

(Recursion Tree)  $\Rightarrow O(n \log n)$

## 5) Matrix Multiplication :

(Square) Given  $A [1 \dots n, 1 \dots n]_{n \times n}$   $B [1 \dots n, 1 \dots n]_{n \times n}$   $C [1 \dots n, 1 \dots n]_{n \times n}$

a)  $A \pm B = C$  : Time Complexity :  $O(n^2)$

for  $i \leftarrow 1 \text{ to } n/2$  }  $O(n^2/4) = O(n^2)$   
 for  $j \leftarrow 1 \text{ to } n/2$  }  
 $c[i, j] = A[i, j] \pm B[i, j]$

$$A \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}_{2 \times 2} * B \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}_{2 \times 2}$$

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \\ = C \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}_{2 \times 2}$$

b)  $A * B = C$  : Time Complexity :  $O(n^3)$  : General Method

for  $i \leftarrow 1 \text{ to } n$   
 for  $j \leftarrow 1 \text{ to } n$   
 $c[i, j] = 0$   
 for  $k \leftarrow 1 \text{ to } n$   
 $c[i, j] = c[i, j] + A[i, k] * B[k, j]$

$O(1)$ : Space

Non-DC  
School Method

$$A_{m \times n} * B_{n \times p} = C_{m \times p} \quad O(\underline{m \times n \times p})$$

$$A \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{2 \times 3} B \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}_{3 \times 3} = C \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{2 \times 2}$$

$$(10) \times 3 = (30)$$

X

$$\textcircled{2 \times 3 \times 5}$$

Can we multiply Two Square Matrices of order  $n \times n$ , using DandC

$$A = \left[ \begin{array}{cc|cc} A_{11} & A_{12} & & \\ \hline 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 1 & 2 & 3 \\ 5 & 6 & 7 & 8 \end{array} \right]_{4 \times 4}$$

$$B = \left[ \begin{array}{cc|cc} B_{11} & B_{12} & & \\ \hline 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 8 \\ \hline 2 & 3 & 1 & 5 \\ 8 & 2 & 3 & 5 \end{array} \right]_{4 \times 4} = C \left[ \begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array} \right]_{4 \times 4}$$

Space complexity:  $\mathcal{O}(n^2)$

$$C_{11} = (A_{11} \cdot B_{11}) \oplus (A_{12} \cdot B_{21}) \quad \text{--- (1)}$$

$$C_{12} = A_{11} \cdot B_{12} \oplus A_{12} \cdot B_{22} \quad \text{--- (2)}$$

$$C_{21} = A_{21} \cdot B_{11} \oplus A_{22} \cdot B_{21} \quad \text{--- (3)}$$

$$C_{22} = A_{21} \cdot B_{12} \oplus A_{22} \cdot B_{22} \quad \text{--- (4)}$$

$T(n) : (n \times n)$

$$T(n) = c, \quad n \leq 2$$

$$= 8T(n/2) + b n^2, \quad n > 2$$

$$\boxed{T(n) = 8T(n/2) + b n^2}$$

$(n \times n)$

$(\frac{n}{2} \times \frac{n}{2})$

$(\frac{n}{4} \times \frac{n}{4})$

$(\frac{n}{4} \times \frac{n}{4})$

$$T(n) = 8T(n/2) + bn^2 \Rightarrow \underline{\underline{O(n^3)}} : \underline{\underline{\text{DandC}}}$$

Strassen's method of Matrix Multiplication

(extension to DandC)

$$T(n) = \underbrace{8T(n/2)}_{\Theta(n^3)} + \underbrace{bn^2}_{\Theta(n^3)} \Rightarrow \Theta(n^3) : \underline{\underline{\text{DandC}}}$$

Strassen's method of Matrix Multiplication

(extension to DandC)

Multiplication  
= (Repeated  
Additions)

$$a * b = (a + a + a + \dots + a)$$

A, B, C:  $n \times n$  matrices

$A_{ij}, B_{ij}, C_{ij} : n/2 \times n/2 \quad (1 \leq i, j \leq n)$

P, Q, R, S, T, U, V :  $\left(\frac{n}{2} \times \frac{n}{2}\right)$





$$\begin{aligned}P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\Q &= (A_{21} + A_{22})B_{11} \\R &= A_{11}(B_{12} - B_{22}) \\S &= A_{22}(B_{21} - B_{11}) \\T &= (A_{11} + A_{12})B_{22} \\U &= (A_{21} - A_{11})(B_{11} + B_{12}) \\V &= (A_{12} - A_{22})(B_{21} + B_{22})\end{aligned}$$

$$\left\{ \begin{array}{l} C_{11} = P + S - T + V \\ C_{12} = R + T \\ C_{21} = Q + S \\ C_{22} = P + R - Q + U \end{array} \right.$$

{ } { }

(3.13)

{ } { }

(3.14)

7: Multip  
=  
18: Add +  
Sub

The resulting recurrence relation for  $T(n)$  is

$$T(n) = \begin{cases} b & n \leq 2 \\ 7T(n/2) + an^2 & n > 2 \end{cases} \quad (3.15)$$

where  $a$  and  $b$  are constants. Working with this formula, we get

$$\begin{aligned}T(n) &= an^2[1 + 7/4 + (7/4)^2 + \cdots + (7/4)^{k-1}] + 7^k T(1) \\&\leq cn^2(7/4)^{\log_2 n} + 7^{\log_2 n}, \text{ } c \text{ a constant}\end{aligned}$$

$\log_4 4 + \log_4 7 + \log_4 4 + \cdots + \log_4 7$

Export PDF

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
Horowitz Sahni.pdf  
1 file / 30.95 MB

Convert To:  
Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

Create PDF

Edit PDF

Combine PDF

Send Files

Store Files

## Time Complexity of Strassen's Approach:

$$T(n) = c, \quad n \leq 2$$

$$= 7 \cdot T(n/2) + b n^2, \quad n > 1$$

$$T(n) = 7 \cdot T(n/2) + b n^2 \quad \textcircled{1}$$

$$T(n/2) = 7 \cdot T(n/4) + b n^2/4 \quad \textcircled{2}$$

$$T(n) = 7 \left( 7T(n/4) + b n^2/4 \right) + b n^2$$

$$= 49T(n/4) + \left(\frac{7}{4}\right)^1 b n^2 + b n^2 \cdot \left(\frac{7}{4}\right)^0$$

$$= 7^2 \cdot T(n/2) + b n^2 \sum_{i=0}^1 \left(\frac{7}{4}\right)^i$$

$$\sum_{i=1}^n x^i < x^{n+1} \quad \left| \begin{array}{l} \frac{n}{2^K} = 1 \\ x = 7 \end{array} \right. \Rightarrow n = 2^K$$

$$T(n) = 7^K \cdot T(n/2^K) + b n^2 \cdot \sum_{i=0}^{K-1} \left(\frac{7}{4}\right)^i$$

$$< 7^{\log_2 n} \cdot T(1) + b n^2 \cdot \left(\frac{7}{4}\right)^K$$

$$< n^{\log_2 7} c + b n^2 \cdot \frac{n^{\log_2 7}}{n^K}$$

$$T(n) < d \cdot n^{\log_2 7} < d \cdot n^{2.81}$$

$$\therefore T(n) : O(n^{2.81})$$



## \* Master Method / Theorem for Solving Divide & Conquer Recurrences:

$$\boxed{\begin{aligned} T(n) &= a \cdot T\left(\frac{n}{b}\right) + f(n), \quad n > d \\ &= c \quad , \quad n \leq d \end{aligned}}$$

$a \geq 1$ ;  $b > 1$ ;  $f(n)$  is +ve fn

Case I: If  $f(n)$  is  $\mathcal{O}(n^{\log_b a - \epsilon})$  for some  $\epsilon > 0$ , then  $T(n)$  is  $\underline{\mathcal{O}(n^{\log_b a})}$

Case II: If  $f(n)$  is  $\mathcal{O}(n^{\log_b a} \cdot \log^k n)$  for some ' $k$ ', s.t

a)  $k \geq 0$ , then  $T(n)$  is  $\mathcal{O}(n^{\log_b a} \cdot \log^{k+1} n)$

b)  $k = -1$ , then  $T(n)$  is  $\mathcal{O}(n^{\log_b a} \cdot \log \log n)$

c)  $k < -1$ , then  $T(n)$  is  $\mathcal{O}(n^{\log_b a})$

Case III: If  $f(n)$  is  $\mathcal{O}(n^{\log_b a + \epsilon})$  and  $a \cdot f(n/b) \leq \delta \cdot f(n)$  for some  $\epsilon > 0$  and  $\delta < 1$  then  $T(n)$  is  $\Theta(f(n))$

$$1) T(n) = 4T(n/2) + n \Rightarrow O(n^2)$$

$$a=4$$

$$b=2$$

$$f(n)=n$$

$$\log_2 4 = 2$$

$$\text{Case I: } n \text{ is it } O(n^{2-\epsilon}) \quad \epsilon=1$$

$$\therefore T(n) \text{ is } \Theta(n^2)$$

$$2) T(n) = 2T(n/2) + n \cdot \log n$$

$$a=2; b=2; f(n)=n \log n$$

$$\log_2 2 = 1$$

Case I:  $n \log n$  is it  $O(n^{1-\epsilon})$  X

Case II:  $n \log n$  is it  $\Theta(n \cdot \log^k n)$

$$k=1$$

$$a) T(n) \text{ is } \Theta(n \cdot \log^2 n)$$

$$3) T(n) = T(n/3) + n$$

$$a=1; \quad b=3; \quad f(n)=n$$

$$\log_3 1 = 0$$

I:  $n$  is it  $\mathcal{O}(n^{0-\epsilon})$  ×

II:  $n$  is it  $\Theta(n^0 \cdot \log n)$  ×

III:  $n$  is it  $\Omega(n^{0+\epsilon})$  ✓  
 $\epsilon = 1$

$$a \cdot f(n/b) \leq \delta \cdot f(n)$$

$$\boxed{1 \cdot \frac{n}{3} \leq \delta \cdot n}$$

$$\delta = \frac{1}{3} < 1$$

∴  $T(n)$  is  $\Theta(n)$  ✓

$$4) T(n) = 9 \cdot T(n/3) + n^{2.5}$$

$$a=9; \quad b=3; \quad f(n)=n^{2.5} = f(n/3) = (n/3)^{2.5} = \frac{n^{2.5}}{3^{2.5}}$$

$$\log_3 9 = 2$$

I:  $n^{2.5}$  is it  $\mathcal{O}(n^{2-\epsilon})$  ×

II:  $n^{2.5}$  is it  $\Theta(n^2 \cdot \log n)$  ×

III.  $n^{2.5}$  is it  $\Omega(n^{2+\epsilon})$  ✓  $\epsilon = 0.5$

$$a \cdot f(n/b) \leq \delta \cdot f(n)$$

$$\checkmark \frac{n^{2.5}}{3^{2.5}} \leq \delta \cdot n^{2.5} \quad \delta = \frac{1}{\sqrt{3}} < 1$$

∴  $T(n)$  is  $\Theta(n^{2.5})$  ✓



1) Mau-Mim:  $T(n) = 2T(\frac{n}{2}) + 2 \quad (\frac{3n}{2} - 2) = O(n)$  ✓

$$a=2; b=2; f(n)=2$$

I:  $n$  is  $O(n^{1-\epsilon})$   $\epsilon=1$   $\therefore T(n) = O(n)$  ✓

2) MergeSort:  $T(n) = 2T(\frac{n}{2}) + n \Rightarrow O(n \underline{\log n})$

$$a=1; b=2; f(n)=n$$

I:  $n$  is  $O(n^{1-\epsilon})$   $\epsilon \geq 0$  ✗

II:  $n$  is  $\Theta(n \cdot \log^K n)$   $K=0$

a)  $T(n)$  is  $\Theta(n \cdot \log n)$  ✓

3) Binary Search:  $T(n) = T(n/2) + c$

$$a=1; b=2; f(n)=c$$

$$\log_2^1 = 0$$

I:  $c$  is if  $\Theta(n^{0-\epsilon})$   $\epsilon > 0$  X

II:  $c$  is if  $\Theta(n^0 \cdot \log^k n)$ ,  $k=0$

$\therefore T(n)$  is  $\Theta(\log n)$  ✓

4) Matrix Multiplication:

a) DandC:  $T(n) = 8T(n/2) + n^2 \Rightarrow n^2$  is if  $\Theta(n^{3-\epsilon})$   $\epsilon = 1$  ✓  
 $\therefore T(n)$  is  $\Theta(n^3)$  ✓

b) Strassen's Approach:  $T(n) = 7T(n/2) + n^2$   
 $n^2$  is if  $\Theta(n^{2.81-\epsilon})$  ✓  $\therefore T(n)$  is  $\Theta(n^{2.81})$  ✓

$$5) T(n) = 2T(\sqrt{n}) + \log n$$

$$\text{Let } n = 2^k$$

$$T(2^k) = 2T(2^{k/2}) + k \quad \textcircled{1}$$

$$\text{Let } T(2^k) = S(k)$$

$$T(2^{k/2}) = S(k/2)$$

$$\boxed{S(k) = 2S(k/2) + k} \quad \textcircled{2}$$

$$T(n) = 2T(n/2) + n$$

$$\underline{\mathcal{O}(n \log n)}$$

$$\underline{T(n) = T(\sqrt{n}) + C}$$

$$T(2^k) = T(2^{k/2}) + C$$

$$\boxed{S(k) = S(k/2) + C}$$

$$\Rightarrow \mathcal{O}(\log k)$$

$$\underline{\mathcal{O}(\log \log n)} \checkmark$$

$$\mathcal{O}(k \cdot \log k)$$

$$\mathcal{O}(\log n \cdot \log \log n)$$



- $T(n) = 2^n T\left(\frac{n}{2}\right) + n^n \times$   
a is not a constant; the number of subproblems should be fixed
- $T(n) = 2T\left(\frac{n}{2}\right) + \frac{n}{\log n}$   
non-polynomial difference between  $f(n)$  and  $n^{\log_b a}$  (see below;  
extended version applies)
- $T(n) = 0.5T\left(\frac{n}{2}\right) + n$   
 $a < 1$  cannot have less than one subproblem
- $T(n) = 64T\left(\frac{n}{8}\right) - n^2 \log n$   
 $f(n)$ , which is the combination time,  
is not positive
- $T(n) = T\left(\frac{n}{2}\right) + n(2 - \cos n)$   
case 3 but regularity violation.

In the second inadmissible example

Save Language Find in article Theme Contents

$$\begin{aligned}
 T(n) &= 2T\left(\frac{n}{2}\right) + \frac{n}{\log n} \\
 &= 2T\left(\frac{n}{2}\right) + n \cdot (\log n)^{-1} \\
 n \cdot (\log n)^{-1} &\in O(n^{k-\epsilon}) \times \\
 n \cdot (\log n)^{-1} &\in \Theta(n \cdot (\log n)^k) \quad k=-1 \\
 b) \quad T(n) &\in \underline{\Theta(n \cdot \log \log n)} \checkmark
 \end{aligned}$$

$$1) T(n) = 16T(n/4) + n$$

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

$$3) T(n) = 4T(n/2) + \log n$$

$$4) T(n) = 2T(n/2) + n^2$$

$$5) T(n) = 3T(n/3) + n/2$$

$$6) T(n) = \sqrt{2} \cdot T(n/2) + \sqrt{n}$$

1) Master Method

→ only the order

→ cannot get the value

2) Back Substitution

Value

order

3) Recursion Tree (A Symmetric Case)

↳ order

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

1. Assume that Merge Sort takes 30sec to Sort 64 elements in worst case.  
What is the approximate number of elements that can be Sorted in the Worst Case using Merge Sort using 6 minutes?

Aposteriori

$$\begin{array}{l} \cancel{30 \text{ s}} \\ \cancel{360 \text{ s}} \end{array} \quad \begin{array}{l} \cancel{64 \text{ elements}} \\ ? \end{array}$$

Time-Complexity :  $n \log n$  (Apriori-time)

$$\begin{array}{l} n = \underline{\underline{64}} \Rightarrow 64 \times \log_2 64 \text{ units} \\ \qquad \qquad \qquad \underline{\underline{6 \times 64 \text{ units}}} \end{array}$$

$$\begin{array}{l} \cancel{30 \text{ s}} \quad \cancel{6 \times 64 \text{ units}} \\ ? \quad \rightarrow 1 \text{ unit} \end{array}$$

$$1 \text{ unit} = \frac{30}{6 \times 64} \text{ s}$$

$$\frac{360 \times 6 \times 64 \text{ ms}}{30}$$

$$1 \text{ unit} \Rightarrow \frac{30}{6 \times 64} \text{ s}$$

$$? \quad \cancel{360 \text{ s}} \quad (n)$$

$$\left\{ \frac{360 \times 6 \times 64}{30} \text{ units} = n \log n \right\} \quad \cancel{n \log n}$$

$$4608 = n \cdot \log_2 n$$

$$2^K \cdot K = 4608$$

$$K = 9$$

$$\therefore n = 2^K = 2^9 = \underline{\underline{512}}$$

2. Consider the following recurrence relation

$$T(n) = \begin{cases} T(n/2) + T(2n/5) + 7n & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

Recursion tree

Which one of the following options is correct?

- (a)  $T(n) = \Theta(n \log n)$
- (b)  $T(n) = \Theta((\log n)^{5/2})$
- (c)  $T(n) = \Theta(n^{5/2})$
- (d)  $T(n) = \Theta(n)$  ✓

3. Consider the recurrence function

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

Then  $T(n)$  in terms of  $\Theta$  notation is

- (a)  $\Theta(\log \log n)$
- (b)  $\Theta(\log n)$  ✓
- (c)  $\Theta(\sqrt{n})$
- (d)  $\Theta(n)$

$$\begin{aligned} T(n) &= T(\sqrt{n}) + C \\ &\Rightarrow \Theta(\log \log n) \end{aligned}$$

4. For constants  $a \geq 1$  and  $b > 1$ , consider the following recurrence defined on the non-negative integers

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

621

Which one of the following options is correct about the recurrence  $T(n)$  ?

- (a) If  $f(n)$  is  $n \log_2(n)$ , then  $T(n)$  is  $\Theta(n \log_2(n))$
- (b) If  $f(n)$  is  $\frac{n}{\log_2(n)}$ , then  $T(n)$  is  $\Theta(\log_2(n))$
- (c) If  $f(n)$  is  $O(n^{\log_b(a)-\varepsilon})$  for  $\varepsilon > 0$ , then  $T(n)$  is  $\Theta(n^{\log_b(a)})$  Case i ✓
- (d) If  $f(n)$  is  $\Theta(n^{\log_b(a)})$ , then  $T(n)$  is  $\Theta(n^{\log_b(a)})$

5. Match the algorithms with their time complexities

**Algorithm**

(P) Towers of Hanoi with  $n$  disks

(Q) Binary search given  $n$  sorted numbers

(R) Heap sort given  $n$  numbers at the worst case

(S) Addition of two  $m \times n$  matrices

**Time complexity**

(i)  $\Theta(n^2)$

(ii)  $\Theta(n \log n)$

(iii)  $\Theta(2^n)$

(iv)  $\Theta(\log n)$

(a) P (iii), Q (iv), R (i), S (ii)

(b) P (iv), Q (iii), R (i), S (ii)

(c) P (iii), Q (iv), R (ii), S (i)

(d) P (iv), Q (iii), R (ii), S (i)

For Micro Notes by the Student



Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:  
 Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:  
 Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
 Change

Convert

► Create PDF  
► Edit PDF  
► Combine PDF  
► Send Files  
► Store Files

$$T(n) = 2T(n-1) + C \Rightarrow O(2^n)$$

## II: Greedy Method:

### Terminology:

- 1) Problem definition: Problem Statement
  - 2) Constraints <u>Implicit</u>: local ✓
  - 3) {Solution Space}: <u>Explicit</u>: Boundary
  - 4) Feasible Solutions: multiple
  - 5) objective function
  - 6) optimal solution
- GM  
DP  
BK  
BS

### n-Queens: ↴

n-Queens  $\langle Q_1 \dots Q_n \rangle$

CHBD:  $n \times n$

$n=4$   $\langle Q_1 \dots Q_4 \rangle :$

|       |   |   |   |   |
|-------|---|---|---|---|
|       | 1 | 2 | 3 | 4 |
| $Q_1$ | . |   |   |   |
| $Q_2$ |   |   | . |   |
| $Q_3$ |   | . |   |   |
| $Q_4$ |   |   |   | . |

$$\begin{array}{c} \langle x_1 \dots x_4 \rangle \\ \{1 \leq x_i \leq 4\} \\ \xrightarrow{x_1 \quad x_4} \langle 1 \quad 2 \quad 3 \quad 4 \rangle \end{array}$$

$$\langle 2 \quad 1 \quad 4 \quad 3 \rangle$$

$$\langle 3 \quad 4 \quad 2 \quad 1 \rangle \times$$

?

$$\begin{array}{c} \langle 3, 1, 4, 2 \rangle \checkmark \\ \langle 2, 4, 1, 3 \rangle \checkmark \end{array}$$

(i) Row  
(ii) Column  
(iii) Diagonal  
Attacking  
Implicit

Soln Space: All possible ways of organizing the IP's, but satisfying explicit constraints;

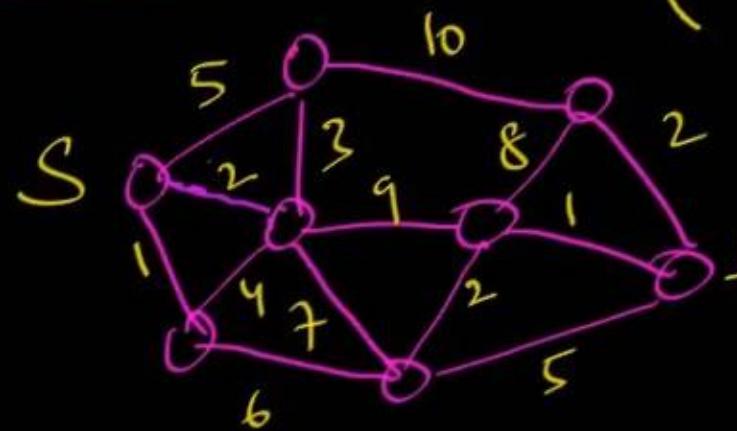
$$\begin{array}{c} \{ \\ \begin{array}{l} (1, 2, 3, 4) \\ (1, 2, 4, 3) \\ (1, 3, 2, 4) \\ (1, 3, 4, 2) \\ \vdots \end{array} \\ \} \\ n! \end{array}$$

✓ feasible Soln: Those Solutions in the Soln Space that satisfy implicit constraints;

✓ objective fn: Few Problems may have objective function; )  
objectivity refers to minimization/maximization of some criteria;

optimal Solution: is that feasible solution that satisfies (unique) (optimization problem) objective fn. (It always refer to value)

Shortest Path:



(Least Cost) : Minimize the distance  
↓  
dist

Problems

Decision

y/n

n - & s  
Banker's Algo  
Search

optimization

Opt-Sols

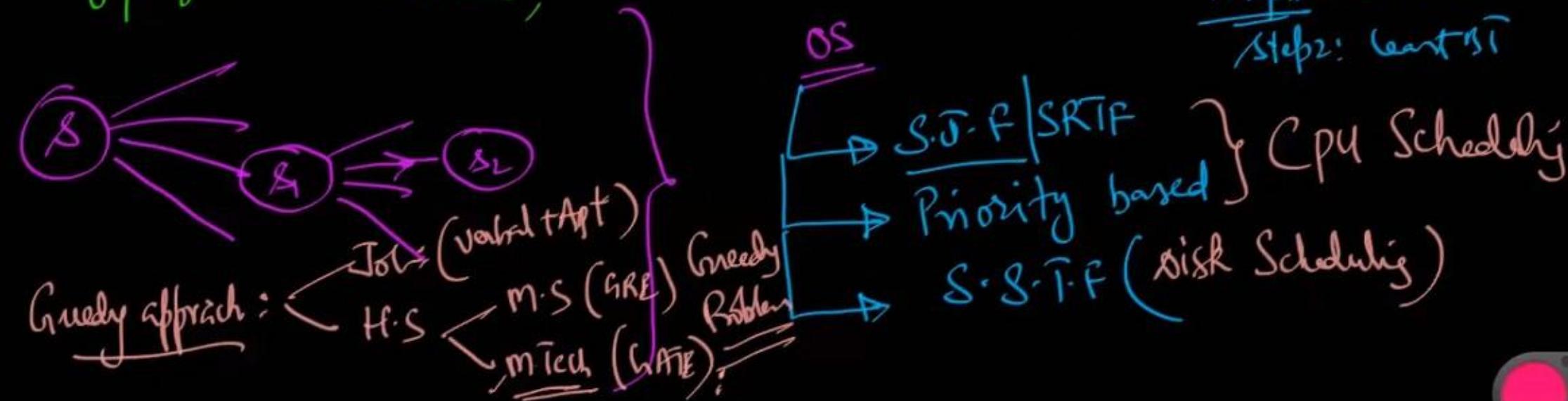
Maximize  
Lmin

Sht-Path  
CPU Scheduling  
KNAP SACK

Profit: criteria  
→ Max  
Penalty  
↳ Min

Greedy Method: is an Algorithm design strategy used for solving problems, whose solutions are viewed as a result of making a set/sequence of decisions;

These decisions are made in a step-wise (Step-by-Step) manner, such that at each step, out of all available options, greedily select that option that satisfies Feasibility / Objective Criteria;



Horowitz Sahni.pdf - Adobe Reader

File Edit View Tools Fill & Sign Comment Sign In

Open Export PDF

Selected PDF File: Horowitz Sahni.pdf 1 file / 30.95 MB

Convert To: Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change Convert

Create PDF Edit PDF Combine PDF Send Files Store Files

Min-time Complexity =  $\underline{\underline{O(n)}}$

---

```
1 Algorithm Greedy(a, n)
2 // a[1 : n] contains the n inputs.
3 {
4     solution := ∅; // Initialize the solution.
5     for i := 1 to n do
6     {
7         x := Select(a);
8         if Feasible(solution, x) then
9             solution := Union(solution, x);
10    }
11    return solution;
12 }
```

**Algorithm 4.1** Greedy method control abstraction for the subset paradigm

function `Greedy` describes the essential way that a greedy algorithm will look, once a particular problem is chosen and the functions `Select`, `Feasible`, and `Union` are properly implemented.

For problems that do not call for the selection of an optimal subset, in the greedy method we make decisions by considering the inputs in some order. Each decision is made using an optimization criterion that can be computed using decisions already made. Call this version of the greedy method the *ordering paradigm*. Sections 4.2, 4.3, 4.4, and 4.5 consider problems that fit the subset paradigm, and Sections 4.6, 4.7, and 4.8 consider problems that fit the ordering paradigm.

## EXERCISE

1. Write a control abstraction for the ordering paradigm.

## 4.2 KNAPSACK PROBLEM

Let us try to apply the greedy method to solve the knapsack problem. We are given  $n$  objects and a knapsack or bag. Object  $i$  has a weight  $w_i$  and the knapsack has a capacity  $m$ . If a fraction  $x_i$ ,  $0 \leq x_i \leq 1$ , of object  $i$  is placed into the knapsack, then a profit of  $p_i x_i$  is earned. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the

## Greedy Paradigm

(Subset Paradigm)

(we need to determine a subset)

Ex: {Knapsack  
JSO.}

(Ordering Paradigm)

→ (Organize the given IP's in some order)

Ex: (Opt-Merge pattern)

Applications $\rightarrow$  Filling)① Knapsack Problem:

$\rightarrow$  Given a Bag (KNAP) with capacity; 'M'

$\rightarrow$  'n' - objects ( $v_i$ )

Weight ( $w_i$ )      Profit ( $p_i$ )

Decision  
 $x_i = ?$

$\rightarrow$  Max. the Profit, subject to condition that the Total wt. of the objects put into the KNAP should not exceed its Capacity;

Explicit criteria:  $\sum_{i=1}^n w_i \leq M ; \sum p_i$   
Trivial  $\Rightarrow (x_i = 1) ?$

$$\left( \sum_{i=1}^n w_i > M \right) \quad \langle x_1, x_2, \dots, x_n \rangle = \begin{cases} < & \\ > & \end{cases}$$

$w_i x_i ; p_i x_i$  int  $x_i$  { Fractional  
Real  
Greedy }

Max.  $\sum_{i=1}^n p_i x_i$ ; obj.

S.T.C  $\sum_{i=1}^n w_i x_i \leq M$ : impl. const

where  $0 \leq x_i \leq 1$  ( $\infty$ )

$$\left\{ \begin{array}{l} \text{Solv Space (Greedy KNP)} : \infty \\ \text{Solv Space (0/1 KNP)} : 2^n \end{array} \right.$$

Erl: i)  $n = 3; M = 20;$

$$\langle w_1, w_2, w_3 \rangle = \langle 18, 15, 10 \rangle$$

$$\langle p_1, p_2, p_3 \rangle = \langle 25, 24, 15 \rangle$$

$$\langle x_1, x_2, x_3 \rangle =$$

a) Greedy about Profit:

$$\left. \begin{array}{l} x_1 = 1 \\ x_2 = 2/15 \\ x_3 = 0 \end{array} \right\} \quad \left. \begin{array}{l} \sum w_i x_i = 18 + \frac{2}{15} \times 15 + 0 = 20 \\ \sum p_i x_i = 25 + \frac{2}{15} \times 24 + 0 \\ = 28.3 \end{array} \right\}$$

$$\left. \begin{array}{l} \langle x_1, x_2, \dots, x_n \rangle \\ \vdots \quad \vdots \quad \ddots \quad \vdots \\ \vdots \quad \vdots \quad \ddots \quad \vdots \end{array} \right\} 2^n \quad x_i = 0/1$$

b) Greedy about weight:

$$\left\{ \begin{array}{l} x_3 = 1 \\ x_2 = 10/15 = 2/3 \\ x_1 = 0 \end{array} \right.$$

$$\sum w_i x_i = 20$$

$$\left. \begin{array}{l} \sum p_i x_i = 0 + \frac{2}{3} \times 24 + 15 \\ = 31 \end{array} \right\} \checkmark$$

c) Greedy about  $P/W$ ;  $M = 20$

$$\frac{P_1}{w_1} = \frac{25}{18} = 1.38$$



$$w_i \rightarrow p_i \\ 1 \rightarrow ? \quad (p_i/w_i)$$

$$\frac{P_2}{w_2} = \frac{24}{15} = 1.6$$

$$\frac{P_3}{w_3} = \frac{15}{10} = 1.5$$

$$x_2 = 1$$

$$x_3 = 5/10 = 1/2$$

$$x_1 = 0$$

$$\left. \begin{array}{l} \sum w_i x_i = 20 \\ \sum p_i x_i = 0 + 24 + \frac{1}{2} \times 15 \end{array} \right\}$$

$$= 31.5 \quad \checkmark$$

Q)  $n = 7; M = 15; \langle p_1 \dots p_7 \rangle = \langle 10, 5, 15, 7, 6, 18, 3 \rangle$   
 $\langle w_1 \dots w_7 \rangle = \langle 2, 3, 5, 7, 1, 4, 1 \rangle$

- a) Greedy about Profit
- b) Greedy about weight
- c) Greedy about  $P/w$

Q)  $n=7; M=15; \langle p_1 \dots p_7 \rangle = \langle 10, 5, 15, 7, 6, 18, 3 \rangle$   
 $\langle w_1 \dots w_7 \rangle = \langle 2, 3, 5, 7, 1, 4, 1 \rangle$

a) Greedy about Profit

b) Greedy about weight

c) Greedy about  $P/w$ :

$$\frac{p_1}{w_1} = \frac{10}{2} = 5 \checkmark$$

$$\frac{p_2}{w_2} = \frac{5}{3} = 1.66$$

$$\frac{p_3}{w_3} = \frac{15}{5} = 3 \checkmark$$

$$\frac{p_4}{w_4} = \frac{7}{7} = 1$$

$$\frac{p_5}{w_5} = \frac{6}{1} = 6 \checkmark$$

$$\frac{p_6}{w_6} = \frac{18}{4} = 4.5 \checkmark$$

$$\frac{p_7}{w_7} = \frac{3}{1} = 3 \checkmark$$

$$x_5 = 1$$

$$x_1 = 1$$

$$x_6 = 1$$

$$x_7 = 1$$

$$x_3 = 1$$

$$x_2 = \frac{2}{3}$$

$$x_4 = 0$$

$$\Rightarrow 1 + 2 + 4 + 1 + 5 + 2 = \underline{\underline{20}}$$

$$\sum p_i x_i = \left( 10 + \frac{2}{3} \times 5 + 15 + 6 + 18 + 3 \right) = \underline{\underline{55.33}}$$

1. Consider the weights and values of items listed below. Note that there is only one unit of each item.

| Item number | Weight (in Kgs) | Value (in Rupees) |
|-------------|-----------------|-------------------|
| 1           | 10              | 60                |
| 2           | 7               | 28                |
| 3           | 4               | 20                |
| 4           | 2               | 24                |

The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by  $V_{opt}$ . A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by  $V_{greedy}$ . The value of  $V_{opt} - V_{greedy}$  is \_\_\_\_\_.

- a) 16 ✓  
 b) 26  
 c) 12  
 d) 8

(DP)

$$M = 11$$

$V_{opt}$  :  $V_{greedy}$  = 44

$$\frac{P_1}{W_1} = \frac{60}{10} = 6$$

$$\frac{P_2}{W_2} = \frac{28}{7} = 4$$

$$\frac{P_3}{W_3} = \frac{20}{4} = 5$$

$$\frac{P_4}{W_4} = \frac{24}{2} = 12$$

$$V_{opt} : \cancel{\underline{60}} = \underline{16}$$

$$\begin{cases} x_1 = 1 \\ x_3 = 1 \end{cases} \quad 44$$

$$2 + 4$$

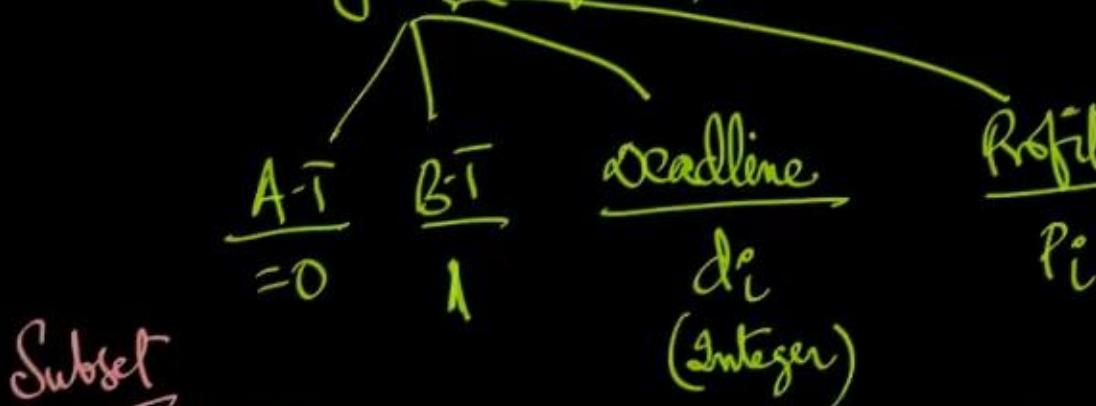
Solve this problem with fractional greedy :

⇒ 78 (Bag will be completely filled always)

## ② Job Sequencing with deadlines (JSD) : CPU Scheduling

→ Given a single CPU using Non-PR Scheduling

→  $n$ -jobs (programs)



→ (If a job is completed within its deadline, then we get its associated Profit)

→ Max. the Profit : objective

{: Select a subset of  $n$ -jobs} S.T, the jobs in the subset are all completable within their deadlines and generates maximum profit ;}

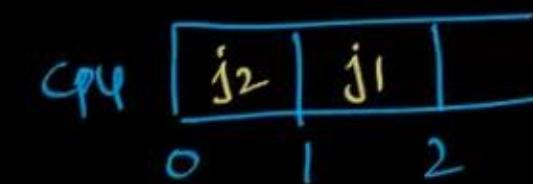
$$\text{Solv State: } \underline{\underline{2^n}}$$

Brute Force:  $O(2^n)$  Exponential

$$n=4; \langle j_1, \dots, j_4 \rangle ; \langle d_1, \dots, d_4 \rangle = \langle \underline{2}, 1, 2, 1 \rangle ; \langle p_1, \dots, p_4 \rangle = \langle 10, 15, 20, 48 \rangle$$

 $J \leftarrow$ 

1)  $J \leftarrow \emptyset$



2)  $|J|=1$

$$J \leftarrow \{j_1\} \checkmark$$

3)  $|J|=2$

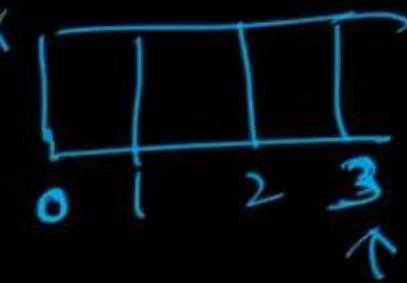
$$J \leftarrow \{j_1, j_2\} \checkmark = 115$$

$$J \leftarrow \{j_1, j_3\} \checkmark$$

$$J \leftarrow \{j_2, j_4\} \times$$

4)  $|J|=3$

$$J \leftarrow \{j_1, j_2, j_3\} \times$$

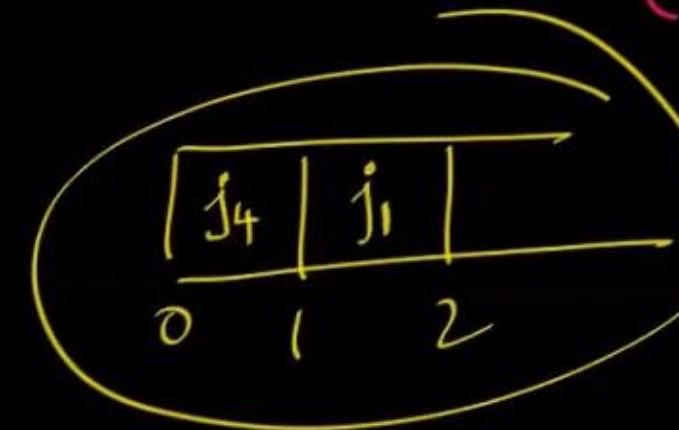


$$n=4; \langle j_1 \dots j_4 \rangle ; \langle d_1 \dots d_4 \rangle = \langle 2, 1, 2, 1 \rangle ; \langle p_1 \dots p_4 \rangle = \langle \underset{x}{10}, \underset{x}{15}, \underset{x}{20}, \underset{x}{48} \rangle$$

Greedy Strategy for JSD:

$$J \leftarrow \{j_1, j_4\} \checkmark = \underline{\underline{148}}$$

1.



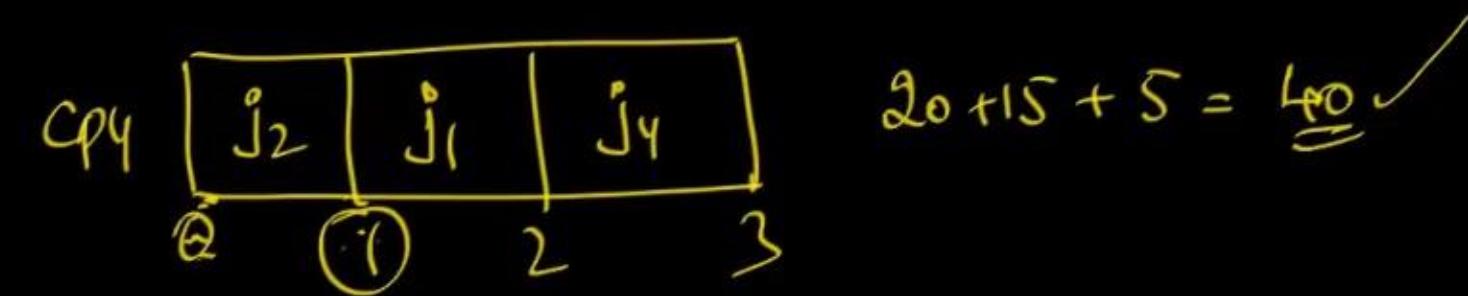
$$2) n=5; \langle j_1 \dots j_5 \rangle$$

$$\langle d_1 \dots d_5 \rangle = \langle 2, 3, 2, 3, 2 \rangle$$

$$\langle p_1 \dots p_5 \rangle = \langle \underset{x}{10}, \underset{x}{15}, \underset{x}{20}, \underset{x}{23}, \underset{x}{30} \rangle$$

cp4  $\frac{j_2 | j_5 | j_4}{0 \ 1 \ 2 \ 3}$  ⑥8 ✓

**Example 4.3** Let  $n = 5$ ,  $(p_1, \dots, p_5) = (20, 15, 10, 5, 1)$  and  $(d_1, \dots, d_5) = (2, 2, 1, 3, 3)$ . Using the above feasibility rule, we have

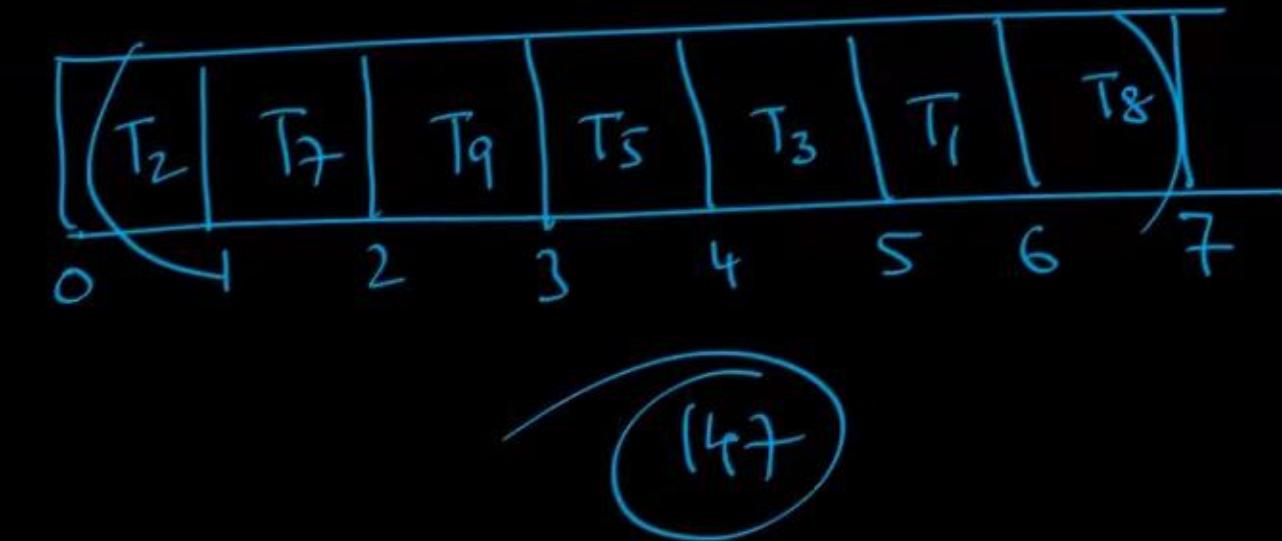


2. We are given 9 tasks  $T_1, T_2, \dots, T_9$ . The execution of each task requires one unit of time. We can execute one task at a time. Each task  $T_i$  has a profit  $P_i$  and a deadline  $d_i$ . Profit  $P_i$  is earned if the task is completed before the end of the Deadline.

| Task     | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | $T_7$ | $T_8$ | $T_9$ |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| Profit   | 15    | 20    | 30    | 18    | 18    | 10    | 23    | 16    | 25    |
| Deadline | 7     | 2     | 5     | 3     | 4     | 5     | 2     | 7     | 3     |

- a. Are all tasks completed in the schedule that gives maximum profit?

- (a) All tasks are completed
- (b)  $T_1$  and  $T_6$  are left out
- (c)  $T_1$  and  $T_8$  are left out
- (d)  $T_4$  and  $T_6$  are left out



## 4.4. JOB SEQUENCING WITH DEADLINES

211

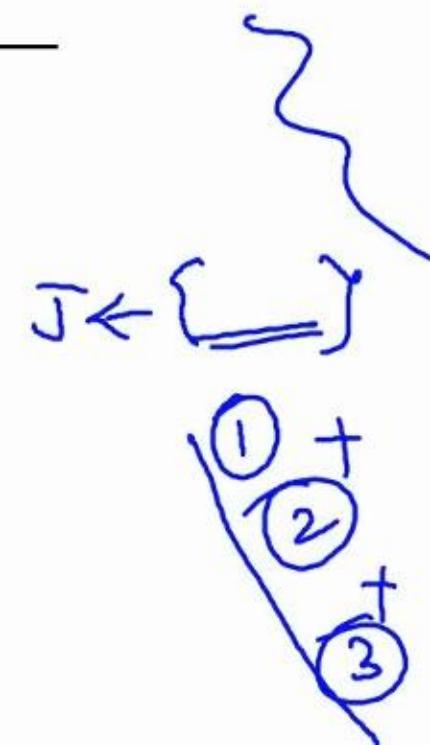
---

```
1 Algorithm GreedyJob( $d, J, n$ ) :  $O(n^2)$ 
2 //  $J$  is a set of jobs that can be completed by their deadlines.
3 {
4      $J := \{1\}$ ;
5     for  $i := 2$  to  $n$  do:  

6     {
7         if (all jobs in  $J \cup \{i\}$  can be completed  

8             by their deadlines) then  $J := J \cup \{i\}$ ;  

9     }
10 }
```



---

**Algorithm 4.5** High-level description of job sequencing algorithm

...,  $J[k]$  are changed after the insertion. Hence, it is necessary to verify

```
1  Algorithm JS( $d, j, n$ )
2  //  $d[i] \geq 1$ ,  $1 \leq i \leq n$  are the deadlines,  $n \geq 1$ . The jobs
3  // are ordered such that  $p[1] \geq p[2] \geq \dots \geq p[n]$ .  $J[i]$ 
4  // is the  $i$ th job in the optimal solution,  $1 \leq i \leq k$ .
5  // Also, at termination  $d[J[i]] \leq d[J[i + 1]]$ ,  $1 \leq i < k$ .
6  {
7       $d[0] := J[0] := 0$ ; // Initialize.
8       $J[1] := 1$ ; // Include job 1.
9       $k := 1$ ;
10     for  $i := 2$  to  $n$  do
11     {
12         // Consider jobs in nonincreasing order of  $p[i]$ . Find
13         // position for  $i$  and check feasibility of insertion.
14          $r := k$ ;
15         while  $((d[J[r]] > d[i]) \text{ and } (d[J[r]] \neq r))$  do  $r := r - 1$ ;
16         if  $((d[J[r]] \leq d[i]) \text{ and } (d[i] > r))$  then
17         {
18             // Insert  $i$  into  $J[ ]$ .
19             for  $q := k$  to  $(r + 1)$  step  $-1$  do  $J[q + 1] := J[q]$ ;
20              $J[r + 1] := i$ ;  $k := k + 1$ ;
21         }
22     }
23     return  $k$ ;
24 }
```

*Insertion  
Sort*

**Algorithm 4.6** Greedy algorithm for sequencing unit time jobs with deadlines and profits

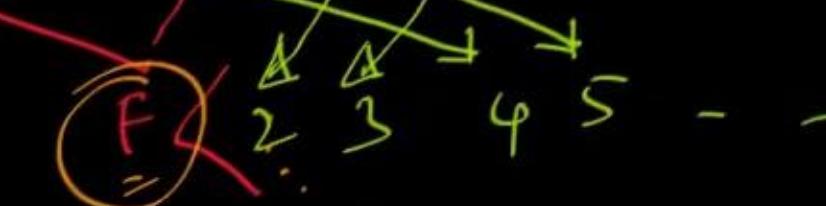
For JS there are two possible parameters in terms of which its complexity can be measured. We can use  $n$ , the number of jobs, and  $s$ , the number of jobs included in the solution  $J$ . The **while** loop of line 15 in Algorithm 4.6 is iterated at most  $k$  times. Each iteration takes  $\Theta(1)$  time. If the conditional of line 16 is true, then lines 19 and 20 are executed. These lines require  $\Theta(k - r)$  time to insert job  $i$ . Hence, the total time for each iteration of the **for** loop of line 10 is  $\Theta(k)$ . This loop is iterated  $n - 1$  times. If  $s$  is

Optimal Merge Pattern (OMP) : Merging of files, with

$F_1: \langle 4, 5, 8, 10 \rangle$

$F_2: \langle 2, 3, 9, 11, 12 \rangle$

2-way Merging



$\Rightarrow F: \langle 2, 3, 4, 5, 8, 9, 11, 12 \rangle$

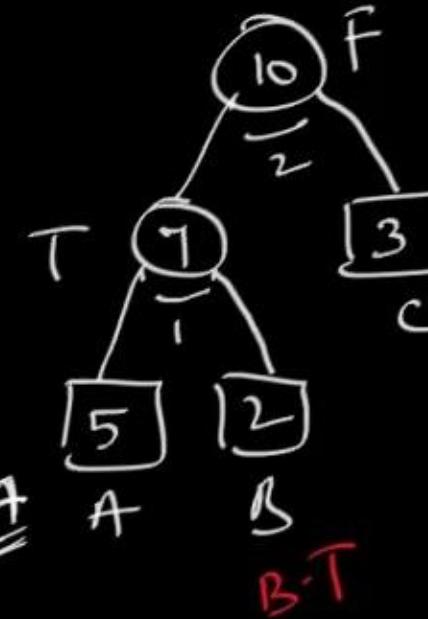
$\Rightarrow$  Total No. of (Record Movements) :  $(n+m)$

: Trivial

$$n = 3$$

$$\langle A, B, C \rangle = \langle \underline{5}, \underline{2}, \underline{3} \rangle$$

$\Rightarrow$  files  
L<sub>n</sub>



$$\langle A \underline{\leq} C \rangle : 17$$

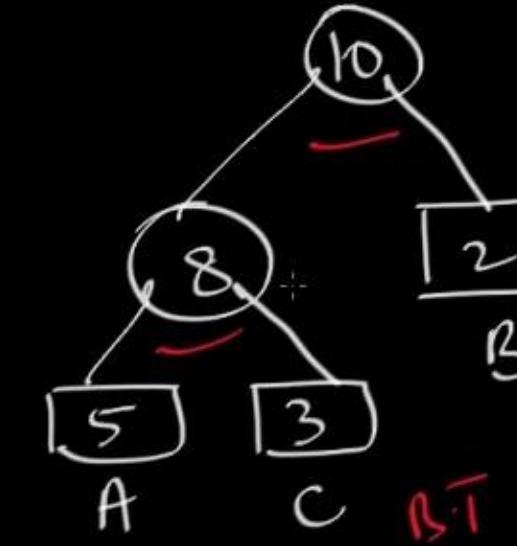
$$\langle A \leq C B \rangle : 18$$

No. of Records

(2-way Merging)

$\underline{\underline{obj}} =$  Min. Total No. of Record Movements

$$\Rightarrow \text{Total Record Movements} = 7 + 10 = 17$$



$$\text{Total Record Mov's : } 18$$

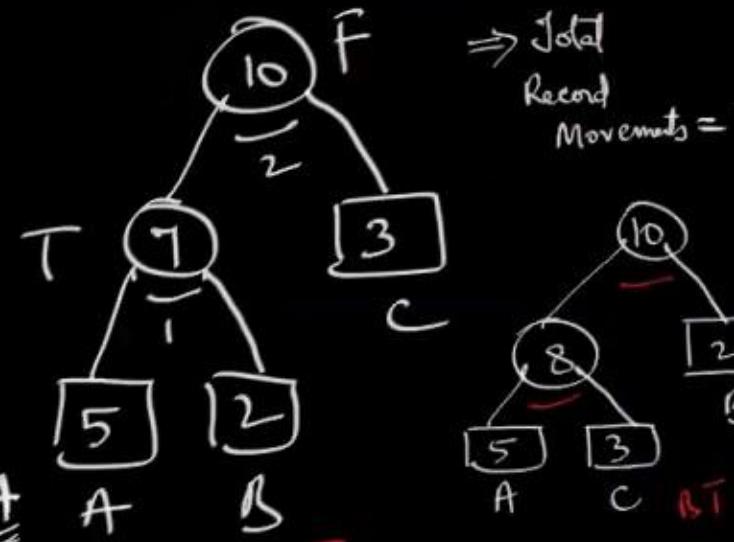
$$n=3$$

$$\langle A, B, C \rangle = \langle \underline{5}, \underline{2}, \underline{3} \rangle$$

file

(Ln)

$$\langle A \underline{\downarrow} C \rangle : \underline{17}$$



No. of Records

$$\Rightarrow \text{Total Record Movements} = 7 + 10 = \underline{\underline{17}}$$

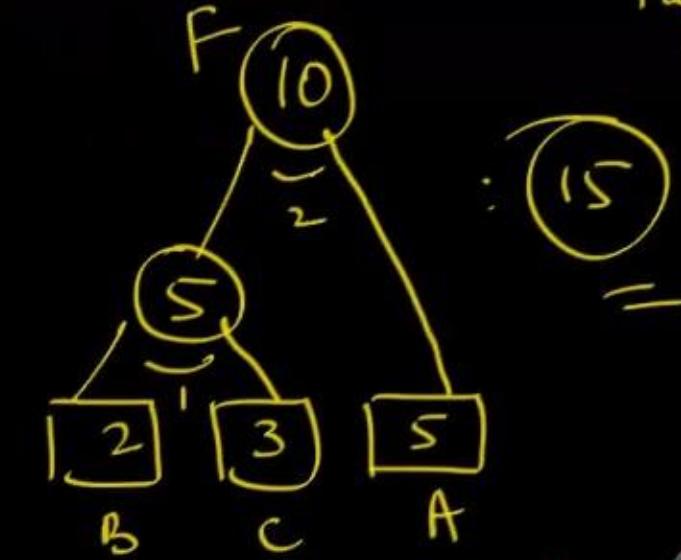
$$\text{Total Record Mov's} : \underline{\underline{18}}$$

(2-way Merging)

= Recor

obj: Min. Total No. of rd movements

(opt 2-way Binary Merge Tree)



$$\langle A \underline{\downarrow} C \underline{\downarrow} B \rangle : \underline{\underline{18}}$$

$$n=7; \langle F_1 \dots F_7 \rangle = \langle 20, 5, 18, 3, 2, 10, 15 \rangle$$

a) 73

b) 181 ✓

c) 178

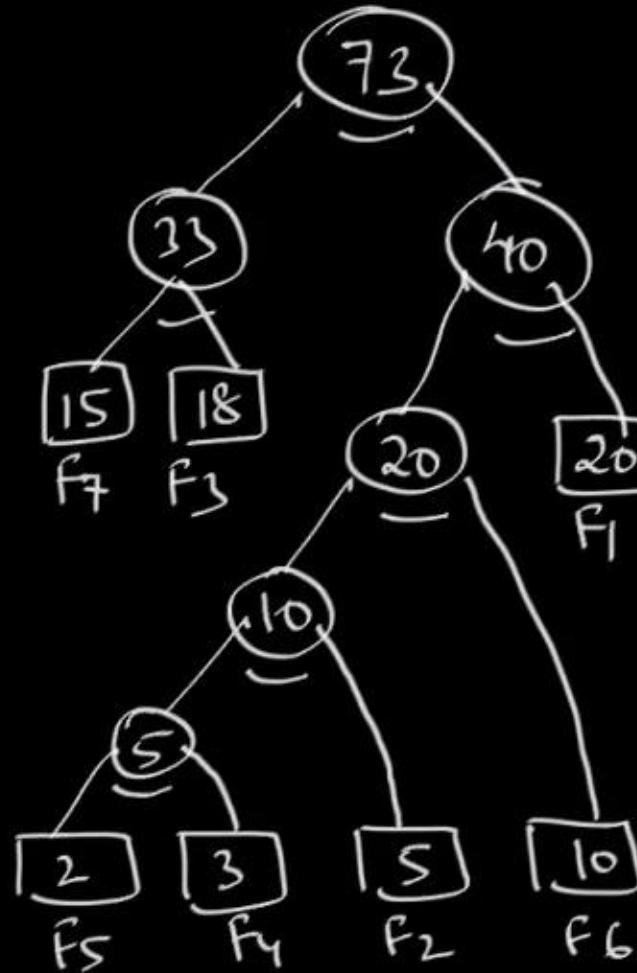
d) 196

e) 166

f) 186

g) 146

h) 226



Total Record Mov's: 73 + 33 + 40  
+ 20 + 10 + 5  
= 181

No. of Record Mov's in  
given by (wtd. External Path  
length)

$$= \sum_{i=1}^n d_i \cdot v_i$$

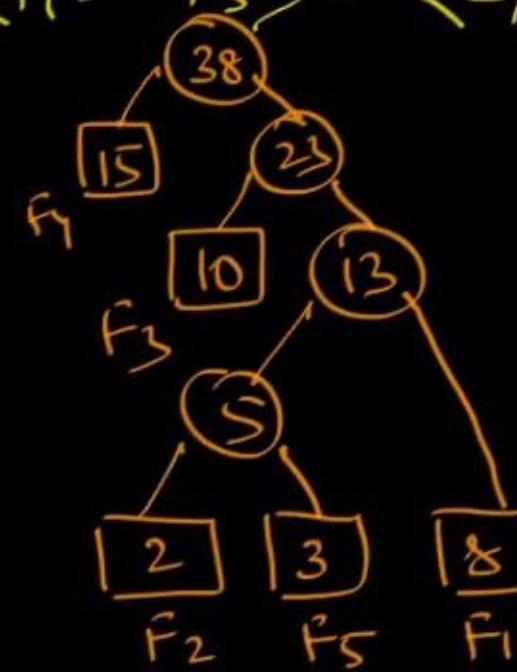
$d_i$  = distance  
from root to  
 $F_i$

$$= 2 \cdot 20 + 4 \cdot 5 + 2 \cdot 18 + 5 \cdot 3  
+ 5 \cdot 2 + 3 \cdot 10 + 2 \cdot 15$$

= 181

$v_i$  = size of  $F_i$

$$n=5; \quad \langle f_1 \dots f_5 \rangle = \langle 8, 2, 10, 15, 3 \rangle$$



$$\therefore (38 + 23 + 13 + 5)$$



```

procedure GREEDY_KNAPSACK( $P$ ,  $W$ ,  $M$ ,  $X$ ,  $n$ )
// $P(1:n)$  and  $W(1:n)$  contain the profits and weights respectively of the  $n$ //
//objects ordered so that  $P(i)/W(i) \geq P(i + 1)/W(i + 1)$ .  $M$  is the//
//knapsack size and  $X(1:n)$  is the solution vector//
real  $P(1:n)$ ,  $W(1:n)$ ,  $X(1:n)$ ,  $M$ ,  $cu$ ;
integer  $i$ ,  $n$ ;
 $X \leftarrow 0$  //initialize solution to zero//
 $cu \leftarrow M$  // $cu$  = remaining knapsack capacity//
for  $i \leftarrow 1$  to  $n$  do
    if  $W(i) > cu$  then exit endif
     $X(i) \leftarrow 1$ 
     $cu \leftarrow cu - W(i)$ 
repeat
    if  $i \leq n$  then  $X(i) \leftarrow cu/W(i)$  endif
end GREEDY_KNAPSACK

```

↓ Spec. order of  $P/W$

}
 Time Complexity  
 $O(n)$   
Space:  $O(1)$

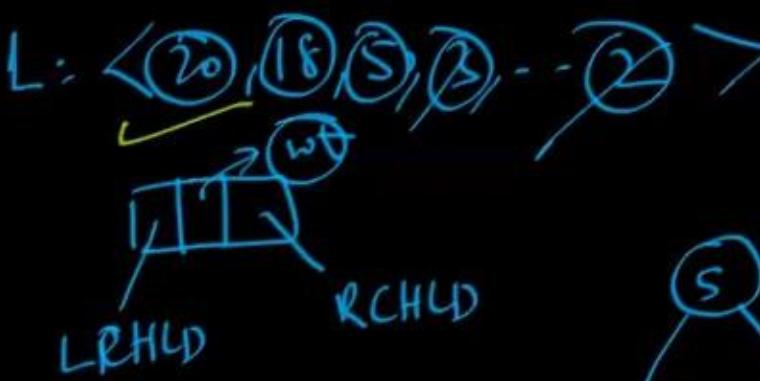
**Algorithm 4.3** Algorithm for greedy strategies for the knapsack problem

```

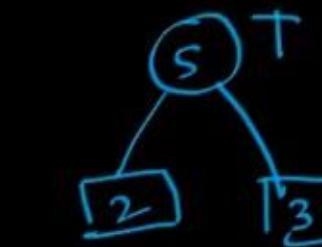
line procedure TREE( $L$ ,  $n$ )
  // $L$  is a list of  $n$  single node binary trees as described above//
  for  $i = 1$  to  $n - 1$  do
    call GETNODE( $T$ ) //merge two trees with//
     $LCHILD(T) = LEAST(L)$  //smallest lengths//
     $RCHILD(T) = LEAST(L)$ 
     $WEIGHT(T) = WEIGHT(LCHILD(T)) + WEIGHT(RCHILD(T))$ 
    call INSERT( $L$ ,  $T$ )
  repeat
  return ( $LEAST(L)$ ) //tree left in  $L$  is the merge tree//
end TREE

```

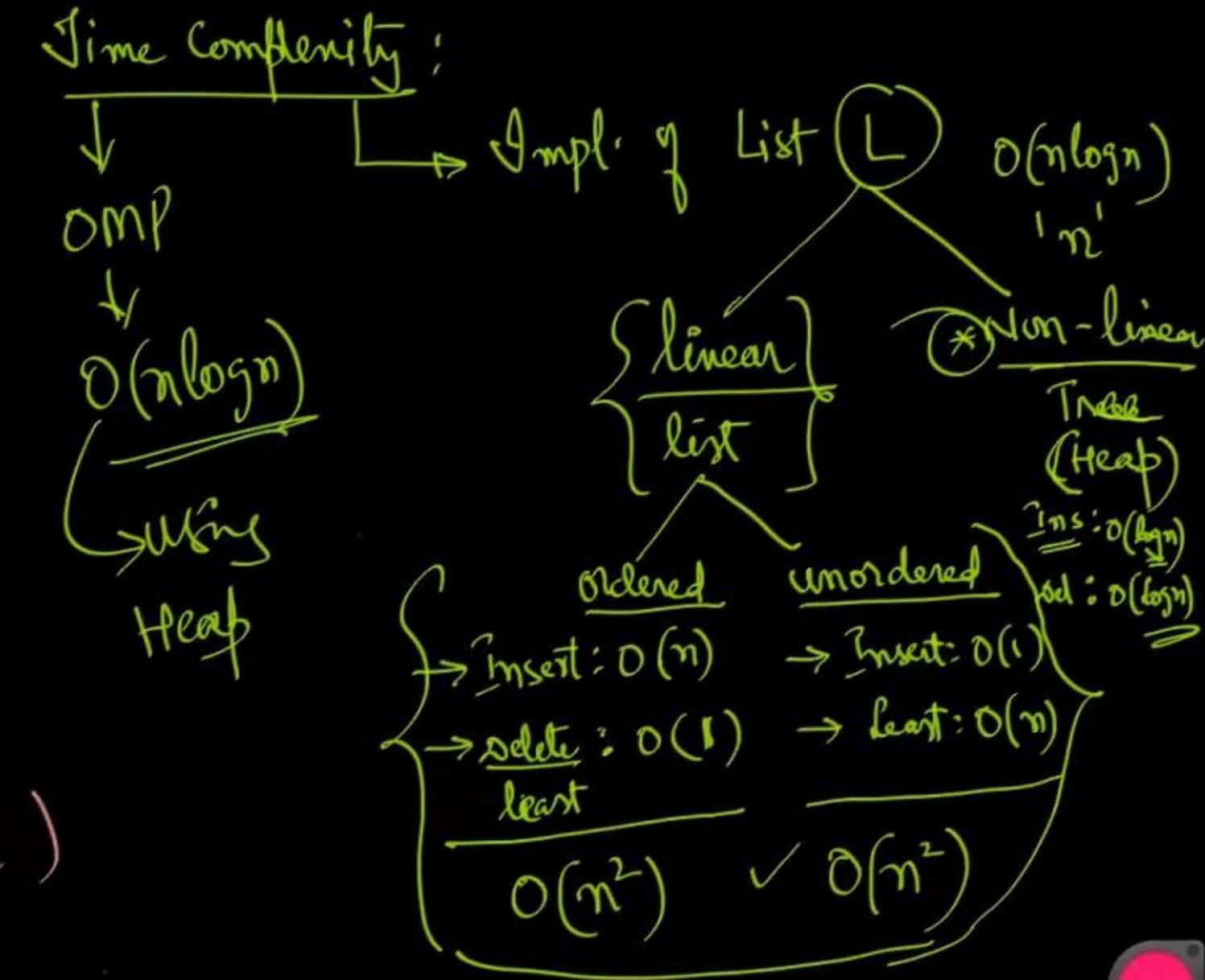
Algorithm 4.7 Algorithm to generate a 2-way merge tree



LEAST: delete  
Insert:



Space:  $O(n)$



Applying OMP: Huffman Coding: (is a Data Encoding + Data Compression Technique)

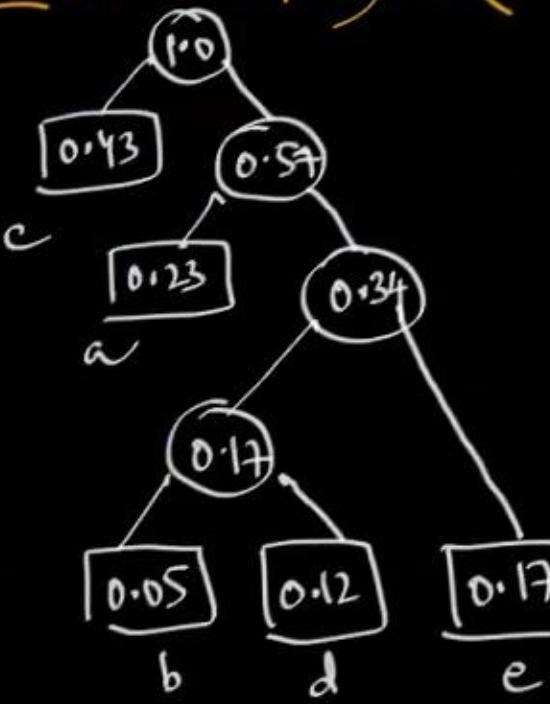
English: (vowels)

↳ (Non-uniform Encoding)

(frequency) depend. Encoding

$$L = \sum (a, b, c, d, e) = (0.23, 0.05, 0.43, 0.12, 0.17)$$

Non-uniform  
Encoding  
3 bits  
char

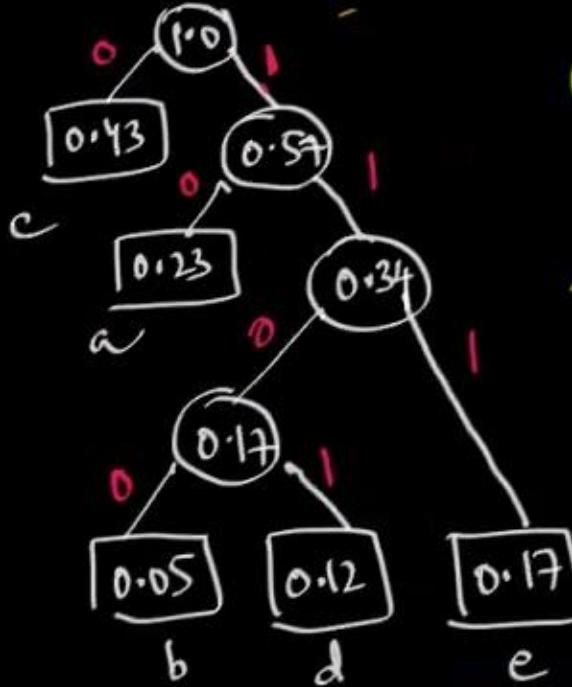


repr. of  
Data (info) in some form

✓ ASCII:  $L = \sum \left( \frac{2^6}{12} + \frac{2^6}{12} + \dots \right)$   
 $\downarrow$   
 $\{7\text{ bits}\}$  ✓ ~ {100}

✓ BCD: (0..9)  $\xrightarrow{\downarrow}$  4 bits

signals  
channel  
MANCHESTER D...  
medium



opt- 2-way Binary Encode Tree

Note: one bit error is sufficient to make whole sent corrupted

$a \rightarrow 10$  (2 bits)  
 $b \rightarrow 1100$  (4 bits)  
 $c \rightarrow 0$  (1 bit)  
 $d \rightarrow 1101$  (4 bits)  
 $e \rightarrow 111$  (3 bits)

Sent:  $\langle c a c c a a e c b c \rangle = 20$

Binary Stream :  $\langle 010000101110110000 \rangle$ ; Total : 17

Binary Stream:  $\langle 0100001010001110 \dots \rangle$   
~~: peace~~

Sent Stream:  $a a c c a a e c e c$

Ans. No. of bits :  $\sum_{i=1}^n d_i \cdot q_i$

$$\cdot 2 * 0.23 + 4 * 0.05 + 1 * 0.43 + 4 * 0.12$$

$$= \boxed{2.08 \text{ bits/character}} \quad \checkmark$$

3. The characters 'a' to 'h' have the set of frequencies based on the first 8 Fibonacci numbers as follows :

a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21

A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code?

110111100111010

- (a) fdhieg ✓
- (b) ecgdf
- (c) dchfg
- (d) fehdg

|                         |    |
|-------------------------|----|
| $a \rightarrow 1111100$ | {} |
| $b \rightarrow 1111101$ |    |
| $c \rightarrow 111111$  |    |
| $d \rightarrow 11110$   |    |
| $e \rightarrow 1110$    |    |
| $f \rightarrow 110$     |    |
| $g \rightarrow 10$      |    |
| $h \rightarrow 0$       |    |

110111100111010  
f d h e g



|   |     |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |      |     |     |    |
|---|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|-----|----|
| 1 | 132 | 132F | 133F | 134F | 135F | 136F | 137F | 138F | 139F | 140F | 141F | 142F | 143F | 144F | 145F | 146F | 147F | 148F | 149F | 150F | 151F | 152F | 153F | 154F | 155F | 156F | 157F | 158F | 159F | 160F | 161F | 162F | 163F | 164F | 165F | 166F | 167F | 168F | 169F | 170F | 268 | Off | On |
|---|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|-----|-----|----|

4. A message is made up entirely of characters from the set  $X = \{P, Q, R, S, T\}$ . The table of probabilities for each of the characters is shown below:

| Character | Probability |
|-----------|-------------|
| P         | 0.22        |
| Q         | 0.34        |
| R         | 0.17        |
| S         | 0.19        |
| T         | 0.08        |
| Total     | 1.00        |

If a message of 100 characters over X is encoded using Huffman coding, then the expected length of the encoded message in bits is 225.

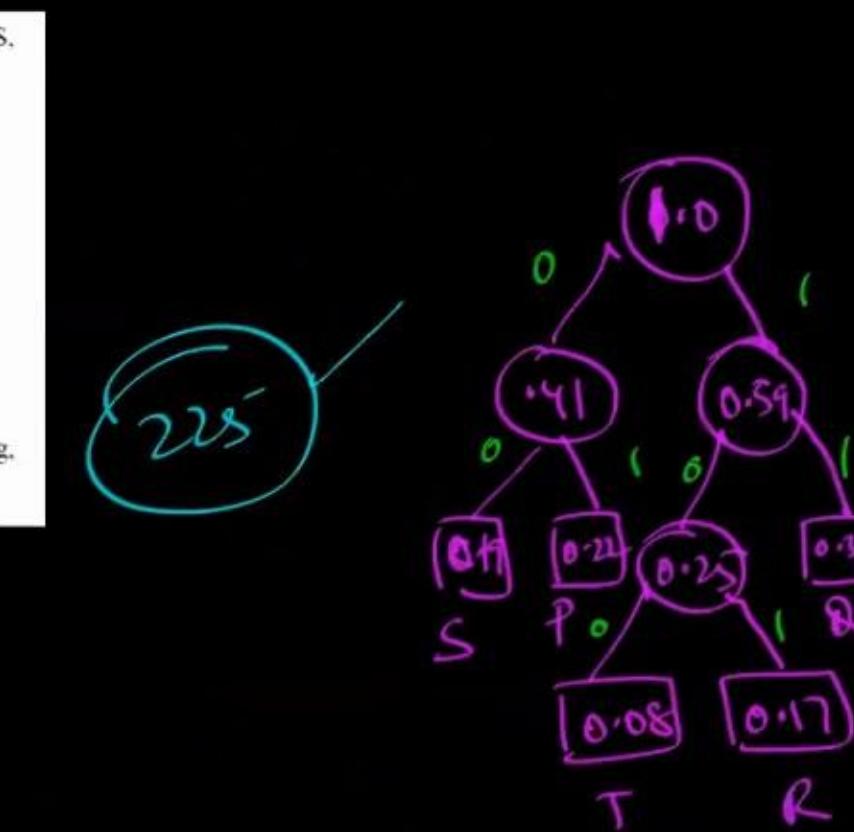
P → 01 (2 bits)

Q → 11 (2 bits)

R → 101 (3 bits)

S → 00 (2 bits)

T → 100 (3 bits)



$$\text{Ans: No. of bits/char: } (2 * 0.22 + 2 * 0.34 + 3 * 0.17 + 2 * 0.19 + 3 * 0.08) \\ = 2.25 \text{ bits/char}$$

5. Consider the string `abbccddeee`. Each letter in the string must be assigned a binary code satisfying the following properties:

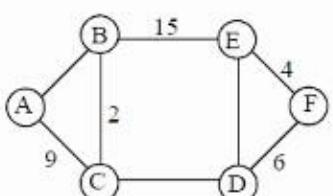
1. For any two letters, the code assigned to one letter must not be a prefix of the code assigned to the other letter.
2. For any two letters of the same frequency, the letter which occurs earlier in the dictionary order is assigned a code whose length is at most the length of the code assigned to the other letter.

Among the set of all binary code assignments which satisfy the above two properties, what is the minimum length of the encoded string?

- (a) 25      (b) 23      (c) 21      (d) 30

6. Consider a Graph whose vertices are points in a plane with integer co-ordinates  $(x, y)$  where  $1 \leq x \leq n$ ,  $1 \leq y \leq n$ ,  $n > 2$  is an integer. 2 vertices  $\langle x_1, y_1 \rangle$  &  $\langle x_2, y_2 \rangle$  are adjacent iff  $|x_1 - x_2| \leq 1$  &  $|y_1 - y_2| \leq 1$ . The cost of such an edge is given by the distance between them. Compute the weight of min cost Spanning Tree of such graph for a value of  $n$ .

7. Consider the following Graph whose Minimum Cost Spanning Tree marked with edge values has a weight of 36. Minimum possible sum of all edges of the graph  $G$  is \_\_\_\_\_. (Assume that all edges have distinct cost).



8. Consider a graph with ' $n$ ' vertices  $n > 2$ . The vertices are numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  are adjacent iff  $0 < |i - j| \leq 2$ . The weight of such an edge is  $i + j$ . The weight of minimum cost Spanning Tree of such a graph

For Micro Notes by the Student



a = 1  
b = 2  
c = 2  
d = 2  
e = 3

Export PDF

Adobe ExportPDF Convert PDF files to Word or Excel online.

Select PDF File: Algorithms Handout 2021.pdf 1 file / 537 KB

Convert To: Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change

Convert

Create PDF

Edit PDF

Combine PDF

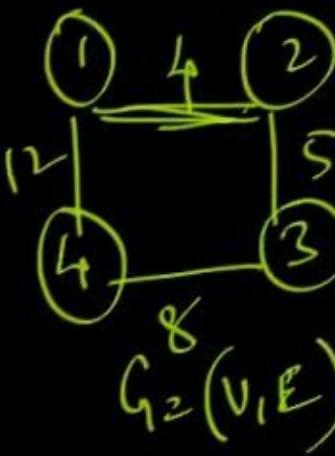
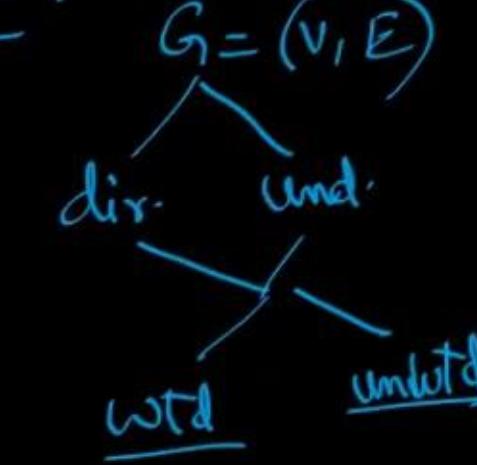
Send Files

Store Files

Graph based Problems:

$$G = (V, E)$$

$$|V| = n; |E| = e$$



$$\begin{array}{c|cccc} C & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 12 \\ 2 & 1 & 0 & 1 & 0 \\ 3 & & & & \\ 4 & & & & \end{array}$$

$\tilde{\Theta}(n^2)$

Repr.

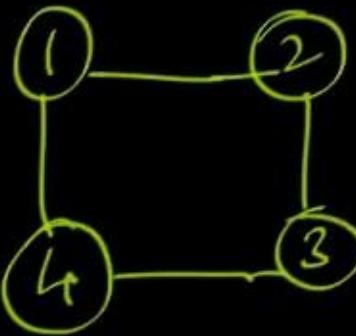
Adj.  
Matrix

(array)  
( $n \times n$ )

(Cost-  
Adj.  
Matrix)

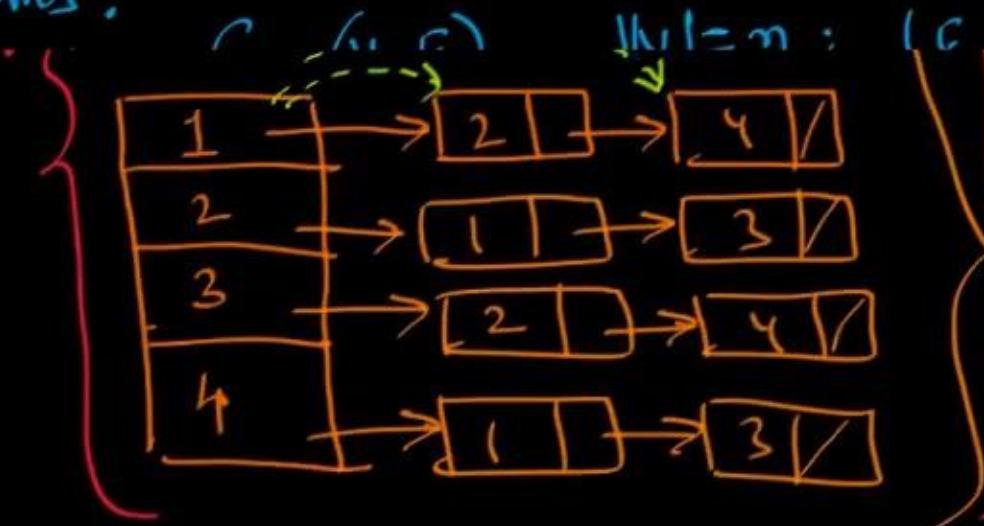
Adj. list  
(linked  
list)

## Graph based Problems:



$$G = (V, E)$$

$$|V| = n; |E| = e$$



$|V|=n; |E|=e$

for und. graph

$$= (m + 2 \cdot e)$$

$$\underline{\underline{O(n+e)}}$$

Ex: for Complete Graph

$$e = \frac{n(n-1)}{2}$$

$$= O(n^2)$$

$$\therefore \underline{\underline{O(n+n^2)}}$$

$$\underline{\underline{= O(n^2)}}$$

for directed graphs

the size of Adj-list

$$= (n+e)$$

$$O(n+e)$$

for complete  $K_n$ :  $O(n+n^2) = O(n^2)$



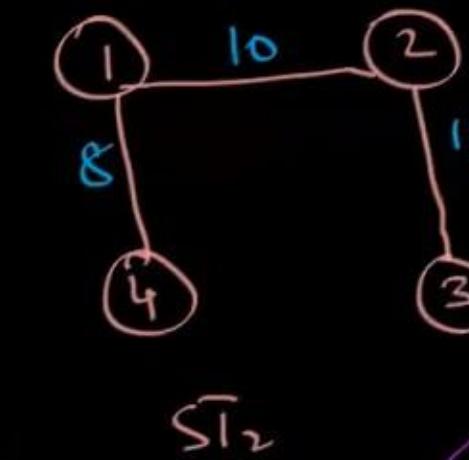
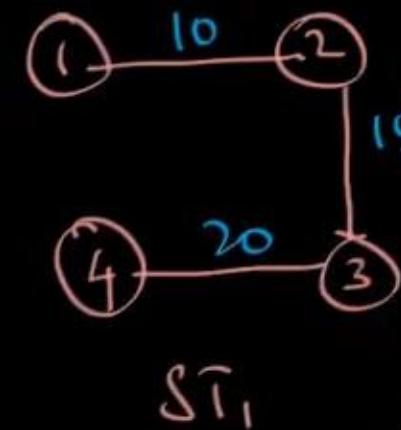
## 5) Min. Cost Spanning Tree (MCST):

Spanning Tree: A Subgraph  $T = (V, E')$  of  $G = (V, E)$ , where  $E' \subseteq E$  is a Spanning Tree iff  $T$  is a Tree;



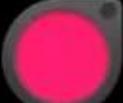
$$|V| = n$$

$$|E| = e$$



No. of Spanning Trees:  
 (Co-factor) of Matrix Method

Solt Space =  $\binom{n-2}{n}$  Caley's formula



Applic. of Sp. Trees:  $N/w \rightarrow$  Routing

↳ MCSI

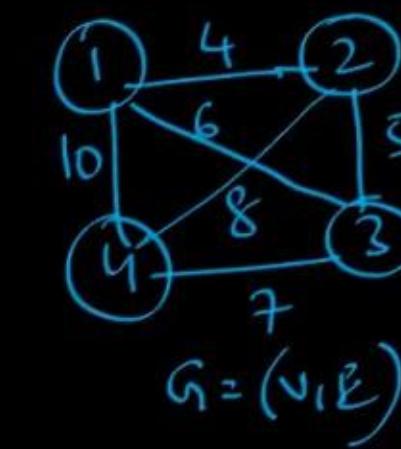
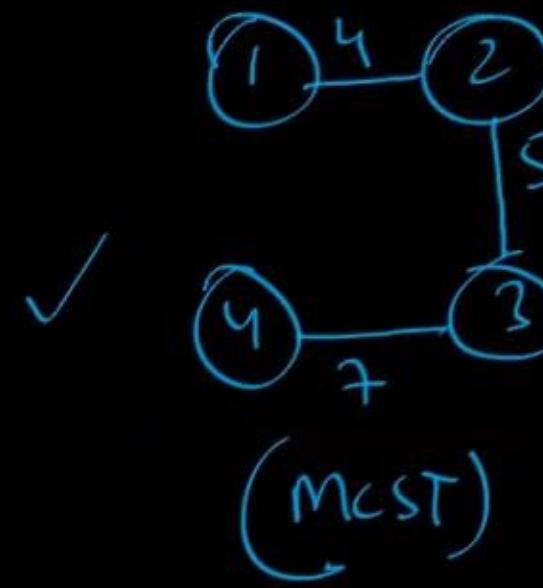
- Electronics :
- Electrical :
- N/w Topologies

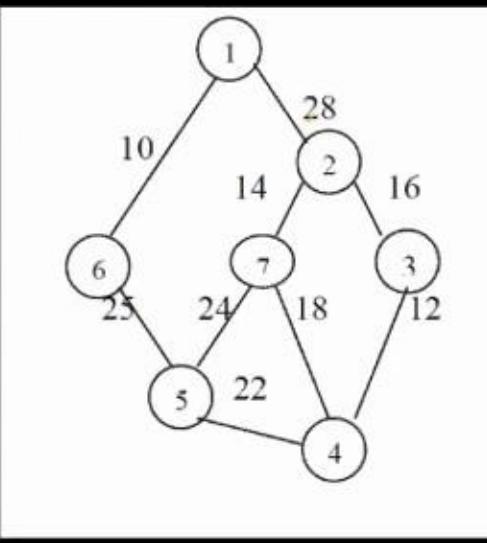
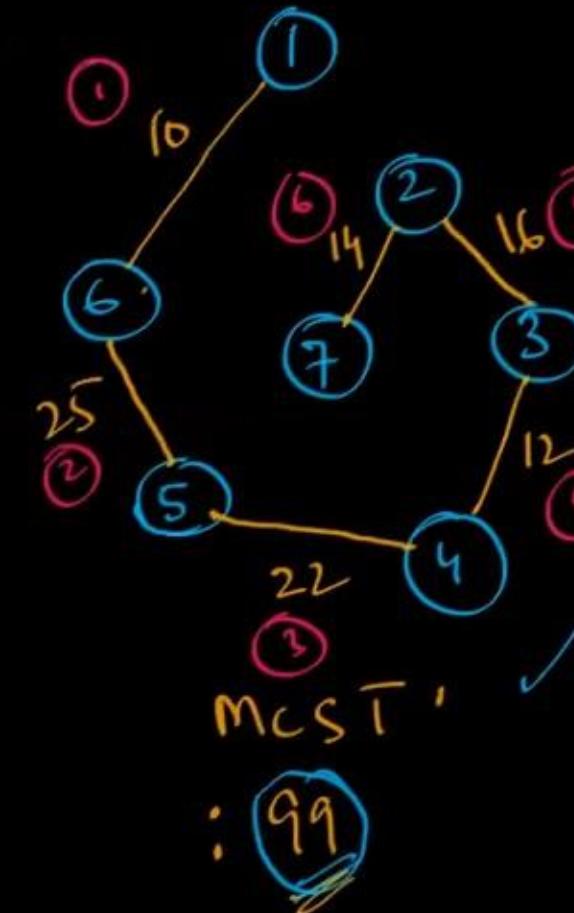
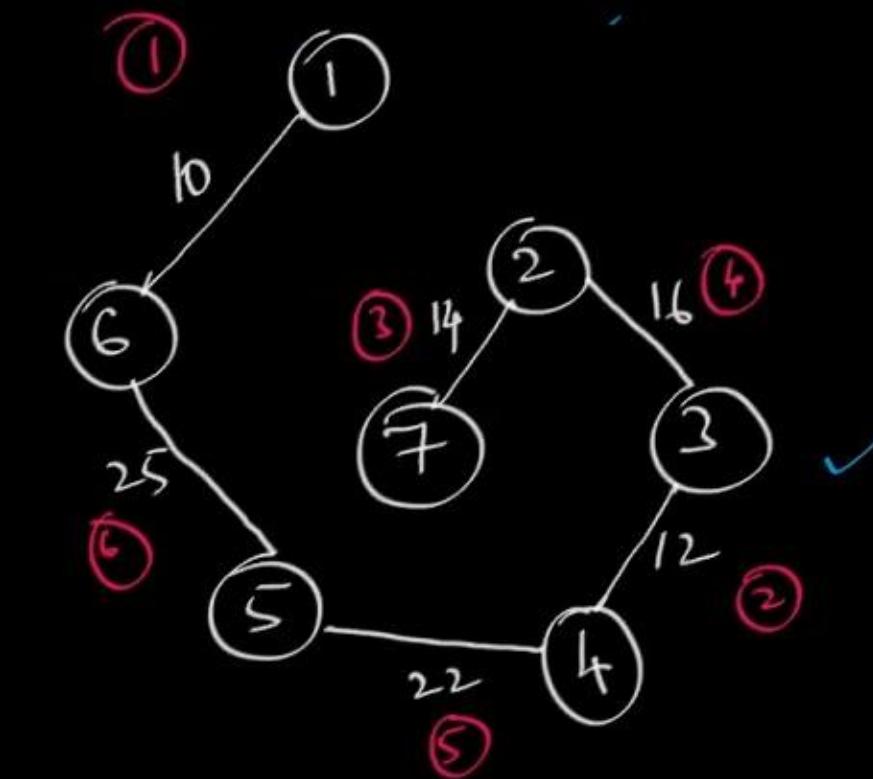


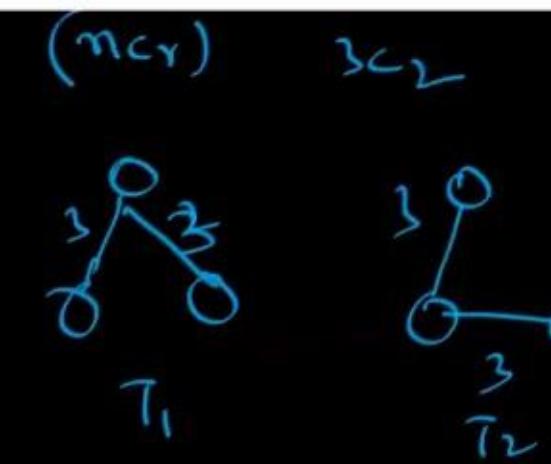
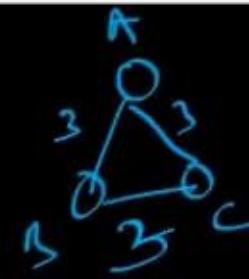
$\rightarrow$  GQ: The no. of edges that must be removed from the graph to get a Sp. Tree is  $\frac{e - (n-1)}{= e - n + 1}$ . (Assume  $G_1$  has  $n$ -vertices &  $e$  - edges)

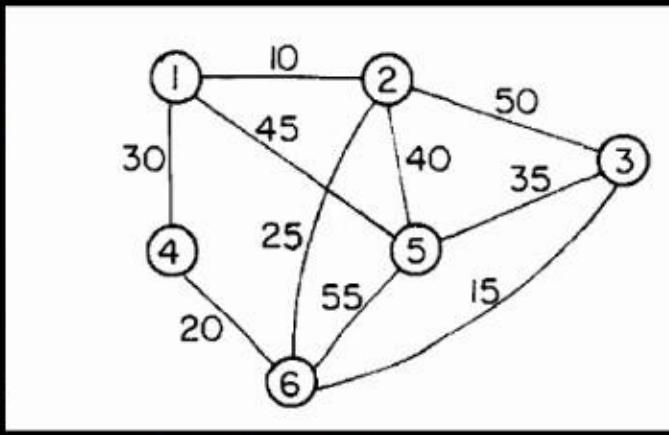
$$\begin{matrix} \rightarrow e \\ G \end{matrix}$$

$$T: \underline{\underline{(n-1) \text{ edges}}}$$

a) Prim'sb) Kruskalsc) Dijkstras

(i) Floyd's Algorithm : (Tree Structure)Connectedness  
PropertyTime:  $O(n^2)$ : $: O((n+e)\log n)$   
(Heap)(ii) Kruskal's Algorithm :edge values: Min-Heap : e  
: 10, 12, 14, 16, 18, 22, 24, 25, 28  
x x x xMay not be Maint  
 $: O(e \cdot \log e)$ 

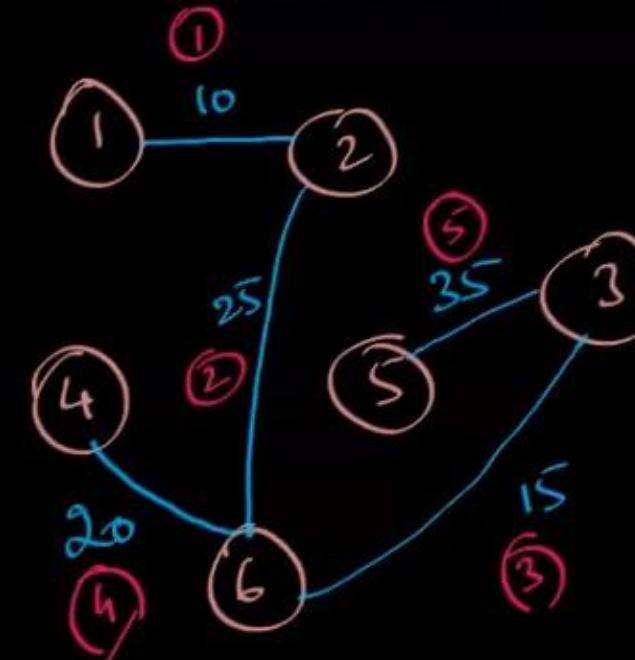




$G = (V, E)$

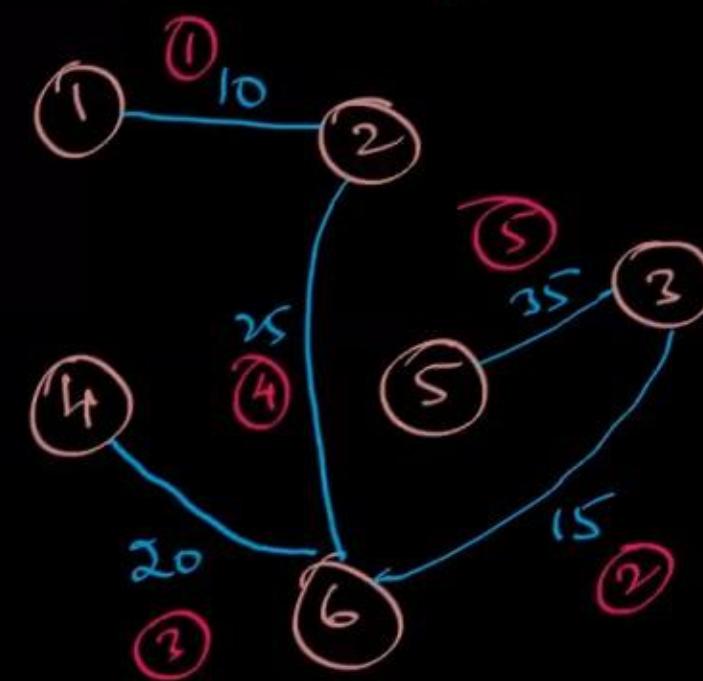
Steps followed in  
Prim's & Kruskal's  
Method

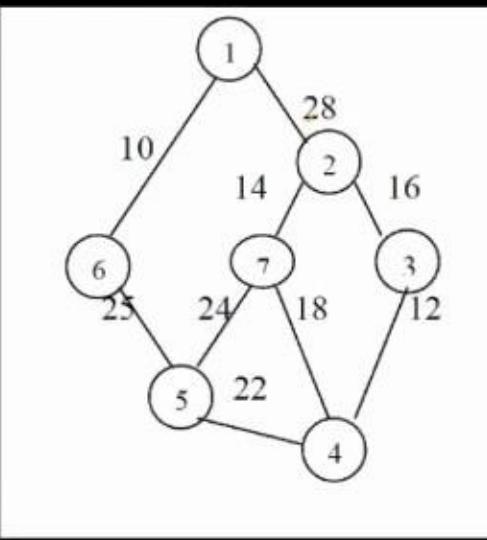
Prim's Method



Kruskal's Method:

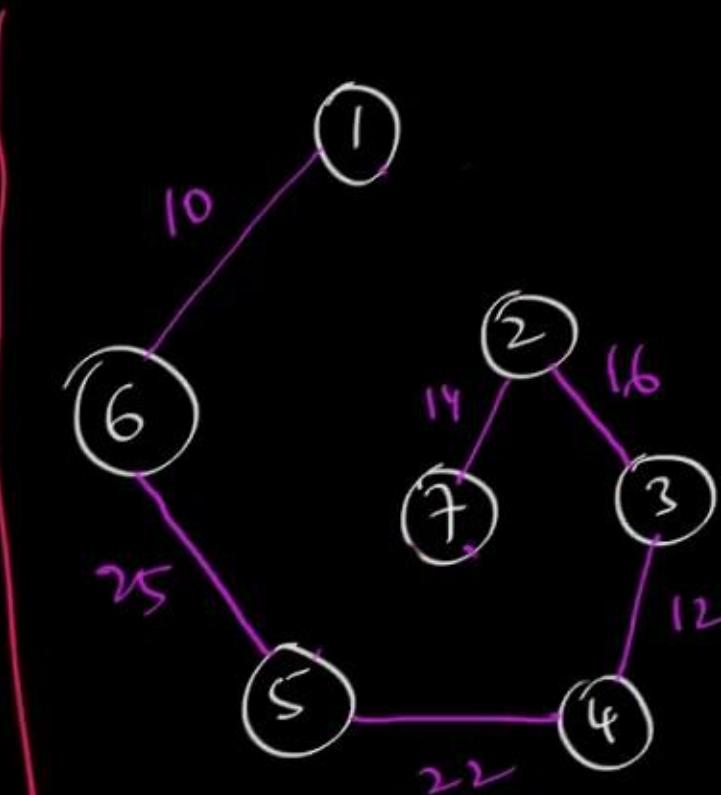
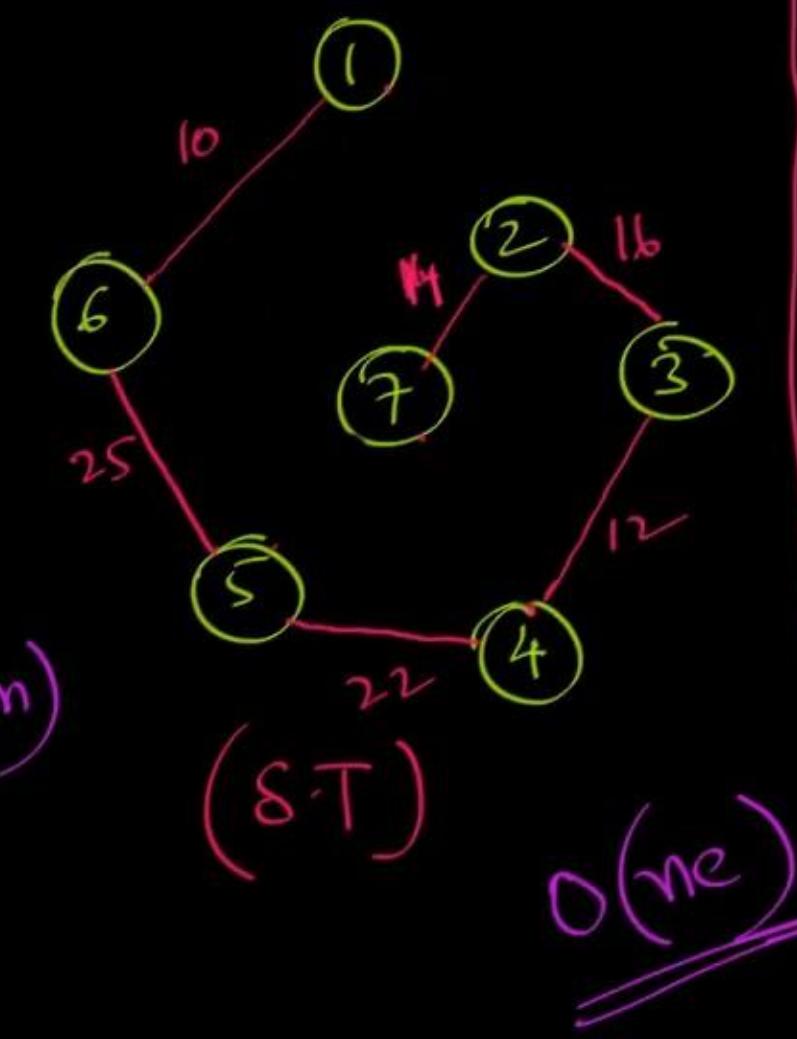
: 10, 15, 20, 28, 30, 35, 40, 45, 50, 55





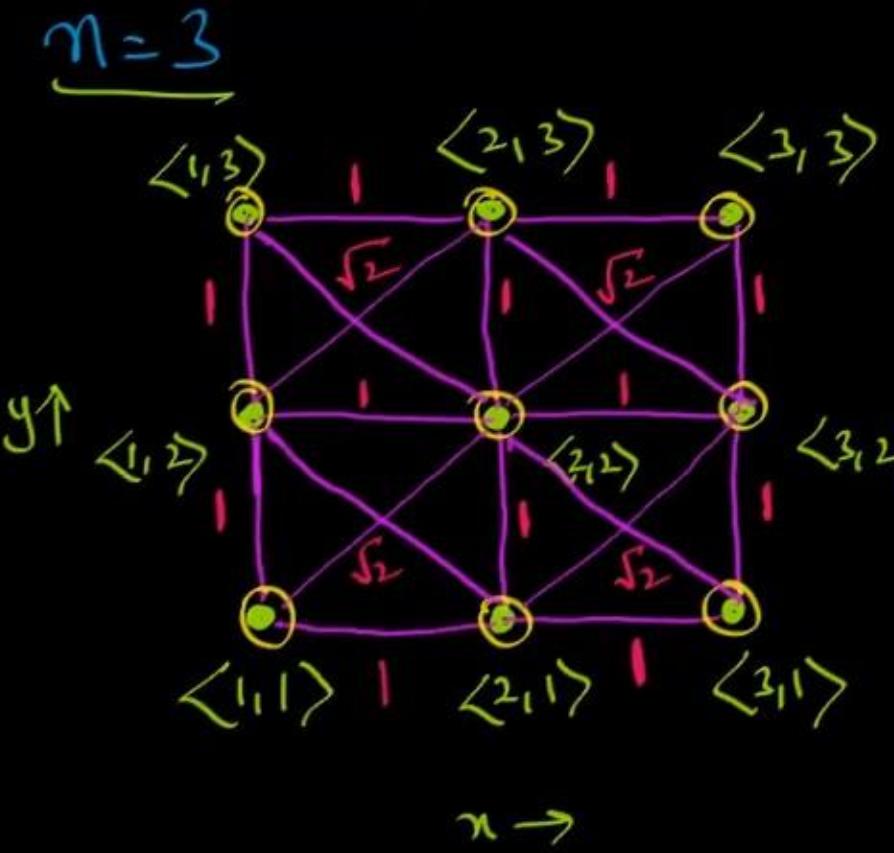
(Based on  
cycle detection)

## Dijkstra's Algo. for MCST :



Complete Graph  
 $O(n^3)$

6. Consider a Graph whose vertices are points in a plane with integer co-ordinates  $(x, y)$  where  $1 \leq x \leq n$ ,  $1 \leq y \leq n$ ,  $n > 2$  is an integer. 2 vertices  $\langle x_1, y_1 \rangle$  &  $\langle x_2, y_2 \rangle$  are adjacent iff  $|x_1 - x_2| \leq 1$  &  $|y_1 - y_2| \leq 1$ . The cost of such an edge is given by the distance between them. Compute the weight of min cost Spanning Tree of such graph for a value of  $n$ .

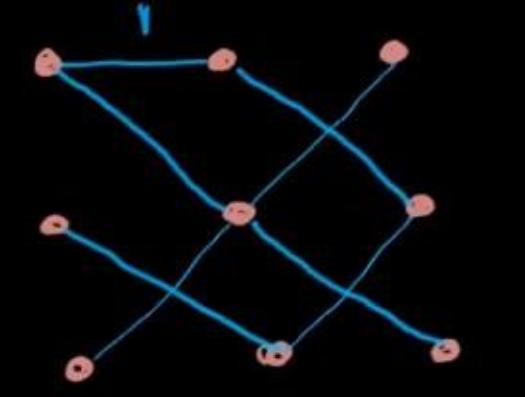


$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



(Min. Cost  
Sp-Tree)  
(8.1)

wt  
Max. Cost Sp-Tree



(7 \* sqrt(2) + 1)  
: (n^2 - 2) \* sqrt(2) + 1

```
line procedure PRIM(E, COST, n, T, mincost)
```

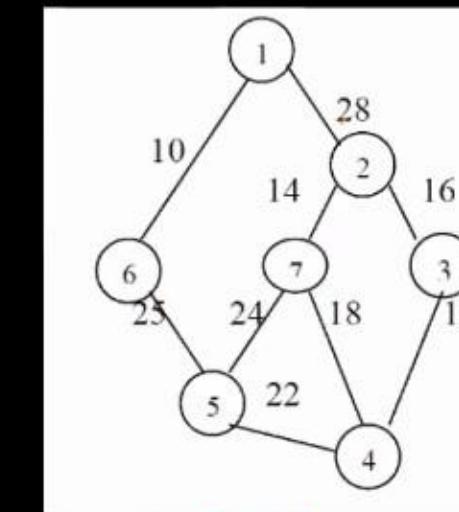
```
//E is the set of edges in G// ✓
//COST(n, n) is the cost adjacency matrix of an n vertex graph//
//such that COST(i, j) is either a positive real number or +∞ if//
//no edge (i, j) exists. A minimum spanning tree is computed and//
//stored as a set of edges in the array T(1:n - 1, 2). (T(i, 1),//
//T(i, 2)) is an edge in the min-cost spanning tree. The final cost//
//is assigned to mincost//
```

```
1   real COST(n, n), mincost;
2   integer NEAR(n), n, i, j, k, l, T(1:n - 1, 2);
3   1.   (k, l) ← edge with minimum cost
4   2.   mincost ← COST(k, l)
5   3.   (T(1, 1), T(1, 2)) ← (k, l)
6   4.   for i = 1 to n do //initialize NEAR//
7       if COST(i, l) < COST(i, k) then NEAR(i) ← l
8           else NEAR(i) ← k endif
9   repeat
10  5.   NEAR(k) ← NEAR(l) ← 0
11  6.   for i = 2 to n - 1 do //find n - 2 additional edges for T//
12      let j be an index such that NEAR(j) ≠ 0 and COST(j, NEAR(j))//
13          is minimum
14      (T(i, 1), T(i, 2)) ← (j, NEAR(j))
15      mincost ← mincost + COST(j, NEAR(j))
16      NEAR(j) ← 0
17      6.   for k = 1 to n do //update NEAR//
18          if NEAR(k) ≠ 0 and COST(k, NEAR(k)) > COST(k, j)//
19              then NEAR(k) ← j
20      endif
21      repeat
22      repeat
23      if mincost ≥ ∞ then print ('no spanning tree') endif
end PRIM
```

Algorithm 4.8 Prim's minimum spanning tree algorithm

$$\text{Cost}(i, j) = \infty$$

$\langle i, j \rangle \notin E$



$$G = (V, E)$$

Relaxation

$$2) j \leftarrow 5$$

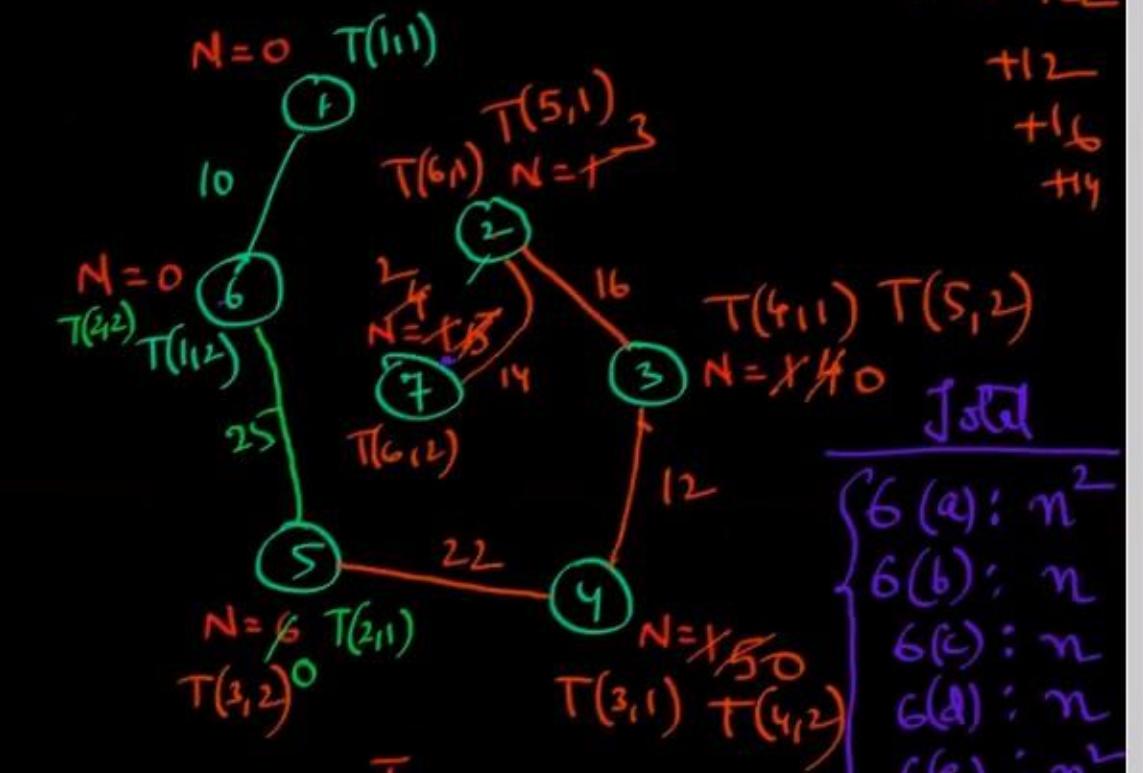
$$3) j \leftarrow 4$$

$$4) j \leftarrow 3$$

$$5) j \leftarrow 2$$

∴  $\langle k, l \rangle = \langle 1, 6 \rangle$

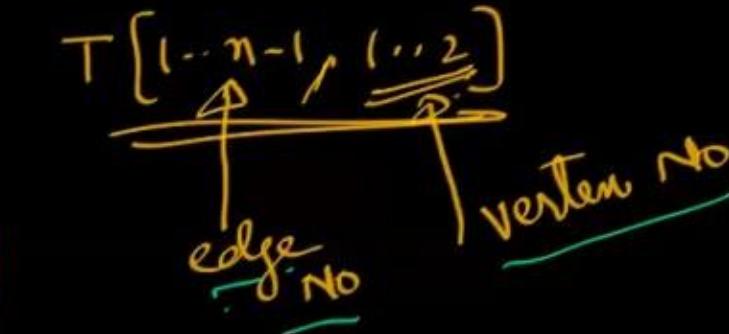
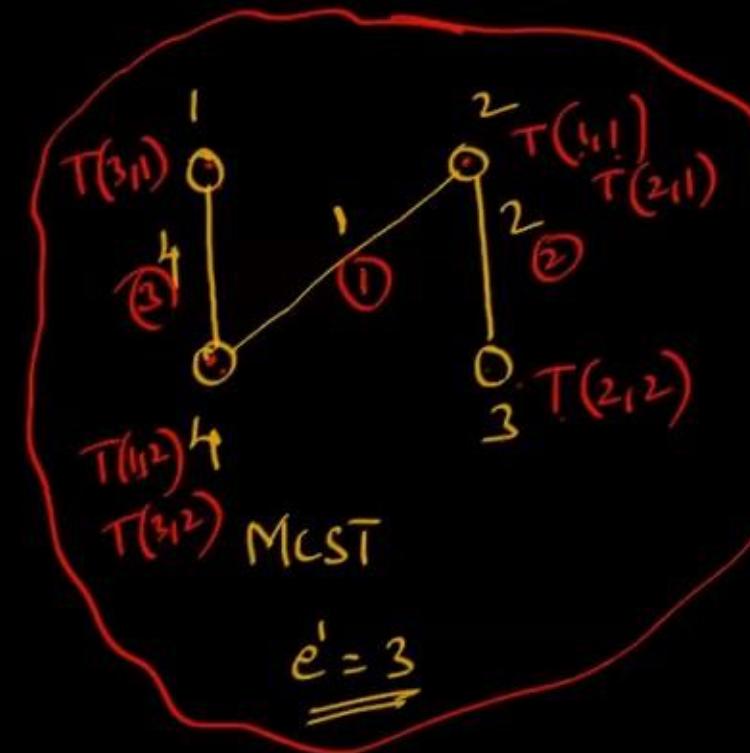
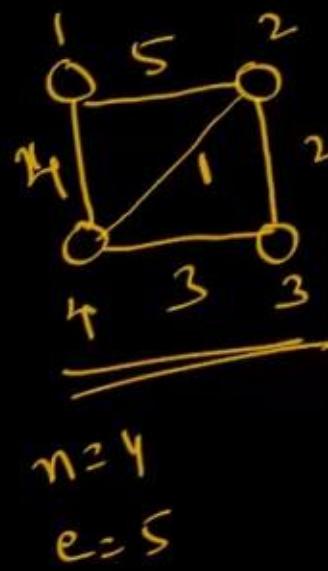
$$\begin{aligned} \text{mincost} &\leftarrow 10 \\ &+ 25 + 22 \\ &+ 12 \\ &+ 16 \\ &+ 14 \end{aligned}$$



Time:  $O(|E|) + 1 + 1 + n + 1 + n^2 = O(|E| + n^2)$

Non-Heap Impl. =  $O(n^2)$

(MST)



```
line procedure PRIM(E, COST, n, T, mincost)
```

```
//E is the set of edges in G// ✓
//COST(n, n) is the cost adjacency matrix of an n vertex graph//
//such that COST(i, j) is either a positive real number or + ∞ if//
//no edge (i, j) exists. A minimum spanning tree is computed and//
//stored as a set of edges in the array T(1:n - 1, 2). (T(i, 1),//
//T(i, 2)) is an edge in the min-cost spanning tree. The final cost//
//is assigned to mincost//
```

1. real COST(n, n), mincost;
2. integer NEAR(n), n, i, j, k, l, T(1:n - 1, 2);
3. (k, l) ← edge with minimum cost
4. mincost ← COST(k, l)
5. (T(1, 1), T(1, 2)) ← (k, l)
6. for i = 1 to n do //initialize NEAR//
7.     if COST(i, l) < COST(i, k) then NEAR(i) ← l
 else NEAR(i) ← k endif
8. repeat
9.     NEAR(k) ← NEAR(l) ← 0
10. 6. for i = 2 to n - 1 do //find n - 2 additional edges for T//
11.     let j be an index such that NEAR(j) ≠ 0 and COST(j, NEAR(j)) is minimum
12.     a) (T(i, 1), T(i, 2)) ← (j, NEAR(j))
13.     mincost ← mincost + COST(j, NEAR(j))
14.     NEAR(j) ← 0
15.     for k = 1 to n do //update NEAR//
16.         if NEAR(k) ≠ 0 and COST(k, NEAR(k)) > COST(k, j) then NEAR(k) ← j endif
17.     repeat
18. repeat
19. if mincost ≥ ∞ then print ('no spanning tree') endif
20. end PRIM

Heap: Insert:  $O(\log n)$   
          Delete:  $O(\log n)$   
          Create:  $O(n)$   
          Dec-key:  $O(\log n)$

## Prim's Algo: Heap Implementation :

1. Create a heap of edges :  $O(e)$ ; delete:  $O(\log e)$  :  $O(n)$

2. 1

3. 1

4. n

5. 1

6. (a) Create a heap of vertices with cost

$n \cdot \log n$

b)  $n$

c)  $n$

d)  $n$

e)  $n^2 \log n \sim e \cdot \log n$

$$\text{Time: } O((n+e) \cdot \log n) \quad e \sim O(n^2) \\ = O(n^2 \log n)$$

$$\text{Time: } e + n + 1 + 1 + n + 1 + n \log n + 3n + e \log n \\ = (n+e) \log n$$

```

1    $T \leftarrow \emptyset$ 
2   while  $T$  contains fewer than  $n - 1$  edges and  $E \neq \emptyset$  do
3     {a choose an edge  $(v, w)$  from  $E$  of lowest cost
4     {b delete  $(v, w)$  from  $E$ 
5       c if  $(v, w)$  does not create a cycle in  $T$ 
6         then add  $(v, w)$  to  $T$ 
7         else discard  $(v, w)$ 
8       endif
9     repeat

```

## Minimum Spanning Trees 181

$\rightarrow$  create Heap( $e$ ):  $O(e)$

} Min-Heap : Log e

$$\frac{O(e \cdot \text{Log } e)}{O(n^2 \log n)} \cdot \text{Mark} = \text{Complete}$$

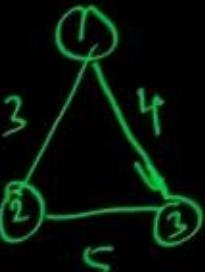
$e \sim n^2$

**Algorithm 4.9** Early form of minimum spanning tree algorithm due to Kruskal

8. Consider a graph with 'n' vertices  $n > 2$ . The vertices are numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  are adjacent iff  $0 < |i - j| \leq 2$ . The weight of such an edge is  $i + j$ . The weight of minimum cost Spanning Tree of such a graph for a value of n is \_\_\_\_\_.

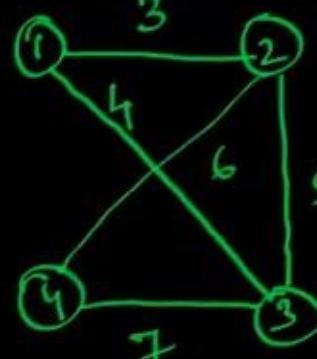
9. Consider a complete weighted Graph with n vertices numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  having edge between them has a cost value of  $2|i - j|$ . The weight of minimum cost Spanning Tree of such a graph is \_\_\_\_\_.

$$n=3:$$



$$\therefore 3+4 \\ 3+2(2)$$

$$n=4$$



$$\therefore 3+4+6 \\ 3+2(2+3)$$

$$n=5$$

$$\therefore 3+4+6+8 \\ \therefore 3+2(2+3+4)$$

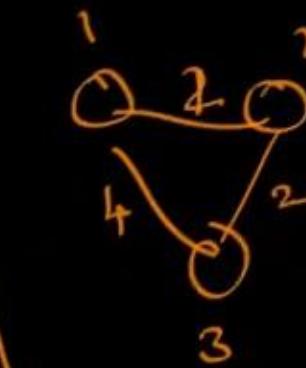
$$(n)$$

$$\therefore 3+2(2+3+\dots+n-1)$$

$$3+2\left(\frac{n(n-1)}{2}-1\right)$$

$$3+2\left(\frac{n^2-n-2}{2}\right)$$

$$n^2-n+1$$



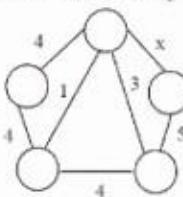
$$\therefore 2(n-1)$$



10. Let G be a complete undirected graph with 4 vertices and edge weights are {1, 2, 3, 4, 5, 6}. The maximum possible weight that a minimum weight Spanning Tree can have is 7.

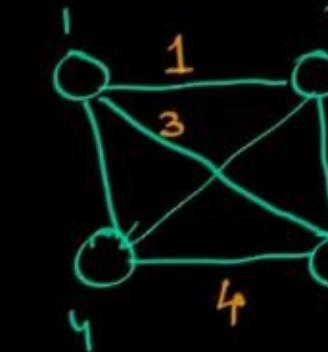
11. Let G be a connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of G is 500. When the weight of each edge of G is increased by five, the weight of a minimum spanning tree becomes 5.

12. Consider the following undirected graph G:



Choose a value for x that will maximize the number of minimum weight spanning trees (MWSTs) of G. The number of MWSTs of G for this value of x is 4.

Q10



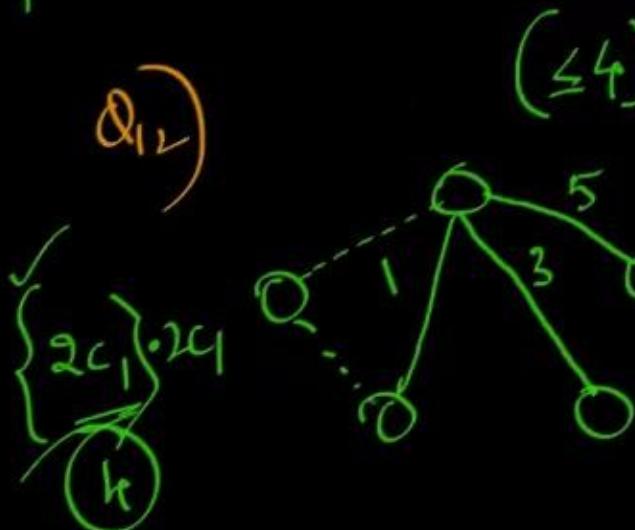
$$\therefore \underline{1+2+4} : \{7\} \quad \checkmark$$

$$\therefore 1+2+3 = \{6\}$$

Q11)  $\{n=100\}$

$$C = \underline{50} + 99 \times 5 = 995 \quad \checkmark$$

- (i)  $\leq 4 : 2$
- (ii)  $\underline{5} : 4 \quad \checkmark$
- (iii)  $> 5 : 2$



# Single Source Shortest Paths (SSSP):

Shortest Paths Problem :

(i) Single-Pair Sht Path:



(ii) Single Source Shortest Paths:

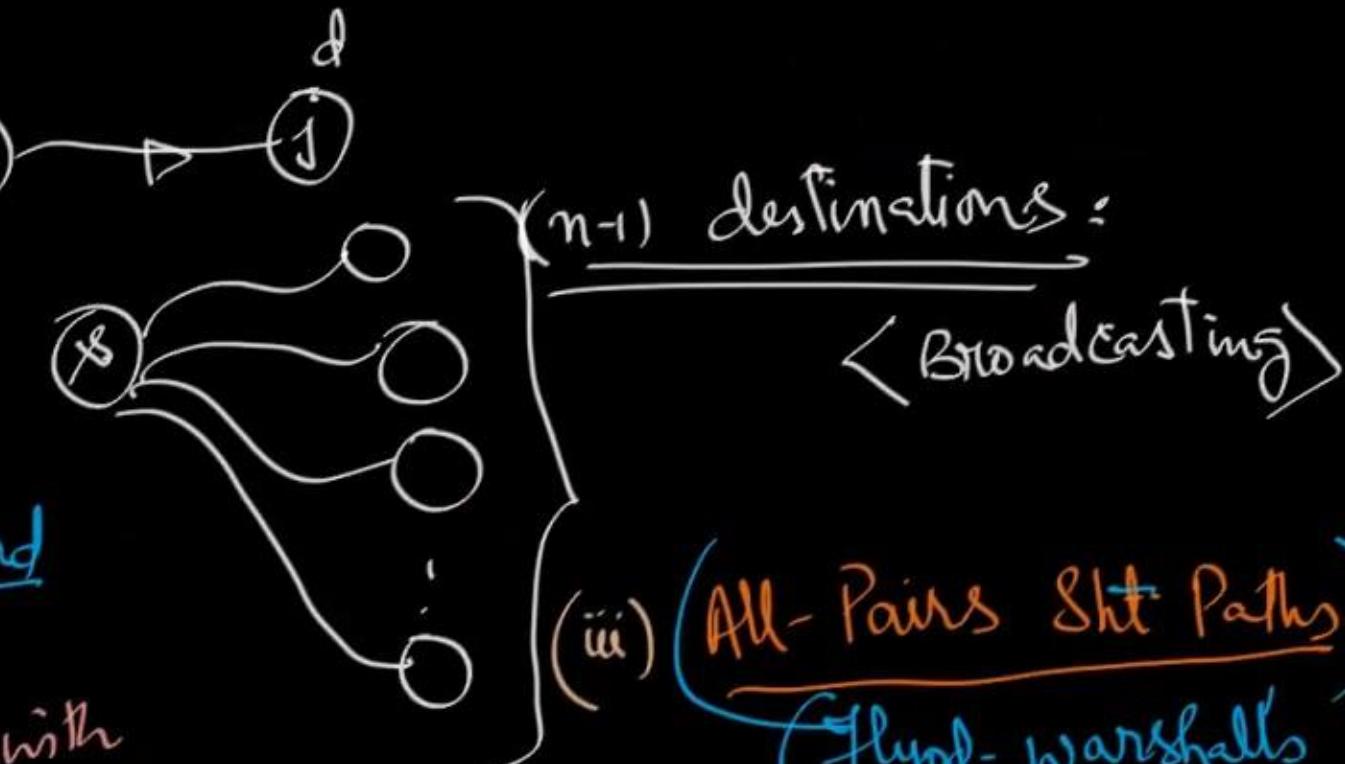
1) Dijkstra's  
Algo

→ Greedy  
→ +ve wt.  
edges

2) Bellman-Ford  
→ (D.P.)

→ Can work with  
-ve wt edges

→ (-ve wt cycle)  
(No Algo. Can work)



(iii) All-Pairs Sht Paths

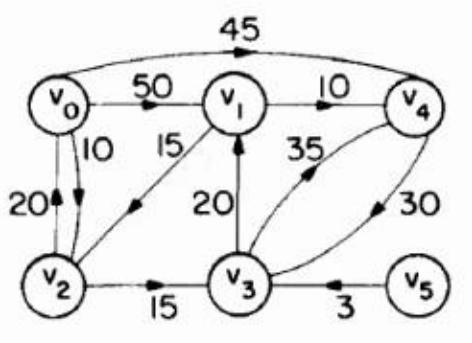
Floyd-Warshall's  
Algo

1 ≤ → DP  
2 ≤  
3 ≤  
n ≤

# Dijkstra's Single Source Sht. Paths:

 $s = v_0$ 

## a) Matrix Method:



destination  $\rightarrow d$ -values

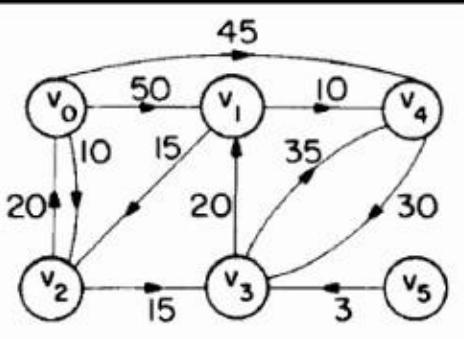
| Vertex Selected               | $v_0$ | $v_1$ | $v_2$ | $v_3$    | $v_4$ | $v_5$    |
|-------------------------------|-------|-------|-------|----------|-------|----------|
| $\{v_0\}$                     | -     | 50    | (10)  | $\infty$ | 45    | $\infty$ |
| $\{v_0, v_2\}$                | -     | 50    | (10)  | (25)     | 45    | $\infty$ |
| $\{v_0, v_2, v_3\}$           | -     | (45)  | (10)  | (25)     | 45    | $\infty$ |
| $\{v_0, v_2, v_3, v_1\}$      | -     | (45)  | (10)  | (25)     | (45)  | 25       |
| $\{v_0, v_1, v_2, v_3, v_4\}$ | -     | (45)  | (10)  | (25)     | (45)  | 20 ✓     |

$d(x)$ : Cost of reaching the vertex 'x' from 's'

 $s = v_5$ 

Relaxation


 $c_1 < c_2$



$$\delta = v_5$$

Vertice Selected

$\{v_5\}$

$\{v_5, v_3\}$

$\{v_1, v_5, v_3\}$

$\{v_5, v_3, v_1, v_4\}$

$\{v_5, v_3, v_1, v_4, v_2\}$

d-values

Matrix - Method

|           | $v_0$    | $v_1$    | $v_2$    | $v_3$ | $v_4$    | $v_5$ |
|-----------|----------|----------|----------|-------|----------|-------|
| $\{v_5\}$ | $\infty$ | $\infty$ | $\infty$ | 3     | $\infty$ | -     |

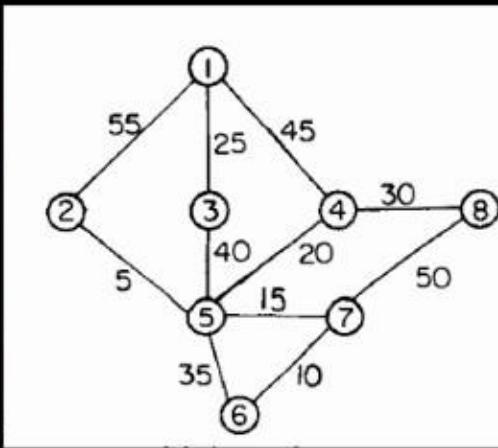
|  |          |    |          |   |    |   |
|--|----------|----|----------|---|----|---|
|  | $\infty$ | 23 | $\infty$ | 3 | 38 | - |
|--|----------|----|----------|---|----|---|

|  |          |    |    |   |    |   |
|--|----------|----|----|---|----|---|
|  | $\infty$ | 23 | 38 | 3 | 33 | - |
|--|----------|----|----|---|----|---|

|  |          |    |    |   |    |   |
|--|----------|----|----|---|----|---|
|  | $\infty$ | 23 | 38 | 3 | 33 | - |
|--|----------|----|----|---|----|---|

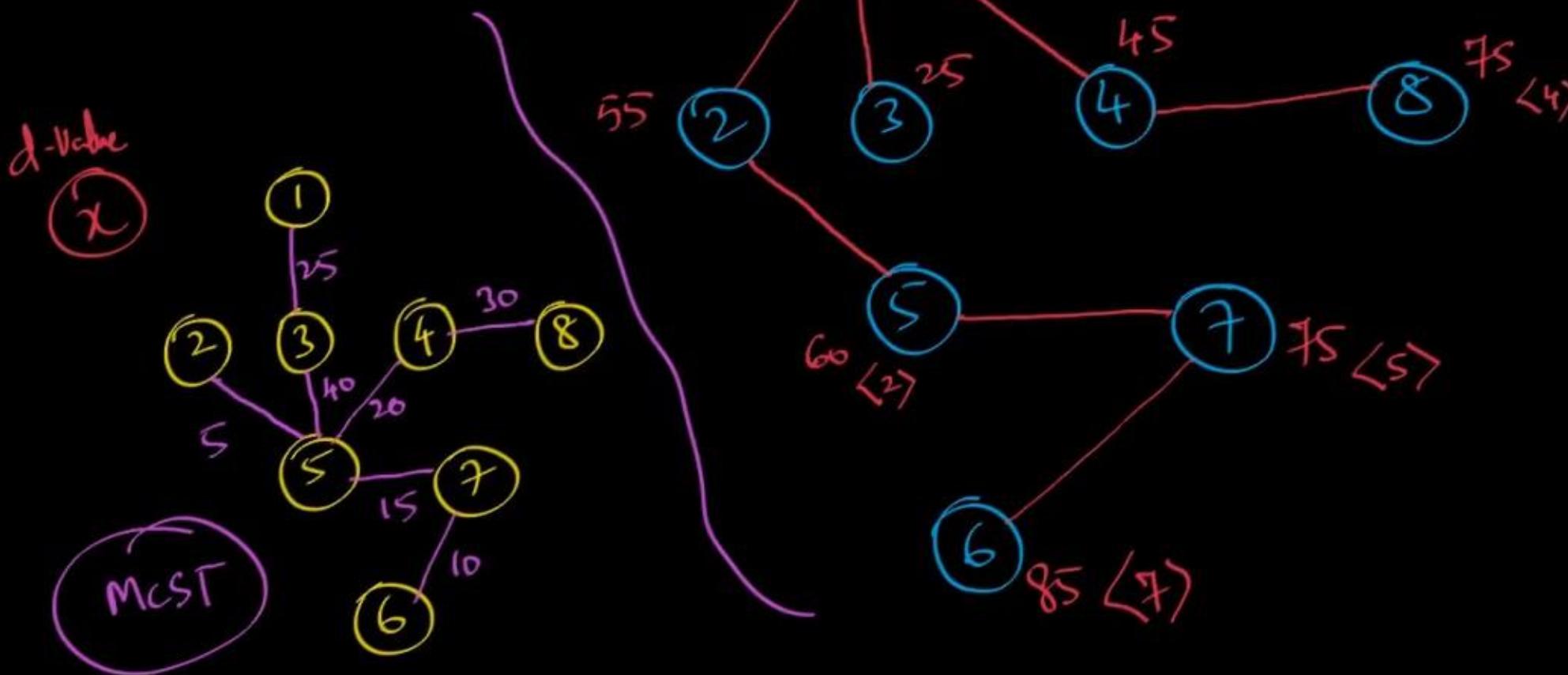
|  |    |    |    |   |    |   |
|--|----|----|----|---|----|---|
|  | 58 | 23 | 38 | 3 | 33 | ✓ |
|--|----|----|----|---|----|---|

Paths are  
not shown.



Spanning Tree Method: of Dijkstra's Algo.  
 $S = \{1\}$

S-S-S-P Sp. Tree



array with  $s(i) = 0$  if vertex  $i$  is not in  $S$  and  $s(i) = 1$  if it is. It is assumed that the graph itself is represented by its cost adjacency matrix with  $\text{COST}(i, j)$  being the weight of the edge  $(i, j)$ .  $\text{COST}(i, j)$  will be set to some large number,  $+\infty$ , in case the edge  $(i, j)$  is not in  $E(G)$ . For  $i = j$ ,  $\text{COST}(i, j)$  may be set to any nonnegative number without affecting the outcome of the algorithm.

*src* → *d-value*

```

procedure SHORTEST-PATHS(v, COST, DIST, n)
    //DIST(j,  $1 \leq j \leq n$  is set to the length of the shortest path//
    //from vertex v to vertex j in a digraph G with n vertices.//
    //DIST(v) is set to zero. G is represented by its cost adjacency//
    //matrix, COST(n, n).//
    boolean S(1:n); real COST(1:n, 1:n), DIST(1:n)
    integer u, v, n, num, i, w
1   1. for i ← 1 to n do //initialize set S to empty//
2     S(i) ← 0; DIST(i) ← COST(v, i)
3     repeat
4   2. S(v) ← 1; DIST(v) ← 0 //put vertex v in set S//
5   3. for num ← 2 to n - 1 do //determine n - 1 paths from vertex v//
6     a choose u such that DIST(u) = min{DIST(w)}
        S(w) = 0
7     b S(u) ← 1 //put vertex u in set S//
8     c for all w with S(w) = 0 do //update distances//
9       DIST(w) ← min(DIST(w), DIST(u) + COST(u, w))
10    repeat
11  repeat
12 end SHORTEST-PATHS

```

**Algorithm 4.11** Greedy algorithm to generate shortest paths

**Analysis of Algorithm SHORTEST-PATHS**

From our earlier discussion, it is easy to see that the algorithm is correct. The time taken by the algorithm on a graph with  $n$  vertices is  $O(n^2)$ . To see this note that the **for** loop of line 1 takes  $\theta(n)$  time. The **for** loop of line 5 is executed  $n - 2$  times. Each execution of this loop requires  $O(n)$  time at

Tools | Fill & Sign | Comment | Sign In

Export PDF  
Convert PDF files to Word or Excel online.

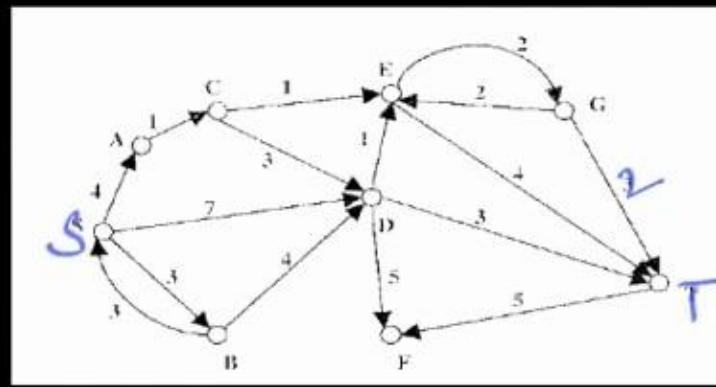
Select PDF File:  
Algorithms - Sahani.pdf  
1 file / 18.65 MB

Convert To:  
Microsoft Word (\*.docx)

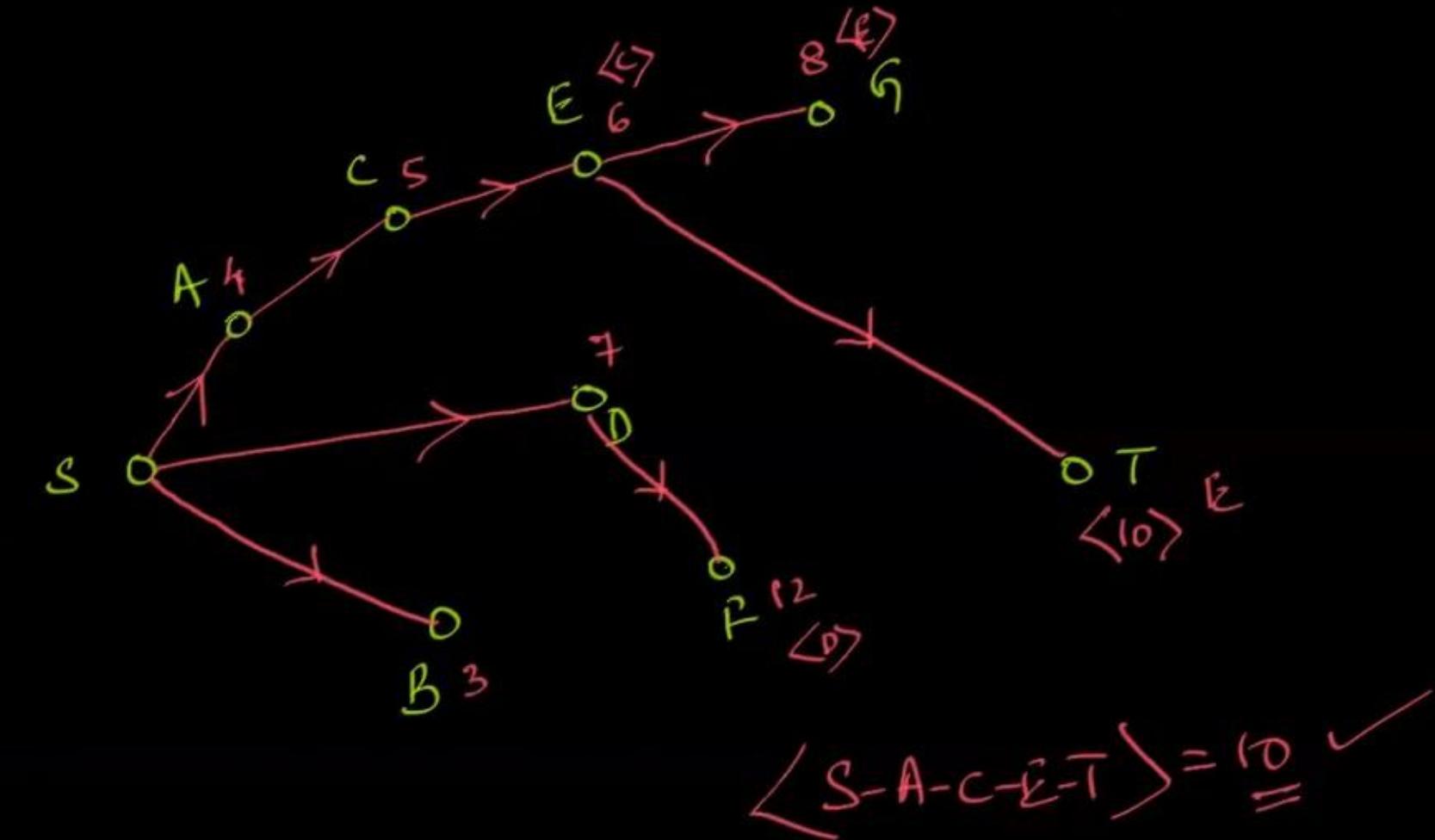
Recognize Text in English(U.S.) Change

Convert

- ▶ Create PDF
- ▶ Edit PDF
- ▶ Combine PDF
- ▶ Send Files
- ▶ Store Files

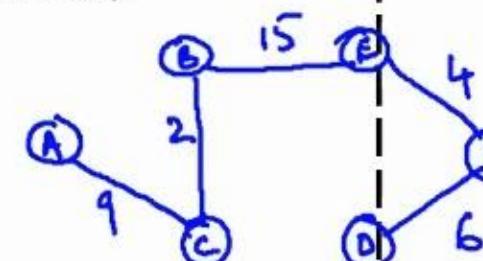
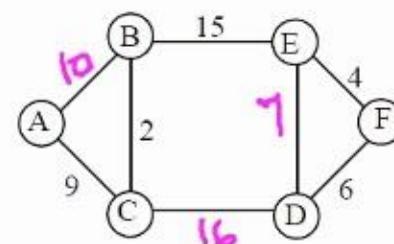


- a)  $S-D-T : 10$
- b)  $S-B-D-T : 10$
- c)  $\cancel{S-A-C-E-T : 10}$
- d)  $S-A-C-E-G-T : 10$



$\langle x_1, y_1 \rangle$  &  $\langle x_2, y_2 \rangle$  are adjacent iff  $|x_1 - x_2| \leq 1$  &  $|y_1 - y_2| \leq 1$ . The cost of such an edge is given by the distance between them. Compute the weight of min cost Spanning Tree of such graph for a value of n.

7. Consider the following Graph whose Minimum Cost Spanning Tree marked with edge values has a weight of 36. Minimum possible sum of all edges of the graph G is 69. (Assume that all edges have distinct cost).



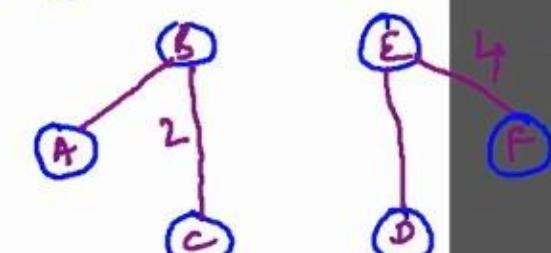
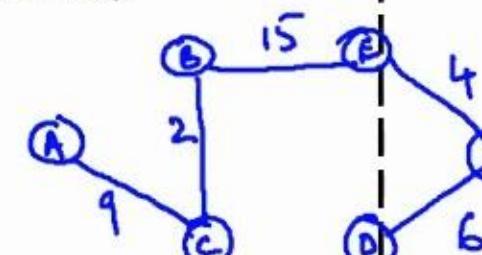
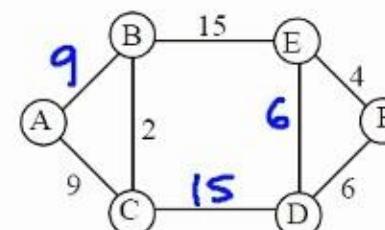
8. Consider a graph with 'n' vertices  $n > 2$ . The vertices are numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  are adjacent iff  $0 < |i - j| \leq 2$ . The weight of such an edge is  $i + j$ . The weight of minimum cost Spanning Tree of such a graph for a value of n is \_\_\_\_\_.

9. Consider a complete weighted Graph with n vertices numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  having edge between them has a cost value of  $2|i - j|$ . The weight of minimum cost Spanning Tree of such a graph is \_\_\_\_\_.

$\langle x_1, y_1 \rangle$  &  $\langle x_2, y_2 \rangle$  are adjacent iff  $|x_1 - x_2| \leq 1$  &  $|y_1 - y_2| \leq 1$ . The cost of such an edge is given by the distance between them. Compute the weight of min cost Spanning Tree of such graph for a value of n.

7. Consider the following Graph whose Minimum Cost Spanning Tree marked with edge values has a weight of 36. Minimum possible sum of all edges of the graph G is \_\_\_\_\_. (Assume that all edges have distinct cost).

No. of mcs Ts = 8



$$(2c_1 + 2c_1 + 2c_1) = 8$$

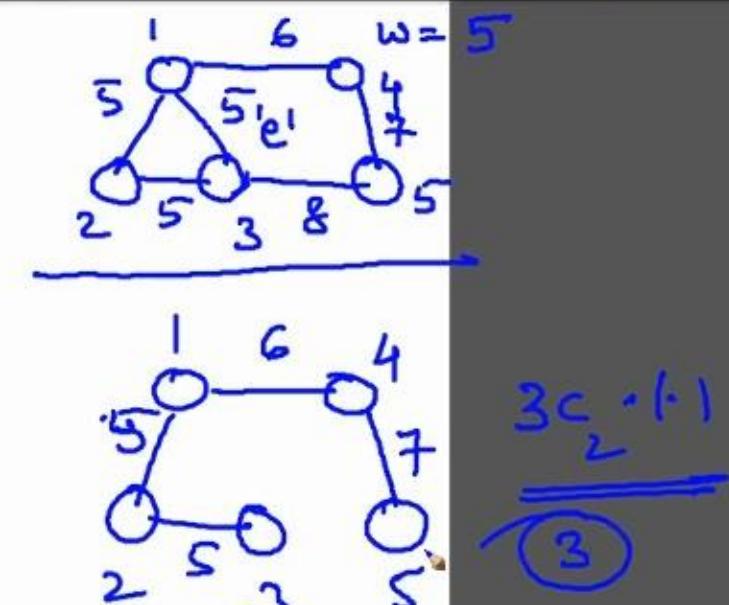
8. Consider a graph with 'n' vertices  $n > 2$ . The vertices are numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  are adjacent iff  $0 < |i - j| \leq 2$ . The weight of such an edge is  $i + j$ . The weight of minimum cost Spanning Tree of such a graph for a value of n is \_\_\_\_\_.

9. Consider a complete weighted Graph with n vertices numbered  $V_1$  to  $V_n$ . Two vertices  $V_i$  &  $V_j$  having edge between them has a cost value of  $2|i - j|$ . The weight of minimum cost Spanning Tree of such a graph is \_\_\_\_\_.

13. Let  $w$  be the minimum weight among all edge weights in an undirected connected graph. Let 'e' be a specific edge of weight ' $w$ '. Which of the following is False?

- i. There is a minimum Spanning Tree containing 'e' always. **T**
- ii. Every minimum Spanning Tree has an edge of weight ' $w$ '. **T**
- iii. 'e' is present in every minimum Spanning Tree. **F**
- iv. If 'e' is not present in a minimum Spanning Tree named 'T' then there will be a cycle formed by adding 'e' to T. **T**

14.  $G = (V, E)$  is an undirected simple graph in which each edge has a distinct weight, and  $e$  is a particular edge of  $G$ . Which of the following statements about the minimum spanning trees (MSTs) of  $G$  is/are **TRUE**?



14.  $G = (V, E)$  is an undirected simple graph in which each edge has a distinct weight, and  $e$  is a particular edge of  $G$ . Which of the following statements about the minimum spanning trees (MSTs) of  $G$  is/are **TRUE**?



- I. If  $e$  is the lightest edge of some cycle in  $G$ , then every MST of  $G$  includes  $e$

✓ II. If  $e$  is the heaviest edge of some cycle in  $G$ , then every MST of  $G$  excludes  $e$       *lightest*

(a) I only      (b) II only  
(c) both I and II      (d) neither I nor II

### *For Micro Notes by the Student*



15. Applying Dijkstra's Algorithm over the given Graph, Which path is reported from 'S' to 'T':



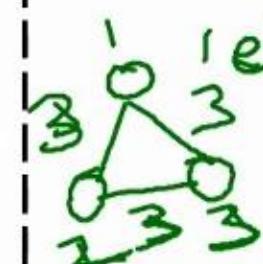
Choose a value for  $x$  that will maximize the number of minimum weight spanning trees (MWSTs) of  $G$ . The number of MWSTs of  $G$  for this value of  $x$  is \_\_\_\_.

13. Let  $w$  be the minimum weight among all edge weights in an undirected connected graph. Let ' $e$ ' be a specific edge of weight ' $w$ '. Which of the following is False?

- i. There is a minimum Spanning Tree containing ' $e$ ' always. ✓
- ii. Every minimum Spanning Tree has an edge of weight ' $w$ '. ✓
- iii. ' $e$ ' is present in every minimum Spanning Tree. ✗
- iv. If ' $e$ ' is not present in a minimum Spanning Tree named ' $T$ ' then there will be a cycle formed by adding ' $e$ ' to  $T$ .

14.  $G = (V, E)$  is an undirected simple graph in which each edge has a distinct weight, and  $e$  is a particular edge of  $G$ . Which of the following statements about the minimum spanning trees (MSTs) of  $G$  is/are TRUE?

$$w = 3$$

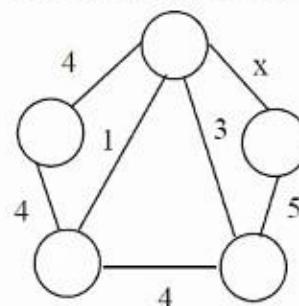




10. Let  $G$  be a complete undirected graph with 4 vertices and edge weights are  $\{1, 2, 3, 4, 5, 6\}$ . The maximum possible weight that a minimum weight Spanning Tree can have is \_\_\_\_\_.

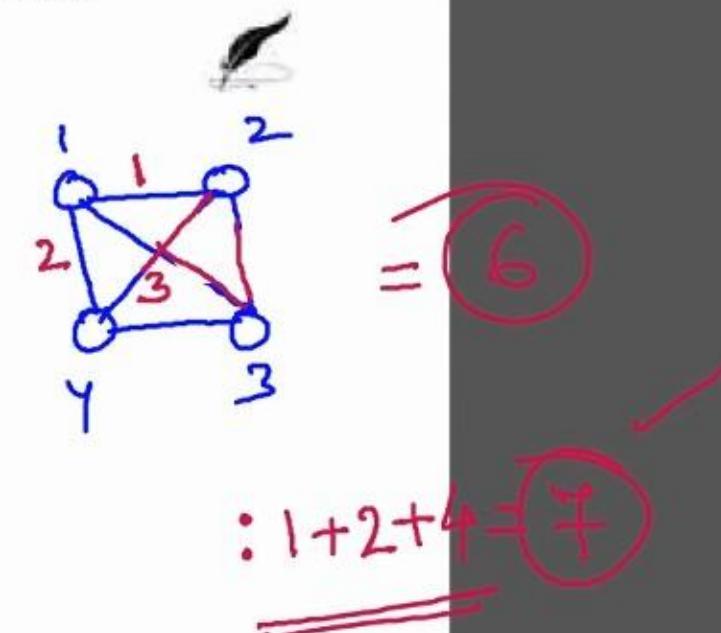
11. Let  $G$  be a connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of  $G$  is 500. When the weight of each edge of  $G$  is increased by five, the weight of a minimum spanning tree becomes \_\_\_\_\_.

12. Consider the following undirected graph  $G$ :



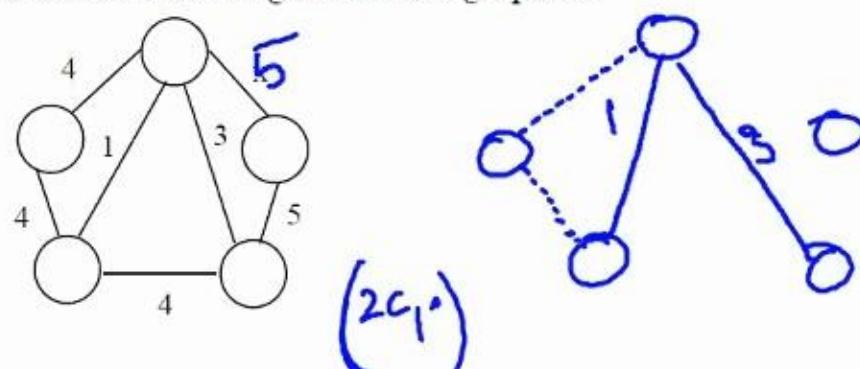
Choose a value for  $x$  that will maximize the number of minimum weight spanning trees (MWSTs) of  $G$ . The number of MWSTs of  $G$  for this value of  $x$  is \_\_\_\_\_.

For Micro Notes by the Student



11. Let  $G$  be a connected undirected graph of 100 vertices and 300 edges. The weight of a minimum spanning tree of  $G$  is 500. When the weight of each edge of  $G$  is increased by five, the weight of a minimum spanning tree becomes \_\_\_\_\_.

12. Consider the following undirected graph  $G$ :



Choose a value for  $x$  that will maximize the number of minimum weight spanning trees (MWSTs) of  $G$ . The number of MWSTs of  $G$  for this value of  $x$  is 4.

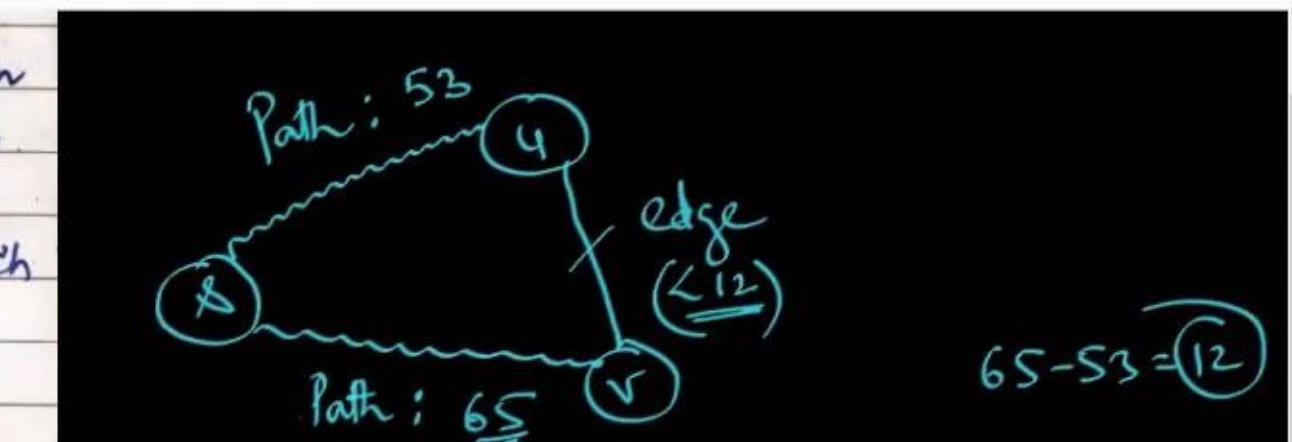
13. Let  $w$  be the minimum weight among all edge weights in an undirected connected graph. Let ' $e$ ' be a specific edge of weight ' $w$ '. Which of the following is False?

- There is a minimum Spanning Tree containing ' $e$ ' always.
- Every minimum Spanning Tree has an edge of weight ' $w$ '.
- ' $e$ ' is present in every minimum Spanning Tree.
- If ' $e$ ' is not present in a minimum Spanning Tree named ' $T$ ' then there will be a cycle formed by adding ' $e$ ' to  $T$ .

1)  $x > 5 : 2c_1$   
2)  $x < 5 : \underline{2c_1}$   
3)  $\underline{x = 5} : (2c_1) - (2c_1)$   
2 (4)

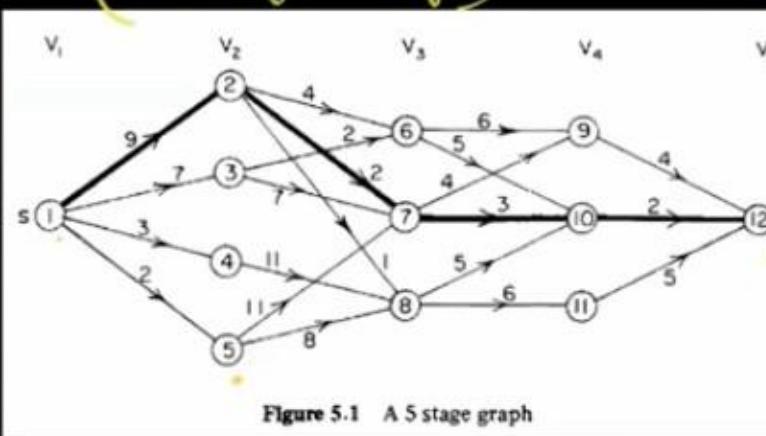
Q) Consider a weighted undirected graph & let  $uv$  be an edge in the graph. It is known that the shortest path from the source vertex 'S' to 'u' has weight 53 & the shortest path from 'S' to 'v' has weight 65. Which is true?

- a) weight  $(u, v) \leq 12$
- b) weight  $(u, v) \leq 12$
- c) weight  $(u, v) > 12$
- d) weight  $(u, v) \geq 12$  ✓



## Dynamic Programming (DP)

( $\ell$ -Stage graph)



**Figure 5.1** A 5 stage gra

## Multi-Stage Graph

$$G = (V, E)$$

$$|v|=n; |E|=e$$

$$(V_i \xrightarrow{=} V_{i+1})$$

Shortest Path : S → t

Richard Bellman (1950's): Rand Corp.

## Greedy Method

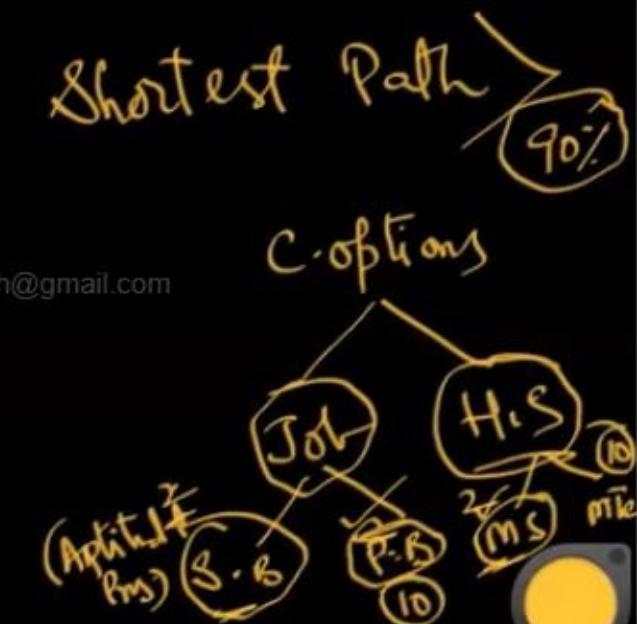
$$\frac{s}{\text{Greedy Number}} \left\langle 1 - 5 - 8 - 10 - 12 \right\rangle : 17 \checkmark$$

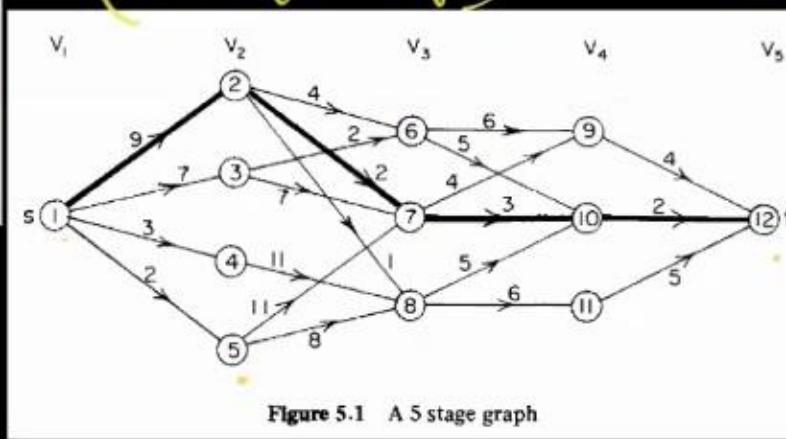
$$\left\{ \begin{array}{l} \langle 1-2-7-10-12 \rangle \\ \langle 1-3-6-10-12 \rangle \end{array} \right\} : \underline{\underline{16}}$$

Greedy Method fails to find Shortest Path  
↳ Short Sighted  
+ Options

akshintalamahvith@gmail.com

Colleges



Dynamic Programming (DP)( $\ell$ -Stage graph)Multi-Stage graph:

$$G = (V, E)$$

$$|V| = n; |E| = e$$

$$(V_i \rightarrow V_{i+1})$$

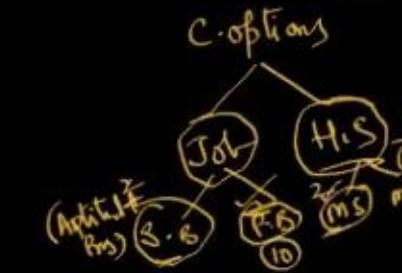
Shortest Path:  $s \rightarrow t$ Richard Bellman (1950's)

{Greedy Method}

$$\checkmark \quad \begin{matrix} & t \\ & \swarrow \\ 1 - 5 - 8 - 10 - 12 \end{matrix} : 17 \quad \checkmark$$

$$\left\{ \begin{matrix} 1 - 2 - 7 - 10 - 12 \\ 1 - 3 - 6 - 10 - 12 \end{matrix} \right\} : 16$$

Greedy Method fails to find shortest Path (90%)  
 ↳ Short Sighted

Rand Corp.

M-S-G: {Single-Pair Short Path}

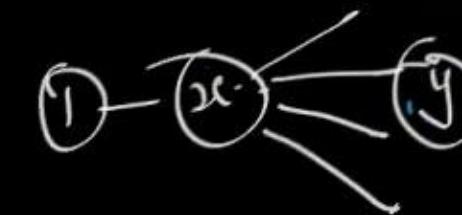
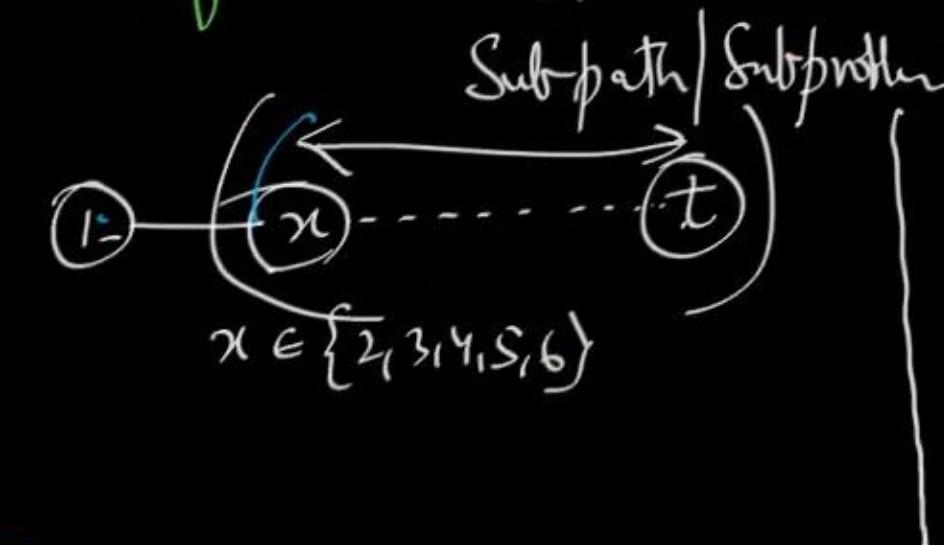
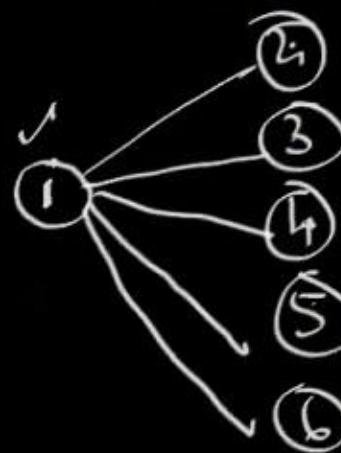
Dynamic ProgrammingBrute Force  
Naive approach  
EnumerationDijkstra's (SSSP)  
(Greedy Method)

$$\text{Cost: } s \rightarrow t : 16 \quad \checkmark$$

Brute Force: enumerate all possibilities &amp; pick up the correct / best

- D.P is an algorithm design strategy, used for solving problems whose solutions are viewed as a result of making a sequence of decisions.
- one way of making these decisions is to make them in a step-wise manner based on local info available at every steps.
- This is true for problems solvable by Greedy Method;
- But for many other problems, like Multistage Graphs, optimal solution is not achievable by making decisions step-wise;
- one way of solving such problems is to enumerate all possibilities & then pickup the best (Brute-Force)
- The drawback of Brute-Force is excessive time & space requirements
- D.P based on enumeration, often tries to curtail (reduce) the amount of enumeration by removing those decision sequences that are either infeasible or sub-optimal. (Hence reduce T.C)

- In Dp the sequence of optimal decisions are made by appealing to the Principle of optimality (Global optimality)  
(optimal substructure  
Property)
  - Principle of optimality: States that whatever the initial state and the decision are the remaining sequence of decisions must constitute an optimal decision sequence with regard to the state resulting from first decision.

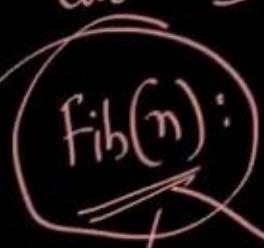


- The fundamental diff. b/w Greedy Method & D-P is that, Greedy Method always generates only one decision sequence, whereas D-P may generate multiple decision sequences,
- Another imp. Property of D-P is that the solutions/results of the subproblems are tabulated (cached) so that if required later can be reused (recall) rather than recompute again ;  
(Can result in reduction of time) [Memoization + Tabulation]
- D-P like DandC also divides the problem into subproblems but in DandC the subproblems are independent whereas in D-P we may have overlapping subproblems ;

## Elements of D.P:

- { (i) The division of problem into subproblems
- (ii) Property of optimal substructure
- (iii) Existence of overlapping subproblems

Case Study



Recursive

Iterative

## Memoization

(Top-down)  
(Recursive Impl.)

## Tabulation

<Bottom-up>

Impl: (Non-Recursive  
Impl)

## Approaches of D.P

: (Mathematically, the solution to the problem is always expressed as a Mathematical Recurrence)



# Case - Study of $n^{\text{th}}$ Fibonacci Number :

$$\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1)$$

$$\begin{cases} \text{fib}(0) = 0 \\ \text{fib}(1) = 1 \end{cases}$$

with Traditional Recursion  
(No Memoization)

$$\left\{ \begin{array}{l} T(n) = T(n-1) + T(n-2) + O(1) \\ T(1) = 1 \end{array} \right.$$

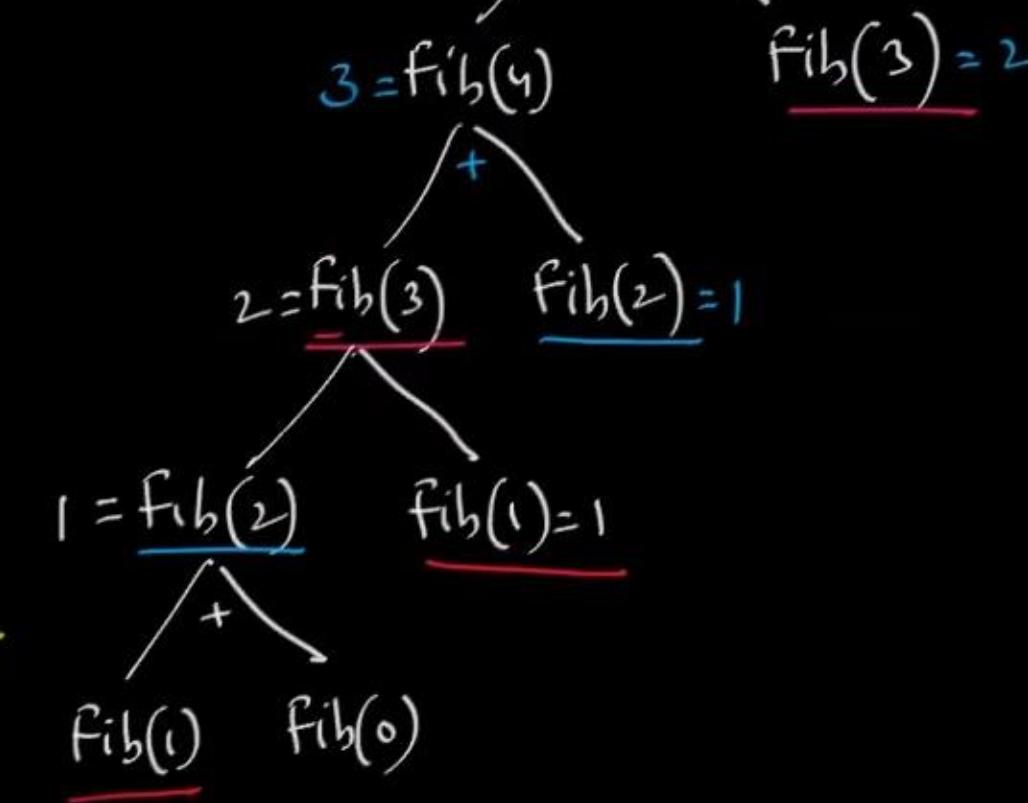
$$T(n-2) < T(n-1)$$

$$T(n) < 2T(n-1) + O(1)$$

$O(2^n)$  Exponential

$$\text{fib}(5) = 5$$

Memoization



## Memoized Implementation of Fib(n):

Memfib(n)

```

{
    if (mem[n] is undefined)
    {
        if (n < 2) return n;
        else result = Memfib(n-2) + Memfib(n-1);
        mem[n] = result; // Caching (Memoizing)
    }
    else return (mem[n]);
}
  
```

(Top-down approach)

↑ Reusing

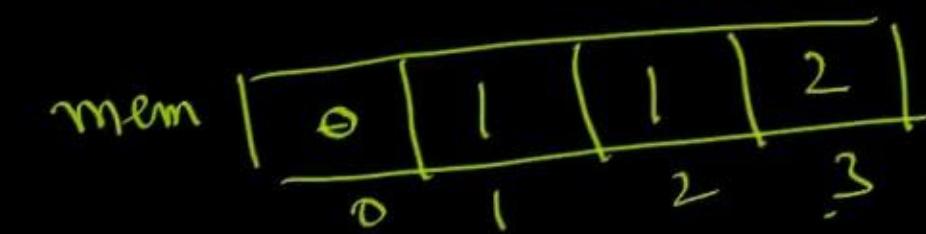
Memfib(3)

→ Memfib(2)

Memfib(1) : cache

Memfib(0) :

→ Memfib(1) // used cached result //



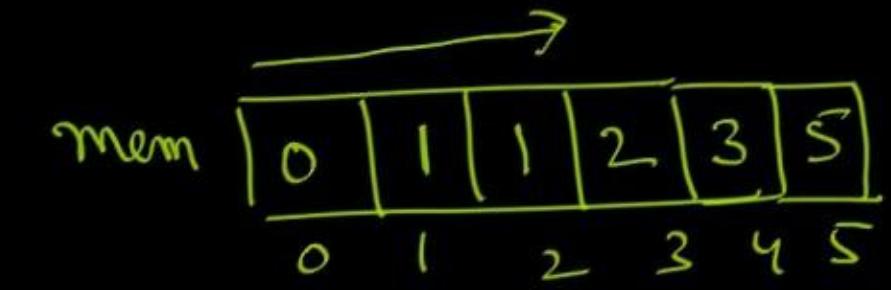
Memfib(3)

Time: O(n)

# Tabulation based implementation of fib(n) | Bottom-up

```

    {
        Tab-fib(n)
        {
            mem[0] = 0; } Boundary Conditions
            mem[1] = 1;
            for i <= 2 to n
                mem[i] = mem[i-2] + mem[i-1];
            return (mem[n]);
        }
    }
    (Iterative)
  
```



Time-Complexity  
 $O(n)$

Space:  $O(n)$

## Multistage Graph : (l-stage)

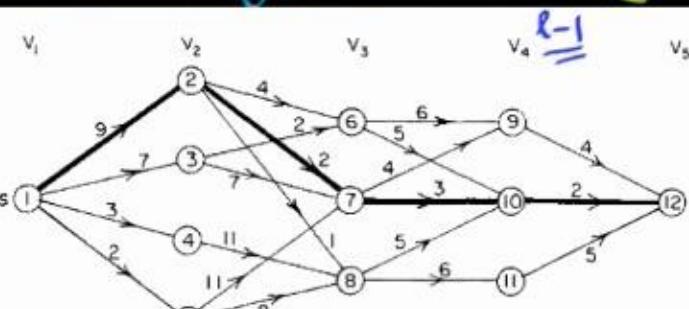


Figure 5.1 A 5 stage graph

Developing Soln to the Problem  
as a Mathematical Recursive  
formula :

| $c$      | 1 | 2 | 3 | $\dots$ | 12 |
|----------|---|---|---|---------|----|
| 1        | - | 9 | 7 | $\dots$ |    |
| 2        |   |   |   | $\dots$ |    |
| $i$      |   |   |   | $\dots$ |    |
| 3        |   |   |   | $\dots$ |    |
| $\vdots$ |   |   |   | $\dots$ |    |
| 12       |   |   |   | $\dots$ |    |

$c(i, j) = \text{edge cost}$   
b/w vertices  
'i' & 'j'

Path Cost :  $s \rightarrow t$  :

Let  $\text{cost}(i, j)$  repr. Cost of the Path from  
vertex 'j' which is in Stage 'i' to the destination  
vertex 't';

$\forall i$

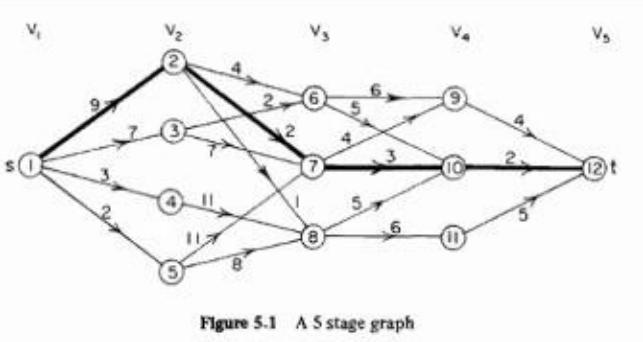
$j \dashrightarrow t : \text{cost}(i, j)$

$$\text{cost}(i, i) = \min_{\forall k} \{ c(i, k) + \text{cost}(2, k) \} \quad \text{--- (1)}$$

$$\text{cost}(i, i) = \min_{\substack{\forall k \\ \{k \in V_2\} \\ \{(i, k) \in E\}}} \{ c(i, k) + \text{cost}(2, k) \}$$

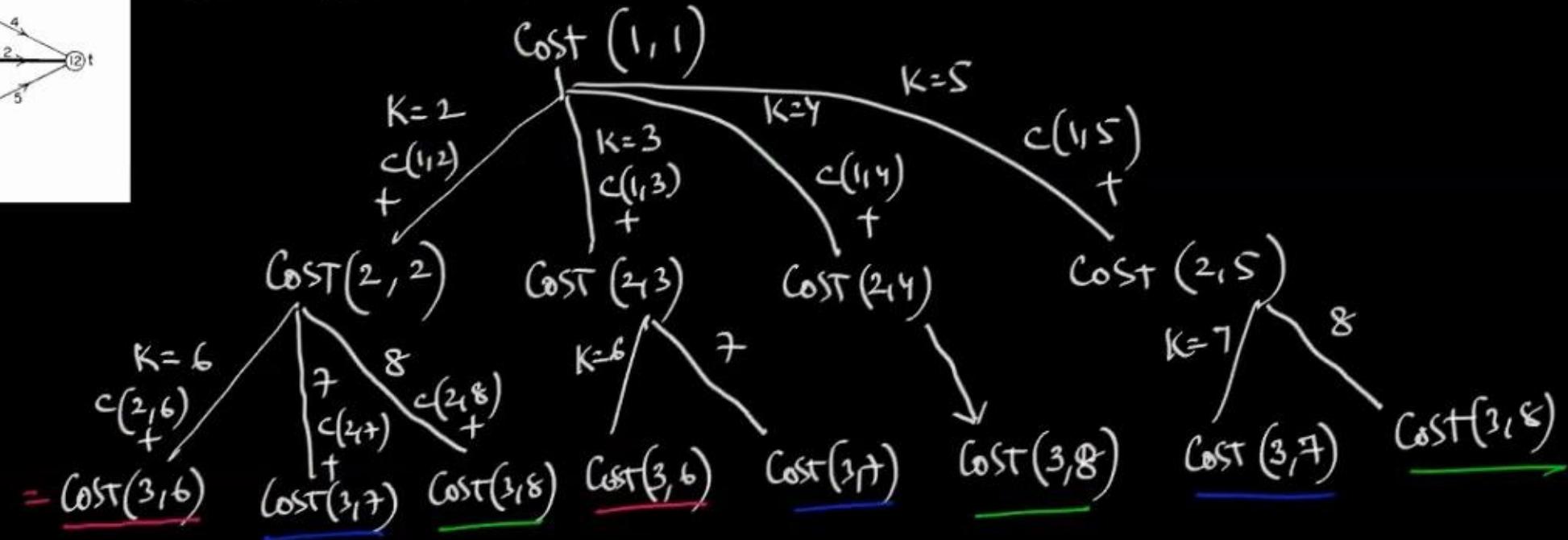
$$\text{cost}(i, j) = \min_{k \in V_{i+1}} \{ c(j, k) + \text{cost}(i+1, k) \} \quad \text{--- (2)}$$

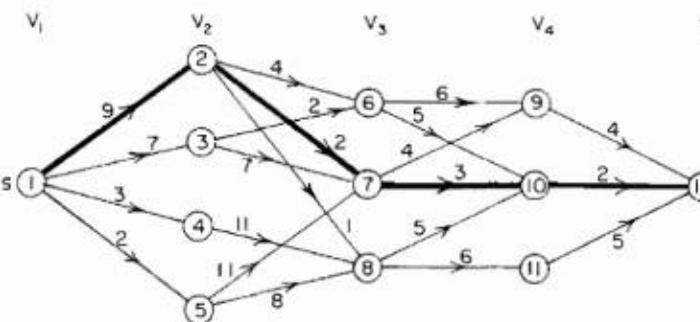
$$\text{cost}(l-1, j) = c(j, t) \quad D(i, j) = 'k' \text{ that minimizes } \text{eq(2)}$$



Demonstrates overlapping Subproblems

### Recursion Tree:





$$\begin{aligned} \text{Cost}(1,1) &= \min \left\{ \begin{array}{l} \underset{K=2}{c(1,2)} + \text{Cost}(2,1), \quad c(1,3) + \text{Cost}(2,3) \\ \underline{9+7} \end{array} \right. \\ &= \underline{\underline{16}} \\ &\quad \left. \begin{array}{l} \underset{K=3}{c(1,4)} + \text{Cost}(2,4), \quad c(1,5) + \text{Cost}(2,5) \\ \underline{11+18} \end{array} \right. \\ &\quad \left. \begin{array}{l} \underset{K=4}{c(1,6)} + \text{Cost}(2,6), \quad c(1,7) + \text{Cost}(2,7) \\ \underline{2+15} \end{array} \right. \end{aligned}$$

$$D(1,1) = 2$$

$$\begin{aligned} \text{Cost}(4,9) &= c(3,4) = c(9,4) = 4 \\ \text{Cost}(4,10) &= 2 \\ \text{Cost}(4,11) &= 5 \end{aligned}$$

$$K=9$$

$$\begin{aligned} \text{Cost}(3,6) &= \min \left\{ \begin{array}{l} \underset{K=6}{c(6,9)} + \text{Cost}(4,9) \\ \underline{6+4} \end{array} \right. \\ &= \underline{\underline{10}} \\ &\quad \left. \begin{array}{l} \underset{K=10}{c(6,10)} + \text{Cost}(4,10) \\ \underline{5+2} \end{array} \right. \end{aligned}$$

$$\begin{aligned} \text{Cost}(3,7) &= 5; \\ D(3,7) &= 10 \end{aligned}$$

$$\begin{aligned} \text{Cost}(3,8) &= 7; \\ D(3,8) &= 10 \end{aligned}$$

$$\begin{aligned} \text{Cost}(2,2) &= 7 \\ D(2,2) &= \underline{\underline{7}} \end{aligned}$$

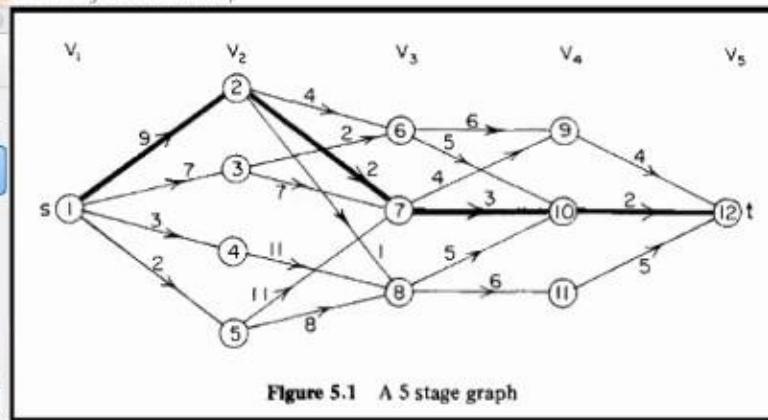
$$\begin{aligned} \text{Cost}(2,5) &= 15 \\ D(2,5) &= \underline{\underline{8}} \end{aligned}$$

$$\text{Cost}(2,3) = 9$$

$$D(2,3) = \underline{\underline{6}}$$

$$\text{Cost}(2,4) = 18$$

$$D(2,4) = \underline{\underline{8}}$$



$\ell$ -stages  $\rightarrow \underline{(\ell-2)}$  decisions

$$D(1, 1) = 2$$

$$D(2, \underline{D(1, 1)}) = D(2, 2) = 7$$

$$D(3, \underbrace{D(2, \underline{D(1, 1)})}_7) = D(3, 7) = 10$$

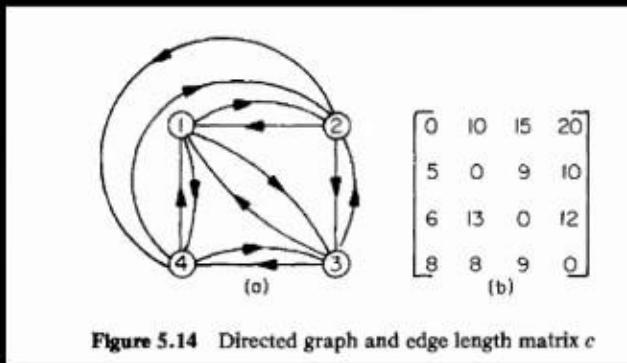
Construction of Path:

$$\boxed{1 - 2 - 7 - 10 - 12} = 16$$

{Sahni}

(Develop an iterative algo (Tabulation)  
for  
Multi Stage Graph)

## ② Travelling Salesperson Problem (TSP)

Figure 5.14 Directed graph and edge length matrix  $c$ 

{Tour}: (Hamiltonian cycle)  
n-cities

Home-city:  $v_0$   
 $(n-1)$ : exactly cities

Pvs(NP)

Intractable: (TSP): (No Poly-time  
(NP)  $\in$  NP Algo exists)

Milk man : postman :

Greedy Method:

$$v_0 = 1$$

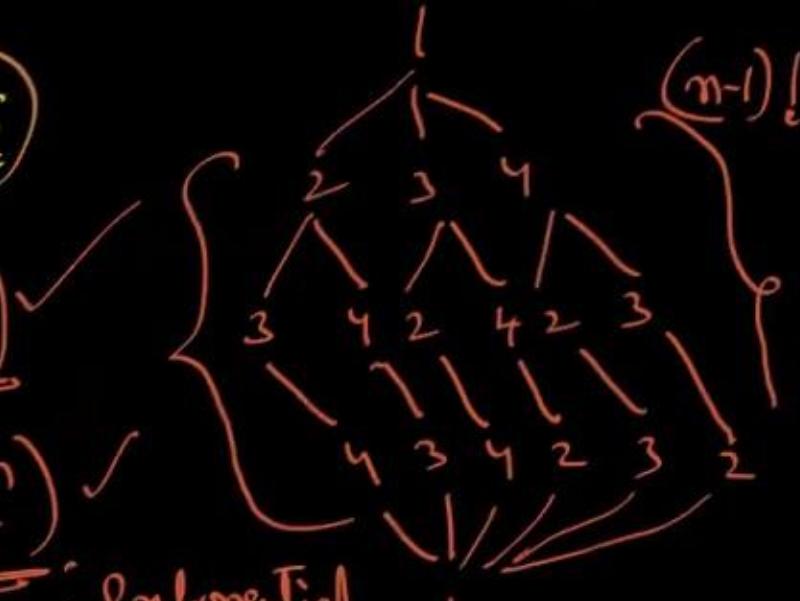
1-2-3-4-1:  $\{39\}$  : Fails

Tour: 1-2-4-3-1: 35

Soln Space:  $(n-1)!$   $\Rightarrow O(n^n)$

Time-complexity:  $O(n^2 n)$

: Exponential





18. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:

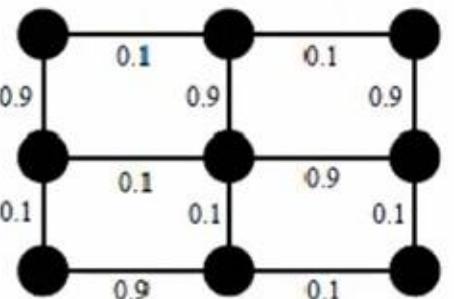
- (I) Minimum Spanning Tree of  $G$  is always unique.
  - (II) Shortest path between any two vertices of  $G$  is always unique.

Which of the above statements is/are necessarily true?



akshintalamahvith@gmail.com

19. Consider the following undirected graph with edge weights as shown



The number of minimum-weight spanning trees of the graph is \_\_\_\_\_

20. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:

*For Micro Notes by the  
Student*



$$\frac{B \rightarrow C : 5}{B - A \rightarrow C : 5}$$



ACE Live Class 1

Select PDF File:

Convert To:  
Microsoft Word (\*.docx)

Convert

- ▶ Create PDF
- ▶ Edit PDF
- ▶ Combine PDF
- ▶ Send Files
- ▶ Store Files



For Micro Notes by the  
Student



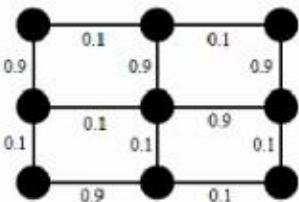
18. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:

- (I) Minimum Spanning Tree of  $G$  is always unique.
- (II) Shortest path between any two vertices of  $G$  is always unique.

Which of the above statements is/are necessarily true?

- (a) (I) only
- (b) (II) only
- (c) both (I) and (II)
- (d) neither (I) nor (II)

19. Consider the following undirected graph with edge weights as shown



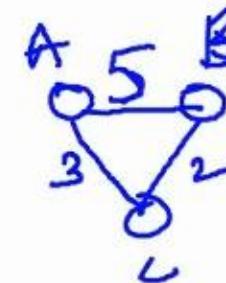
The number of minimum-weight spanning trees of the graph is \_\_\_\_\_

20. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:

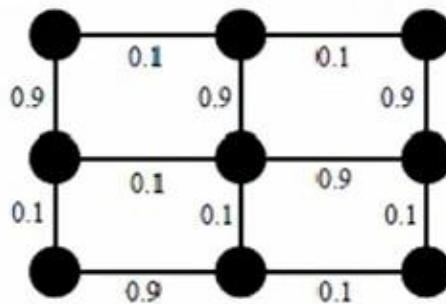
- (I) Minimum Spanning Tree of  $G$  is always unique.
- (II) Shortest path between any two vertices of  $G$  is always unique.

Which of the above statements is/are necessarily true?

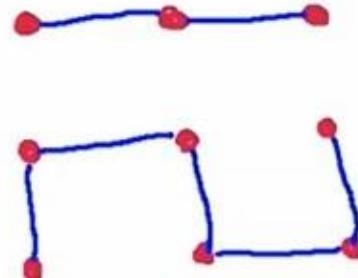
- (a) (I) only
- (b) (II) only
- (c) both (I) and (II)
- (d) neither (I) nor (II)



19. Consider the following undirected graph with edge weights as shown



$$3C_1 = \underline{3}$$



The number of minimum-weight spanning trees of the graph is 3

20. Let  $G = (V, E)$  be any connected undirected edge-weighted graph. The weights of the edges in  $E$  are positive and distinct. Consider the following statements:

- (I) Minimum Spanning Tree of  $G$  is always unique.  
(II) Shortest path between any two vertices of  $G$  is always unique.

Which of the above statements is/are necessarily true?

- (a) (I) only  
(b) (II) only  
(c) both (I) and (II)  
(d) neither (I) nor (II)

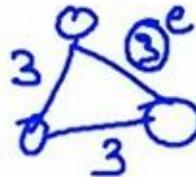


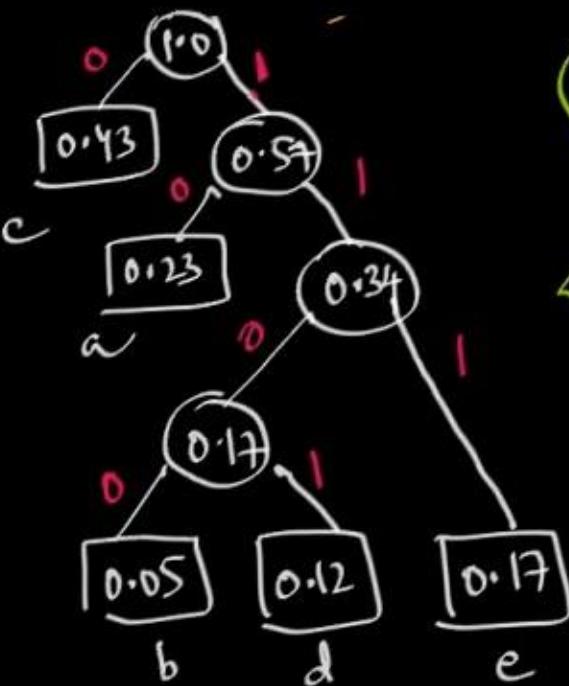
21. Let G be a connected undirected weighted graph. Consider the following two statements.

- S1: There exists a minimum weight edge in G which is present in every minimum spanning tree of G
- S2: If every edge in G has distinct weight, then G has a unique minimum spanning tree.

Which one of the following options is correct?

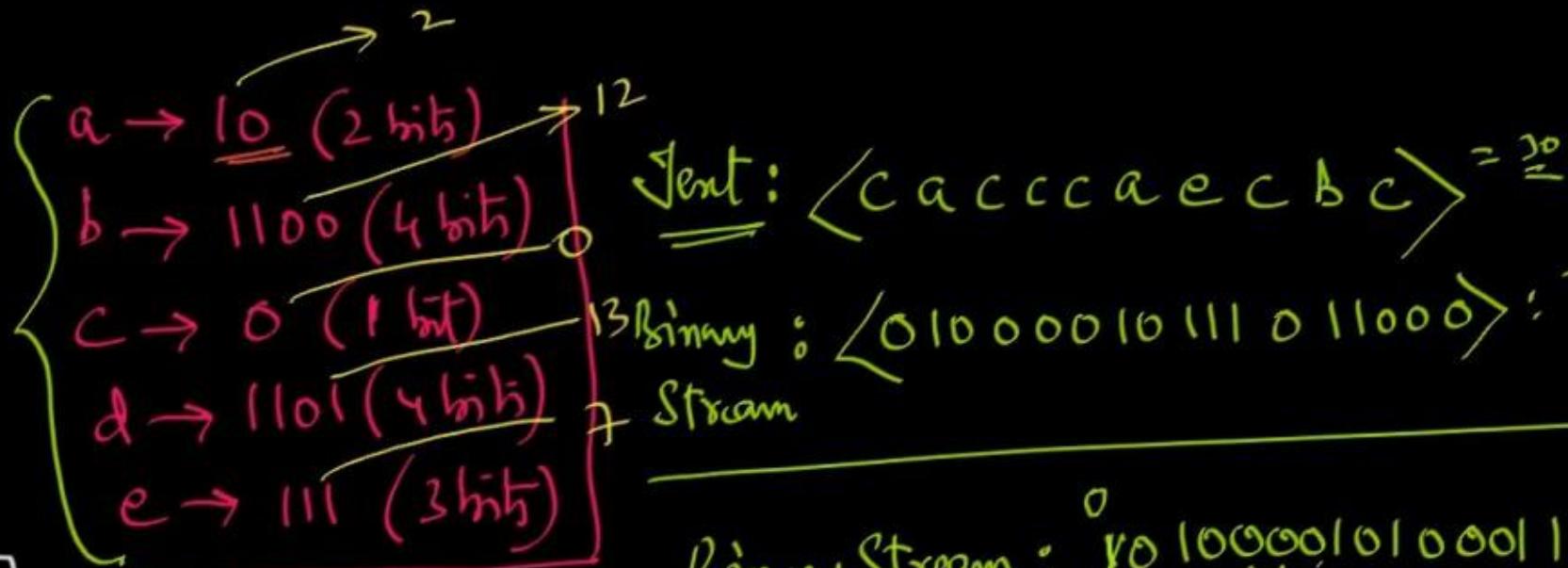
- (a) Both S<sub>1</sub> and S<sub>2</sub> are true
- (b) S<sub>1</sub> is true and S<sub>2</sub> is false
- (c) S<sub>1</sub> is false and S<sub>2</sub> is true
- (d) Both S<sub>1</sub> and S<sub>2</sub> are false.





opt- 2-Way Binary Encode Tree

Note: one bit error is suff to make whole Text corrupted



sent:  $\langle c a c c c a e c b c \rangle = \underline{\underline{30}}$

Binary Stream:  $\langle 01000010111011000 \rangle$ : Total:  $\underline{\underline{17}}$

Binary Stream:  $\langle 0100001010001110 \dots \rangle$

sent Stream:  $\underline{\underline{a a c c c a e c e c}}$

$$\text{Ans. No. of bits} = \sum_{i=1}^n p_i \cdot q_i$$

$$= 2 * 0.23 + 4 * 0.05 + 1 * 0.43 + 4 * 0.12$$

$$= (2.08 \text{ bits/character}) + 3 * 0.17$$

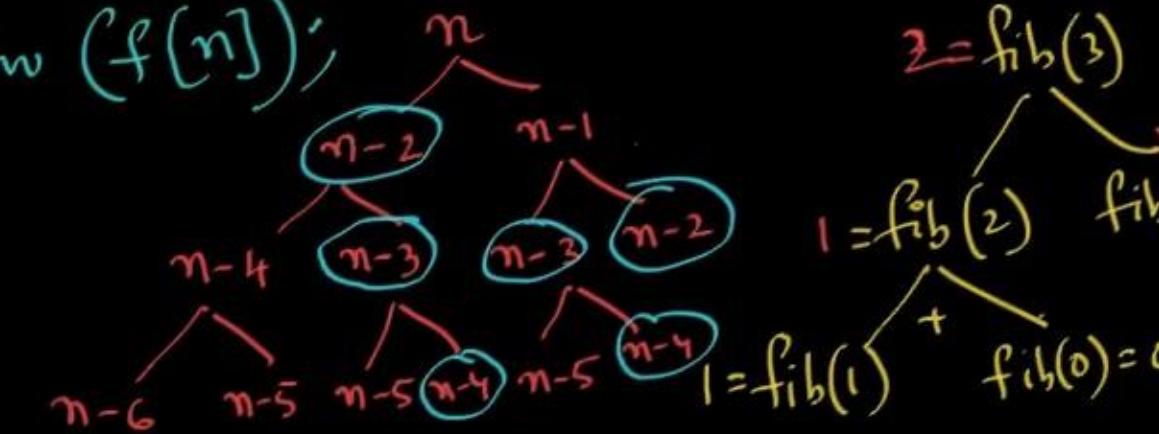
## Memoized Implementation of $\text{fib}(n)$ :

integer  $f[1000]$ ;

$$f[0] = 0; f[1] = 1;$$

$$f[i] = -1, i \geq 2$$

```
int fib(n)
{
    if (f[n] == -1)
        f[n] = fib(n-1) + fib(n-2);
    return (f[n]);
}
```



|   |   |   |   |   |   |    |
|---|---|---|---|---|---|----|
| 0 | 1 | 2 | 3 | 4 | 5 | -  |
| 0 | 1 | 1 | 2 | 3 | 5 | -1 |

$$\begin{aligned} \text{fib}(5) &= \\ 3 &= \text{fib}(4) & \text{fib}(3) &= 2 \\ & & & \end{aligned}$$

$$\begin{aligned} 2 &= \text{fib}(3) & \text{fib}(2) &= 1 \\ & & & \end{aligned}$$

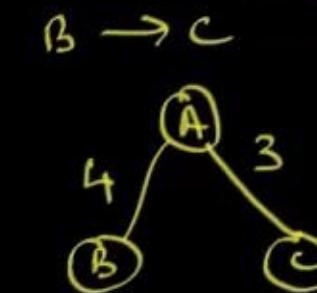
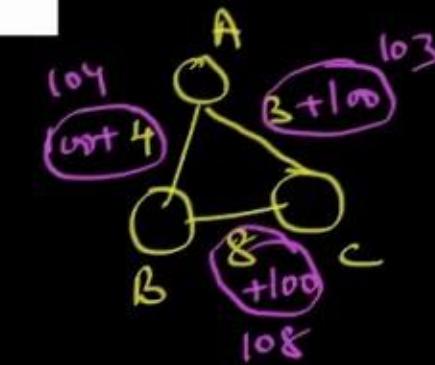
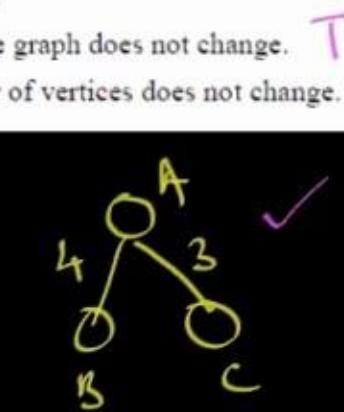
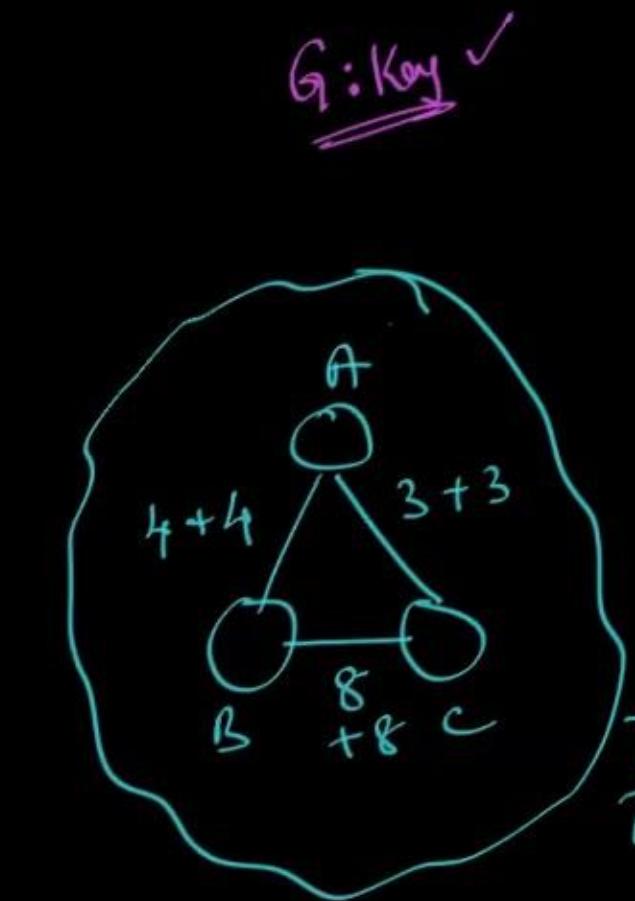
$$\begin{aligned} 1 &= \text{fib}(2) & \text{fib}(1) &= 1 \\ & & & \end{aligned}$$

$$1 = \text{fib}(1) + \text{fib}(0) = 0$$

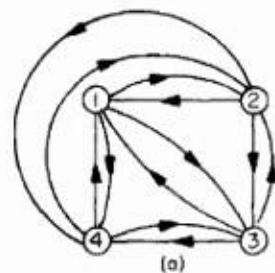
A **prefix code** is a type of **code** system distinguished by its possession of the "prefix property", which requires that there is no whole **code word** in the system that is a **prefix** (initial segment) of any other code word in the system. It is trivially true for fixed-length code, so only a point of consideration in variable-length code.

For example, a code with code words  $\{9, 55\}$  has the prefix property; a code consisting of  $\{9, 5, 59, 55\}$  does not, because "5" is a prefix of "59" and also of "55". A prefix code is a **uniquely decodable code**: given a complete and accurate sequence, a receiver can identify each word without requiring a special marker between words. However,

16. Let  $G$  be a weighted connected undirected graph with distinct positive edge weights. If every edge weight is increased by the same value, then which of the following statements is/are true?
1. Minimum spanning Tree of the graph does not change. **T**
  2. Shortest path between any pair of vertices does not change. **F**



## ② Travelling Salesperson Problem (TSP)



|   |    |    |    |
|---|----|----|----|
| 0 | 10 | 15 | 20 |
| 5 | 0  | 9  | 10 |
| 6 | 13 | 0  | 12 |
| 8 | 8  | 9  | 0  |

(a)

|   |    |    |    |
|---|----|----|----|
| 0 | 10 | 15 | 20 |
| 5 | 0  | 9  | 10 |
| 6 | 13 | 0  | 12 |
| 8 | 8  | 9  | 0  |

(b)

Figure 5.14 Directed graph and edge length matrix  $c$ 

{Tour}: (Hamiltonian cycle)  
n-cities

Home-city:  $V_0$

$(n-1)$ : exactly cities

$P_{vs}(NP)$

Greedy Method:

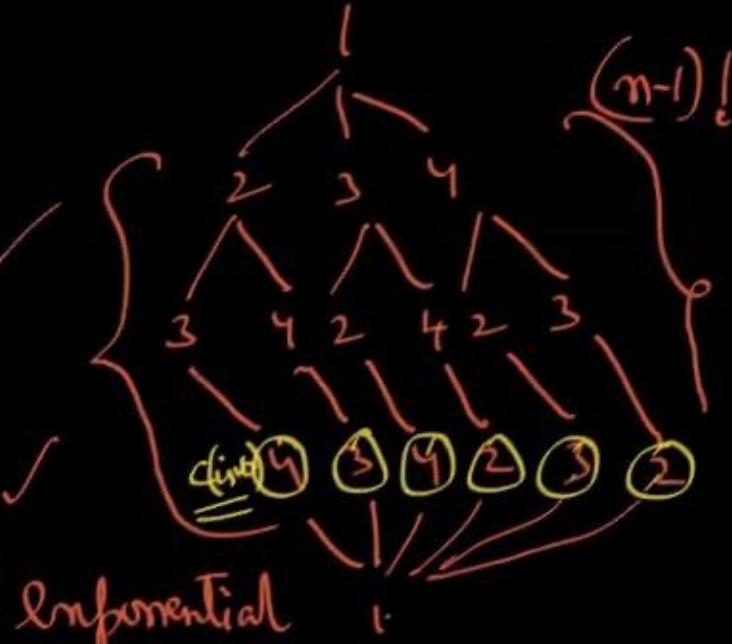
$$V_0 = 1$$

1-2-3-4-1 :  $\{39\}$  : fails

Tour :  $\overline{1-2-4-3-1}$  :  $\{35\}$

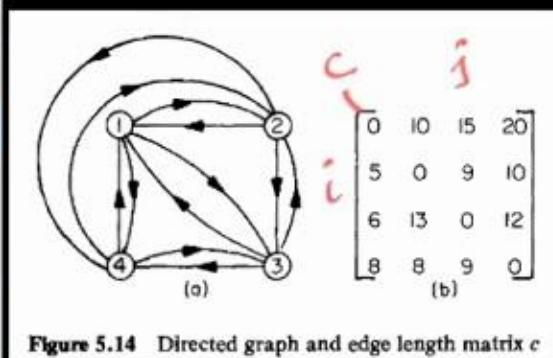
Solv Space :  $(n-1)!$   $\Rightarrow \Theta(n^n)$

Time-complexity :  $\Theta(n^2 n)$



Intractable: (TSP) : (No poly-time  
(NPC)  $\Leftrightarrow$  NP Algo exists)

Exponential



Let  $g(i, s)$  represent cost of the tour from vertex 'i' & visit all the vertices in the set 'S' exactly once and come back (return) to starting vertex (city) ( $v_0$ )  $v_0 = \text{source}$

$$v_0 = \underline{\underline{g}}(1, \{2, 3, 4\}) = \min_{k \in S} \left\{ c(1, k) + g(k, S - \{k\}) \right\} \quad \text{--- ①}$$

edge    Sub tour

o-tour :    
 $\sum_{k \in S} \quad S - \{k\}$

$c(1, k) \in E$

$$\left\{ \begin{array}{l} g(i, s) = \min_{k \in s} \left\{ c(i, k) + g(k, s - \{k\}) \right\} \end{array} \right. - (2)$$

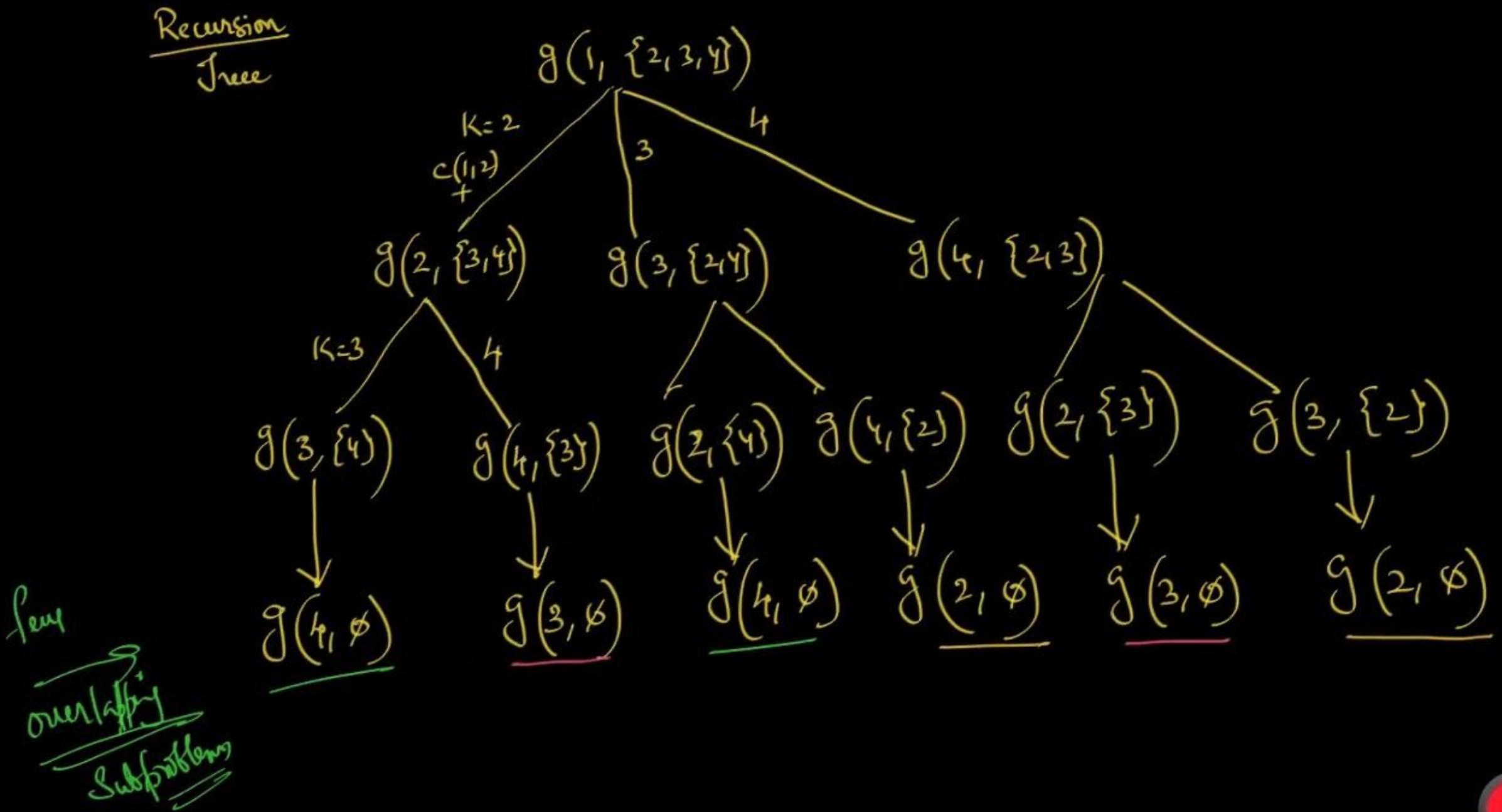
$J(i, s)$  = 'k' that minimizes Eq (2)

$$g(i, \phi) = c(i, v_0) \quad i \neq v_0$$

$$v_0 = 1$$

$g(1, \{2, 3, 4\})$

$\stackrel{\text{edge}}{i \neq v_0} = c(i, v_0)$





$$g(2, \{3, 4\}) = \min \left\{ \underbrace{c(2, 3) + g(3, \{4\})}_{9 + 20}, \underbrace{c(2, 4) + g(4, \{3\})}_{10 + 15} \right\} = 25$$

$$J(2, \{3, 4\}) = 4$$


---

Your construction :

$$\langle 1 - 2 - 4 - 3 - 1 \rangle = \textcircled{35}$$

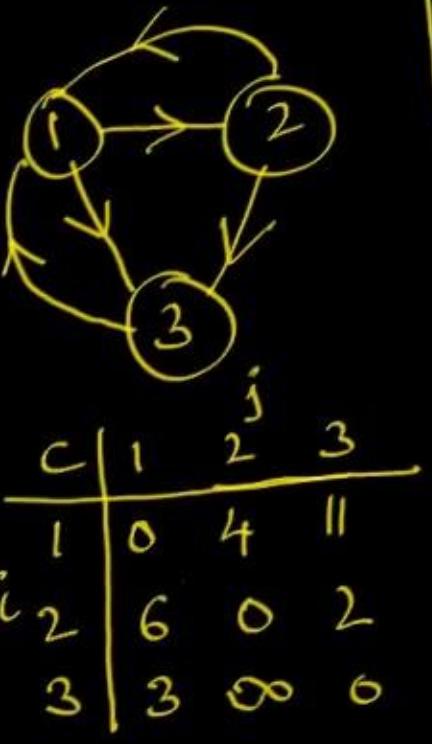
$$\checkmark J(1, \{2, 3, 4\}) = 2$$

$$J(2, \{3, 4\}) = 4$$

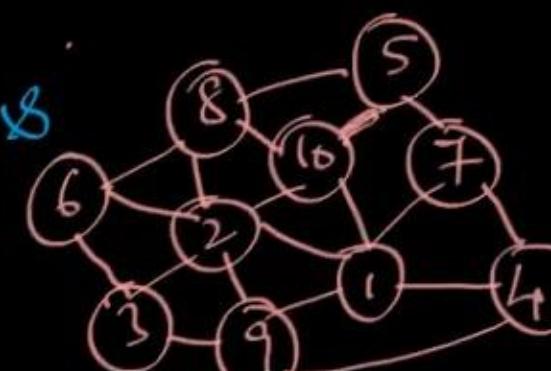
### 3) All-Pairs Shortest Paths < Floyd-Warshall's Algo ->

: finding shortest paths b/w each pair of vertices in the graph

- 1) use Dijkstra's Algo by varying source vertex (apply it  $n$ -times);  $\langle i, j \rangle$   
 $i \leq (i, j) \leq n$
- 2) using D.P (Floyd-Warshall's Algo)

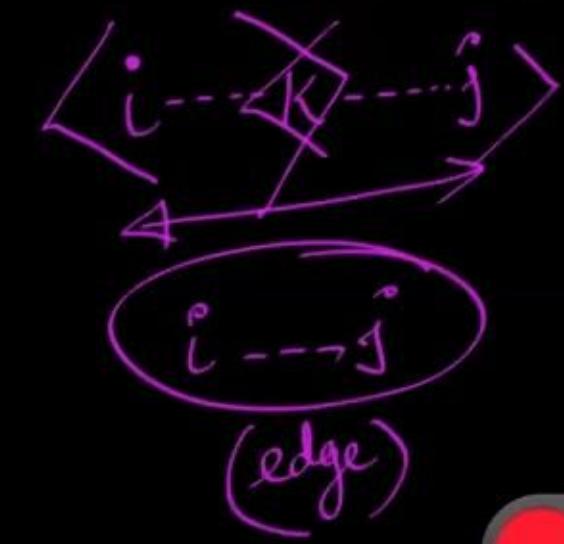


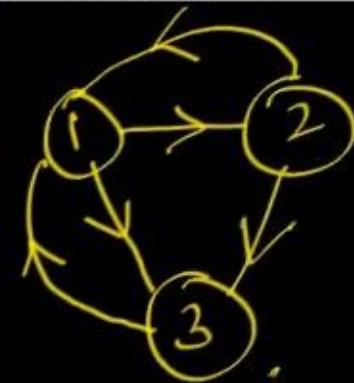
Let  $A^k(i, j)$  repr. cost of the Path from vertex 'i' to vertex 'j'  
 Not going thru intermediate vertex of index greater than 'k'



$$G = (V, E)$$

6-8-5-7-4  
 6-2-1-4  
~~6-3-2-10-5-7-4~~  
 6-3-9-1-4  
 $i \dots \underline{n} \dots j$



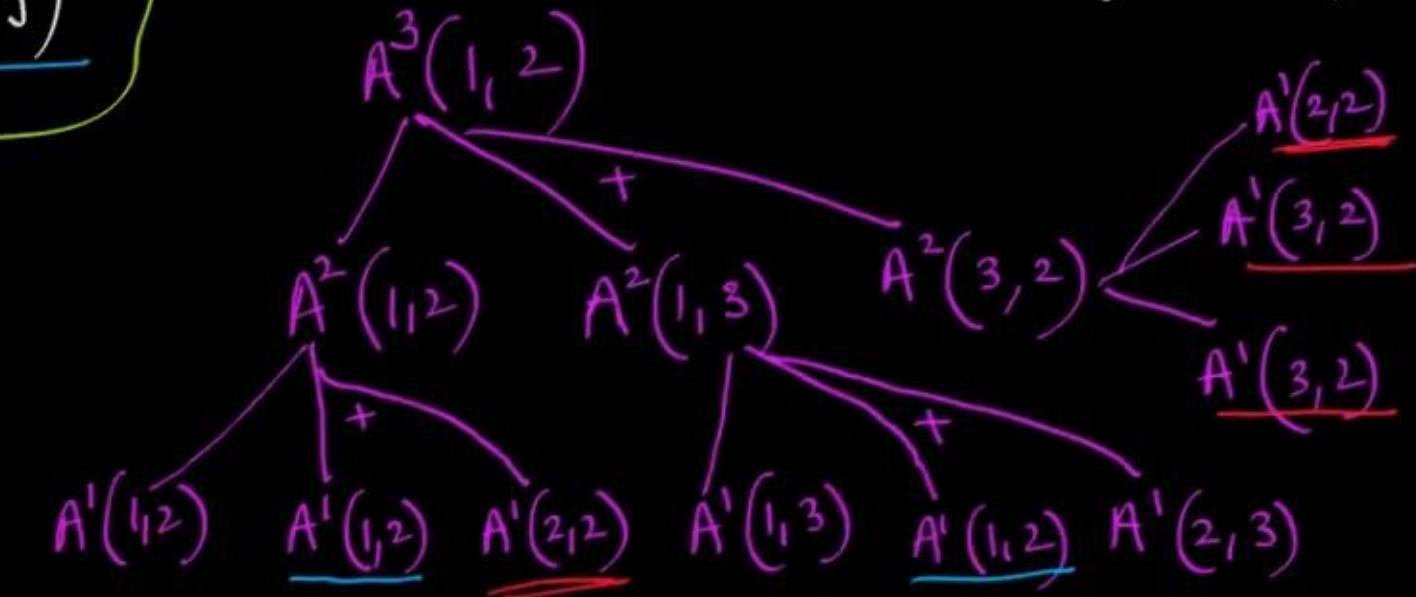
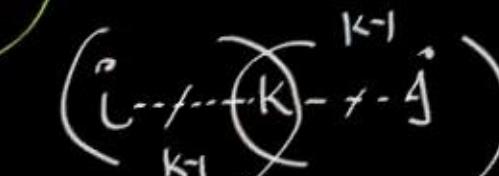


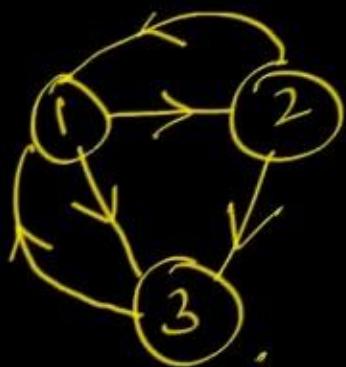
|   | 1 | 2        | 3  |
|---|---|----------|----|
| 1 | 0 | 4        | 11 |
| 2 | 6 | 0        | 2  |
| 3 | 3 | $\infty$ | 6  |

Let  $A^k(i, j)$  repr. cost of the Path from vertex 'i' to vertex 'j'  
Not going thru intermediate vertex of index greater than ' $k$ '.

$$A^k(i, j) = \min_{1 \leq k \leq n} \{ A^{k-1}(i, j), A(i, k) + A(k, j) \}$$

$$A^0(i, j) = c(i, j)$$





|   | 1 | 2 | 3  |
|---|---|---|----|
| 1 | 0 | 4 | 11 |
| 2 | 6 | 0 | 2  |
| 3 | 3 | ∞ | 6  |

$$A^0(1,2) = \min \left\{ A^0(1,1), A^0(1,1) + A^0(1,2) \right\}$$

$$A^3 \Rightarrow A^2 \Rightarrow A^1 \Rightarrow \underline{A^0}$$

| $A^0$ | 1 | 2 | 3  |
|-------|---|---|----|
| 1     | 0 | 4 | 11 |
| 2     | 6 | 0 | 2  |
| 3     | 3 | ∞ | 6  |

| $A^1$ | 1 | 2 | 3  |
|-------|---|---|----|
| 1     | 0 | 4 | 11 |
| 2     | 6 | 0 | 2  |
| 3     | 3 | 7 | 0  |

$$2 - 1 - 3$$

$$\overbrace{A^0(2,1) + A^0(1,3)}^{6+11} = 17$$

$$3 - \underline{1} - 2$$

$$\overbrace{A^0(3,1) + A^0(1,2)}^{3+4} = 7$$

| $A^3$ | 1 | 2 | 3 |
|-------|---|---|---|
| 1     | 0 | 4 | 6 |
| 2     | 5 | 0 | 2 |
| 3     | 3 | 7 | 0 |

| $A^2$ | 1 | 2 | 3 |
|-------|---|---|---|
| 1     | 0 | 4 | 6 |
| 2     | 6 | 0 | 2 |
| 3     | 3 | 7 | 0 |

i ≠ ji = ----- j :Algo FW ( $G, n, c, A$ )  
 $\{ \quad \underline{\underline{c[1..n, 1..n]}}, \quad \underline{\underline{A[1..n, 1..n]}}$ 1. for  $i \leftarrow 1$  to  $n$ for  $j \leftarrow 1$  to  $n$   
 $A^0[i, j] = \underline{\underline{c[i, j]}}$ :  $n^2$ Time:  $O(n^3)$ Space:  $O(\underline{\underline{n^2}})$ 2. for  $k \leftarrow 1$  to  $n$  : Intermediate vertex :  $n^3$ for  $i \leftarrow 1$  to  $n$  : Sourcefor  $j \leftarrow 1$  to  $n$  : Destination $A[i, j] = \min \{ A[i, j], A[i, k] + A[k, j] \}$ 

}

The Transitive closure of a Matrix, representing a graph with  $n$ -vertices can be found using Floyd-Warshall's Algorithm with a time complexity of  $O(n^3)$ ;

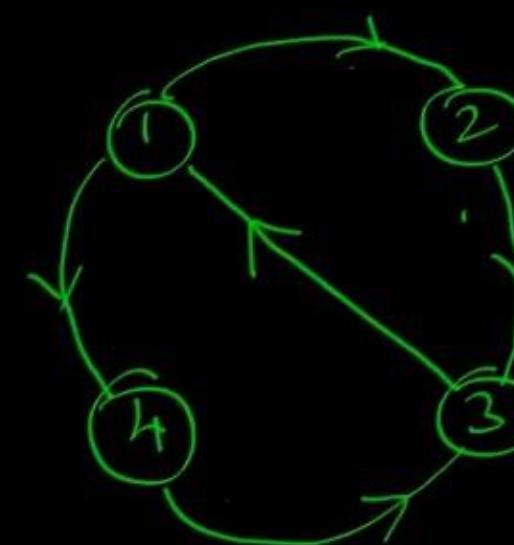
### Transitive closure:

Let 'A' be the adjacency Matrix of a directed graph  $G = (V, E)$ .  
The transitive closure of Matrix A be represented as  $A^+$  is defined as:

$A^+(i, j) = 1$ , iff ' $G$ ' has a directed path containing atleast one edge from vertex ' $i$ ' to vertex ' $j$ ';

$= 0$ , otherwise

{Reflexive Transitive closure}



$$G = (V, E)$$

| A | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 |

| A <sup>T</sup> | 1 | 2 | 3 | 4 |
|----------------|---|---|---|---|
| 1              | 0 | 1 | 1 | 1 |
| 2              | 1 | 0 | 1 | 1 |
| 3              | 1 | 1 | 0 | 1 |
| 4              | 1 | 1 | 1 | 0 |

(Transitive closure)

#### 4) 0/1 Knapsack:

$$\rightarrow n\text{-objects } (o_i) \leftarrow \begin{matrix} w_i \\ p_i \end{matrix}$$

$\rightarrow$  knapsack of capacity  $\leq M$

$$\rightarrow \langle x_1, x_2, \dots, x_n \rangle$$

Max.  $\sum_{i=1}^n p_i x_i$

S.T.C  $\sum_{i=1}^n w_i x_i \leq M$

where  $x_i = 0/1$

$$x_i \rightarrow w_i x_i \quad \sum_{i=1}^n w_i > M$$

$$\langle n \rangle ; M; \langle p_1, \dots, p_n \rangle; \langle w_1, \dots, w_n \rangle$$

$$\langle x_1, \dots, x_n \rangle$$

Let  $OIKNAP(n, M)$  repr. the profit obtainable with  $n$ -objects and a knapsack of capacity ' $M$ ';

$$OIKNAP(n, M) = OIKNAP(n-1, M), (w_n > M)$$

$$= \max \left\{ \begin{array}{l} OIKNAP(n-1, M), \\ x_n = 0 \\ OIKNAP(n-1, M-w_n) + p_n \end{array} \right\}$$

$$x_n = 1 \quad OIKNAP(n-1, M-w_n) + p_n, \quad w_n \leq M$$

$$= 0, \quad n=0, M \geq 0$$

$$= 0, \quad n>0, M=0$$

$$n = 4; M = 3$$

$OIKNAP(n, M) = OIKNAP(n-1, M)$ ,  $(w_n > M)$   
 $= \max \left\{ \begin{array}{l} OIKNAP(n-1, M), \\ w_n = 0 \\ OIKNAP(n-1, M-w_n) + p_n, w_n \leq M \end{array} \right\}$   
 $= 0, n=0, M \geq 0$   
 $= 0, n>0, M=0$

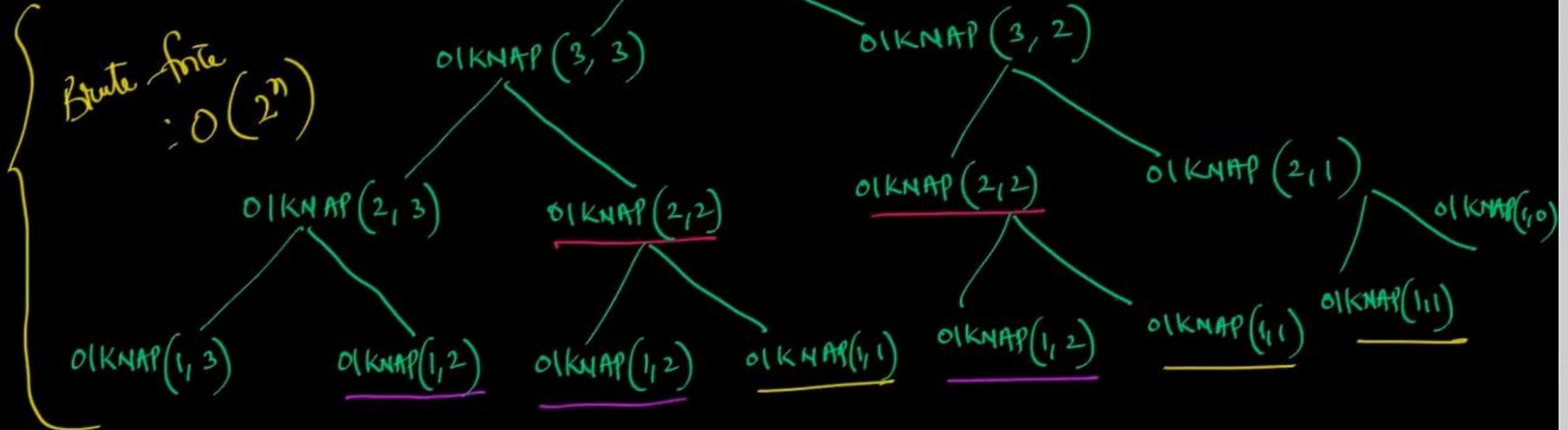
$$\langle p_1 \dots p_4 \rangle = \langle 10, 20, 30, 40 \rangle$$

$$\langle w_1 \dots w_4 \rangle = \langle 1, 1, 1, 1 \rangle$$

$$OIKNAP(4, 3)$$

$$\langle x_1 \dots x_n \rangle$$

$$\left\{ \begin{array}{l} x_i = 0 \text{ or } 1 \\ \sum x_i = 2 \end{array} \right\}$$



$$n=4; M=8; \langle w_1 \dots w_4 \rangle = \langle 2, 3, 4, 5 \rangle; \langle p_1 \dots p_4 \rangle = \langle 1, 2, 5, 6 \rangle$$

$T[0..n, 0..M]$

$\rightarrow M$

$\langle x_1 x_2 x_3 x_4 \rangle = \langle$

$T[n, M] =$

(final profit)

|   |       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8   |
|---|-------|---|---|---|---|---|---|---|---|-----|
|   |       | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0   |
|   |       | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1   |
|   | $w_i$ | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3   |
| 1 | 2     | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1   |
| 2 | 3     | 2 | 0 | 0 | 1 | 2 | 2 | 3 | 3 | 3   |
| 3 | 4     | 3 | 0 | 0 | 1 | 2 | 5 | 5 | 6 | 7   |
| 4 | 5     | 4 | 0 | 0 | 1 | 2 | 5 | 6 | 6 | 7   |
|   |       |   |   |   |   |   |   |   |   | 8 ✓ |

$$T[1, 2] = \max\{T[0, 2], T[0, 0] + 1\}$$

$w_n \leq M$

$T[2, 3] =$

$T[1, 1] =$

$T[0, 1] =$

$T[1, 2] =$

$x_1 = 0$

$T[2, 1] =$

$x_2 = 1$

$x_4 = 1; x_3 = 0$

Algo 01(KNAP)  $(n, M, w[1..n], p[1..n], x[1..n])$

$$\{ \quad T[0..n, 0..M]$$

for  $i \leftarrow 0$  to  $n$   
 { for  $j \leftarrow 0$  to  $M$

{ if ( $i == 0$  or  $j == 0$ )  
 $T[i, j] = 0$ ;

else if ( $w[i] \leq j$ )  
 $T[i, j] = \max\{T[i-1, j], T[i-1, j-w[i]] + p_i\}$

else  $T[i, j] = T[i-1, j]$ ;

} } }

Time:  $O(n \cdot M)$  ✓  
 :  $O(n \cdot 2^n)$  if  $M = 2^n$

Space:  $O(n \cdot m)$  ✓

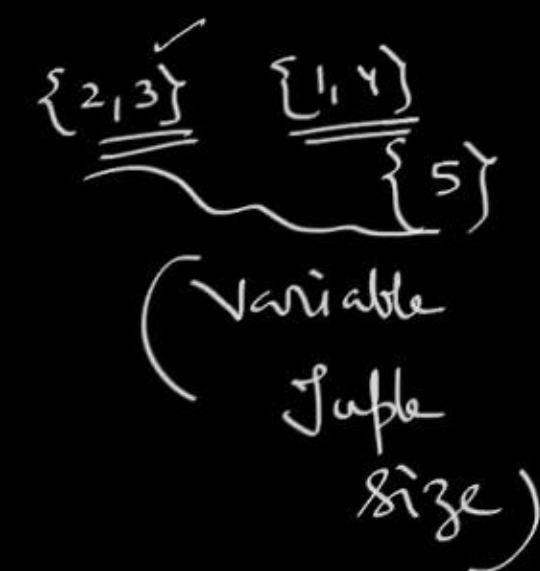
## 5) Sum of Subsets (SOS):

$$n=5; M=50; A: \langle \frac{1}{10}, \frac{2}{20}, \frac{3}{30}, \frac{4}{40}, \frac{5}{50} \rangle$$

Subset-Sum : Given a set of  $n$ -elements and another element ' $M$ ', does there exists a subset of given elements whose sum is equal to  $M$ ;

$$\left\{ \begin{array}{l} x: \langle 0, 1, 1, 0, 0 \rangle \\ x: \langle 1, 0, 0, 1, 0 \rangle \\ x: \langle 0, 0, 0, 0, 1 \rangle \end{array} \right\}$$

fixed  
tuple  
size



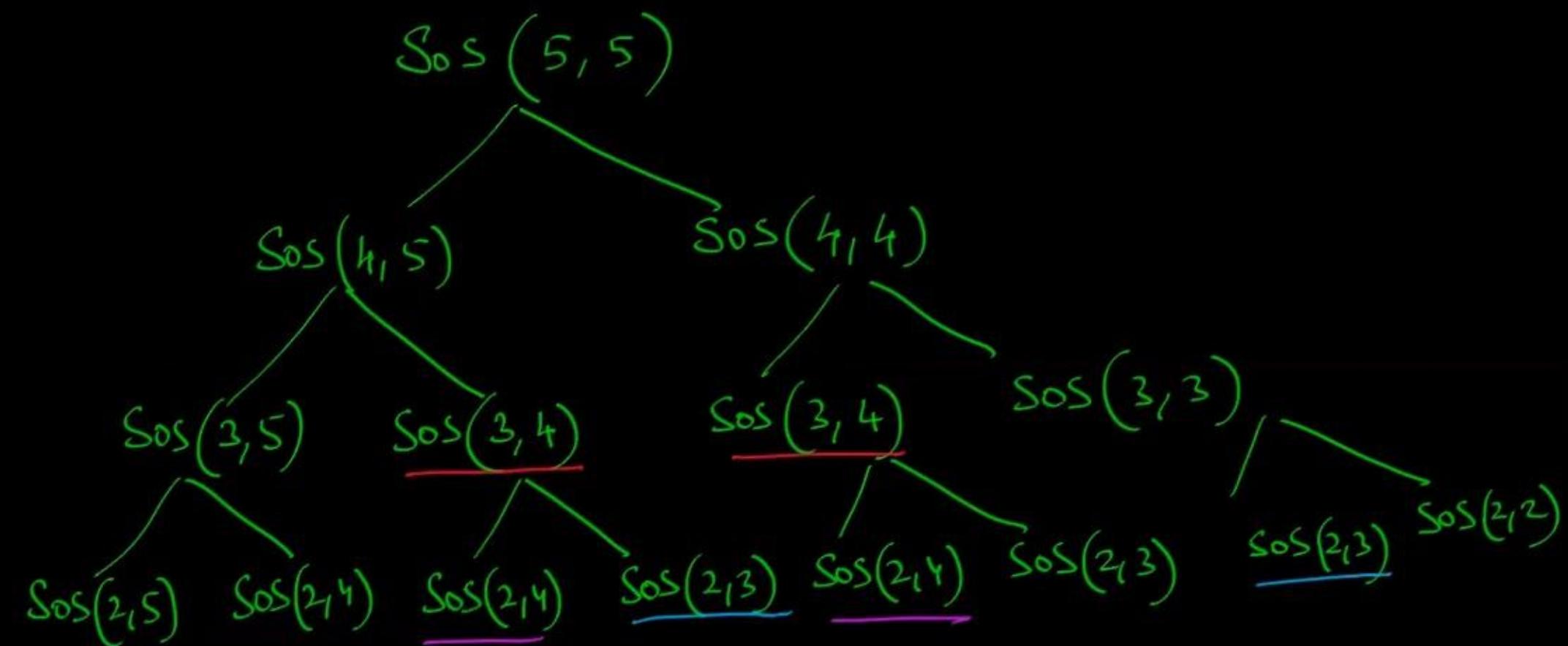
## Derivation of Recurrence:

'n' elements A:  $\langle a_1, a_2, \dots, a_n \rangle$ ; 'M'

Let  $SoS(n, M)$  repr. the subset of 'n' elements that may sum to 'M':  
 $SoS(n, M) = T(F$

$$\left\{ \begin{array}{l} SoS(n, M) = SoS(n-1, M), \quad A[n] > M \\ \quad = SoS(n-1, M) \text{ or} \\ \quad \quad \quad SoS(n-1, M - A[n]) \}, \quad A[n] \leq M \\ \quad = T, \quad n \geq 0, M = 0 \\ \quad = F, \quad n = 0, M > 0 \end{array} \right.$$

$n=5; M=5; A: \langle 1, 1, 1, 1, 1 \rangle$



$$n=5; \quad A = \langle 2, 8, 4, 11, 9 \rangle \quad ; \quad M=6$$

$$T[n+1, M+1] \Rightarrow T[0 \dots n, 0 \dots M]$$

$T$

| $A_i \downarrow$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|------------------|---|---|---|---|---|---|---|
| 2                | T | F | F | F | F | F | F |
| 8                | T | F | T | F | F | F | F |
| 4                | T | F | T | F | T | F | T |
| 11               | T | F | T | F | T | F | T |
| 9                | T | F | T | F | T | F | T |

$$T[1,1] = T[0,1]$$

$$T[1,2] = T[0,2] \text{ or } T[0,0]$$

$$= F \text{ or } T$$

"How to find what elements are included in the subset"

$$T[5,6]$$

Time:  $O(n \cdot M)$

Space:  $O(n \cdot M)$

## 6) Longest Common Subsequence (LCS) "String Matching"

Substrings

$\rightarrow \langle A \rangle \langle AB \rangle \langle ABC \rangle$   
 $\langle A \rangle \langle B \rangle \langle C \rangle \langle BC \rangle \langle BCA \rangle$   
 $\langle A \rangle \langle AB \rangle \langle ABC \rangle \langle ACAB \rangle$   
 $\langle A \rangle \langle B \rangle \langle C \rangle \langle D \rangle \langle AB \rangle \langle BC \rangle \langle CD \rangle \langle ABC \rangle \langle BCA \rangle \langle CAB \rangle$

I.  $X = \langle A \ B \ C \ D \ A \ B \rangle$   $A \ B \ B \ C \ X$   
 $'n'$

II.  $X = \langle A \ B \ C \ D \ A \ B \rangle$   
 $Y = \langle B \ D \ C \ A \ B \ A \rangle_m$

A Subsequence that is common to both  
 $'X'$  &  $'Y'$  is known as Common Subsequence

$$\langle BC \rangle = 2 ; \quad \langle BBA \rangle = 3 \\ \langle BCA \rangle = 4$$

"String"  
Substring  
" a group of  
"more  
characters  
that are taken  
in contiguous  
from the  
"String"

"A group of  
"more  
characters  
taken  
from a string  
that may not  
be contiguous  
nevertheless they  
must be in  
order"

→ Every Substring is a Subsequence.  
 → for a given string of  $n$ -characters.

No. of Substrings

$x = "ABCD"$

$$\begin{array}{lcl} 1 & : & \boxed{4} \\ 2 & : & \boxed{3} \\ 3 & : & \boxed{2} \\ 4 & : & \boxed{1} \end{array}$$

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

$$= O(n^2)$$

No. of Subsequences ( $2^n$ )

- 1:  $\langle \rangle$
- 2:  $\langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle$   
 $\quad\quad\quad \langle AB \rangle, \langle AC \rangle, \langle AD \rangle$
- 3:  $\langle ABC \rangle, \langle BCD \rangle$   
 $\quad\quad\quad \langle ABD \rangle, \langle ACD \rangle$
- 4:  $\langle ABCD \rangle$

$$O(2^n)$$

Defn: Given Two Strings of length ' $n$ ' & ' $m$ ' characters respectively,  
It is required to find a Common Subsequence of both ' $X$ ' & ' $Y$ '  
of longest length;

### Applications:

- (i) Searching in Google ✓
- (ii) Plagiarism (Copyright)
- (iii) Software (open Source)
- (iv) DNA Sampling

↳ Strand : <

LCS : <



>✓

>✓

"Ravi ate apple"  
"Apple was eaten  
by Ravi"

Derivation of Recurrence: Let  $x$  &  $y$  be strings of length ' $n$ ' & ' $m$ '  
 Let ' $i$ ' & ' $j$ ' be indices into the strings  $x$  &  $y$  as shown below;

$$x: \langle x_0, x_1, x_2, \dots, \underline{x_i} \rangle$$

$$y: \langle y_0, y_1, y_2, \dots, \underline{y_j} \rangle$$

→ Let  $L[i, j]$  denote the length of common subsequence of the strings  $x$  &  $y$  as defined above.

Case I:  $L[i, j] = 1 + L[i-1, j-1], \quad x[i] = y[j]$

Case II:  $L[i, j] = \max \{L[i, j-1], L[i-1, j]\}, \quad x[i] \neq y[j]$

Boundary Conditions:  $L[-1, j] = 0, \quad j = -1, 0, 1, 2, \dots$

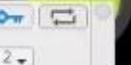
$$L[i, -1] = 0, \quad i = -1, 0, 1, 2, \dots$$

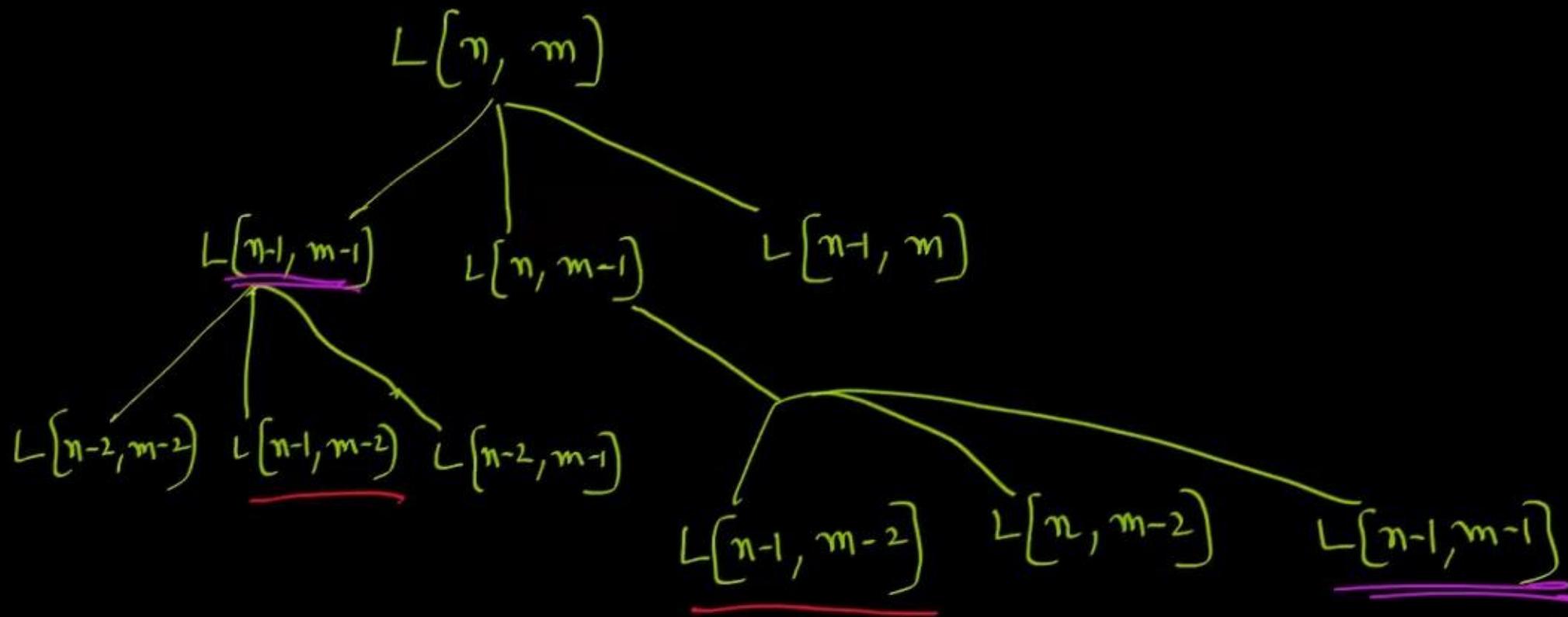
$$x = \langle A \underline{B} C B D A \rangle$$

$$y = \langle S \underline{C} B D B A \rangle$$

$$x = \langle A \underline{B} C B D A \rangle$$

$$y = \langle B \underline{C} C B A \rangle$$





14

ABAB

$$\text{LCS}("ABABA\overset{?}{B}", "ACABA\overset{?}{B}") = 4$$

$$\downarrow \begin{matrix} + \\ \text{B} \end{matrix}$$
$$\text{LCS}("ABABA\overset{?}{A}", "ACABA\overset{?}{A}") = 3$$

$$\downarrow \begin{matrix} + \\ \text{A} \end{matrix}$$
$$\text{LCS}("ABAB\overset{?}{B}", "ACAB") = 2$$

$$\downarrow \begin{matrix} + \\ \text{B} \end{matrix}$$
$$\text{LCS}("AB\overset{?}{B}", "AC") = 1$$

A  
man

$$1 = \text{LCS}("AB", "A")$$

$$A$$
$$\text{LCS}("A", "AC") = 1$$

$$0 = \text{LCS}("AB", "")$$

$$1 = \text{LCS}("A", "A")$$

$$1 = \text{LCS}("A", "A")$$
$$\downarrow \begin{matrix} + \\ \text{A} \end{matrix}$$
$$\text{LCS}("", "AC") = 0$$

$$\downarrow \begin{matrix} + \\ \text{A} \end{matrix}$$
$$\text{LCS}("", "") = 0$$

$$0 = \text{LCS}("", "")$$

Case I:  $L[i, j] = l + L[i-1, j-1]$ ,  $x[i] = y[j]$

Case II:  $L[i, j] = \max\{L[i, j-1], L[i-1, j]\}$ ,  $x[i] \neq y[j]$

Boundary Conditions:  $L[-1, j] = 0$ ,  $j = -1, 0, 1, 2, \dots$   
 $L[i, -1] = 0$ ,  $i = -1, 0, 1, 2, \dots$

$x = \langle \dots \dots i \dots \dots \rangle$        $y = \langle \dots \dots j \dots \dots \rangle$

$x = \langle \dots \dots i \dots \dots \rangle$        $y = \langle \dots \dots j \dots \dots \rangle$

$\leftarrow$

$L[0, j] = 0$ ,  $j = 0, 1, 2, \dots$   
 $L[i, 0] = 0$ ,  $i = 0, 1, 2, \dots$

$$X = \langle A \ B \ C \ B \ D \ A \ B \rangle$$

1 2 3 4 5 6 7

$$Y = \langle B \ D \ C \ A \ B \ A \rangle$$

1 2 3 4 5 6

$$L[0 \dots 7, 0 \dots 6]$$

$$L[7, 6]$$

| $i$ | $j$ | 0     | 1               | 2  | 3              | 4               | 5               | 6               |
|-----|-----|-------|-----------------|----|----------------|-----------------|-----------------|-----------------|
| $i$ | $j$ | $y_i$ | B               | D  | C              | A               | B               | A               |
| 0   | 0   | 0     | 0               | 0  | 0              | 0               | 0               | 0               |
| 1   | A   | 0     | 0↑              | 0↑ | 0↑             | ↖1 <sub>A</sub> | ↖1 <sub>A</sub> | ↖1 <sub>A</sub> |
| 2   | B   | 0     | ↖1 <sub>B</sub> | 1  | ↖1             | ↖1 <sub>B</sub> | ↖2 <sub>B</sub> | ↖2              |
| 3   | C   | 0     | 1               | 1  | 2 <sub>C</sub> | 2               | 2               | 2               |
| 4   | B   | 0     | 1               | 1  | 2              | 2               | 3 <sub>B</sub>  | 3               |
| 5   | D   | 0     | 1               | 2  | 2              | 2               | 3 <sub>D</sub>  | 3               |
| 6   | A   | 0     | 1               | 2  | 2              | 3               | 3               | ↖4 <sub>A</sub> |
| 7   | B   | 0     | 1               | 2  | 2              | 3               | 4               | ↖4 <sub>B</sub> |

AB

$$L[0, j] = 0$$

$$L[i, 0] = 0$$

$$L[i, j] = \min \{ L[i-1, j], L[i, j-1]$$

$$L[0, 0] = \min \{ L[0, 0], L[0, 0] \}$$

$$L[1, 4] = 1 + L[0, 3]$$

BCBA ✓



Algorithm  $LCS(x, y)$   
 $\{ x[1 \dots m], y[1 \dots n] \}$

for  $i \leftarrow 1$  to  $m$   
 $L[i, 0] = 0;$

for  $j \leftarrow 0$  to  $n$   
 $L[0, j] = 0;$

for  $i \leftarrow 1$  to  $m$

for  $j \leftarrow 1$  to  $n$

if ( $x[i] = y[j]$ )  
 $L[i, j] = 1 + L[i-1, j-1]$

else  
 $L[i, j] = \max \{ L[i-1, j], L[i, j-1] \}$

}

Time:  $O(m \cdot n)$   
Space:  $O(m \cdot n)$

Time (Brute-force)

:  $O(2^m \cdot n)$

:  $O(2^n \cdot m)$

exponential

(MCP)

6) Matrix chain Product (MCP): "Non-Square Matrices can be multiplied provided they are compatible"

$A_{m \times n}; B_{n \times n}; C_{n \times n}$

$$\left\{ \begin{array}{l} 1) A + B = C; O(n^2) \\ 2) \underline{A * B = C}; O(n^3) \end{array} \right\}$$


---

"No. of columns of A = No. of rows of B"

$$A \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}_{2 \times 3} B \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{3 \times 4} = C \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{2 \times 4}$$

(3)  $\cdot$  2  $\cdot$  4 = No. of scalar multiplications

$$A_{m \times n} * B_{n \times r} = C_{m \times r}$$

for i  $\leftarrow$  1 to m

for j  $\leftarrow$  1 to r

$$c[i,j] = 0$$

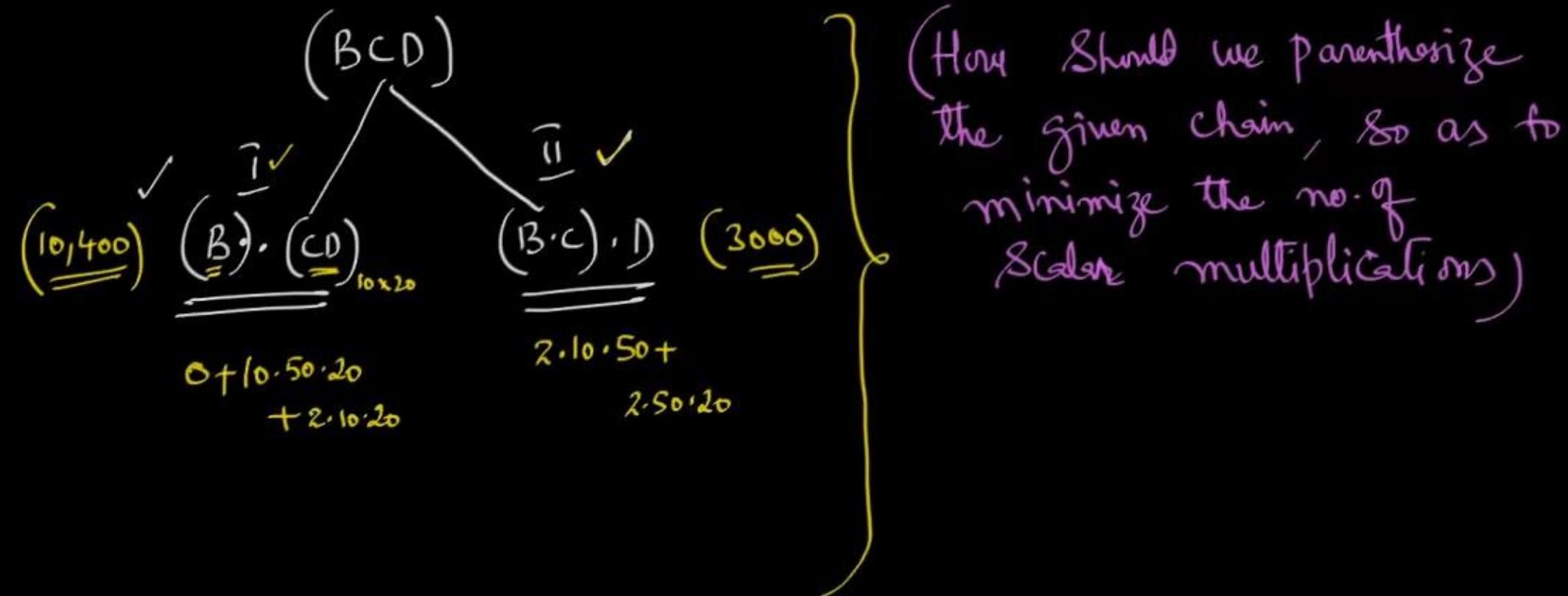
for k  $\leftarrow$  1 to n

$$c[i,j] = c[i,j] + A[i,k] * B[k,j]$$

No. of scalar multiplications =  $(m \cdot n \cdot r)$

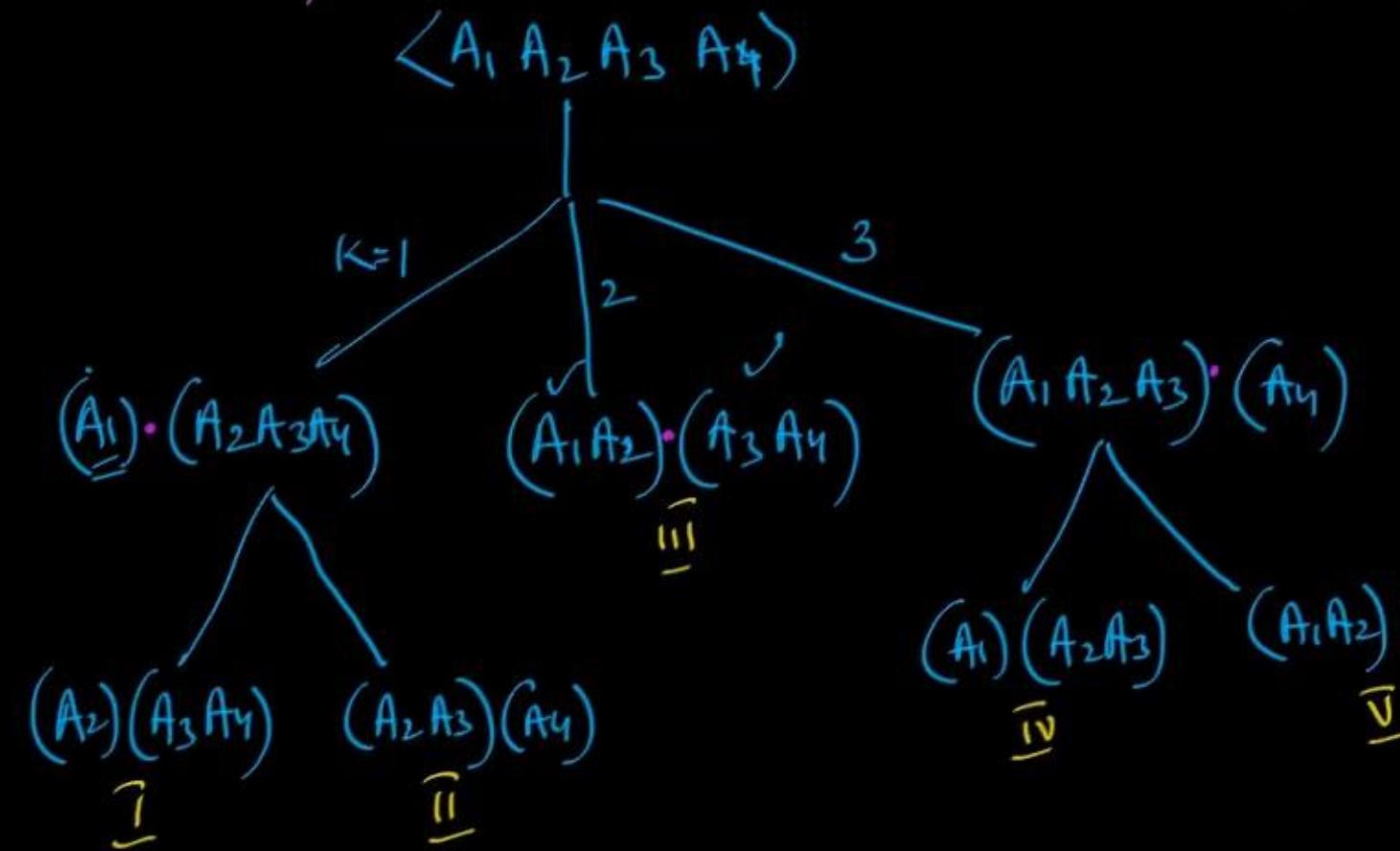
Given a chain of Matrices: It is required to multiply them, so as to minimize the overall No. of Scalar multiplications

i)  $\langle B \cdot C \cdot D \rangle$      $B_{2 \times 10}$ ;  $C_{10 \times 50}$ ;  $D_{50 \times 20} = A_{2 \times 20}$



$$2) \quad \langle A_1 \ A_2 \ A_3 \ A_4 \rangle$$

$k = \text{split point}$



- 1)  $(A_1)(A_2)(A_3 A_4)$
  - 2)  $(A_1)(A_2 A_3)(A_4)$
  - 3)  $(A_1 A_2)(A_3 A_4)$
  - 4)  $((A_1)(A_2 A_3))(A_4)$
  - 5)  $((A_1 A_2)(A_3))(A_4)$

Derivation of Recurrence:

→ Let  $A_{i..j}$  be the Matrix that results from multiplying  $(A_i \cdot A_{i+1} \cdot A_{i+2} \cdots A_j)$  where  $A_i \rightarrow \underline{P_{i-1}} \times \underline{P_i}$

$$\langle A_1 \dots A_7 \rangle = \underline{\underline{A_{1..7}}}$$

→ Any optimal Parenthesization, must split the chain  $\langle A_i \dots A_j \rangle$  between matrices  $A_k$  &  $A_{k+1}$  such that the no. of scalar multiplication is minimal

$$A_{i..j} = \langle A_i \cdot A_{i+1} \cdot A_{i+2} \downarrow \overset{A_k}{\text{---}} \cdots A_j \rangle \quad (i \leq k < j)$$

$\swarrow \quad \searrow$

$$\langle A_i \dots A_k \rangle \quad \langle A_{k+1} \dots A_j \rangle$$

Let  $m[i, j]$  repr. the no. of scalar multiplications to get the matrix  $A_{i..j}$

$$\underline{A_{i..j}} = \langle A_i \cdot A_{i+1} \cdot A_{i+2} \cdots A_j \rangle \quad (i \leq k < j)$$

$\swarrow \quad \searrow$

$$\langle A_i \cdots A_k \rangle \quad \langle A_{k+1} \cdots A_j \rangle$$

Let  $m[i,j]$  keep the no. of scalar multiplications to get the matrix  $A_{i..j}$

$$m[i,j] = \min_{i \leq k < j} \left\{ m[i,k] + m[k+1,j] + p_{i-1} * p_k * p_j \right\} - ①$$

$$m[i,i] = m[j,j] = 0 \quad D(i,j) = 'k'$$

=====

$$A_i \rightarrow P_{i-1} \times P_i$$

$$A_1 \rightarrow 2 \times 5 \rightarrow P_0 \times P_1$$

$$A_2 \rightarrow 5 \times 8 \rightarrow P_1 \times P_2$$

$$A_3 \rightarrow 8 \times 3 \rightarrow P_2 \times P_3$$

$$A_4 \rightarrow 3 \times 4 \rightarrow P_3 \times P_4$$

$$A_5 \rightarrow 4 \times 6 \rightarrow P_4 \times P_5$$

$$A_6 \rightarrow 6 \times 7 \rightarrow P_5 \times P_6$$

$$\langle A_1 \cdot A_2 \cdot A_3 \cdots A_6 \rangle$$

$k=3$

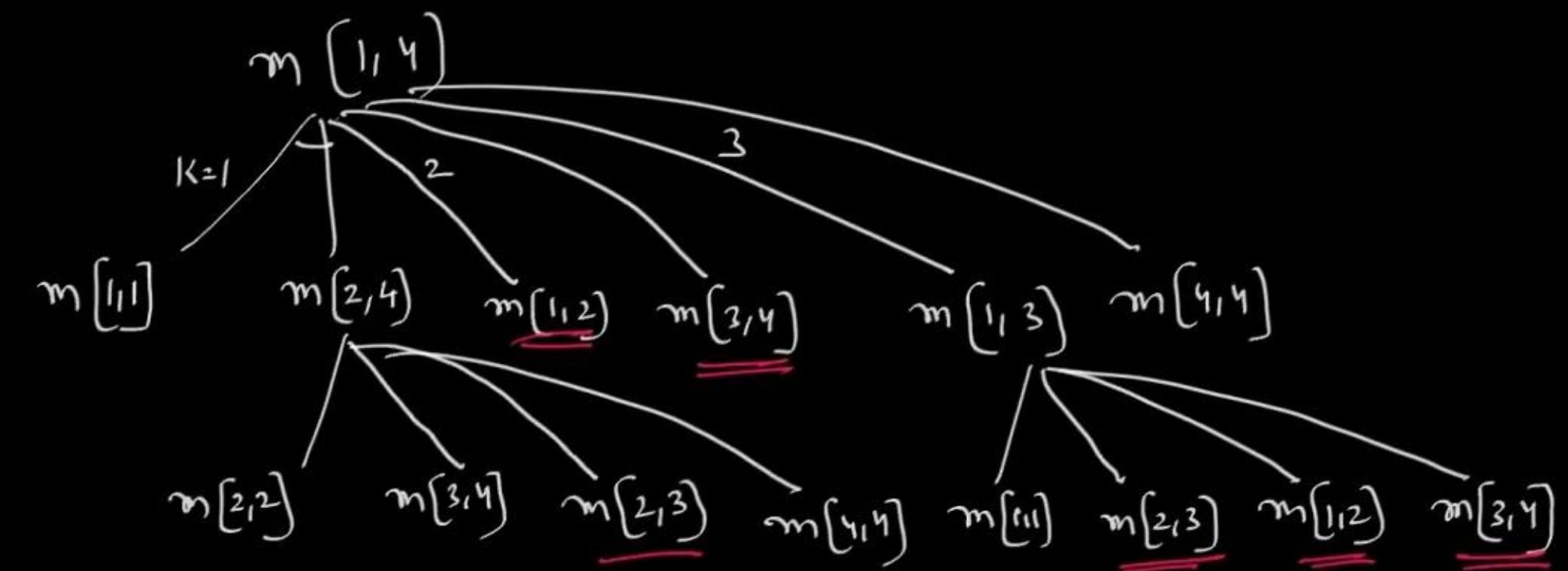
$$\langle A_1 \cdots A_3 \rangle \quad \langle A_4 \cdots A_6 \rangle$$

$\frac{2 \times 3}{P_0 \times P_1}$        $\frac{3 \times 7}{P_2 \times P_3}$





$$\langle A_1 \ A_2 \ A_3 \ A_4 \rangle$$



$$A_1 \rightarrow 2 \times 5$$

$$A_2 \rightarrow 5 \times 8$$

$$A_3 \rightarrow 8 \times 3$$

$$A_4 \rightarrow 3 \times 4$$

$$m[1,4] = \min_{k=1,2,3} \left\{ m[1,1] + m[2,4] + p_0 \cdot p_1 \cdot p_4 , m[1,2] + m[2,4] + p_0 \cdot p_2 \cdot p_4 , \right. \\ \left. m[1,3] + m[4,4] + p_0 \cdot p_3 \cdot p_4 \right\}$$

$$(j-i) = 3 \Rightarrow (j-i) = 2 \Rightarrow (j-i) = 1 \Rightarrow (j-i) = 0$$

$$\underbrace{j-i=0}_{\begin{cases} m[1,1]=0 \\ m[2,2]=0 \\ m[3,3]=0 \\ m[4,4]=0 \end{cases}}$$

$$\begin{cases} j-i=1 \\ m[1,2]=2 \cdot 5 \cdot 8 \\ m[2,3]=5 \cdot 8 \cdot 3 \\ m[3,4]=8 \cdot 3 \cdot 4 \end{cases}$$

$$\begin{cases} j-i=2 \\ m[1,3]= \\ m[2,4]= \end{cases}$$

Jahrelation



$$A_1 \rightarrow 2 \times 5$$

$$A_2 \rightarrow 5 \times 8$$

$$A_3 \rightarrow 8 \times 3$$

$$A_4 \rightarrow 3 \times 4$$

$$(A_1)(A_2 A_3 A_4)$$

$2 \cdot 5 \cdot 4 + 0 + 180$

$$(A_2)(A_3 A_4)$$

$0 + 8 \cdot 3 \cdot 4$

$$+ 5 \cdot 8 \cdot 4$$

$$\begin{array}{c} \langle A_1 A_2 A_3 A_4 \rangle = 152 \\ | \\ K=1 \quad 220 \\ | \\ 2 \quad 240 \\ | \\ (A_1 A_2) (A_3 A_4) \end{array}$$

$$2 \cdot 5 \cdot 8 + 8 \cdot 3 \cdot 4$$

$+ 2 \cdot 8 \cdot 4$

$$5 \cdot 8 \cdot 3 + 0 +$$

$5 \cdot 3 \cdot 4$

$$(A_1 A_2 A_3) (A_4)$$

$128 + 0 + 2 \cdot 3 \cdot 4$

$$150 \quad 128$$

$$(A_1) (A_2 A_3) \quad (A_1 (A_2) (A_3))$$

$$0 + 5 \cdot 8 \cdot 3$$

$+ 2 \cdot 5 \cdot 3$

$2 \cdot 5 \cdot 8 + 0 + 2 \cdot 8 \cdot 3$

$$\langle ((A_1 A_2) \cdot (A_3)) \cdot (A_4) \rangle$$

$(2 \cdot 5 \cdot 8 + 0 + 2 \cdot 8 \cdot 3) + 0 + 2 \cdot 3 \cdot 4$

Time :  $O(n^3)$

Where  $n = \text{no. of matrices in the chain}$

MSH

TSP

ALP

Dijkstra

SOS

LCS

MCP

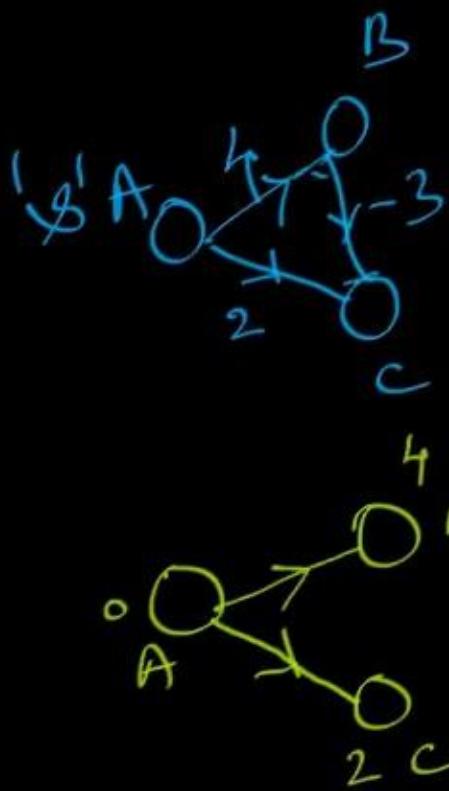
BF

BST

PSD



# 8) Bellman Ford : Single Source Shortest Paths (SSSP)

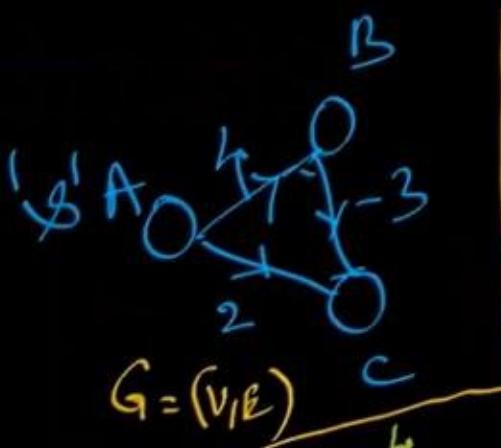


| vertex sel. | d-values |        |   |
|-------------|----------|--------|---|
|             | A        | B      | C |
| {A}         | -        | 4      | ② |
| {A, C}      | -        | ④      | ② |
| {A, B, C}   | -        | {④, ②} |   |

(dijkstra's algo. fail)

+ve - edge: cost (distance, delay)  
 (-ve) edge: Rate of loss, Temperature

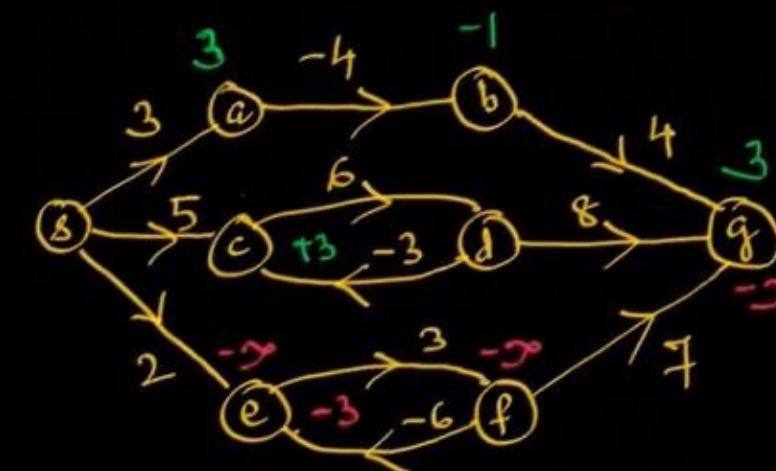
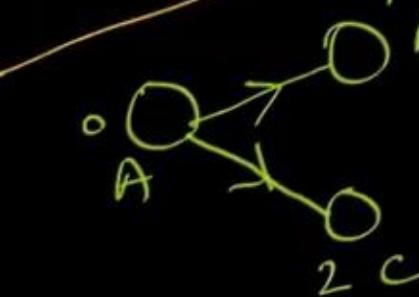
## 8) Bellman Ford : Single Source Shortest Paths (SSSP)



| Vertice<br>sel. | d-values |   |   |
|-----------------|----------|---|---|
|                 | A        | B | C |
| {A}             | -        | 4 | ② |
| {A, C}          | -        | ④ | ② |
| {A, B, C}       | -        | ④ | ② |

+ve-edge: Cost (distance, delay)

(-ve) edge: Rate of loss, Temperature



$$\begin{aligned} s-c: & 5 \\ (s-c-d): & 8 \end{aligned}$$

$$\begin{aligned} (s-c-d-c): & 8 \\ s-c-d-c: & 14 \end{aligned}$$

$$\begin{aligned} s-e: & 2 \\ s-e-f: & -1 \\ s-e-f-e: & -4 \end{aligned}$$

$$\begin{aligned} s-c-f: & 5 \\ s-c-f-e: & 2 \\ s-c-f-e-f: & -1 \end{aligned}$$

akshintalamahith@gmail.com

Note 1: In case if the graph has a +ve wt cycle (with -ve wt edges) then the shortest path to the vertices in the cycle are always determinable;

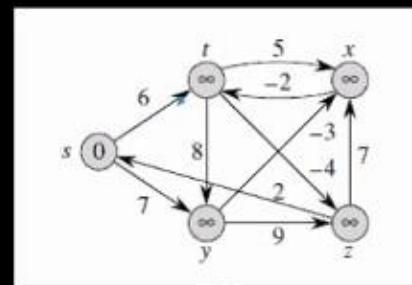
Note 2: In case if the graph has a -ve wt cycle reachable from some vertex, then the shortest path to the vertices in the cycle and also those vertices that are reachable via the cycle are not determinable;

BELLMAN-FORD( $G, w, s$ )

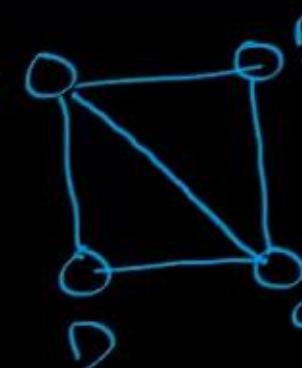
```

1 INITIALIZE-SINGLE-SOURCE( $G, s$ )
2 for  $i \leftarrow 1$  to  $|V[G]| - 1$ 
3   do for each edge  $(u, v) \in E[G]$ 
4     do RELAX( $u, v, w$ )
5 for each edge  $(u, v) \in E[G]$ 
6   do if  $d[v] > d[u] + w(u, v)$ 
7     then return FALSE
8 return TRUE

```

 $\mathcal{O}(n \cdot e)$ 

→ In a graph having  $n$ -vertices and  $e$ -edges,  
\* the max. no. of edges that an unrestricted path can  
 have b/w any 2-vertices, without any cycles is  $\frac{n-1}{}$



$\langle A \underline{\underline{B}} \rangle$ : 1 edge  
 $\langle A \underline{\underline{C}} \rangle$ : 2 edges  
 $\langle A \underline{\underline{D}} \underline{\underline{C}} \rangle$ : 3 edges  
A-D-C-B:

Time:  $O(n) + O(n \cdot e)$   
 $+ O(e)$   
 $= O(n \cdot e)$

$s \rightarrow \{t, x, y, z\}$   
 $x \rightarrow \{6, 4, 7, 2\}$   
 $\underline{\underline{\{2, 4, 7, -2\}}}$

(We are going to carryout Relaxation process for  $(n-1)$  times on each vertex w.r.t to edge costs)

INITIALIZE-SINGLE-SOURCE( $G, s$ )

```

1 for each vertex  $v \in V[G]$ 
2   do  $d[v] \leftarrow \infty$ 
3    $\pi[v] \leftarrow \text{NIL}$ 
4  $d[s] \leftarrow 0$ 

```

RELAX( $u, v, w$ )

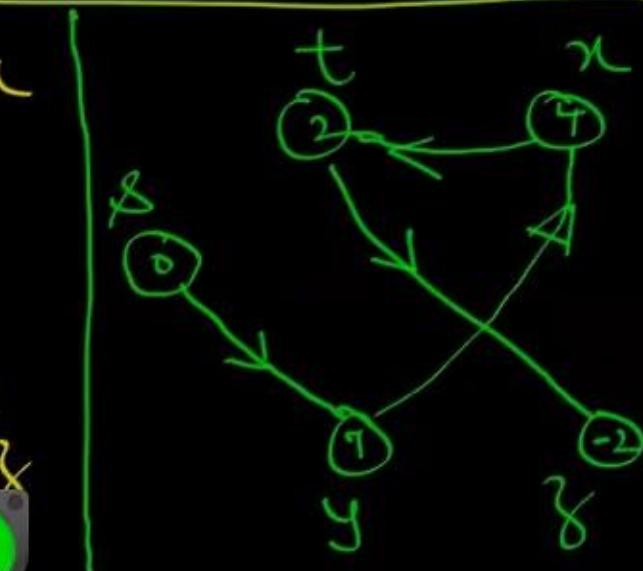
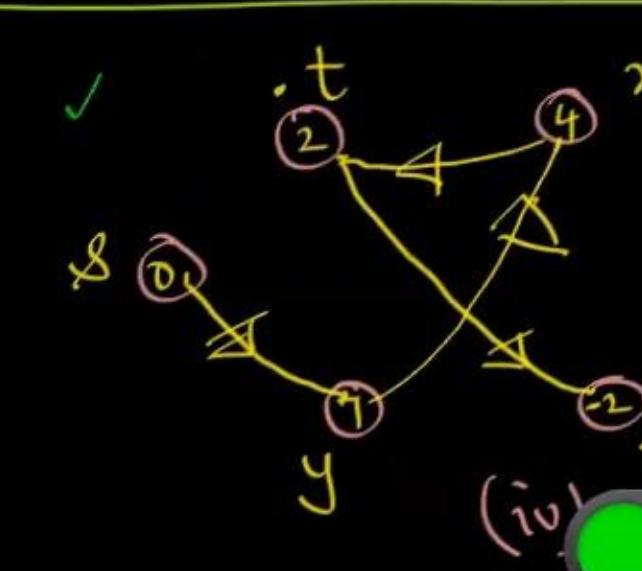
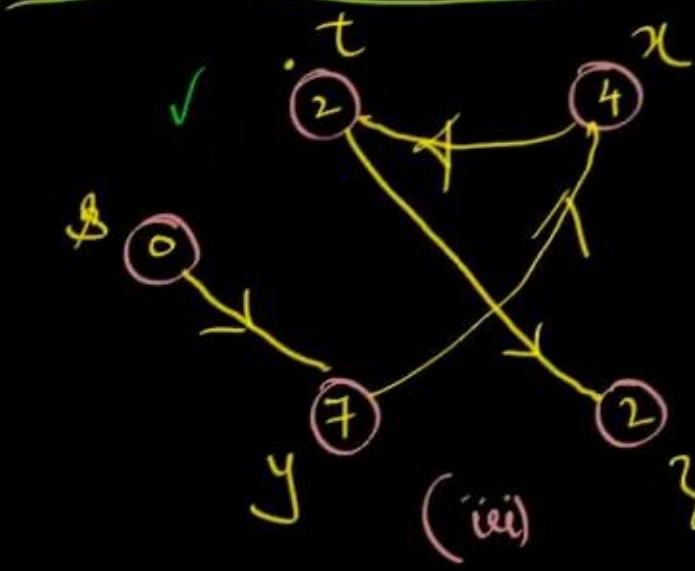
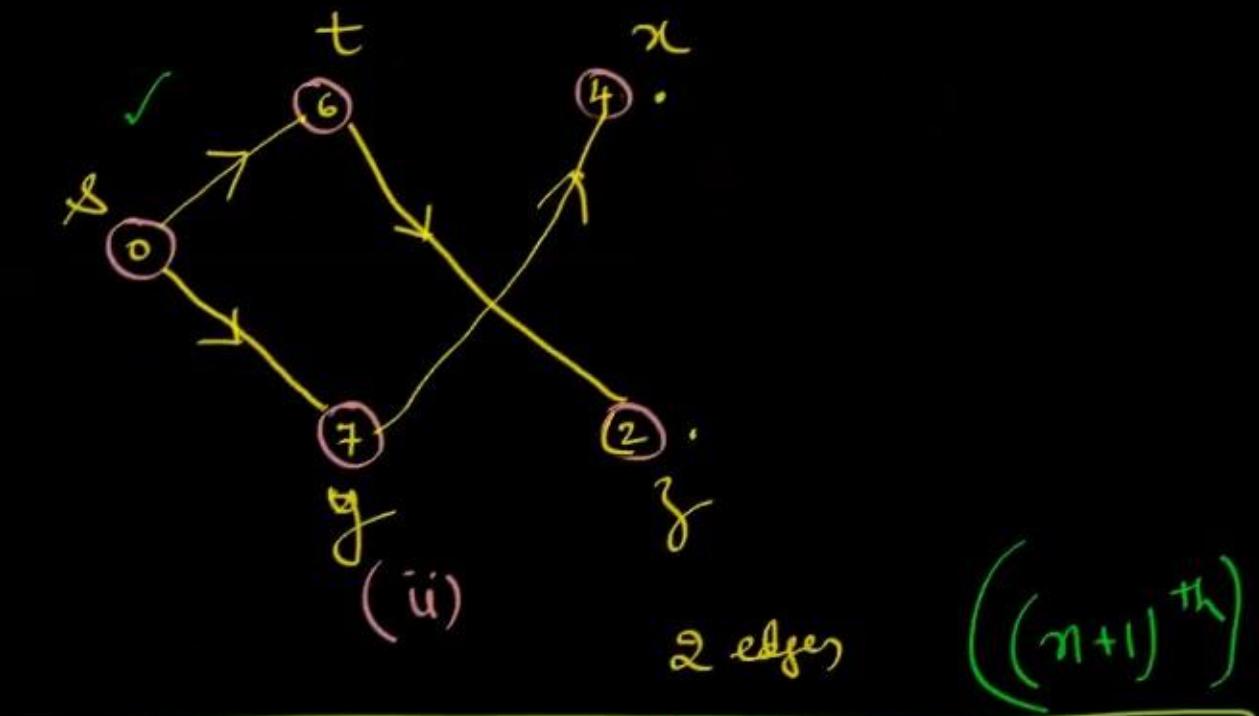
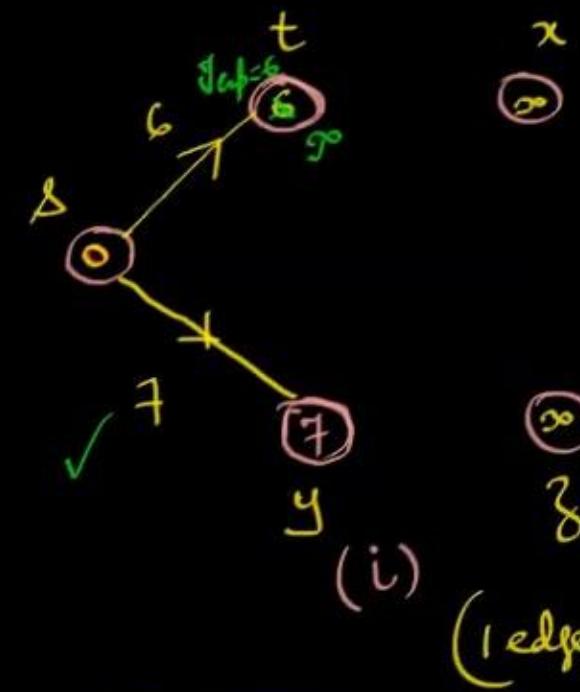
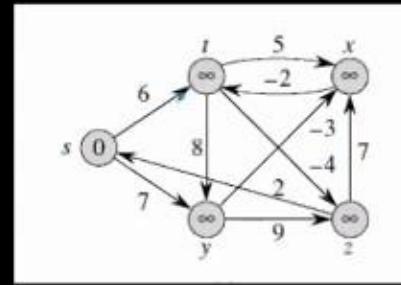
```

1 If  $d[v] > d[u] + w(u, v)$ 
2   then  $d[v] \leftarrow d[u] + w(u, v)$ 
3    $\pi[v] \leftarrow u$ 

```

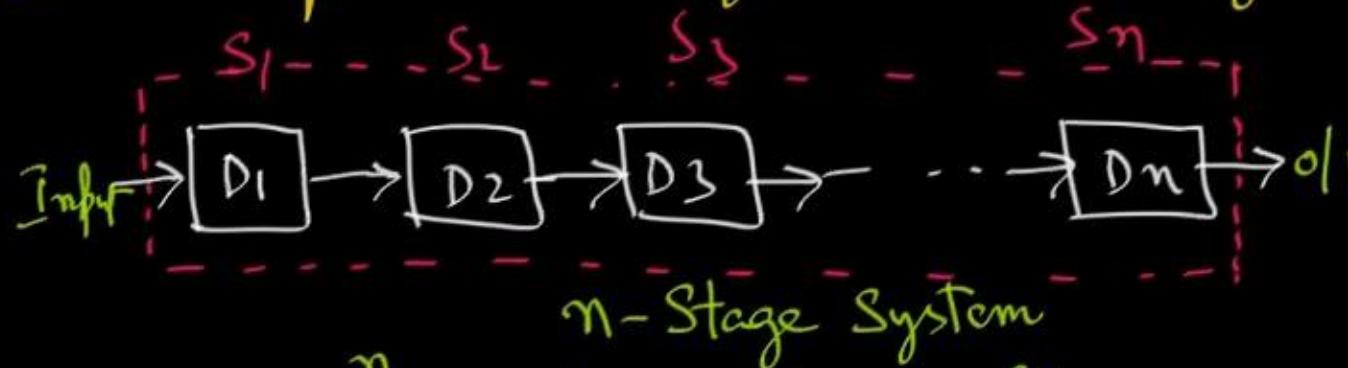
$d[v] > d[u] + w(u, v)$

$d[x]$ : Cost of reaching  
 b/c from source  
 is;



### 9) Reliable System Design (Reliability design)

"It is required to design an  $n$ -stage system, with Man. Reliability"

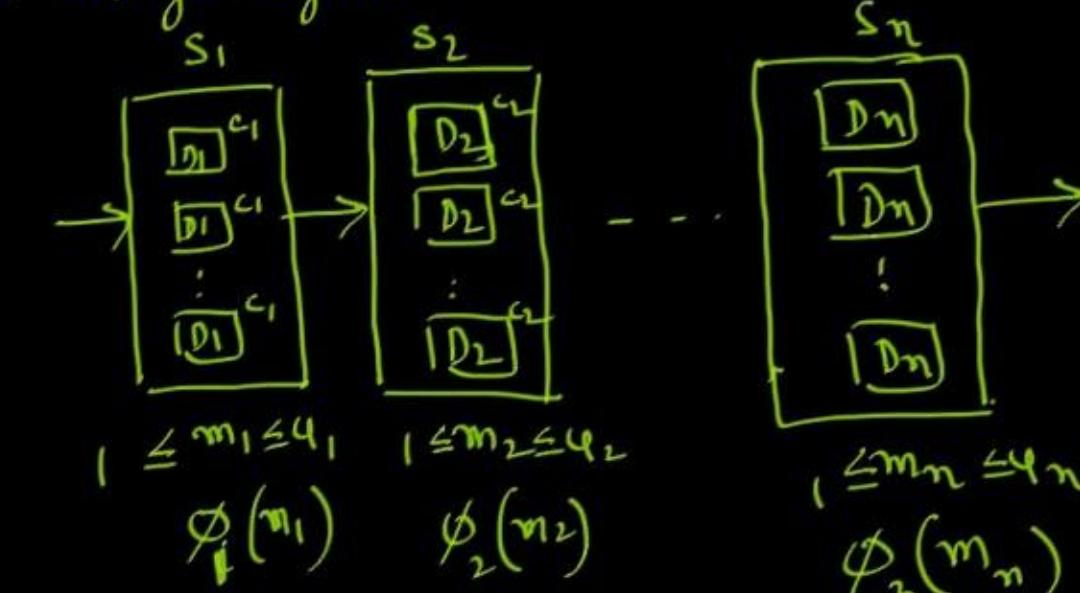


$D_i$  Cost :  $c_i$   
reliability :  $r_i$   
 $0 \leq r_i \leq 1$

$$\text{Total Cost} : \sum_{i=1}^n c_i$$

$$\text{Total Reliability} : \prod_{i=1}^n r_i$$

$$\left[ 1 - (1 - r_i)^{m_i} \right]$$



$$\phi_i(m_i) = 1 - (1 - r_i)^{m_i}$$

$$\text{Total Cost} = \sum_{i=1}^n c_i \cdot m_i$$

$$\text{Rel. of Sys.} = \prod_{i=1}^n \phi_i(m_i)$$

$$\text{where } \phi_i(m_i) = 1 - (1 - r_i)^{m_i}$$

$$\text{Proj Cost: } C$$



Max  $\prod_{i=1}^n \phi_i(m_i) \leq$

$$\text{S.T. } \sum_{i=1}^n c_i m_i \leq C$$

Where  $\phi_i(m_i) = 1 - (1 - r_i)^{m_i}$

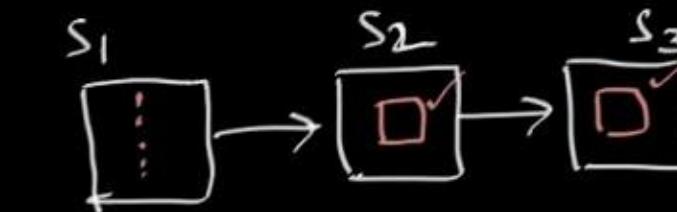
$1 \leq m_i \leq u_i$

$$u_i = \left\lfloor \frac{C - \sum_{j=1}^{i-1} c_j + c_i}{c_i} \right\rfloor$$

$$u_1 = \frac{C - (c_1 + c_2 + c_3) + c_1}{c_1} = \frac{C - (c_2 + c_3)}{c_1}$$

$$N=3; \quad \langle c_1 \dots c_3 \rangle = \langle 30, 15, 20 \rangle; \quad C = 105;$$

$$\langle r_1 \dots r_3 \rangle = \langle 0.9, 0.8, 0.5 \rangle$$



$$u_1 = \left\lfloor \frac{C - (c_2 + c_3)}{c_1} \right\rfloor$$

$$u_2 = \left\lfloor \frac{C - (c_1 + c_3)}{c_2} \right\rfloor$$

Recurrence:

Let  $f_n(C)$  repr. the Rel. of  $n$ -stage system with a total cost of  $C$ ,

$$\begin{cases} f_n(C) = \max \left\{ \phi_n(m_n) * f_{n-1}(C - c_n \cdot m_n) \right\} \\ 1 \leq m_n \leq u_n \\ f_0(x) = 1 \end{cases}$$

$$\begin{aligned} f(n) &= n * f(n+1) \\ f(0) &= 1 \end{aligned}$$



$$f_n(c) = \min \left\{ \phi_1(m_n), \phi_2(m_n), f_{n-1}(c - \underline{m_n}) \right\}$$

$$N=3; \quad \langle c_1 \dots c_3 \rangle = \langle 30, 15, 20 \rangle; \quad C = 105;$$

$$\langle r_1 \dots r_3 \rangle = \langle 0.9, 0.8, 0.5 \rangle$$

$$u_1 = 2 \quad \checkmark$$

$$u_2 = 3$$

$$u_3 = 3$$

Jufle Method: Let  $f_n \sim S^n = \{(Rel, Cost)\}$

$$S^n \sim S^{n-1} \sim S^{n-2} \dots S^1 \sim S^0$$

$$S^0 = \{(1, 0)\}$$

$$S^1 =$$

$$\phi_1(m_1=1) = 0.9$$

$$c_1 = 30$$

$$S_1^1 = \{(0.9, 30)\} \rightarrow \boxed{S_1^1 = \{(0.9, 30)\} \quad (0.99, 60)}$$

$$\begin{aligned} \phi_1(m_1=2) \\ = 1 - (1-0.9)^2 \\ = 0.99 \end{aligned}$$

$$\boxed{S^2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}}$$

$$\boxed{S_1^2 = \{(0.72, 45), (0.792, 75)\}}$$

$$\frac{15}{1 - (1-0.8)^2} = 0.96 \quad S_2^2 = \{(0.864, 60)\}$$

$$1 - (1-0.8)^3 = 0.992 \quad S_3^2 = \{(0.8928, 75)\}$$

$$N=3; \quad \langle c_1 \dots c_3 \rangle = \langle 30, 15, 20 \rangle; \quad C = 105;$$

$$f_n(c) = \min \left\{ \frac{f_n(m_1)}{f_{n-1}(c-m_1)}, \dots, \frac{f_n(m_N)}{f_{n-1}(c-m_N)} \right\}$$

$$\begin{aligned} u_1 &= 2 \\ u_2 &= 3 \\ u_3 &= 3 \end{aligned}$$

Jtuple method: Let  $f_n \sim S^N = \{(Rel, Cost)\}$

$$S^N \sim S^{N-1} \sim S^{N-2} \dots \sim S^1 \sim S^0$$

$$S^0 = \{(1, 0)\}$$

$$\begin{aligned} m_1 &= 0.9 \\ c_1 &= 30 \end{aligned}$$

$$S^1 = \{(0.9, 30)\}$$

$$\begin{aligned} m_2 &= 0.8 \\ c_2 &= 15 \\ 1 - (1 - 0.8)^2 &= 0.36 \\ 1 - (1 - 0.9)^2 &= 0.992 \end{aligned}$$

$$S^2 = \{(0.72, 45), (0.864, 60)\}$$

$$\phi_m(m_3=1) = 0.5$$

$$\phi_m(m_3=2) = 1 - (1 - 0.5)^2$$

$$\phi_m(m_3=3) = 1 - (1 - 0.5)^3$$

$$S^3 = \{(0.36, 65), (0.432, 80), (0.4464, 95)\}$$

$$S^3 = \{(0.54, 85), (0.648, 100)\}$$

$$S^3 = \{(0.63, 105)\}$$

$$(0.9) \cdot (1 - (1 - 0.8)^2) \cdot (1 - (1 - 0.5)^2) = 0.648$$

$$\underline{\underline{0.648, 100}}$$

$$(x, \underline{\underline{60}}) \in S^1$$

$$(y, \underline{\underline{30}}) \in S^1$$

$$Rel: \underline{\underline{0.648}}$$

$$Cost: \underline{\underline{100}}$$

$$m_3 = 2$$

$$\underline{\underline{100 - 40}} = 60$$

$$m_2 = 2$$

$$m_1 = 1$$

$$S^3 = \{(0.36, 65), (0.432, 80), (0.54, 85), (\underline{\underline{0.648, 100}})\}$$

## (10) Min. Cost Binary Search Tree : (OBST | MBST)

Defn: is a Binary Tree with the property that the value at each Node is greater than the values of nodes in left Subtree & less than the values of nodes in right Subtree;

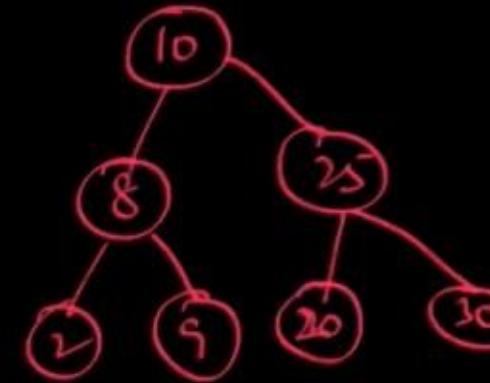
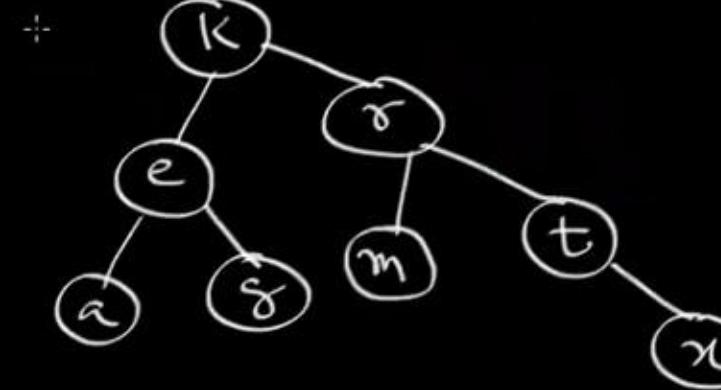
$$1) |L| < a_k < |R| : \text{B.S.T} : (\text{Searching})$$



$$\left. \begin{array}{l} 2) |L, R| < a_k : \text{Max-Heap} \\ 3) a_k < |L, R| : \text{Min-Heap} \end{array} \right\} \begin{array}{l} C.B.T \\ \text{Complete B.T} \end{array}$$

Appl: If B.S.T is in the Impl. of Keyword in Compilers;

akshintalamahvith@gmail.com

BSTBST

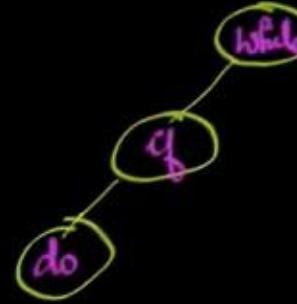
$a \leq b \leq c \dots \leq z$  : Lexical

$$\frac{1}{m+1} \cdot 2^n C_m$$

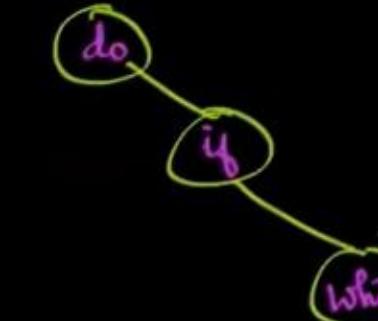
 $n=3$ 

5

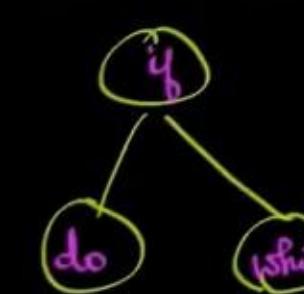
$$(do, if, while) = (a_1, a_2, a_3)$$



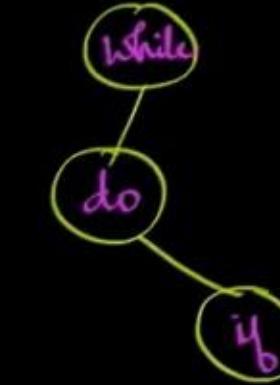
(1)



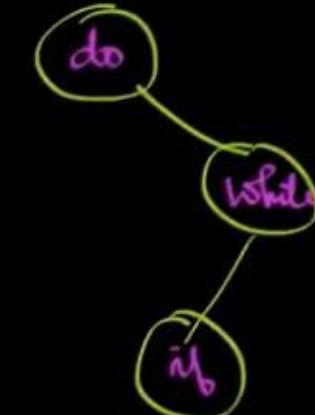
(2)



(3)

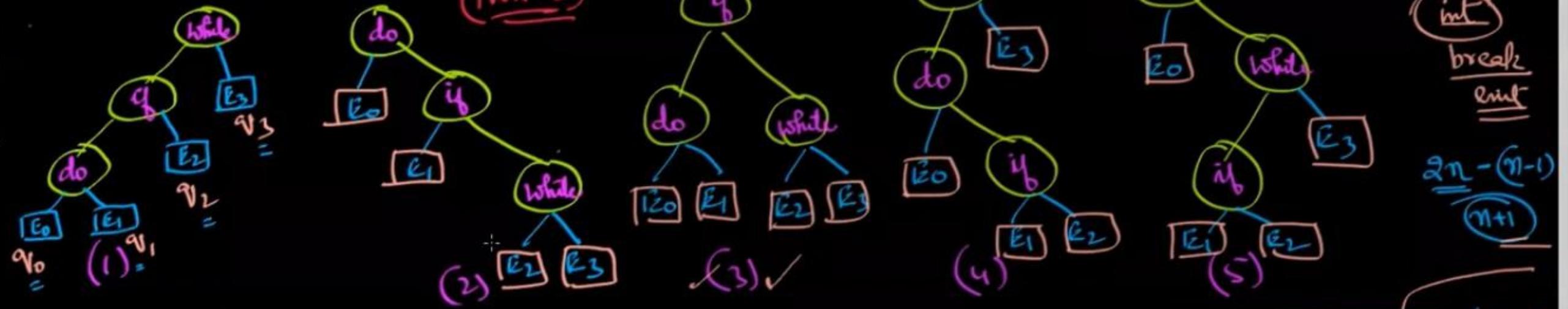


(4)



(5)

$$(do, if, while) = (a_1, a_2, a_3) \quad n=3; \quad \langle p_1-p_3 \rangle = \langle 0.5, 0.1, 0.05 \rangle \quad | \quad \langle q_{v_0} \dots q_{v_3} \rangle = \langle 0.15, 0.1, 0.05, 0.05 \rangle$$



$$\text{Cost}(T) = \text{Cost}(\text{Successful Searches}) + \text{Cost}(\text{Unsuccessful Searches})$$

$$\text{Cost}(\text{Succ. Searches}) = \sum_{i=1}^n \text{Cost}(a_i) = \sum_{i=1}^n p_i * \text{level}(a_i)$$

$\text{Cost}(a_i) \propto \text{level}(a_i)$   
 $= p_i * \text{level}(a_i)$ .

$p_i = \text{Prob. of using 'a}_i\text{' in applications}$

$$\text{Cost}(\text{Unsucc. Searches}) = \sum_{i=0}^n \text{Cost}(E_i)$$

$$\text{Cost}(E_i) \propto (\text{level}(E_i) - 1)$$

$$\text{Cost}(E_i) = q_{v_i} * (\text{level}(E_i) - 1)$$

$$= \sum_{i=0}^n q_{v_i} * (\text{level}(E_i) - 1)$$

int  
break  
exit

$$2n - (n-1) \\ \underline{\underline{n+1}}$$

{i, j, k}

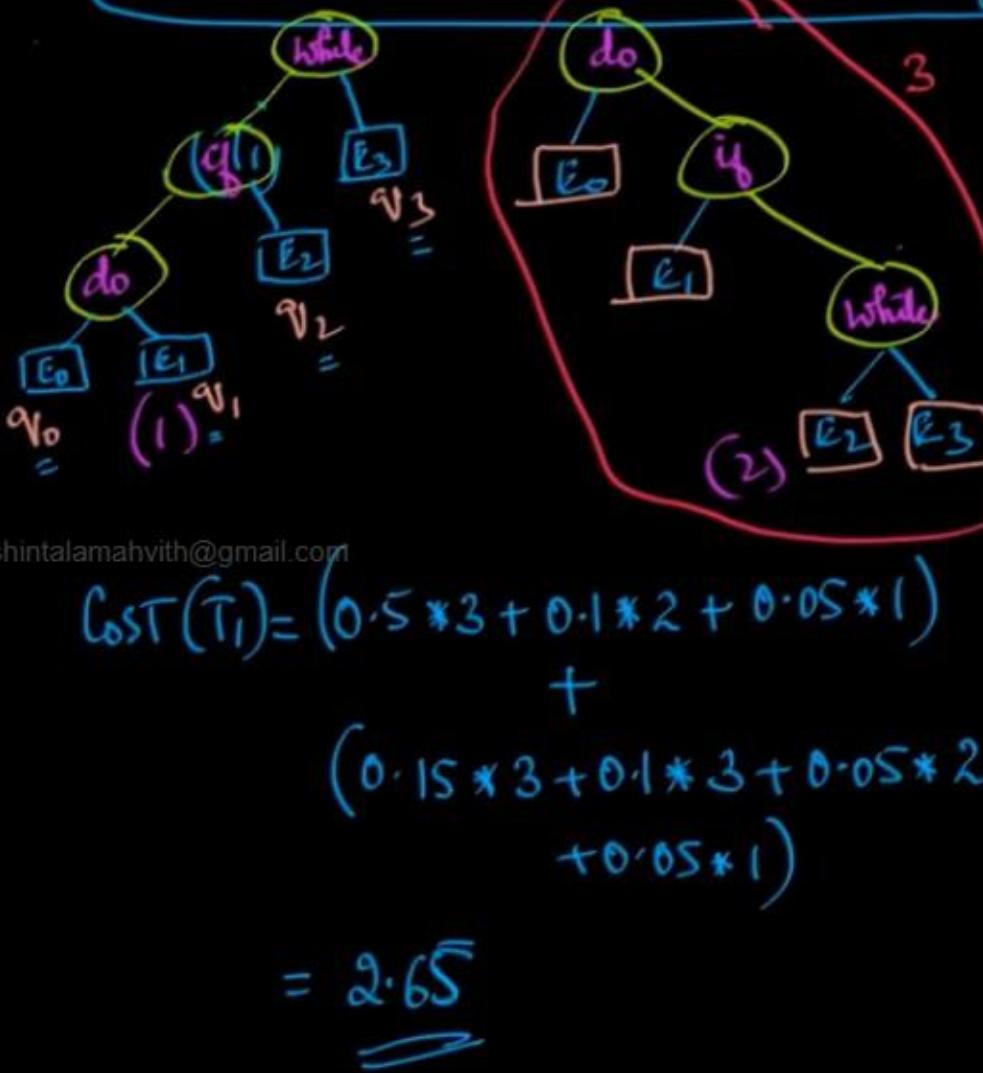
int t, P, K;

int a, b;





$$\text{Cost } (\tau) = \sum_{i=1}^n \hat{t}_i * \text{level}(a_i) + \sum_{i=0}^m v_i * (\text{level}(e_i) - 1)$$



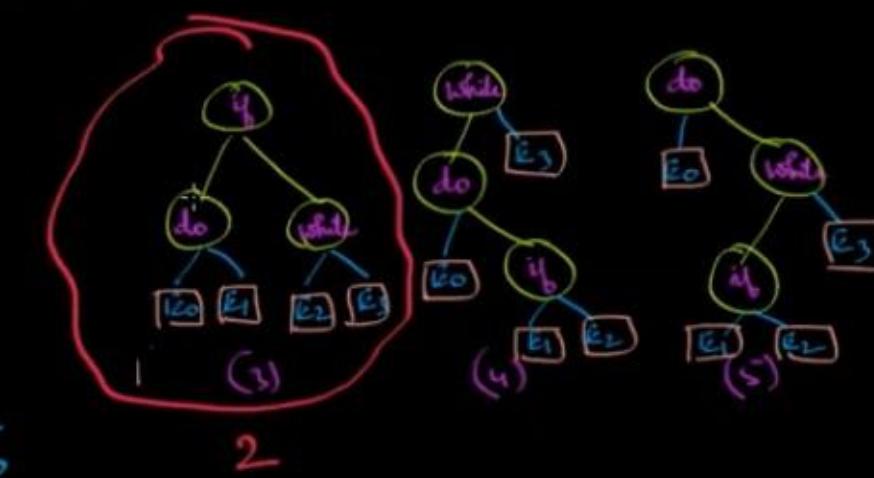
$$\begin{aligned}
 \text{Cost}(\tau_1) &= (0.5 * 3 + 0.1 * 2 + 0.05 * 1) \\
 &\quad + \\
 &\quad (0.15 * 3 + 0.1 * 3 + 0.05 * \\
 &\quad \quad + 0.05 * 1) \\
 &= \underline{\underline{2.65}}
 \end{aligned}$$

$$\text{Cost } (\tau_3) = 1$$

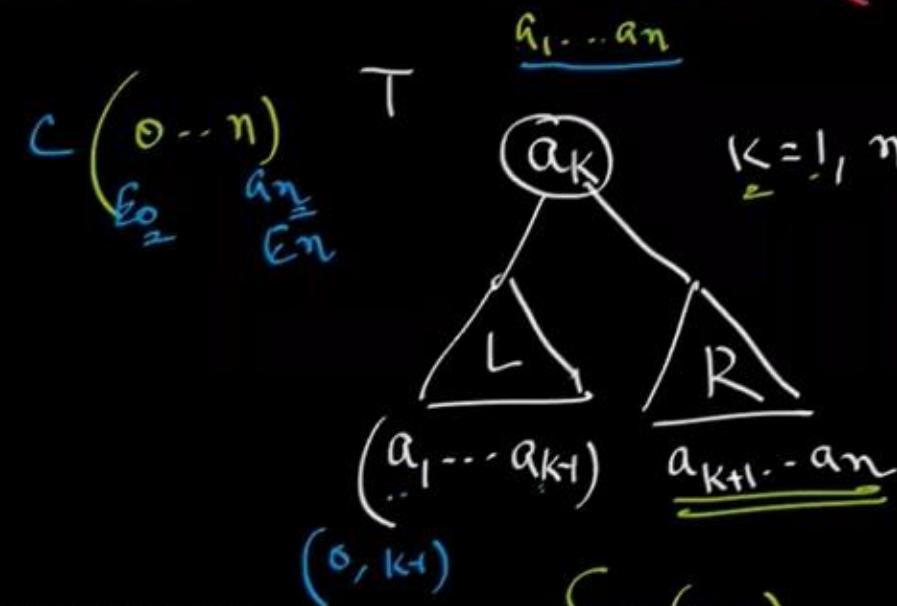
$$\text{Cost}(T_4) = 2.1$$

$$\text{Cost}(\tilde{\tau}_5) = 1$$

$$\text{Cost} \propto (\text{Height} / \text{Prob})$$



Given a set of  $n$ -identifiers  $(a_1, a_2, \dots, a_n)$  with  $(P_1, \dots, P_n)$   
 How to construct a OBST  $(M.C.R.S.T)$   $(V_0, \dots, V_n)$



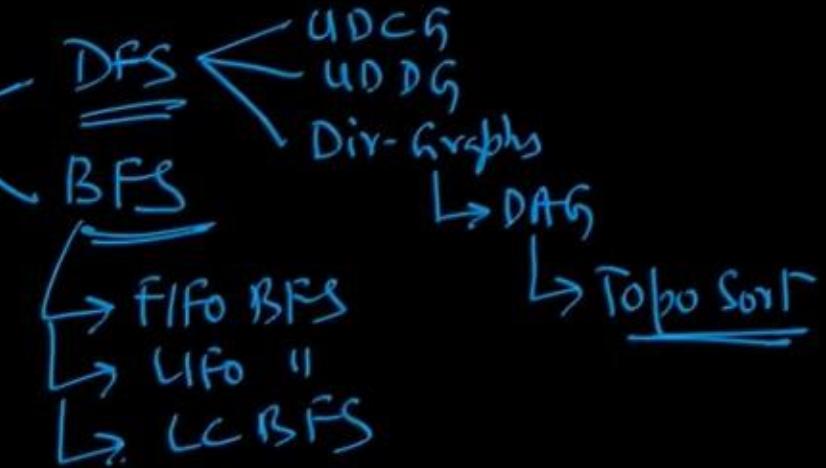
$$\begin{aligned} \text{Cost}(T) &= \text{Cost}(\text{Root}) + \text{Cost}(L) + \\ &\quad \text{Cost}(R) \\ &= P_k + \text{Cost}(L) + \text{Cost}(R) \end{aligned}$$

Let  $\text{Cost}(T) = \underline{\underline{c(0, n)}}$

$$\left\{ \begin{array}{l} c(0, n) = \min_{1 \leq k \leq n} \{ c(0, k-1) + c(k, n) + P_k \} \\ c(i, i) = 0 \end{array} \right\} \quad (\text{MCP})$$

akshintalamahvith@gmail.com



(i) Graph Techniques (Traversals)(ii) Heap Algorithms:

→ Insert Method:  $O(n \log n)$   
 → Heapsort  $\rightarrow O(n)$

Insert

Delete

Inc/Dec Key

→ Heapsort

DC  
 → LIM  
 Rec-Tree

(iii) Sorting Algo.:

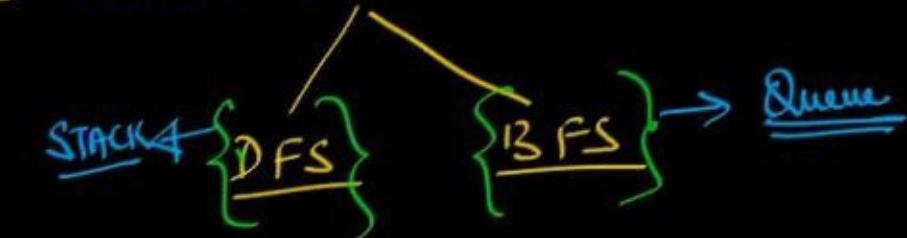
→ Sel  
 → Ins  
 → B-S  
 → Radix Sort

→ {Back vs B-B}

akshintalamahvith@gmail.com

## GRAPH TECHNIQUES: Traversals

### 1) Status of the Node during Traversal:



a) E-Node : The node which is currently being explored or whose children is being generated.  
 (only one E-Node during Traversal)

b) Live Node : The nodes which are not fully explored or all of whose children has not been generated is live Node.  
 (There can be live Nodes during Traversal)

c) dead Node : The node which has been fully explored or all of whose children has been generated.

## b) Timing Values Associated with Nodes:



$\textcircled{x} \frac{d}{f}$

1) discovery-time: The time at which the Node is visited for the first time.

2) finishing-time: The time at which the Node becomes dead.

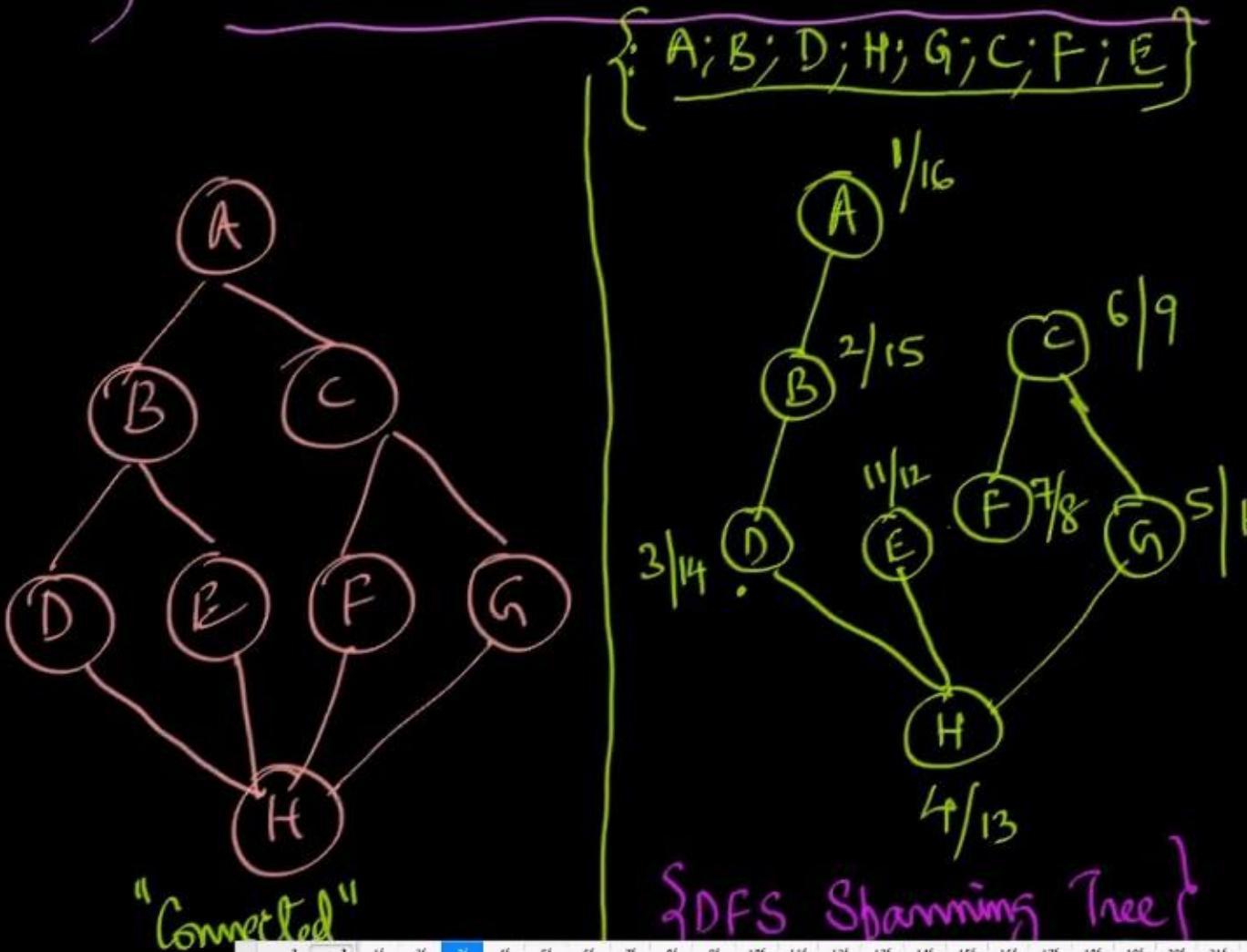
[ 'd' & 'f' are more meaningful in DPS ]

App's:

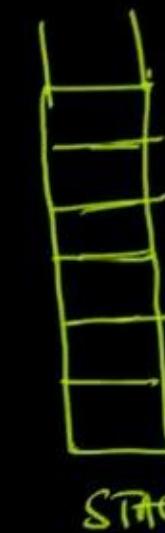
- 1) To determine the topology of graph
- 2) Parenthesization theorem to determine types of edges
- 3) To determine linear order of a DAG (Topological Sort)

i) DFS (Depth First Search):

a) DFS in undirected connected graph:



Time: Rep. graph  $\xrightarrow{\text{Matrix}} O(n^2)$   
 Adj. list  $O(n+e)$

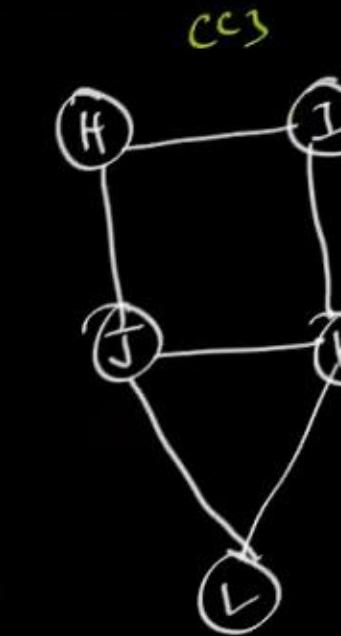
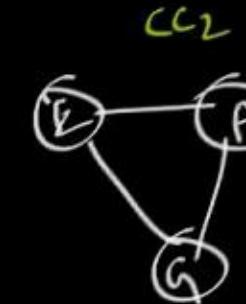
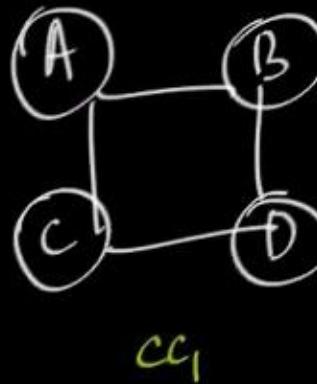


VIS DFS

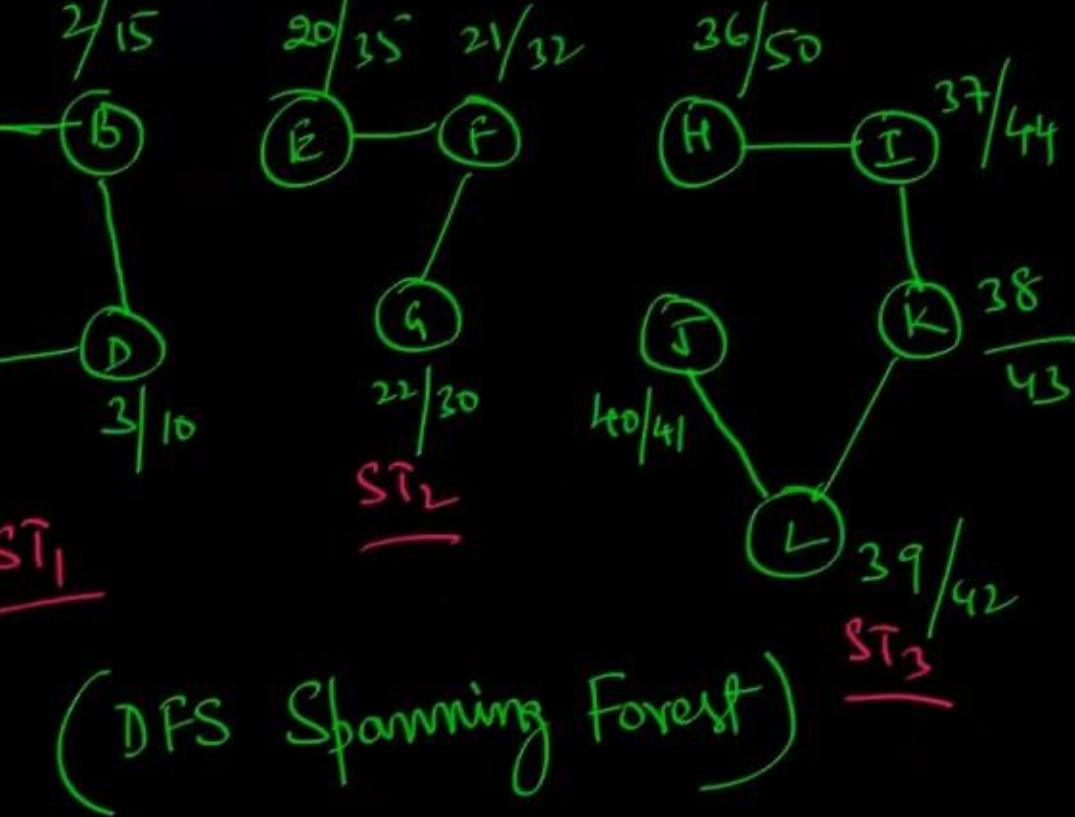
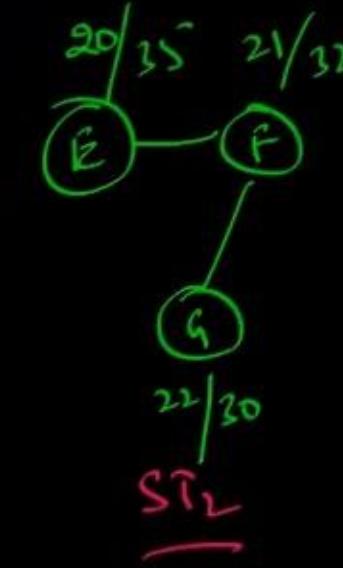
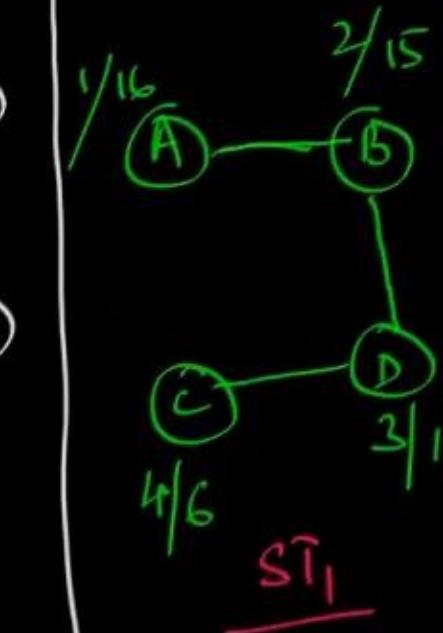
- ✓ a) A; B; E; H; F; C; G; D
- ✓ b) H; D; B; E; A; C; F; G
- ✗ c) A; B; D; E; H; F; G; C
- ✗ d) H; F; C; G; B; A; D; E
- ✓ e) G; H; F; C; A; B; E; D

b) DFS in undirected disjoint graphs

(DFT (Depth First Traversal))



$$G = (V, E)$$



Q) Consider a DFS performed on a undirected graph with 4 vertices ( $P, Q, R, S$ ). Their 'd' & 'f' values are as given:

- |    | $P$        | $Q$        | $R$        | $S$        |
|----|------------|------------|------------|------------|
| 1) | $d=1, f=2$ | $d=2, f=3$ | $d=3, f=4$ | $d=4, f=5$ |
- 1)  $\langle S, 12 \rangle \langle 8, 11 \rangle \langle 9, 10 \rangle \langle 13, 14 \rangle \rightarrow \text{N.C. } \langle P \& R \rangle \langle S \rangle$
- 2)  $\langle 8, 12 \rangle \langle 6, 11 \rangle \langle 9, 10 \rangle \langle 15, 18 \rangle \rightarrow \text{Invalid}$
- 3)  $\langle 8, 9 \rangle \langle 5, 10 \rangle \langle 2, 12 \rangle \langle 1, 15 \rangle \rightarrow \text{Connected } \langle P \& R \& S \rangle$
- 4)  $\langle 15, 20 \rangle \langle 5, 8 \rangle \langle 25, 40 \rangle \langle 10, 12 \rangle \rightarrow \text{N.C. } \langle Q \rangle \langle S \rangle \langle P \rangle \langle R \rangle$
- 5)  $\langle 6, 15 \rangle \langle 5, 12 \rangle \langle 8, 9 \rangle \langle 20, 22 \rangle \rightarrow \text{Invalid}$

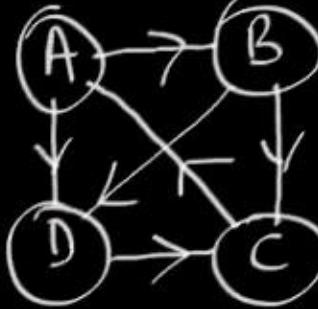
Whether ' $G$ ' is connected/Not & if its disconnected (No. of connected components)



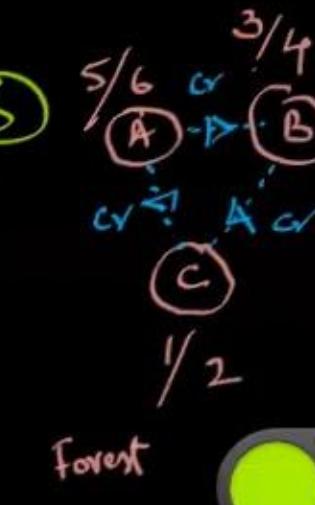
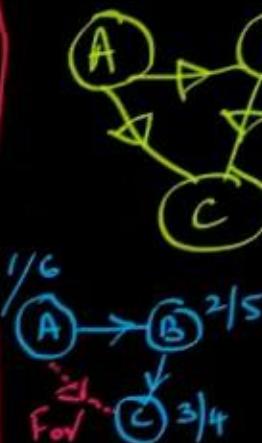
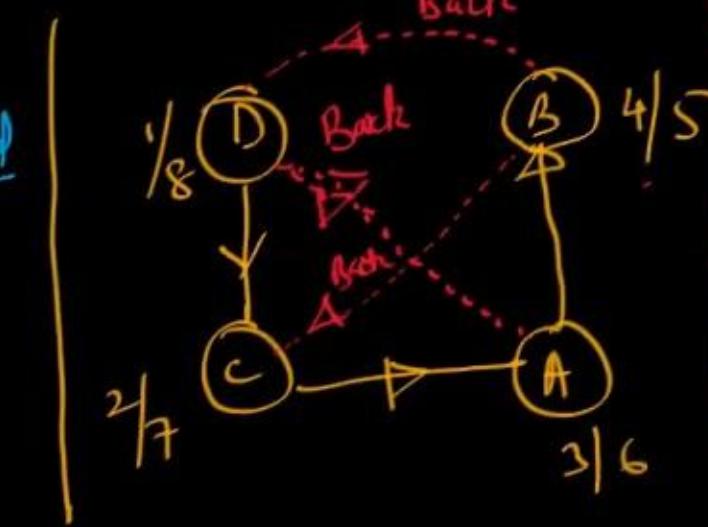
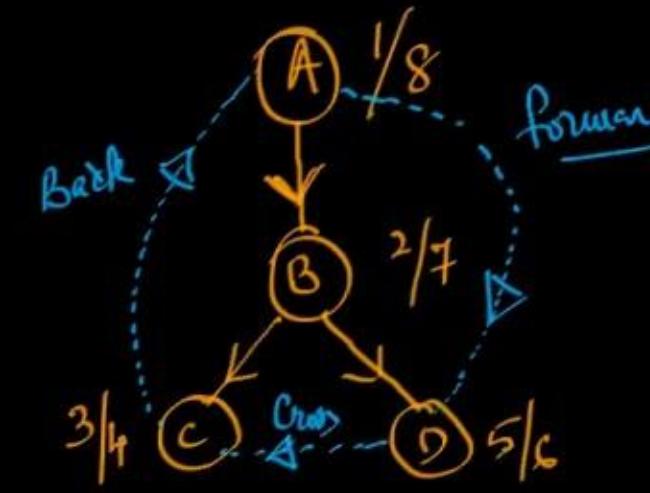
### ③ DFS in directed graphs:

DFS when carried out in directed graphs, will lead to the following edges:

- a) Tree edges: are part of DFS Spanning Tree/Forest
- b) Forward Edge: lead from a Node to its non-child Descendent
- c) Back Edge: lead from a Node to its ancestor
- d) Cross Edge: " " " " another node which is neither its ancestor nor its descendent



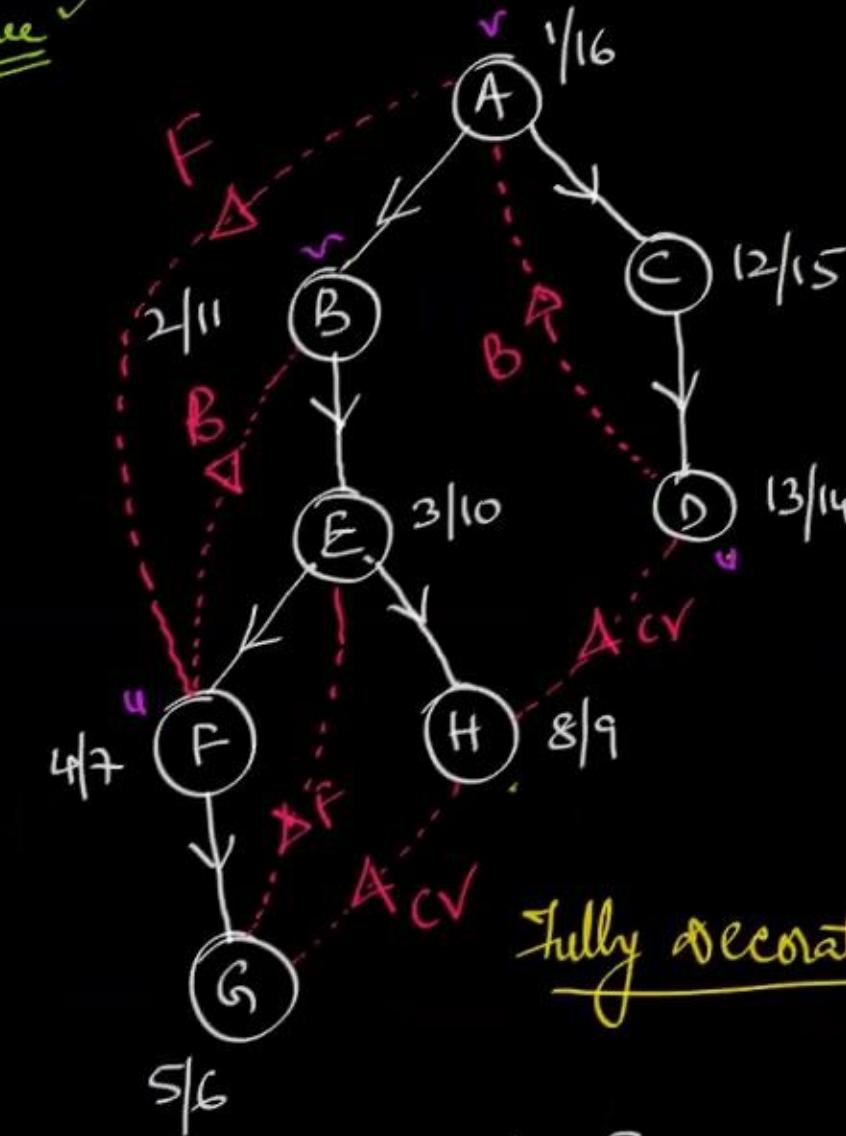
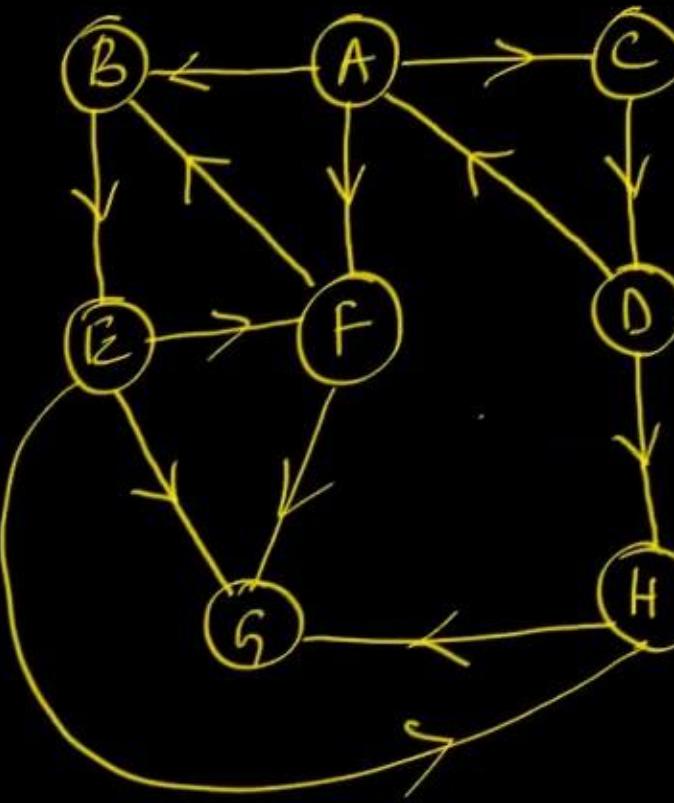
$$G = (V, E)$$



Forest



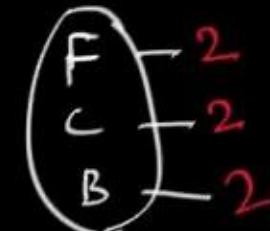
$$\left\{ \begin{array}{l} 1 \\ 2 \\ 2 \\ 11 \\ 4 \end{array} \right\} \left[ \begin{array}{l} r \\ r \\ 16 \end{array} \right] : \text{tree} \checkmark$$



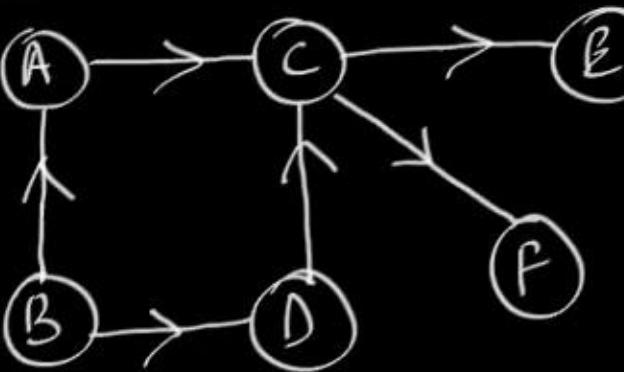
DFS Spanning Tree ?

Fully decorated DFS Spanning Tree/Forest

Role of Back Edge  
 (Reveals the presence of a cycle)



4) DFS in DAG: <Directed Acyclic Graph>: Task (Activities)  
Source: Sink, linear order: Topological order: Sort

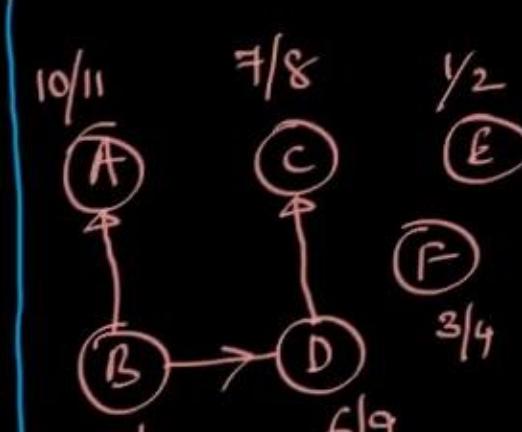
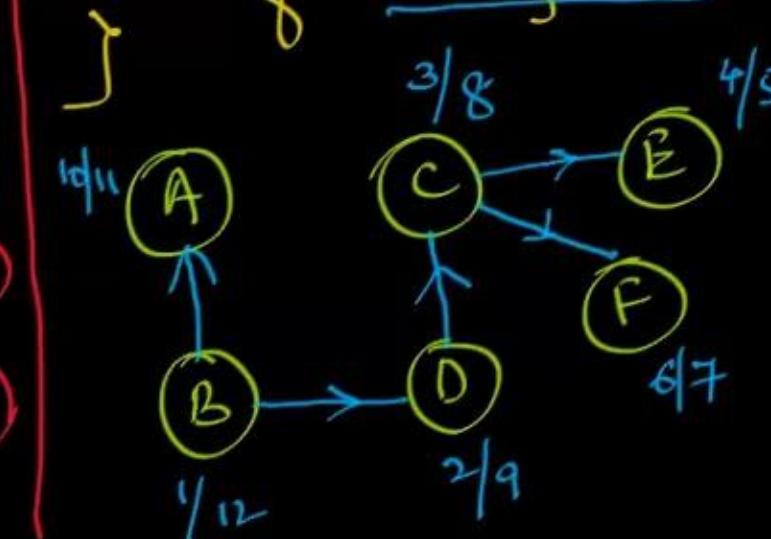


$G_F(V, E)$

A-D-C < E-F-①  
 B < A-D-C < E-F-②  
 D-A-C < E-F-③  
 F-E-④

Algo Topo (r)

1. DFS( $V$ );
2. Arrange all the vertices in the decreasing order of finishing times



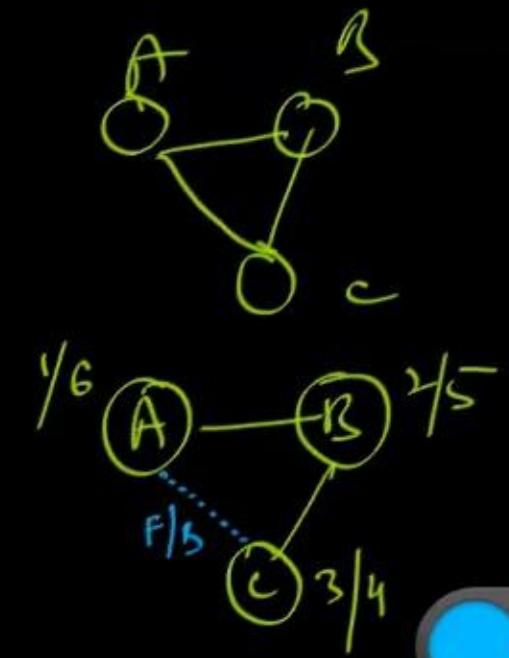
## Parenthesization Theorem:

In any DFS of a directed (undirected) graph, for any two vertices  $(u, v)$  having an edge, exactly one of the following conditions holds;  $d$ : discovery time ;  $f$  = finishing time

i.  $\left[ \begin{array}{c} [d[u] \\ d[v]] \\ f[v] \\ f[u] \end{array} \right]$  : Tree edge / Forward Edge.

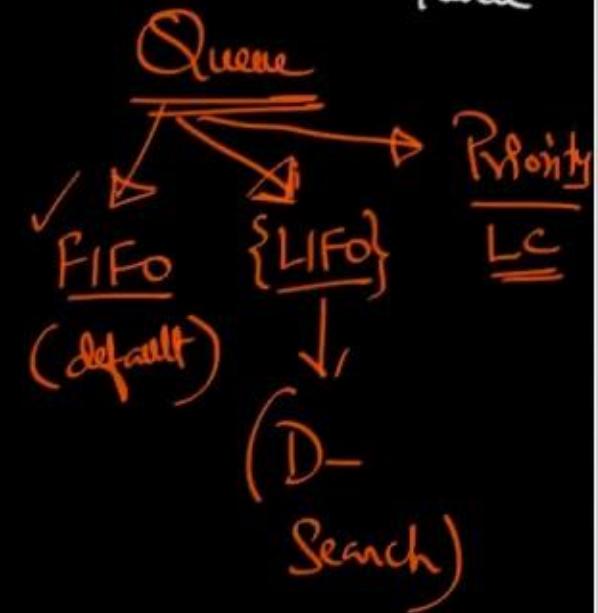
ii.  $\left[ \begin{array}{c} [d[v] \\ d[u]] \\ f[u] \\ f[v] \end{array} \right]$  : Back Edge

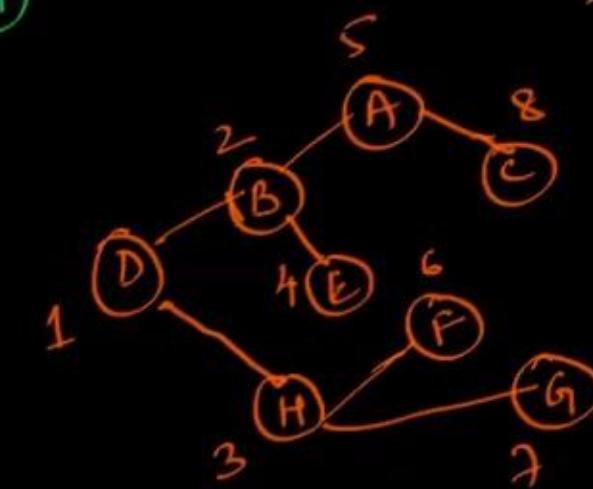
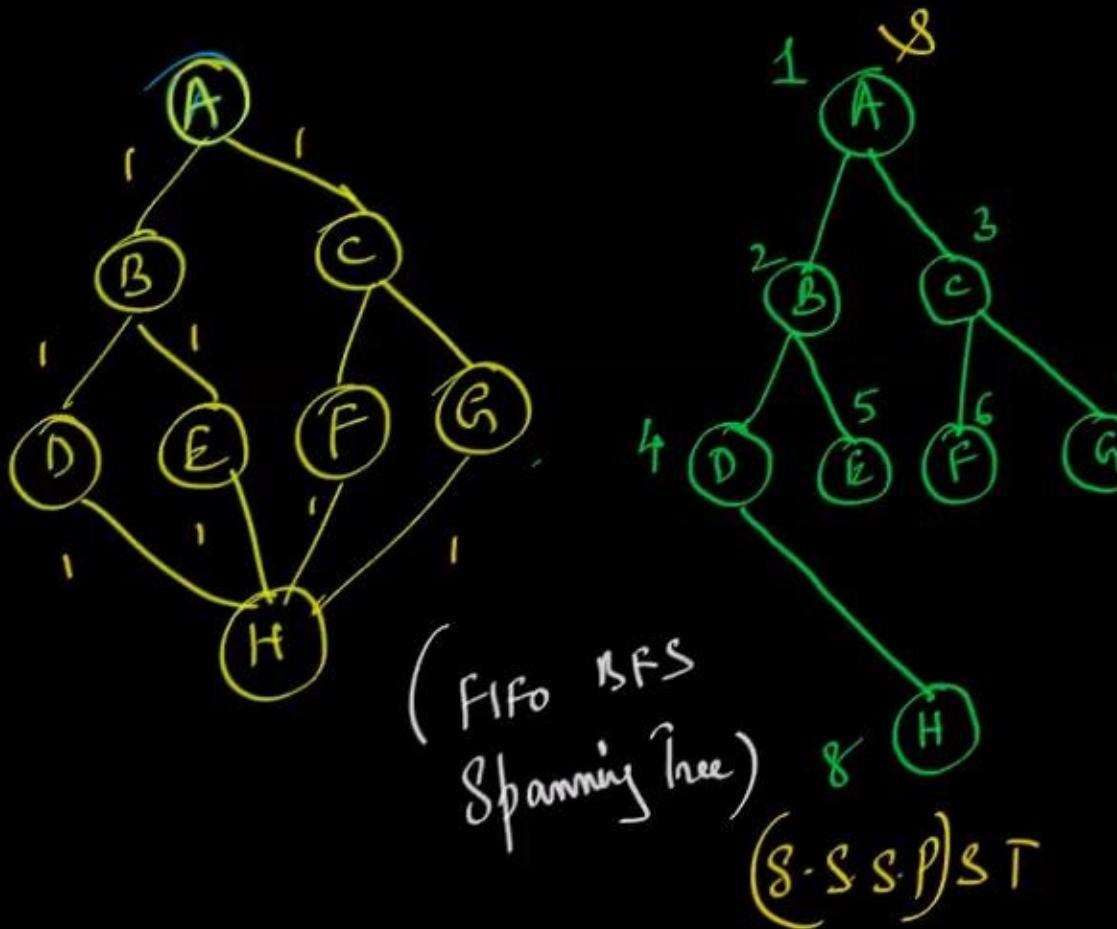
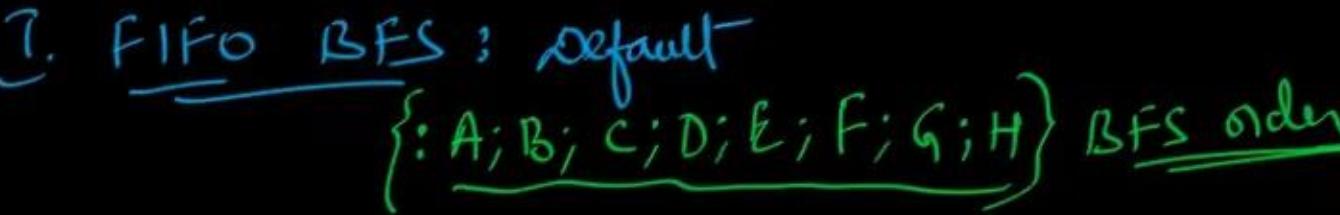
iii.  $\left[ \begin{array}{c} [d[u] \\ f[u]] \\ [d[v] \\ f[v]] \end{array} \right]$  : Cross Edge



ii. Breadth First Search (BFS): current S-node generates all its children & then among those that has not been visited will be added to Queue as live nodes. And the current S-node becomes Dead Node.

- FIFO- BFS (✓)
- LIFO- BFS (D-Search) ✓
- Priority- BFS (LC Search) ✓





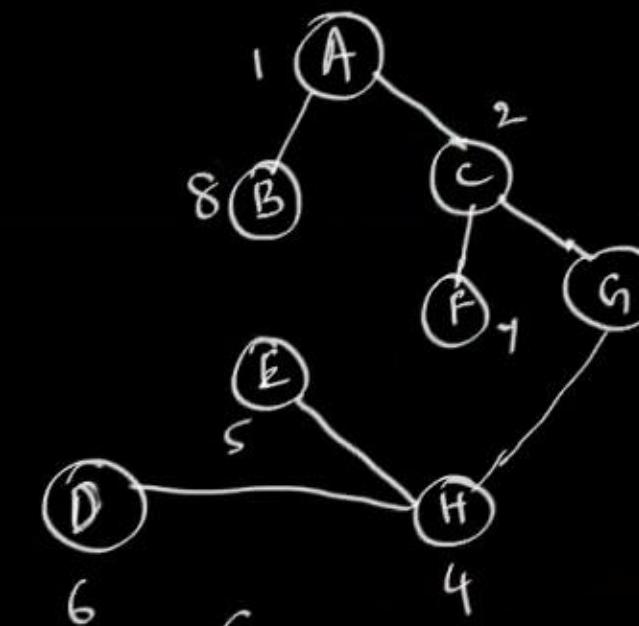
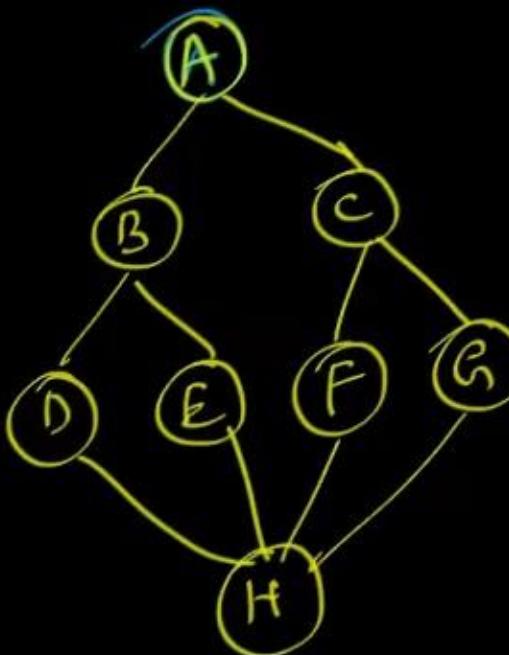
|   |   |   |   |   |   |   |   |   |  |   |   |   |   |   |   |   |  |
|---|---|---|---|---|---|---|---|---|--|---|---|---|---|---|---|---|--|
| $\emptyset$<br>$\left\{ \begin{array}{l} \text{Parent} \\ \text{Line Node} \end{array} \right.$ | <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">A</td><td style="padding: 5px;">A</td><td style="padding: 5px;">B</td><td style="padding: 5px;">B</td><td style="padding: 5px;">C</td><td style="padding: 5px;">C</td><td style="padding: 5px;">D</td><td style="padding: 5px;"></td></tr> <tr> <td style="padding: 5px;">B</td><td style="padding: 5px;">C</td><td style="padding: 5px;">D</td><td style="padding: 5px;">E</td><td style="padding: 5px;">F</td><td style="padding: 5px;">G</td><td style="padding: 5px;">H</td><td style="padding: 5px;"></td></tr> </table> | A | A | B | B | C | C | D |  | B | C | D | E | F | G | H |  |
| A   | A   | B | B | C | C | D |   |   |  |   |   |   |   |   |   |   |  |
| B   | C   | D | E | F | G | H |   |   |  |   |   |   |   |   |   |   |  |

a) H D E F G B C A  
b) D - B - H - E - A - F - G - C

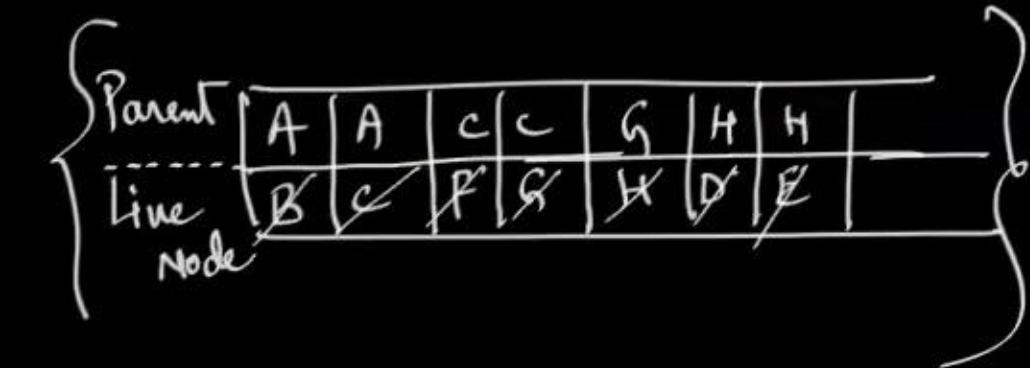
|     |   |   |   |   |   |   |   |
|-----|---|---|---|---|---|---|---|
| Q   | D | D | B | B | H | H | A |
| L-N | B | W | E | A | F | G | C |

2) LIFO - BFS ( D-Search)  
depth

{ : A; C; G; H; E; D; F; B }



( LIFO BFS Spanning Tree )



( Is D-Search same  
as DFS? ) X

### 3) Priority BFS: (LC BFS) / Least Cost:

(15-Puzzle problem)

$$\{ \hat{C}(x) = f(x) + h(x) \}$$

$f(x)$  = cost of reaching ' $x$ ' from <sup>root</sup>

$h(x)$  = cost of reaching the soln from ' $x$ ';

(approx)

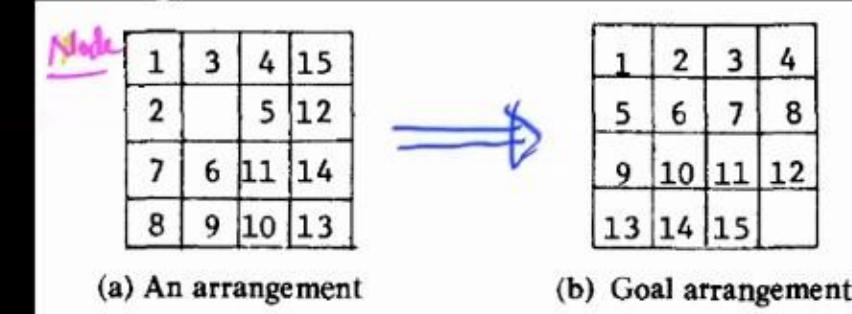
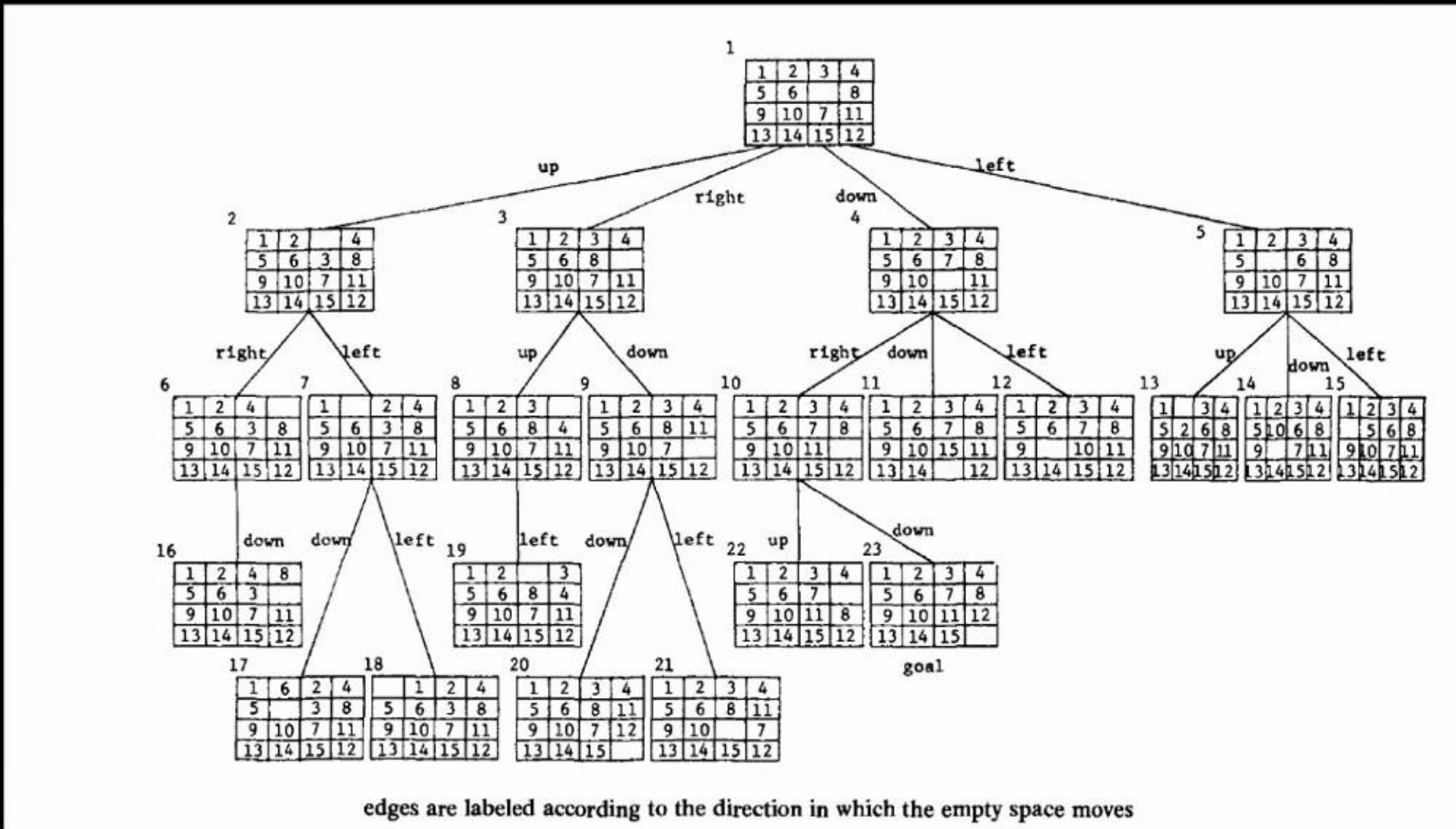


Figure 8.2 15-puzzle arrangements

→ Every live Node will be associated  
with a cost function:  $\hat{C}(x)$   
 $x$ : Node

FIFO & LIFO BFS  
are rather Blind  
Search Strategies.  
They do not give  
preference to that  
live Node for making  
as E-Node, from  
where there is a  
possibility of reaching  
the ans solution)  
faster;

{ Max #g moves : 4 ✓, 3 ✗ }  
Min #g moves : 2 ✓



edges are labeled according to the direction in which the empty space moves

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  |    | 8  |
| 9  | 10 | 7  | 11 |
| 13 | 14 | 15 | 12 |

(LC-Search)

$h(x)$ : (No. of tiles that are not in their goal position)

(2)

|    |    |    |    |
|----|----|----|----|
| 1  | 2  |    | 4  |
| 5  | 6  | 3  | 8  |
| 9  | 10 | 7  | 11 |
| 13 | 14 | 15 | 12 |

(3)

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 |    | 11 |
| 13 | 14 | 15 | 12 |

1

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  |    | 8  |
| 9  | 10 | 7  | 11 |
| 13 | 14 | 15 | 12 |

up

down

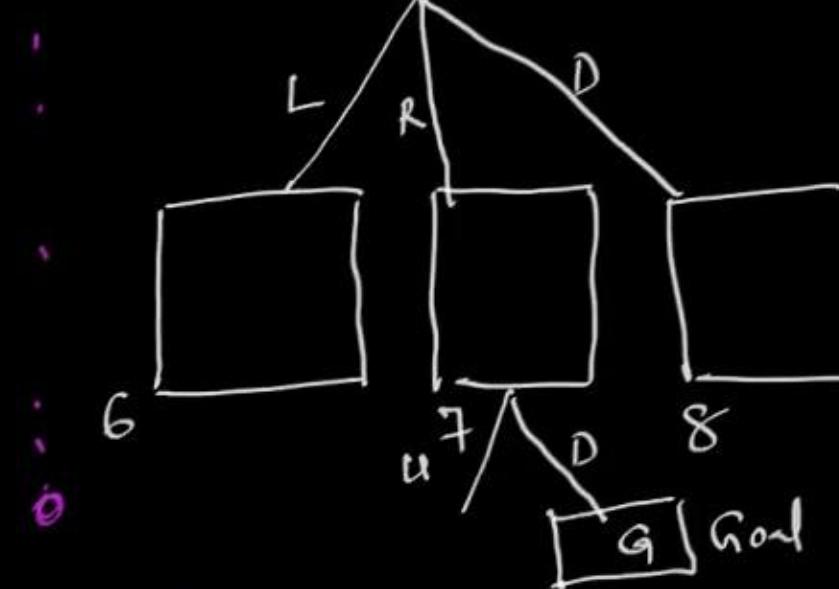
right

left

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 |    | 11 |
| 13 | 14 | 15 | 12 |

$$\begin{aligned}C(2) &= 1+4 = 5 \\ \rightarrow C(1) &= 1+2 = 3 \\ C(4) &= 1+4 = 5 \\ C(5) &= 1+4 = 5\end{aligned}$$

$$\begin{aligned}C(6) &= 2+3 = 5 \\ \rightarrow C(7) &= 2+1 = 3 \\ C(8) &= 2+3 = 5\end{aligned}$$



(LC-BFS)

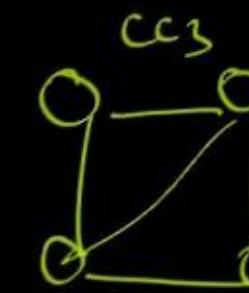
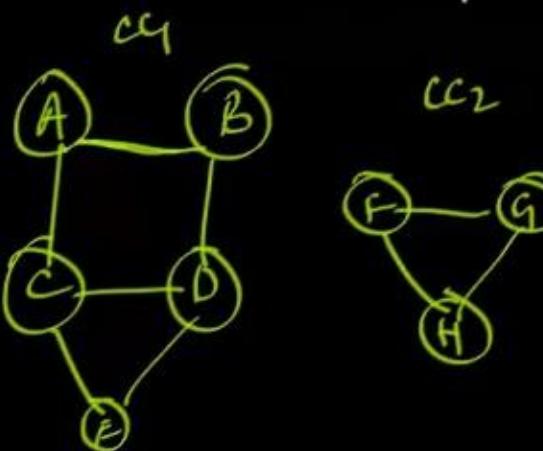
## Applications of DFS/BFS:

- The Time complexity of DFS/BFS depends of the repr. of graph.  
If Graph is represented as Matrix then the time complexity is  $O(n^2)$   
otherwise for Adj. list is  $O(n+e)$
- DFS & BFS can be used to find the presence of cycle in the graph.
- DFS & BFS can be used to find out whether the given graph is connected or not;
- DFS & BFS can be used to find out whether there is a path b/w two vertices of graph.
- (BFS (FIFO) when carried out over an undirected connected graph ~ Single Source Shortest Paths)
- Both BFS & DFS can be used to determine, conc. components,  
Strongly conc. components, Bi-connected comp., Art. Points

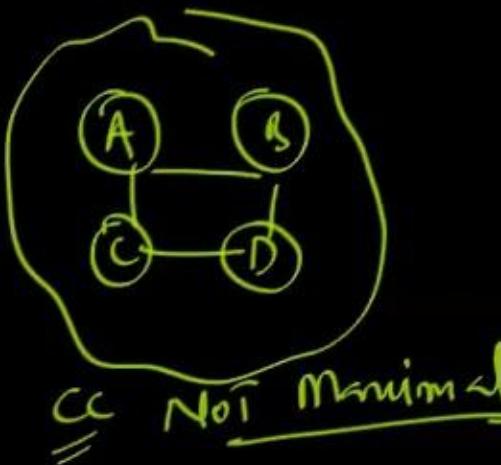
Components:

1. Connected Components: { Maximal Connected Subgraph of a given graph }

<undirected>

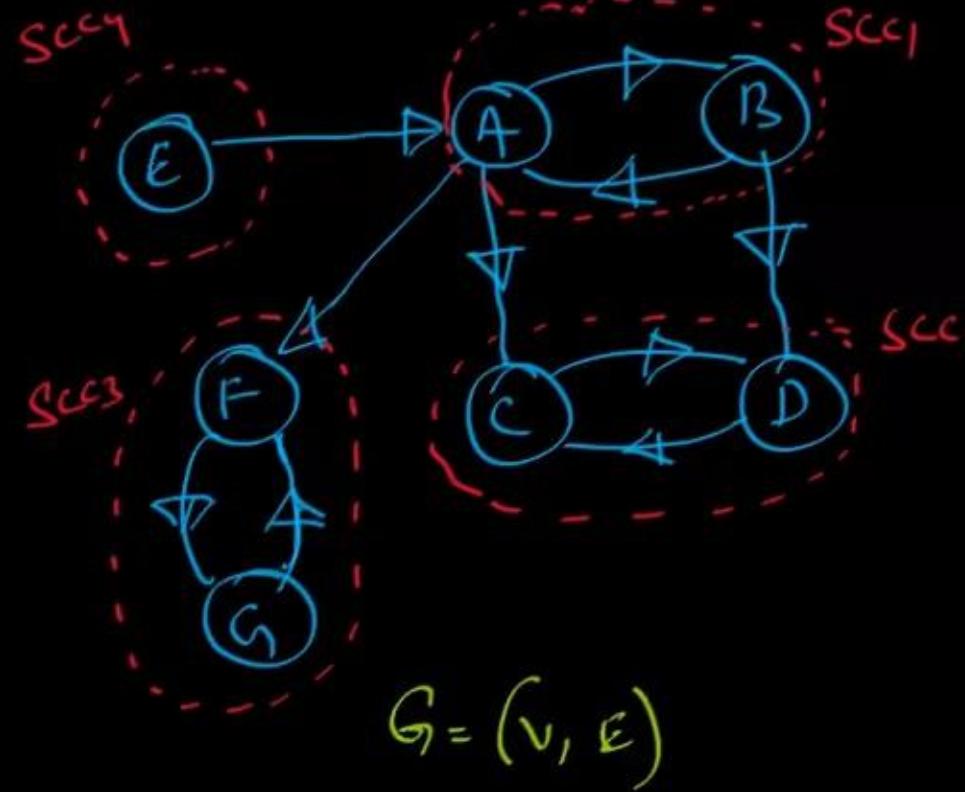


No. of Con. Components = (No. of  $\frac{\text{DFS}}{\text{BFS}}$  Sp. Traces)



## 2) Strongly Connected Components (SCC) ; <directed graph>

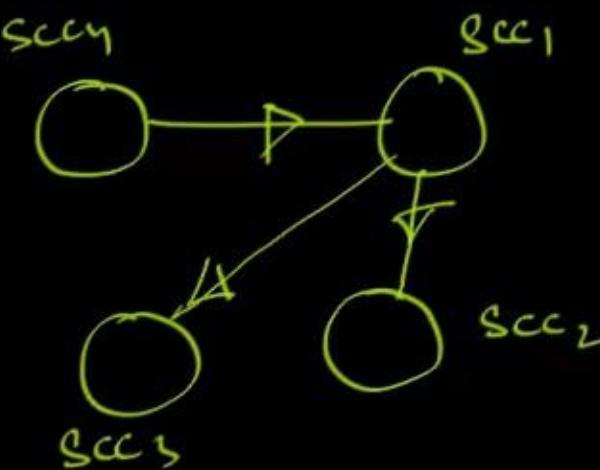
This relation partitions the vertex set into disjoint (maximal) sets known as strongly connected components ;

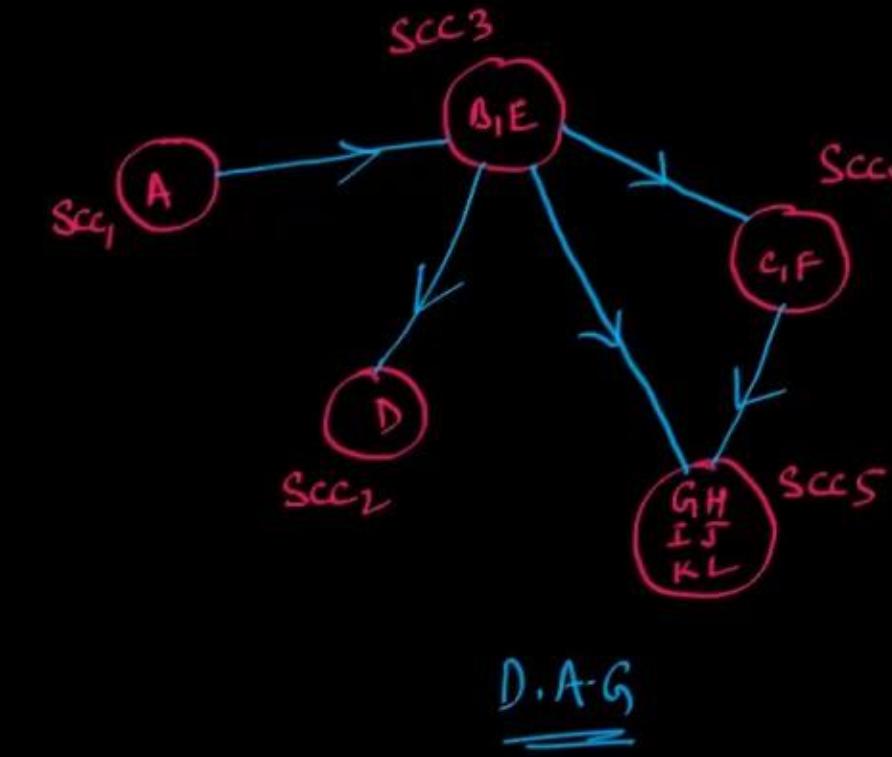
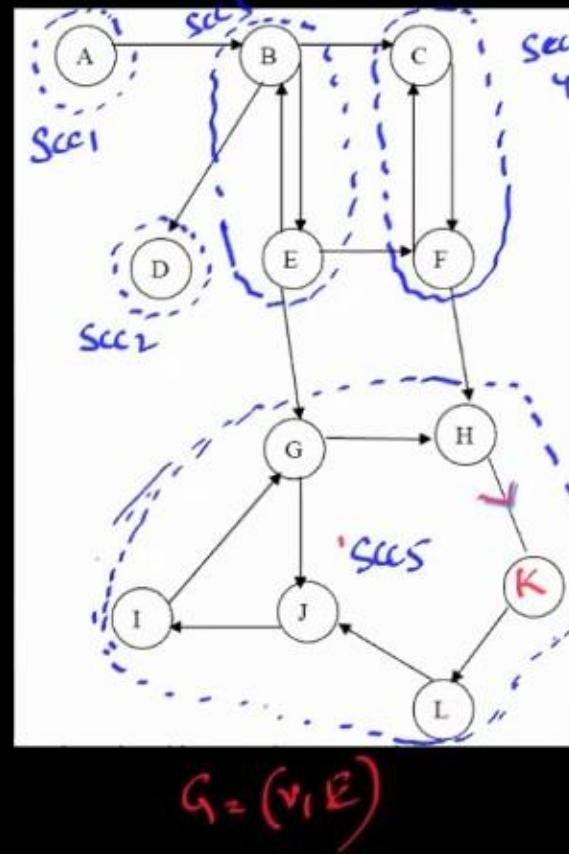


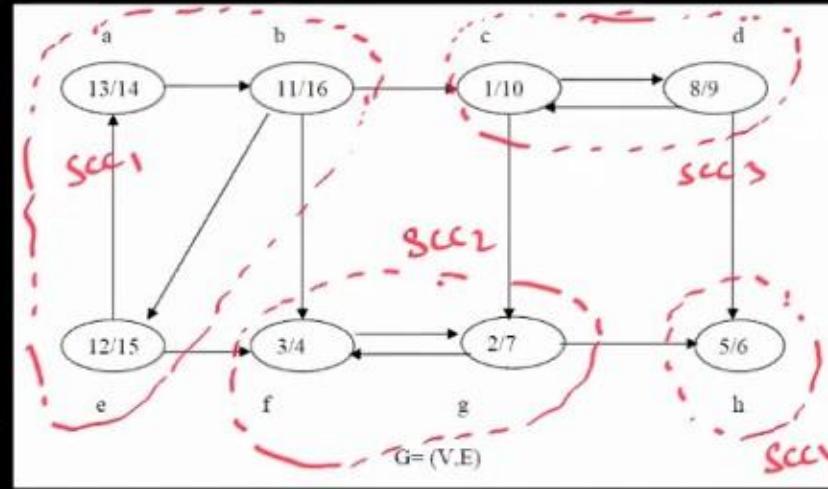
Two vertices 'u' & 'v' of a di-graph are connected, iff there is a path from 'u' to 'v' and a path from 'v' to 'u';

$$V = \{A, B, C, D, E, F, G\}$$

$$\left\{ \underline{\{A, B\}}, \underline{\{C, D\}}, \underline{\{F, G\}}, \underline{\{E\}} \right\}$$







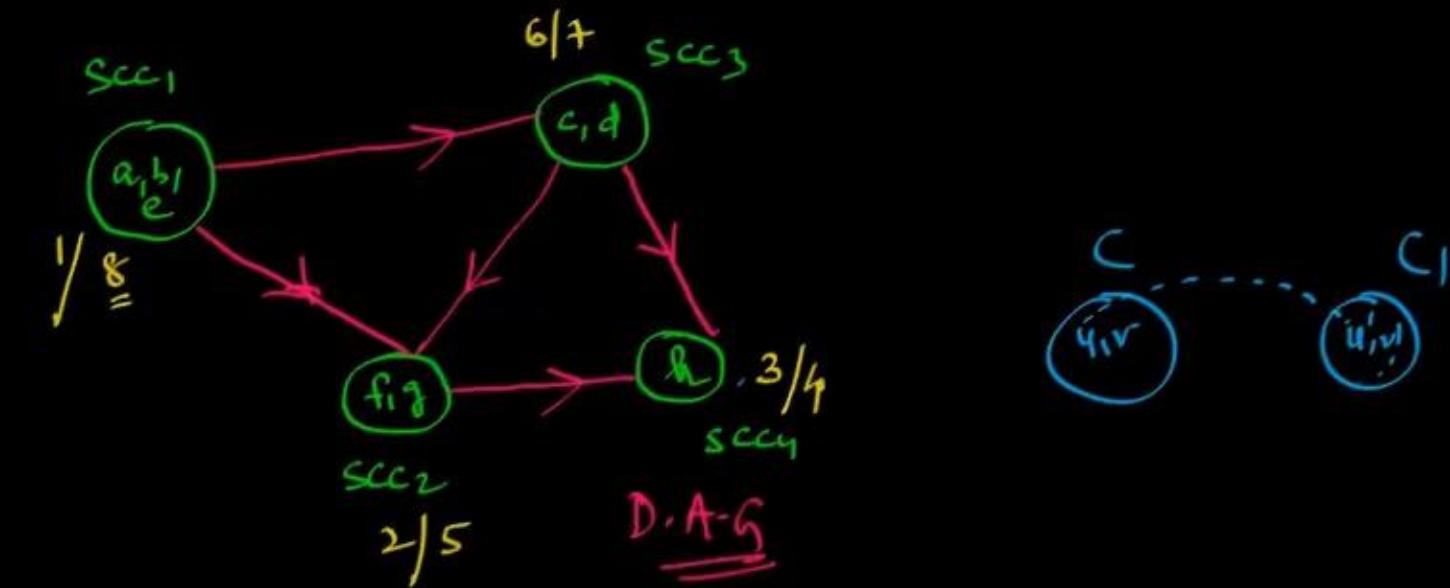
### Properties of S.C.C:

→ Every Directed graph is a D.A.G of its Strongly connected Components;

→ Let  $C$  &  $C'$  be two distinct S.C.C, if there a Path from a vertex in  $C$  to a vertex in  $C'$  then

$$P(C) > f(C')$$

$f$ : finishing time.



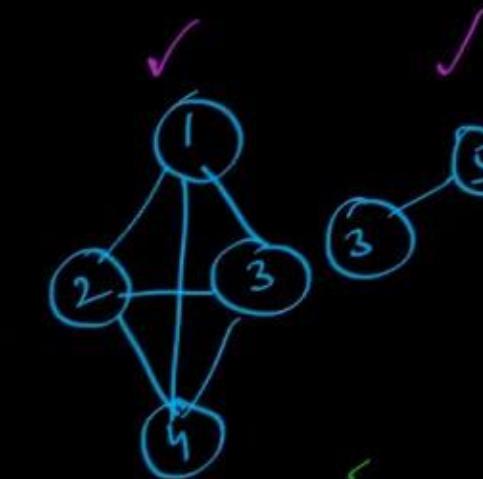
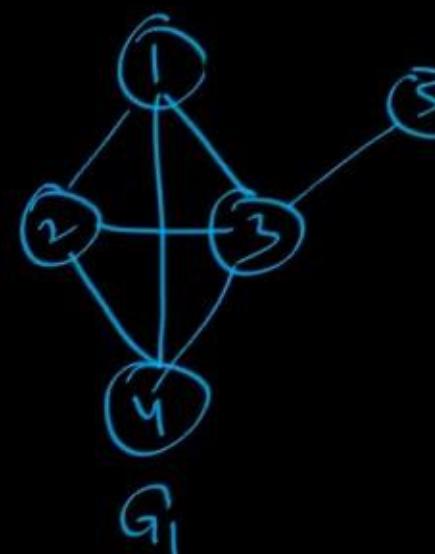
→ Let  $C$  &  $C'$  be two distinct Strongly Connected Components, let  $(u, v) \in C$  &  $(u', v') \in C'$ , then if there is a path from  $u$  to  $u'$  in ' $G$ ' then there cannot be a path from  $v'$  to  $v$  in ' $G$ '

## Bi Connected Components ; Articulation Points ; (Bridge edge)

Articulation point / cut vertex : is that vertex in the graph, the removal of which partitions the graph into 2 or more non-empty components;

BiConnected Graph! Graph is said to be biconnected, iff it does not have any articulation point.

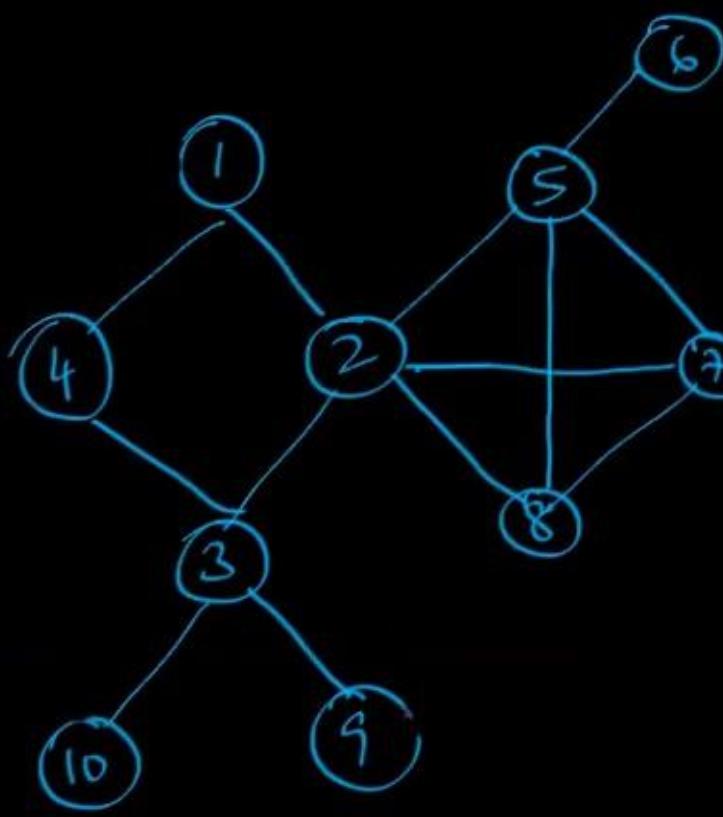
BiConnected Component : Maximal Subgraph that is biconnected is known as Biconnected Component.



No. of entrw Edges  
≤ No. of A Pts

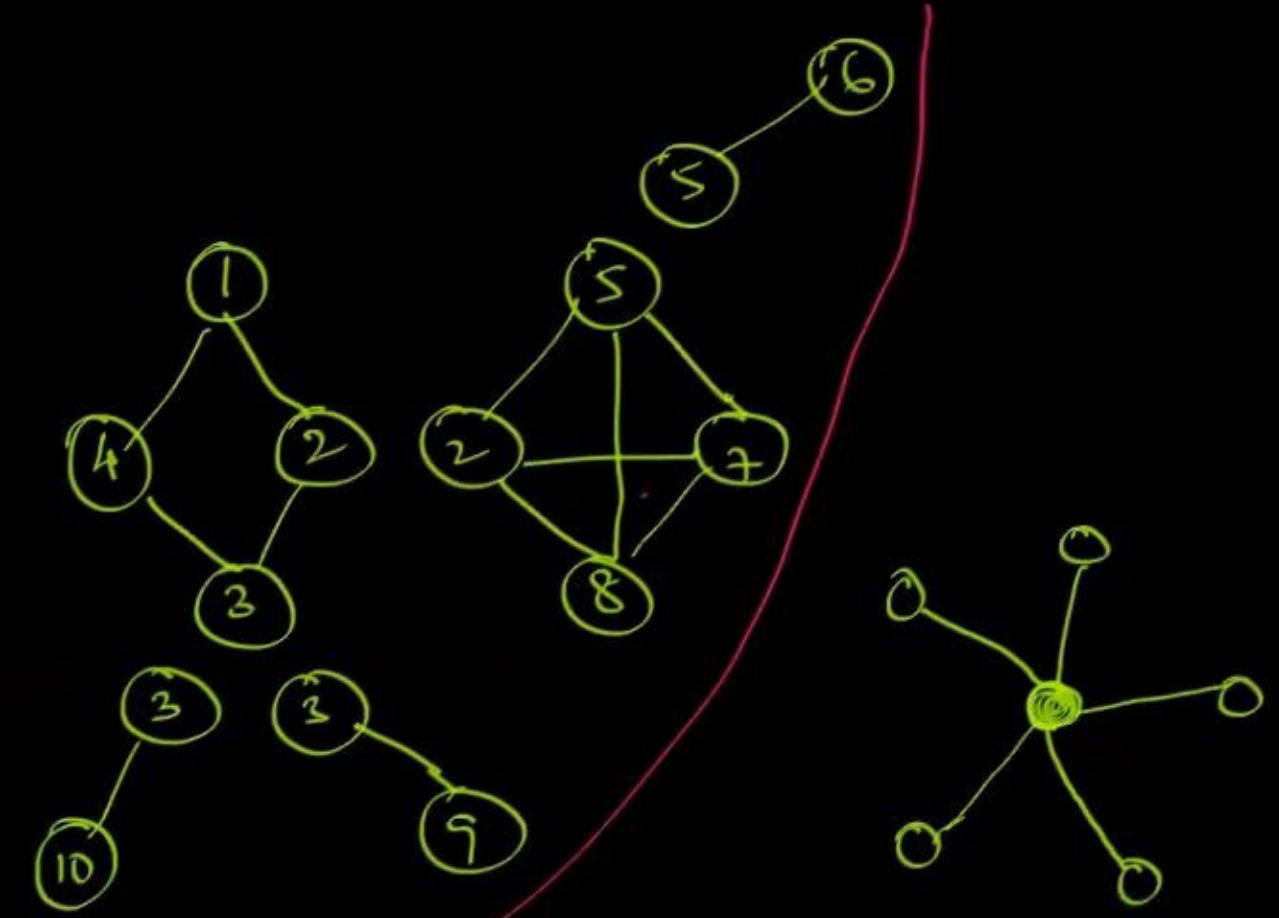
How to make a  
non-biconnected  
graph, BiConnected

↓  
(Adding Entrw  
edges)

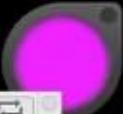


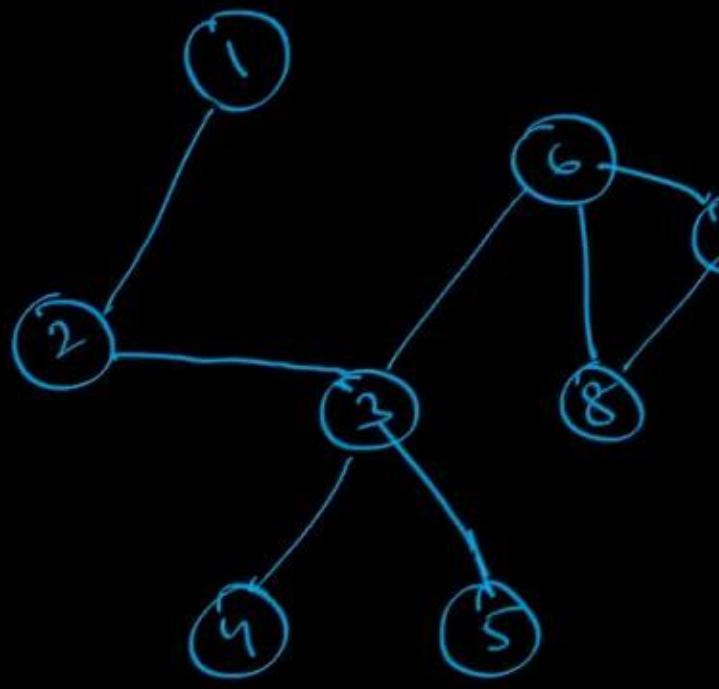
G<sub>1</sub>  
2 edges

✓ 3 edges



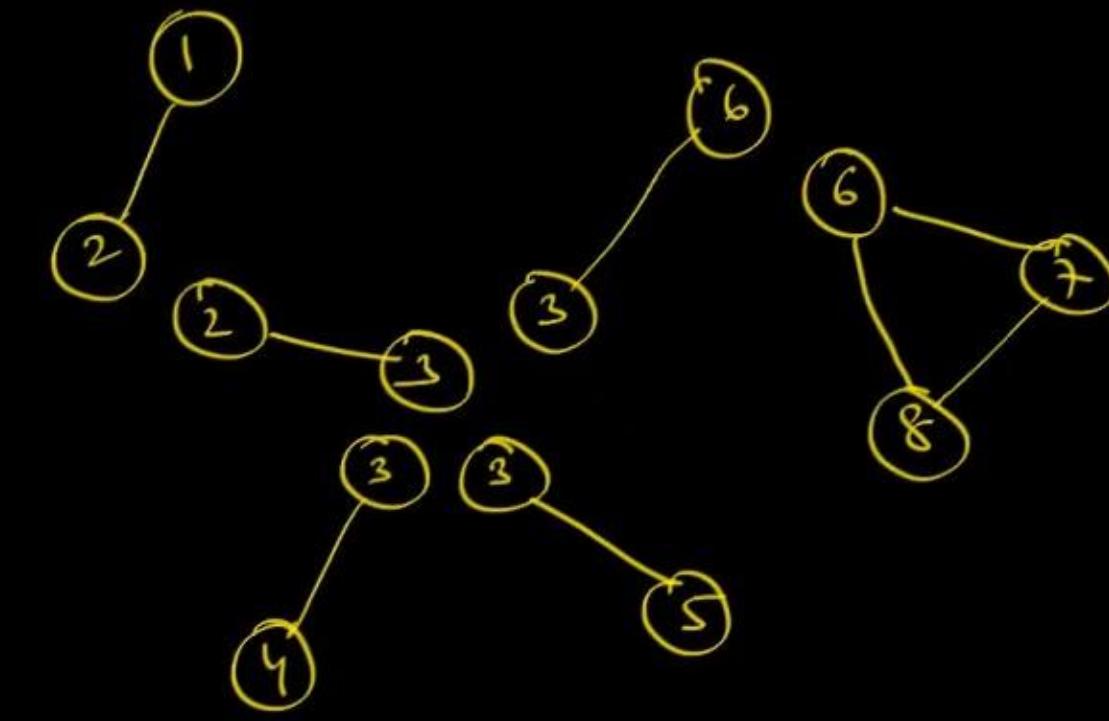
(Start-Graph) G<sub>4</sub>





63

(N Possible) {2-edge}

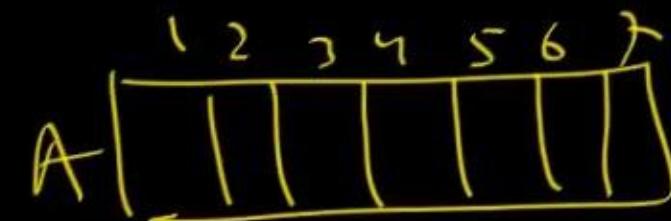
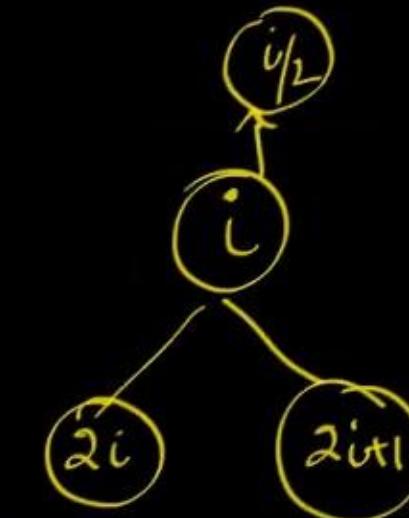
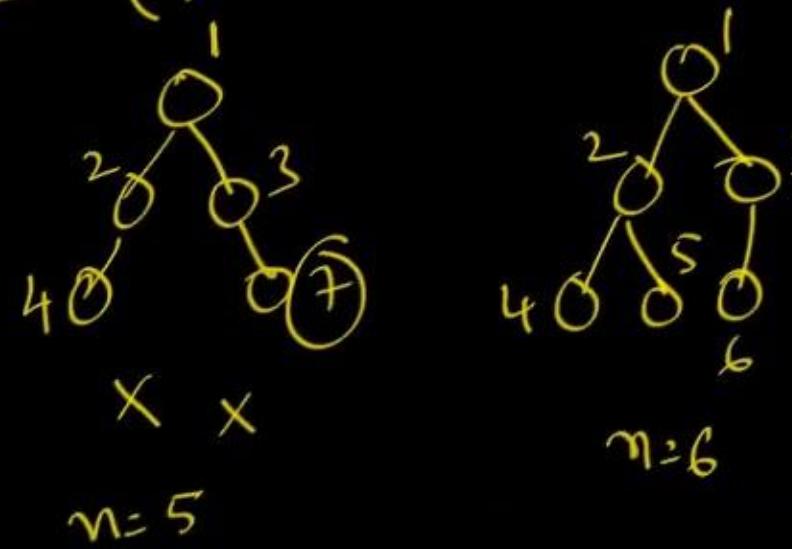


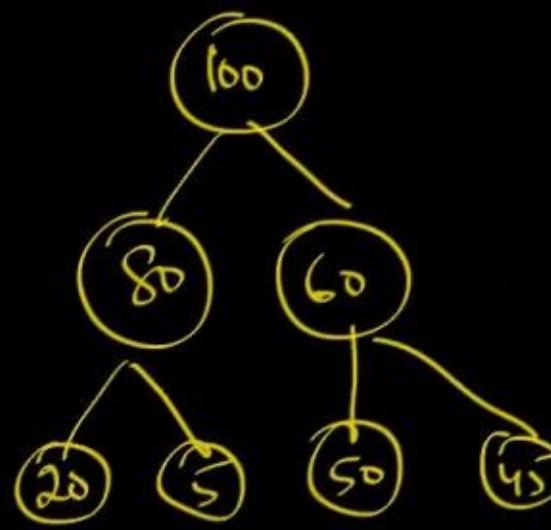
a) 4  
b) 5  
c) 6

Heaps : (Priority Queue)

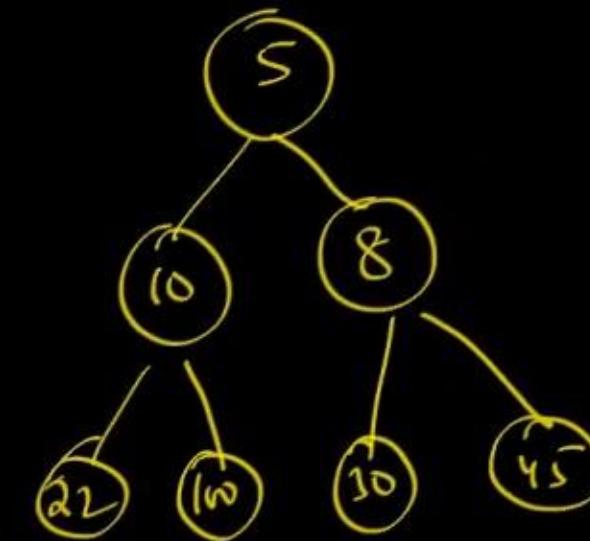
: is a C-B-T with the property that the value at each node is greater than (smaller than) the values of its children in both Left & right subtrees ; (Max/Min)

C-B-T : (if the level order indices all lies in the range of  $1 \dots n$ )





Max-Heap



(Min-Heap)

Heap operations :

→ Create   
(Insertion Method)  
Build-Heap  
(Heapify)

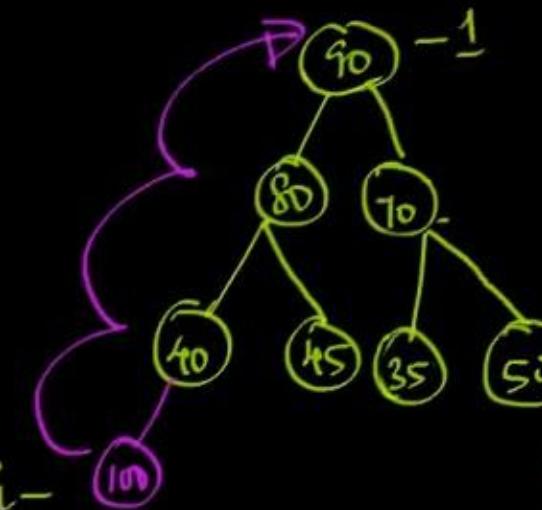
Insertion Method:  $\langle 40, 80, 35, 90, 45, 50, 70 \rangle$  Construct a Heap (max) out of given  $n$ -elements

by inserting one element at a time starting from an empty tree;

(i) Time Complexity:

a) Best Case: ~~Inc/ASC~~:  $O(n)$

b) Worst Case: ~~Inc/ASC~~:  $O(n \log n)$



→ Max. No. of Nodes @ level ' $i$ ' of a B.T.:  $2^{i-1}$

→ The No. of level moves/comps for any node :  $(i-1)$  being inserted @ level ' $i$ '

→ Total level moves/comps for all nodes :  $(i-1) \cdot 2^{i-1}$

$$\rightarrow \text{Total Time} = \sum_{i=1}^k (i-1) \cdot 2^{i-1} = \frac{1}{2} \left[ \sum_{i=1}^k i \cdot 2^i - \sum_{i=1}^k 2^i \right]$$

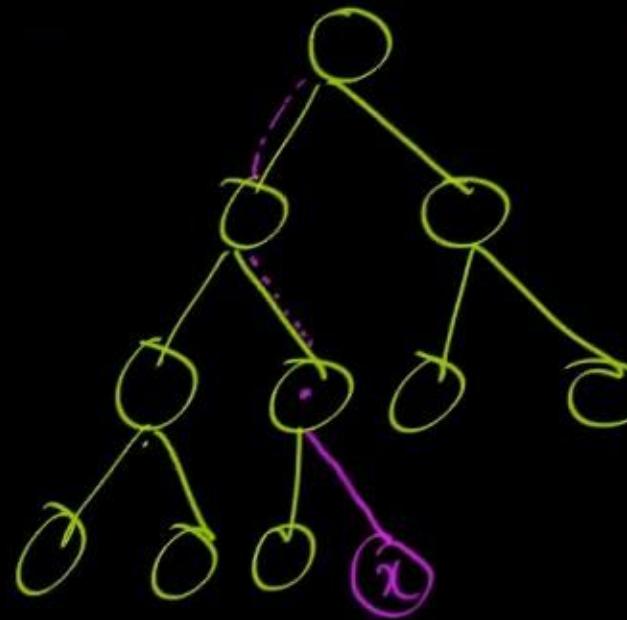
$$\begin{aligned} &= \frac{1}{2} \left[ (k-1) \cdot 2^{k+1} + 2 - 2^{k+1} + 2 \right] = k \cdot 2^{k+1} - 2^{k+1} + 2 \\ &= (n \log n - 2n + 2) = O(n \log n) \end{aligned}$$

$$\sum_{i=1}^n i \cdot 2^i = (n-1) \cdot 2^{n+1} + 2$$

$$\sum_{i=1}^n 2^i = 2^{n+1} - 2$$

$$\frac{n \cdot 2^k}{k = \log n}$$

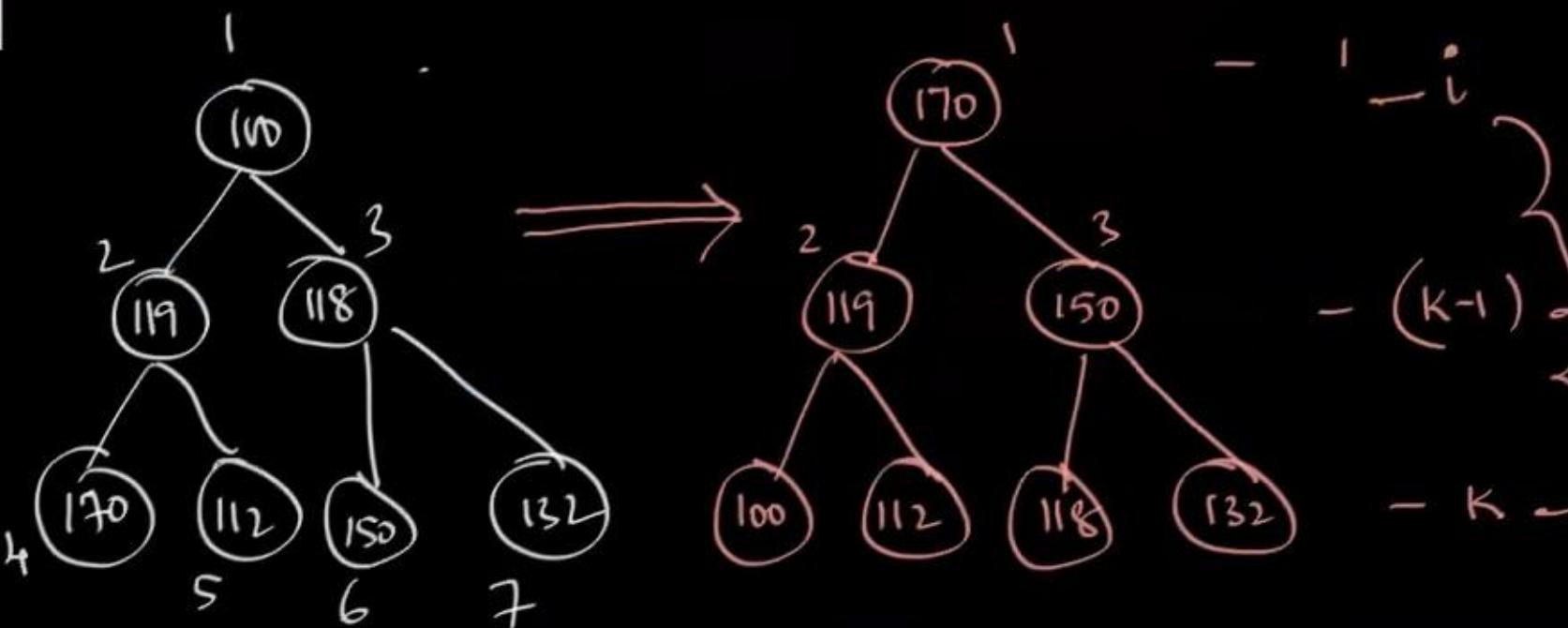
2) Insert-operation: Given a Heap of  $n$ -elements, it is required to Insert an element ' $x$ '



Best-Case:  $O(1)$

Worst-Case:  $\tilde{O}(\log n)$

Heapify Method : Build Heap:  
with Adjust  $\rightarrow O(n)$



|   | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
|---|-----|-----|-----|-----|-----|-----|-----|
| A | 100 | 119 | 118 | 170 | 112 | 150 | 132 |
| A | 170 | 119 | 150 | 100 | 112 | 118 | 132 |

→ Max. no. of nodes at any level 'i' =  $2^{i-1}$

→ Max. no. of level moves/comps for any node being adjusted at level 'i' =  $(K-i)$

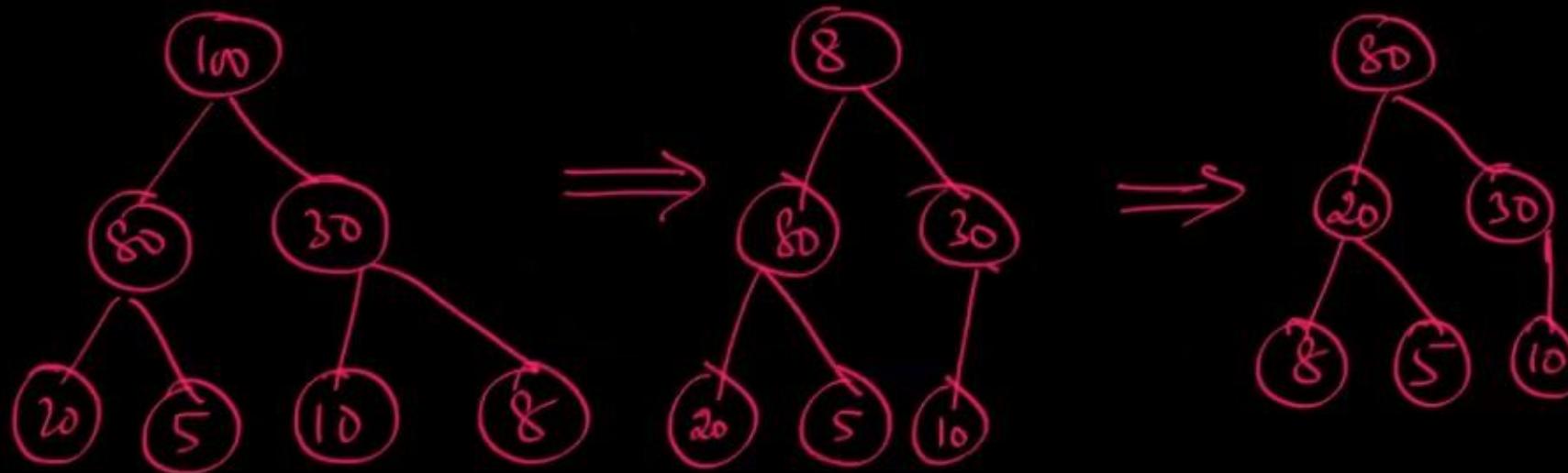
→ Total comps for all nodes at level 'i' =  $(K-i) \cdot 2^{i-1}$

$$\begin{aligned}
 \text{Time} &= \sum_{i=1}^K (K-i) \cdot 2^{i-1} = \frac{1}{2} \left[ \sum_{i=1}^K K \cdot 2^i - \sum_{i=1}^K i \cdot 2^i \right] \\
 &= \frac{1}{2} \left[ K \cdot (2^{K+1} - 2) - [(K-1) \cdot 2^K + 2] \right] \\
 &= \frac{1}{2} \left[ \cancel{K/2}^{K+1} - 2K - \cancel{K/2}^{K+1} + 2^{K+1} - 2 \right] \\
 &= \frac{2^{K+1} - 2K - 2}{2} = 2^K - K - 1 = n - \log n - 1 \\
 &\quad = O(n)
 \end{aligned}$$

$$n = 2^K$$

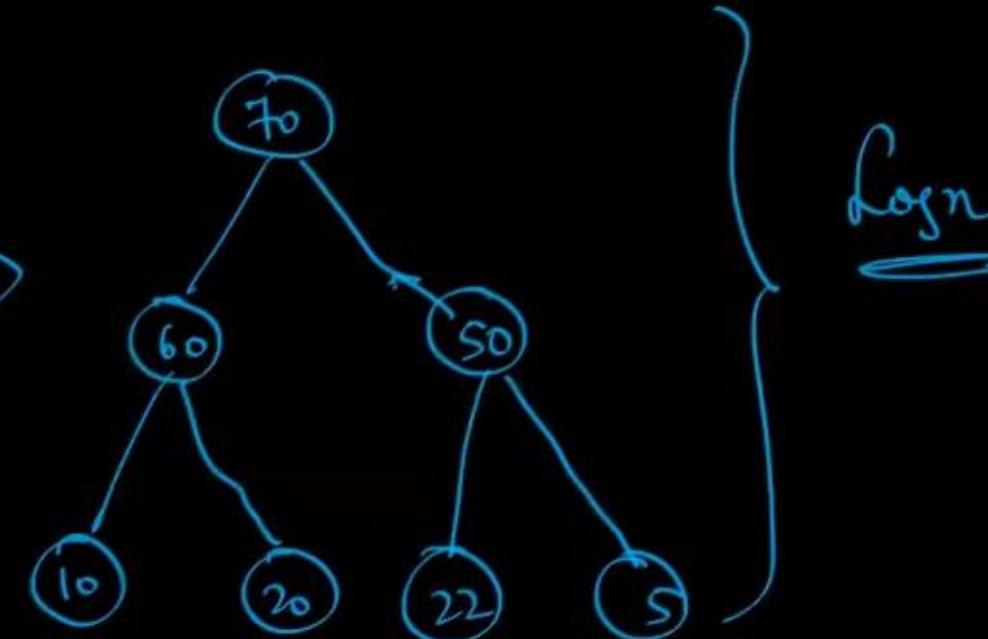
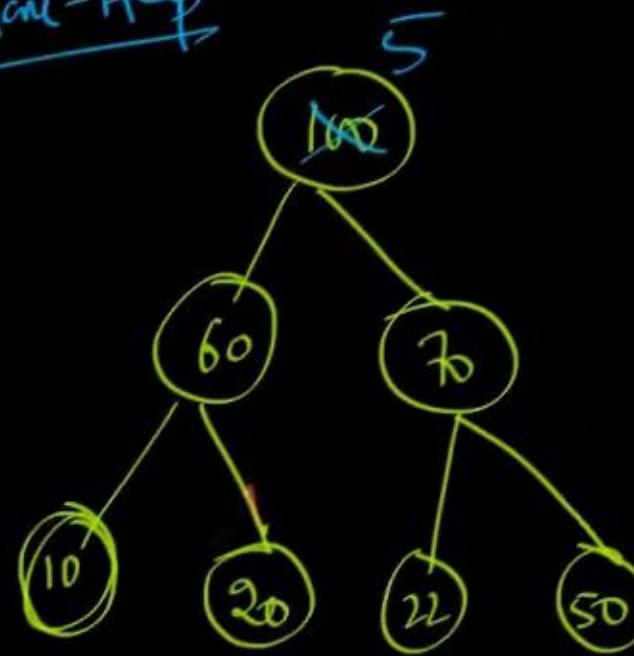
$$K = \log n$$

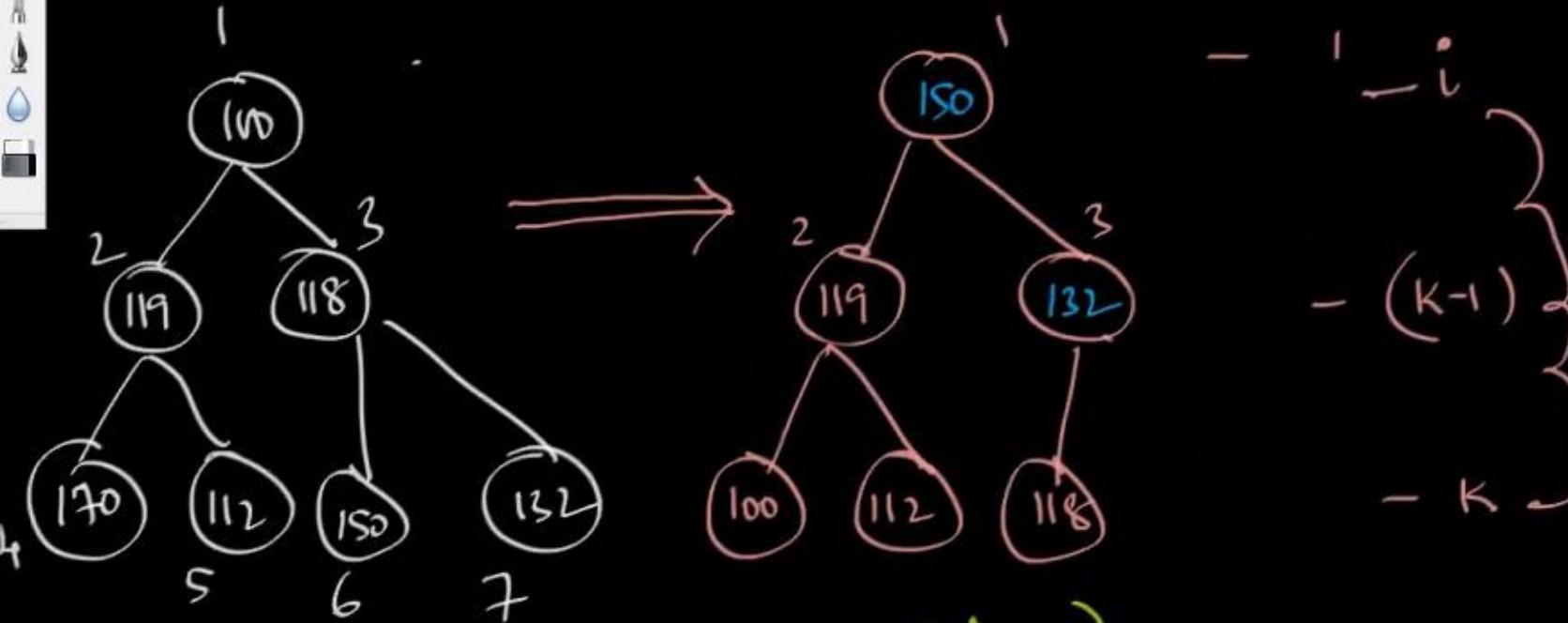
Delete operation: (Deletion is of root) : Time:  $\underline{\mathcal{O}(\log n)}$



Dec-Key:

Max-Heap



Heapsort

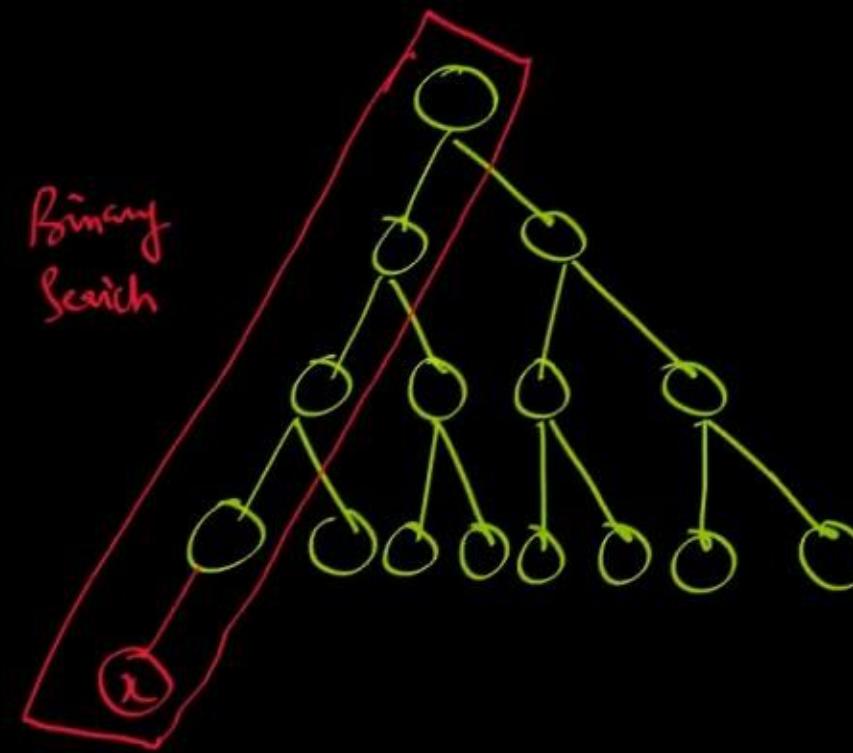
Algo Hs( $A, n$ ) :  $O(n \log n)$

1. Heapsify( $A, n$ ) ;  $\rightarrow O(n)$
2. for  $i \leftarrow n$  down to 2 by -1
 

$\{$   
 SWAP( $A[i], A[1]$ );  
 Adjust( $A[1]$ );  $\log n$

$\}$

|     | 1   | 2   | 3   | 4   | 5   | 6   | 7   |
|-----|-----|-----|-----|-----|-----|-----|-----|
| $A$ | 100 | 119 | 118 | 170 | 112 | 150 | 132 |
| $A$ | 170 | 119 | 150 | 100 | 112 | 118 | 132 |
|     | 132 | 119 | 150 | 100 | 112 | 118 | 170 |
|     | 150 | 119 | 132 | 100 | 112 | 118 | 170 |
|     | 118 | 159 | 132 | 100 | 112 | 150 | 170 |



$$\underline{BS} = \underline{\mathcal{O}(\log n)}$$

$$h = \underline{\log n}$$

$$\underline{\mathcal{O}(\log \log n)} \checkmark$$

Approx. No. of Elements that can be sorted in  $O(\log n)$  using H.S is —

- a)  $O(1)$
- b)  $O(n)$
- c)  $O(\log n)$
- d)  $O\left(\frac{\log n}{\log \log n}\right)$  ✓

Time :  $O(n \log n)$   $\Rightarrow$  'n' elements

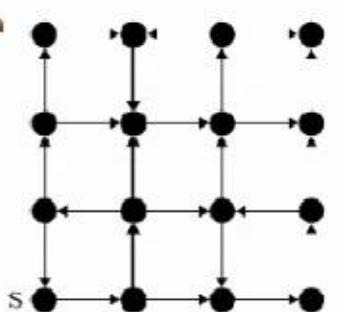
$\cancel{O(\log n \cdot \log \log n)}$

$$\frac{\log n}{\log \log n} * \log\left(\frac{\log n}{\log \log n}\right) = \frac{\log n}{\log \log n} * \cancel{\log \log n} - \frac{\log n}{\log \log n} * \cancel{\log \log n}$$

$$= O(\log n) \quad = \cancel{\log n} - \left( \frac{\log n}{\log \log n} * \cancel{\log \log n} \right)$$

## V. Graph Techniques & Components

1. A DFS is performed on DAG. Which of the following is true for all edges  $(u, v)$  in the graph?
  - (a)  $d[u] < d[v]$
  - (b)  $d[u] < f[v]$
  - (c)  $f[u] < f[v]$
  - (d)  $f[u] > f[v]$
  
2. Consider a DFT of an undirected graph having ' $n$ ' vertices. In the traversal,  $k$  edges are marked as Tree edges then the number of connected components in the graph is given by
  - (a)  $k$
  - (b)  $k + 1$
  - (c)  $n - k$
  - (d)  $n - k - 1$
  
3. Consider the following directed graph



Which of the following is/are correct about the graph?

- (a) The graph does not have a strongly connected component.
- (b) A depth-first traversal starting at vertex S classifies three directed edges as back edges.
- (c) For each pair of vertices  $u$  and  $v$ , there is a directed path from  $u$  to  $v$ .
- (d) The graph does not have a topological order.

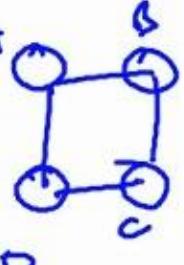
Student



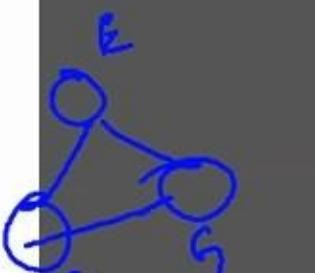
$$k = n - 1$$

$$n - (n - 1)$$

$$\dots = 1$$



3



2 = 5

$$n = 7$$

$$k = 5$$

Export PDF

Adobe ExportPDF Convert PDF files to Word or Excel online.

Select PDF File:

Algorithms Handout 2021.pdf 1 file / 537 KB

Convert To:

Microsoft Word (\*.docx)

Recognize Text in English(U.S.) Change

Convert

Create PDF

Edit PDF

Combine PDF

Send Files

Store Files

Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

22 / 31 100% | Student

## V. Graph Techniques & Components

1. A DFS is performed on DAG. Which of the following is true for all edges  $(u, v)$  in the graph?

(a)  $d[u] < d[v]$       (b)  $d[u] < f[v]$   
(c)  $f[u] < f[v]$       (d)  $f[u] > f[v]$

2. Consider a DFT of an undirected graph having 'n' vertices. In the traversal, k edges are marked as Tree edges then the number of connected components in the graph is given by

(a) k      (b)  $k + 1$   
(c)  $n - k$       (d)  $n - k - 1$

3. Consider the following directed graph

MSQ

There is no source vertex

Which of the following is/are correct about the graph?

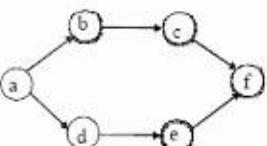
(a) The graph does not have a strongly connected component. X  
(b) A depth-first traversal starting at vertex S classifies three directed edges as back edges. ✓  
(c) For each pair of vertices  $u$  and  $v$ , there is a directed path from  $u$  to  $v$ . X  
(d) The graph does not have a topological order. ✓

ACE Engineering Academy → Hyderabad | Delhi | Bhopal | Pune | Bhubaneswar | Lucknow | Patna | Bengaluru | Chennai | Vijayawada | Visag | Tirupati | Kukatpally | Kolkata

Export PDF  
Selected PDF File: Algorithms Handout 2021.pdf  
Convert To: Microsoft Word (\*.docx)  
Recognize Text in English(U.S.) Change  
Convert  
Create PDF  
Edit PDF  
Combine PDF  
Send Files  
Store Files



4. Consider the following directed graph:

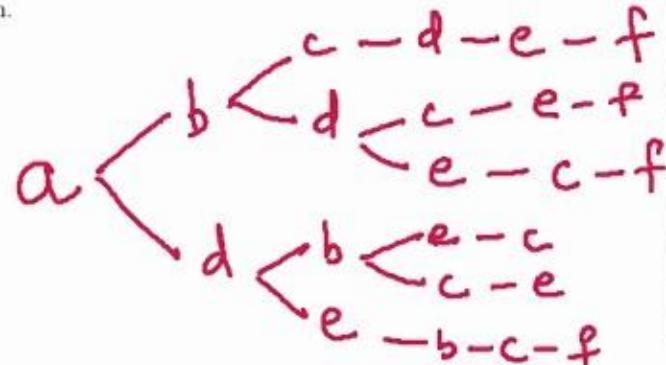


The number of different topological orderings of the vertices of the graph is 6.

5. An articulation point in a connected graph is a vertex such that removing the vertex and its incident edges disconnects the graph into two or more connected components.

Let T be a DFS tree obtained by doing DFS in a connected undirected graph G. Which of the following options is/are correct?

- (a) Root of T can never be an articulation point in G
- (b) If u is an articulation point in G such that x is an ancestor of u in T and y is a descendant of u in T, then all paths from x to y in G must pass through u.
- (c) A leaf of T can be an articulation point in G
- (d) Root of T is an articulation point in G if and only if it has 2 or more children.



For Micro Notes by the Student



▼ Export PDF

Adobe ExportPDF  
Convert PDF files to Word or Excel online.

Select PDF File:

Algorithms Handout 2021.pdf

1 file / 537 KB

Convert To:

Microsoft Word (\*.docx)

Recognize Text in English(U.S.)  
Change

Convert

► Create PDF

► Edit PDF

► Combine PDF

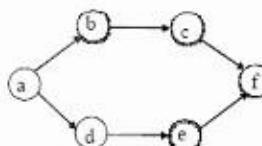
► Send Files

► Store Files

a) 8  
b) 6  
c) 9



4. Consider the following directed graph:

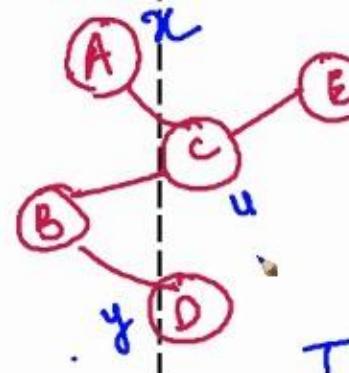
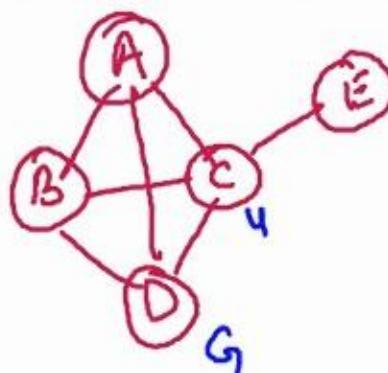


The number of different topological orderings of the vertices of the graph is \_\_\_\_\_.

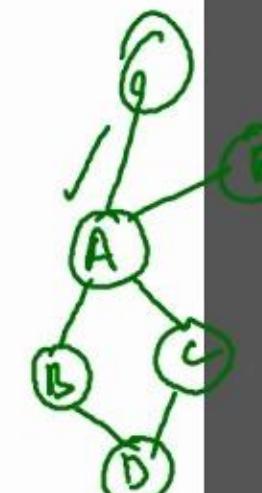
5. An articulation point in a connected graph is a vertex such that removing the vertex and its incident edges disconnects the graph into two or more connected components.

Let  $T$  be a DFS tree obtained by doing DFS in a connected undirected graph  $G$ . Which of the following options is/are correct?

- (a) Root of  $T$  can never be an articulation point in  $G$ .
- (b) If  $u$  is an articulation point in  $G$  such that  $x$  is an ancestor of  $u$  in  $T$  and  $y$  is a descendant of  $u$  in  $T$ , then all paths from  $x$  to  $y$  in  $G$  must pass through  $u$ .
- (c) A leaf of  $T$  can be an articulation point in  $G$ .
- (d) Root of  $T$  is an articulation point in  $G$  if and only if it has 2 or more children.



For Micro Notes by the  
Student



Root



DFS Sp. Tree

▼ Export PDF

Adobe ExportPDF  
Convert PDF files to Word or Excel  
online.

Select PDF File:

Algorithms Handout 2021.pdf  
1 file / 537 KB

Convert To:

Microsoft Word (\*.docx)

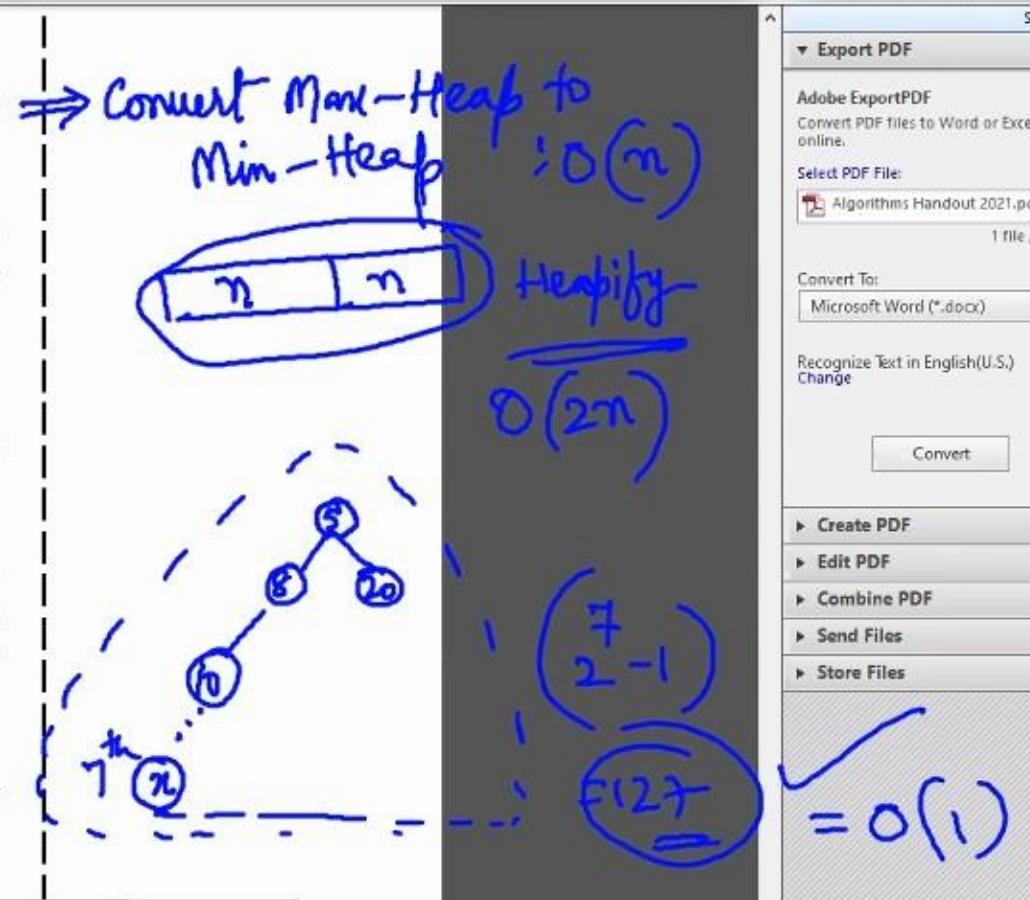
Recognize Text in English(U.S.)  
Change

Convert

- ▶ Create PDF
- ▶ Edit PDF
- ▶ Combine PDF
- ▶ Send Files
- ▶ Store Files

Insert: <1 & 7>; What is the resultant Level order Traversal?

5. In a binary max-Heap with  $n$  elements, the smallest element can be found in time of  $O(n)$
6. Given binary Heap with ' $n$ ' elements & it is required to insert ' $n$ ' more elements not necessarily one after another into this Heap. Total time required for this operation is:  
 (a)  $O(n^2)$       (b)  $n \log n$       ✓(c)  $n$       (d)  $n^2 \log n$
7. Given binary Heap in Array with the smallest at the root, the 7<sup>th</sup> smallest element can be found in time complexity of  $O(1)$  ✓
8. Consider binary Heap in an Array with  $n$  elements it is desired to insert an element into the Heap if a binary search is performed along the path from newly inserted element to the root then the no. of comparison made is order of \_\_\_\_\_.
9. The approximate no. of elements that can be Sorted in  $O(n \log n)$  time using Heap Sort is \_\_\_\_\_.



Recording

Open | File | Edit | View | Insert | Tools | Fill & Sign | Comment | Sign In

Export PDF  
Selected PDF File: Algorithms Handout 2021.pdf  
Convert To: Microsoft Word (\*.docx)  
Recognize Text in English(U.S.) Change  
Convert

Insert: <1 & 7>; What is the resultant Level order Traversal?

5. In a binary max-Heap with n elements, the smallest element can be found in time of \_\_\_\_\_.

6. Given binary Heap with 'n' elements & it is required to insert 'n' more elements not necessarily one after another into this Heap. Total time required for this operation is:  
(a)  $O(n^2)$       (b)  $n \log n$       (c)  $n$       (d)  $n^2 \log n$

7. Given binary Heap in Array with the smallest at the root, the 7<sup>th</sup> smallest element can be found in time complexity of \_\_\_\_\_.

8. Consider binary Heap in an Array with n elements it is desired to insert an element into the Heap if a binary search is performed along the path from newly inserted element to the root then the no. of comparison made is order of  $O(\log \log n)$

9. The approximate no. of elements that can be Sorted in  $O(\log n)$  time using Heap Sort is \_\_\_\_\_.

ACE Engineering Academy → Hyderabad | Delhi | Bhopal | Pune | Bhubaneswar | Lucknow | Patna | Bengaluru | Chennai | Vijayawada | Vizag | Tirupati | Kukatpally | Kolkata



## Sorting Methods :

### Classification :

→ all elements are to be present in Memory;  
 I. Internal vs External → Need Not be  
 ↳ Inversion Sort

II. Comparison vs Non Comparison based:  
 ↳ Radix Sort

III. Recursive vs Iterative

IV. In place vs Not in place  
 ↳ Merge Sort  
 ↳ No extra Mem.

Spacer  
 Recursive  
 Impl:  $O(\log n)$   
 Otherwise  $O(1)$

V. Stable vs Unstable



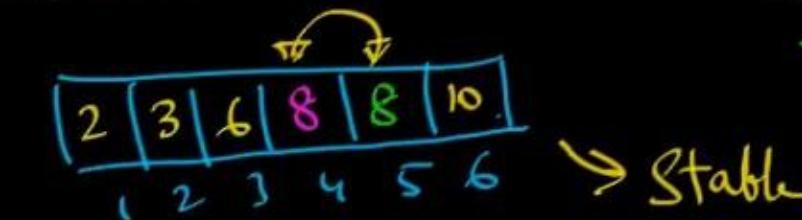
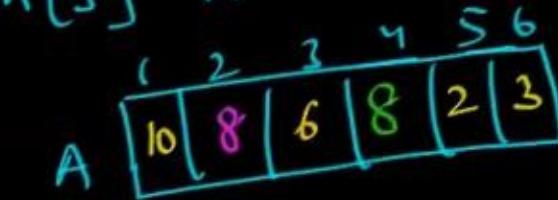
Stable Sort:

Let  $A[1..n]$  be an array of elements;

Let  $i, j$  be indices into the array ( $i \neq j$ )

If  $(A[i] = A[j])$  and if  $(A[i] \text{ precedes } A[j] \text{ in Presorted list})$ , then the sorting method is said to be stable, iff  $A[i] \text{ precedes } A[j] \text{ in the final sorted list as well.}$

Relative of non-distinct elements  
Should be maintained in  
the sorted list



$$i=2, j=4$$

$$A[2] = A[4]$$

$\overbrace{\hspace{10em}}$   
 $A[2]$  occurs  
before  $A[4]$

→ Stable

## Parameters that Influence Sorting Time (for comparison based sorting)

Time =  $f(\text{No. of Element Comparisons}, \text{Swaps(Exchanges)})$

Time = Complexity =  $\Theta(\text{No. of Comps}) + \Theta(\text{No. of Swaps})$

No. of Swaps  $\propto \left\{ \begin{array}{l} \text{No. of Inversions} \\ \text{No. of Comparisons} \end{array} \right\}$

Let  $A[1..n]$  be an array of  $n$  elements  
 $i, j$  - be indices into Array A;

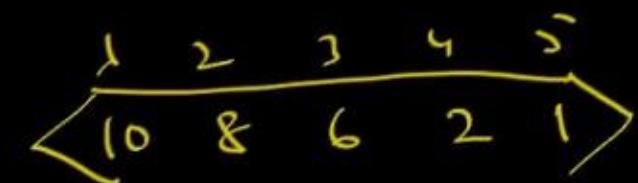
if  $(i < j)$  and  $(A[i] > A[j])$  then the pair  $\langle i, j \rangle$  is an inversion of 'A';

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| A | 1 | 2 | 3 | 4 | 5 |
|   | 9 | 4 | 7 | 8 | 6 |

$\left\{ \begin{array}{l} \langle 1,2 \rangle \langle 1,3 \rangle \langle 1,4 \rangle \langle 1,5 \rangle \\ \langle 3,5 \rangle \langle 4,5 \rangle \end{array} \right\}$

measure of IP size

- Max no. of Inversions:  $\lceil \text{Dcr/ Dsc order} \rceil = \frac{n(n-1)}{2} = O(n^2)$
- Min no. of Inversions: '0'  $\lceil \text{Inc/ Asc order} \rceil$
- Avg. " " " :  $\frac{n(n-1)}{4}$



**Adaptability of Sorting Technique:** The complexity of the sorting method changes based on pre-sorted input.  
Re-sorted Input affects the running time.

$$\begin{aligned}
 & 1, 2; 1, 3, 14, 1, 5 = 4 \\
 & 2, 3; 24, 25 \leftarrow 3 \\
 & 3, 4; 35 = 2 \\
 & 4, 5 = 1
 \end{aligned}$$

# I. Bubble Sort :

Algorithm BS( $A, n$ )

```

    for pass ← n down to 1
    {
        flag = 0;
        for i ← 1 to (pass-1)
        {
            if ( $A[i] > A[i+1]$ )
                swap( $A[i], A[i+1]$ );
            flag = 1;
        }
        if (flag = 0) exit;
    }
    Space : O(1)
  
```

Pass I:

|        | 1  | 2  | 3  | 4  | 5  | 6  |
|--------|----|----|----|----|----|----|
| $i=1:$ | 80 | 60 | 20 | 15 | 40 | 10 |
| $i=2:$ | 60 | 80 | 20 | 15 | 40 | 10 |
| $i=3:$ | 60 | 20 | 15 | 80 | 40 | 10 |
| $i=4:$ | 60 | 20 | 15 | 40 | 80 | 10 |
| $i=5:$ | 60 | 20 | 15 | 40 | 10 | 80 |

Pass II:

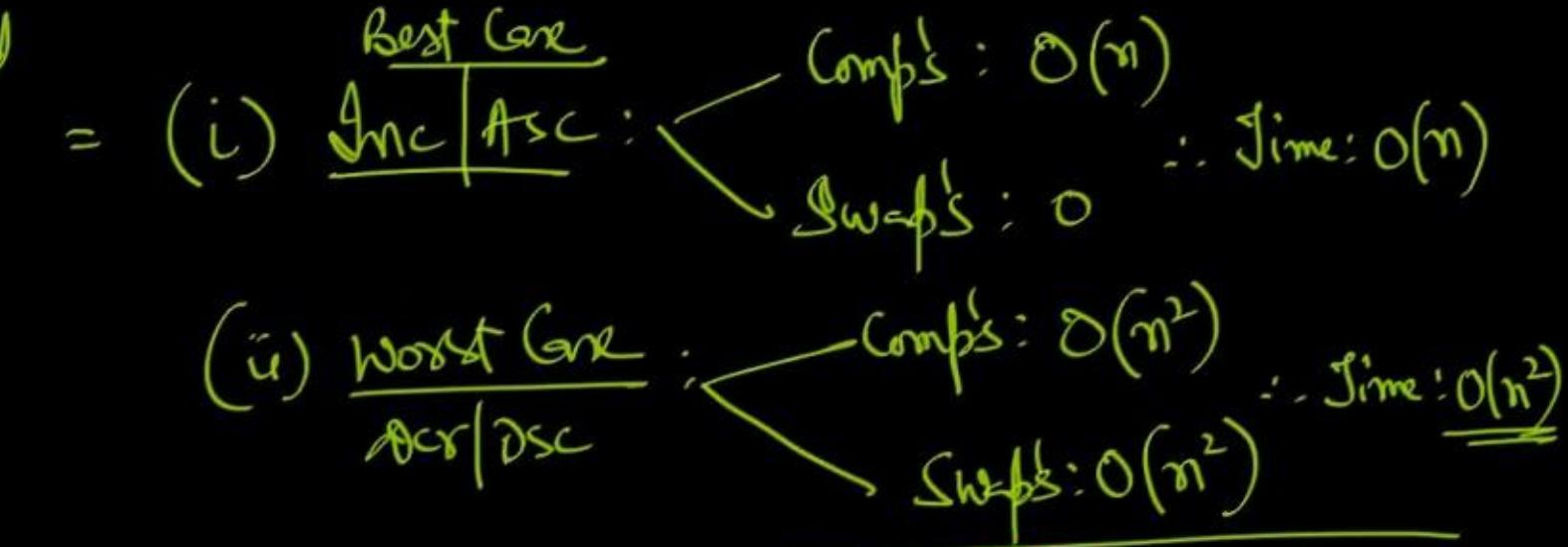
Perf:

- No. of Comps :  $\frac{n(n-1)}{2} = O(n^2)$
- No. of swaps : 0 ; Ascending [always]

$$\frac{n(n-1)}{2} = O(n^2)$$

overall time =  $O(n^2)$

Time Complexity of Revised  
Impl. of B-S



If  $T(n)$  repr. Time Complexity of B-S, then W-C

$T(n)$  as a recurrence:

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

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

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

## II. SELECTION SORT :

Algorithm SS( $A, n$ )

```

for i ← 1 to (n-1)
{
    min ← i;
    for j ← (i+1) to n
    {
        if (A[j] < A[min])
            min ← j;
    }
    swap(A[min], A[i]);
}
  
```

(Requires least  
No. of swaps:  $O(n)$   
in worst case)  
also

|          | 1  | 2  | 3  | 4  | 5  | 6  |
|----------|----|----|----|----|----|----|
| A:       | 80 | 60 | 20 | 15 | 40 | 5  |
| min ← 1: | 5  | 60 | 20 | 15 | 40 | 80 |
| i:       | 5  | 15 | 20 | 60 | 40 | 80 |
| j:       | 4  | 4  | 4  | 4  | 4  | 4  |
| min ← 2: | 5  | 15 | 20 | 60 | 40 | 80 |
| min ← 3: | 5  | 15 | 20 | 60 | 40 | 80 |
| min ← 4: | 5  | 15 | 20 | 60 | 40 | 80 |
| min ← 5: | 5  | 15 | 20 | 60 | 40 | 80 |
| min ← 6: | 5  | 15 | 20 | 60 | 40 | 80 |

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

Performance:

→ No. of Comb's:  $\frac{n(n-1)}{2} = O(n^2)$

\* No. of swaps:  $O(n)$  is always



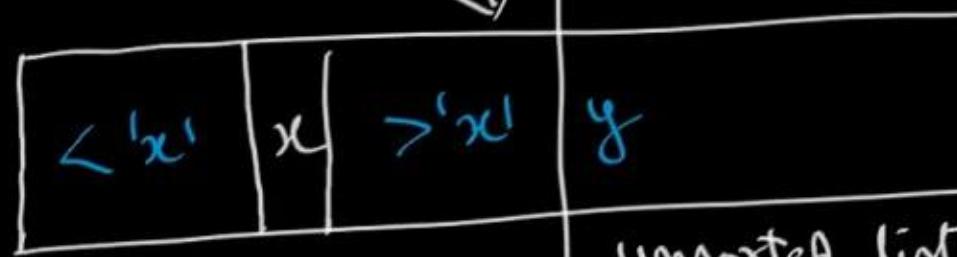
### 3) Insertion Sort:

: Insert an element from unsorted list into correct place in sorted list

Partial Sorted list



↓ Pass



Partial Sorted list

unsorted list

A:

1      2      3      4      5      6  
8      2      4      9      3      5

unsorted list

P.S.L

A: 8 | 2      4      9      3      5

: 2    8 | 4      9      3      5

2    4    8 | 9

2    4    8    9 | 3      5

2    3    4    8    9 | 5

INSERTION-SORT( $A$ )

```

1  for  $j \leftarrow 2$  to  $\text{length}[A]$ 
2    do  $key \leftarrow A[j]$ 
3       $\triangleright$  Insert  $A[j]$  into the sorted
        sequence  $A[1..j-1]$ .
4       $i \leftarrow j-1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6        do  $A[i+1] \leftarrow A[i]$ 
7           $i \leftarrow i-1$ 
8       $A[i+1] \leftarrow key$ 

```

$i$        $j-1$

Performance of IS :

(i) Best Case:  $\text{ASC} : \text{No. of Inv's} : '0'$   
 $\therefore \text{Time} : O(n)$

(ii) Worst Case:  $\text{ASC} | \text{DCR} : \text{No. of Inv's} : O(n^2)$   
 $\text{No. of swaps} : O(n^2)$   
 $\therefore \text{Time} : O(n^2)$

Insertion Sort  $\sim$  No. of Inversions  
 $\text{Time} : O(n+d)$        $d = \text{No. of Inversions}$

RADIX-SORT :
 $A: \frac{1}{101}, \frac{2}{12}, \frac{3}{18}, \frac{4}{22}, \frac{5}{99}, \frac{6}{654}, \frac{7}{736}, \frac{8}{520}, \frac{9}{824}, \frac{10}{33}, \frac{11}{88}$ 

$\rightarrow \underline{\text{base}} = 10$   $A: 520, 101, 12, 22, 33, 654, 824, 736, 18, 88, 99$

$A: 101, 12, 18, 520, 22, 824, 33, 736, 654, 88, 99$

$A: 12, 18, 22, 33, 88, 99, 101, 520, 654, 736, 824$

Time Complexity:

$O(d(n+b))$

$O(dn + \underline{db}) \sim O(dn)$

$d = \text{Max No. of digits}$

$b = \underline{\text{base}}$

Limitations:

(-ve Nos)

Drawback:  
Space Req's:

Pass 3 :

99

Pass 2 :

88

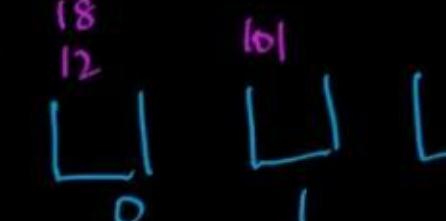
33

22

Pass 1 :

18

12



100

$O(n \cdot \log_b K)$

# Recursion Tree Method for Solving DandC Recurrences : (Asymmetric)

1) Symmetric Recurrences:

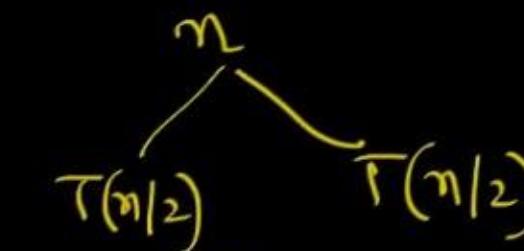
$$T(n) = 2 \cdot T(n/2) + n$$

$$\overline{T(n/2)} = \overline{2T(n/4)} + \overline{n/2}$$

$$a+b$$

$$a \quad b$$

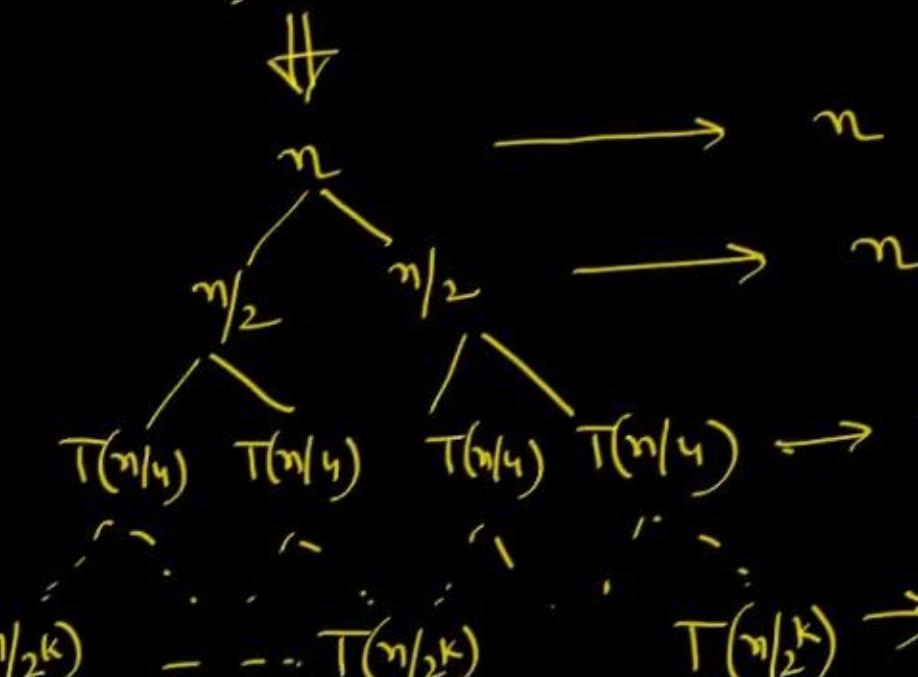
$$\begin{aligned} T(n) &= n^{(k+1)} \\ &= n(\log n + 1) \\ &= O(n \log n) \end{aligned}$$



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

$$n = 2^k$$

$$k = \log n$$



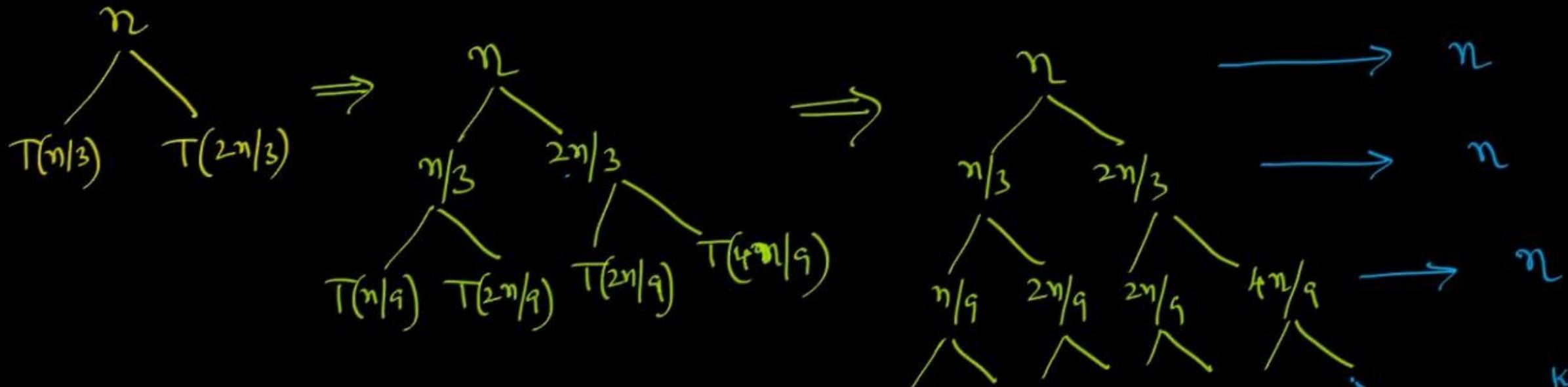
$$\begin{aligned} T(n/2^k) &\rightarrow \dots \rightarrow T(n/2^k) \\ T(n) &= n^{(k+1)} \end{aligned}$$

## 2) Asymmetric Case :

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

$$T(n/3) = T(n/9) + T(2n/9) + n/3 ; \quad T(2n/3) = T(2n/9) + T(4n/9) + 2n/3$$

$$T(n) = T(\alpha n) + T((1-\alpha)n) + n$$



$$\left(\frac{2}{3}\right)^k \cdot n = 1$$

$$\Rightarrow n = \left(\frac{3}{2}\right)^k$$

$$k = \log_{\frac{3}{2}} n$$

$$= n^{\log_{\frac{3}{2}} 2}$$

$$T(n) = n(k+1) = n(\log n + 1)$$

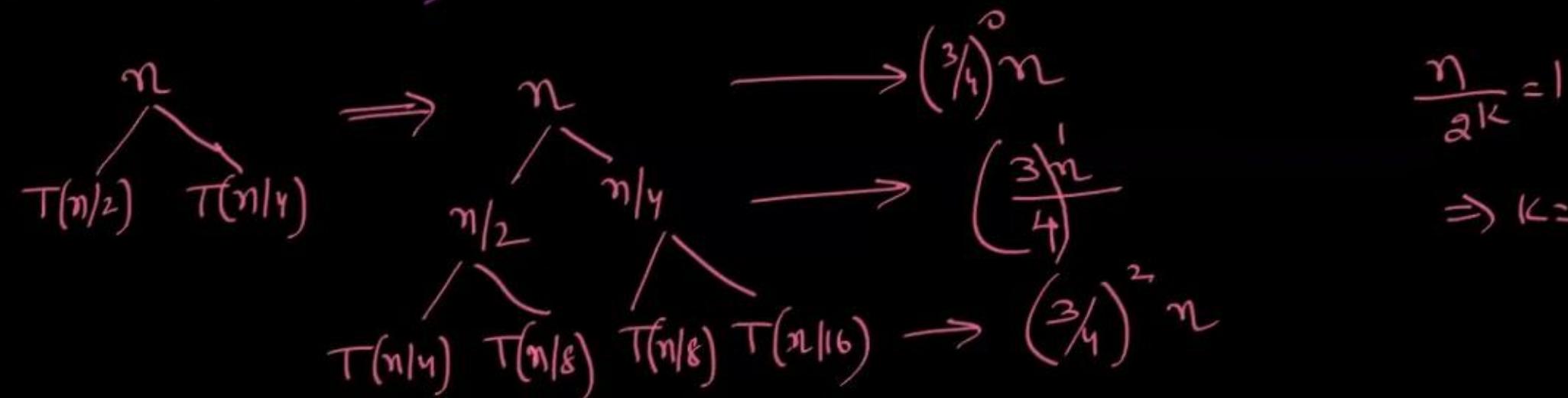
$$= \Theta(n \log n)$$



$$3) T(n) = 3T(n/4) + n^2 \quad \left| \begin{array}{l} \frac{n}{4^K} = 1 \\ \Rightarrow n = 4^K \\ \Rightarrow K = \log_4 n \end{array} \right.$$

$$\begin{aligned}
 T(n) &= n^2 \sum_{i=0}^{K-1} \left(\frac{3}{16}\right)^i + 3^K \cdot T(1) \\
 &= n^2 \cdot " + 3^{\log_4 n} \\
 &= n^2 \cdot ( ) + n^{\log_4 3} \\
 &\leq n^2 \cdot \left(\frac{1}{1 - 3/16}\right) + n^{\log_4 3} \leq cn^2 + cn \\
 &= O(n^2) \\
 \rightarrow \left(\frac{3}{16}\right)^2 n^2 & \\
 \rightarrow \left(\frac{3}{16}\right)^{K-1} \cdot n^2 & \\
 \therefore T(n/4^K) &\rightarrow 3 \cdot T(n/4^K) \\
 T(n) &
 \end{aligned}$$

$$\text{P) } T(n) = T(n/2) + T(n/4) + n$$



$$\begin{aligned}
 T\left(\frac{n}{2^k}\right) & - - - - - \rightarrow 2^k \cdot T\left(\frac{n}{2^k}\right) \\
 T(n) &= n \cdot \underbrace{\sum_{i=0}^{k-1} \left(\frac{3}{4}\right)^i}_{\text{geometric series}} + 2^k \cdot T(1) \\
 &= c \cdot n + n \cdot b = O(n)
 \end{aligned}$$

A:  $1 \left| \begin{matrix} 2 \\ 5 \end{matrix} \right| 8 \quad 10 \quad 12 \quad 15$

$$(n-1) \cdot 1 = O(n)$$

Agenda:

- 1) L.I.M (D and C)
- 2) Back Tracking vs Branch Bound

+  
Summary Discussion



## VII. Sorting Methods

1. Which of the following Sorting algorithms has lowest worst-case complexity?  
i. Bubble Sort      ii. Merge Sort ✓  
iii. Quick Sort      iv. Selection Sort

2. Which of the following in place Sorting algorithm needs minimum number of swaps?  
(a) Selection Sort ✓      (b) Insertion Sort  
(c) Heap Sort      (d) Quick Sort

3. What would be the worst case complexity of Insertion Sort if the inputs are restricted to permutation of 1 to n with at most ' $n$ ' Inversions? :  $O(n)$

4. Let 'S' be a Sorted Array of 'n' integers and  $T(n)$  denote the time taken for the most efficient algorithm to determine if there are 2 elements in the Array with the sum  $< 1000$ .  
(a)  $T(n)$  is  $O(1)$       (b)  $n \leq T(n) \leq n \log n$   
(c)  $T(n) =^n C_2$       (d)  $n \log n \leq T(n) =^n C_2$

For Micro Notes by the Student



:  $(n + \frac{d}{2})$

4. Let 'S' be a Sorted Array of 'n' integers and  $T(n)$  denote the time taken for the most efficient algorithm to determine if there are 2 elements in the Array with the sum  $< 1000$ .

- ✓(a)  $T(n)$  is  $O(1)$       (b)  $n \leq T(n) \leq n \log n$   
(c)  $T(n) =^n C_2$       (d)  $n \log n \leq T(n) =^n C_2$

5. The traditional Insertion Sort to Sort an Array can be of  $n$  elements uses linear search to identify the position where an element is to be inserted into already Sorted part of the Array, if instead binary search is used to identify the position of newly inserted element then the worst case complexity will be order of \_\_\_\_\_.

6. In using Quick Sort suppose the central element of the Array is always chosen as the Pivot then the worst case complexity of the Quick Sort may be \_\_\_\_\_.

7. The Median on Array of size  $n$  can be found in  $O(n)$  time. If Median is selected as Pivot, then the worst case complexity of Quick Sort is \_\_\_\_\_.

8. In applying Quick Sort to an unsorted list if  $(n/4)$  the element is selected as Pivot then the Time Complexity of Quick Sort will be \_\_\_\_\_.

A [ ] : inc  
if ( $A[1] + A[2] < (1000)$ ) yes  
else - if ( $A[n] + A[n-1] < (1000)$ ) yes  
else no

4. Let 'S' be a Sorted Array of 'n' integers and  $T(n)$  denote the time taken for the most efficient algorithm to determine if there are 2 elements in the Array with the sum  $< 1000$ .

- (a)  $T(n)$  is  $O(1)$       (b)  $n \leq T(n) \leq n \log n$   
(c)  $T(n) =^n C_1$       (d)  $n \log n \leq T(n) =^n C_2$

5. The traditional Insertion Sort to Sort an Array can be of  $n$  elements uses linear search to identify the position where an element is to be inserted into already Sorted part of the Array, if instead binary search is used to identify the position of newly inserted element then the worst case complexity will be order of \_\_\_\_\_.

6. In using Quick Sort suppose the central element of the Array is always chosen as the Pivot then the worst case complexity of the Quick Sort may be \_\_\_\_\_.

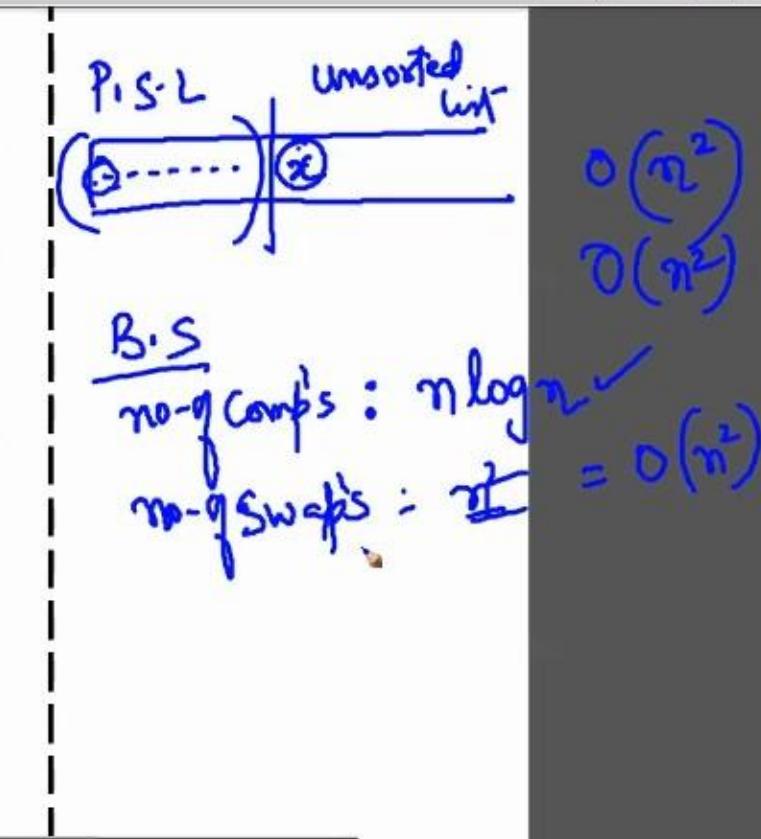
7. The Median on Array of size  $n$  can be found in  $O(n)$  time. If Median is selected as Pivot, then the worst case complexity of Quick Sort is \_\_\_\_\_.

8. In applying Quick Sort to an unsorted list if  $(n/4)$  the element is selected as Pivot then the Time Complexity of Quick Sort will be \_\_\_\_\_.

(i) 2 It's of Sel Sort  
 $: O(n) + O(n) = O(n)$

(ii) a) Build-Heap:  $O(n)$   
b) delete:  $O(\log n)$   
c) delete:  $O(\log n)$   
 $O(n)$

5. The traditional Insertion Sort to Sort an Array can be of  $n$  elements uses linear search to identify the position where an element is to be inserted into already Sorted part of the Array, if instead binary search is used to identify the position of newly inserted element then the worst case complexity will be order of \_\_\_\_\_.
6. In using Quick Sort suppose the central element of the Array is always chosen as the Pivot then the worst case complexity of the Quick Sort may be \_\_\_\_\_.
7. The Median on Array of size  $n$  can be found in  $O(n)$  time. If Median is selected as Pivot, then the worst case complexity of Quick Sort is \_\_\_\_\_.
8. In applying Quick Sort to an unsorted list if  $(n/4)$  the element is selected as Pivot then the Time Complexity of Quick Sort will be \_\_\_\_\_.



5. The traditional Insertion Sort to Sort an Array can be of  $n$  elements uses linear search to identify the position where an element is to be inserted into already Sorted part of the Array, if instead binary search is used to identify the position of newly inserted element then the worst case complexity will be order of \_\_\_\_\_.
6. In using Quick Sort suppose the central element of the Array is always chosen as the Pivot then the worst case complexity of the Quick Sort may be  $O(n^2)$  ✓
7. The Median on Array of size  $n$  can be found in  $O(n)$  time. If Median is selected as Pivot, then the worst case complexity of Quick Sort is \_\_\_\_\_.
8. In applying Quick Sort to an unsorted list if  $(n/4)$  the element is selected as Pivot then the Time Complexity of Quick Sort will be \_\_\_\_\_.



$$T(n) = O(n) + O(n) + 2T(n/2)$$

---

$$O(n \log n)$$

$$T(n) = O(n) + O(n) + T(n/4) + T(3n/4)$$
$$= O(n \log n)$$

9. Consider the Quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two sublists each of which contains at least one-fifth of the elements. Let  $T(n)$  be the number of comparisons required to sort  $n$  elements. Then

- (a)  $T(n) \leq 2T(n/5)+n$       (b)  $T(n) \leq T(n/5)+T(4n/5)+n$  ✓  
(c)  $T(n) \leq 2T(4n/5)+n$       (d)  $T(n) \leq 2T(n/2)+n$

For Micro Notes by the Student



10. Which one of the following in place sorting algorithms needs the minimum number of swaps?

- (a) Quick sort      (b) Insertion sort  
(c) Selection sort ✓      (d) Heap sort

11. The worst case running times of Insertion sort, Merge sort and Quick sort, respectively, are:

$n^2$        $n \log n$        $n^2$

- (a)  $\Theta(n \log n)$ ,  $\Theta(n \log n)$ , and  $\Theta(n^2)$   
(b)  $\Theta(n^2)$ ,  $\Theta(n^2)$ , and  $\Theta(n \log n)$   
(c)  $\Theta(n^2)$ ,  $\Theta(n \log n)$ , and  $\Theta(n \log n)$   
(d)  $\Theta(n^2)$ ,  $\Theta(n \log n)$ , and  $\Theta(n^2)$  ✓

12. If one uses straight two-way merge sort algorithm to sort the following elements in ascending order:

20, 47, 15, 8, 9, 4, 40, 30, 12, 17

then the order of these elements after second pass of the algorithm is:

- (a) 8, 9, 15, 20, 47, 4, 12, 17, 30, 40  
(b) 8, 15, 20, 47, 4, 9, 30, 40, 12, 17

13. You have  $n$  lists, each consisting of  $m$  integers sorted in ascending order.

Merging these lists into a single sorted list will take time:

- (a)  $O(nm \log m)$
- (b)  $O(mn \log m)$
- (c)  $O(m + n)$
- (d)  $O(mn)$



14. If we use Radix Sort to sort  $n$  integers in the range  $(n^{k/2}, n^k]$ , for some  $k > 0$  which is independent of  $n$ , the time taken would be

- (a)  $\Theta(n)$
- (b)  $\Theta(kn)$
- (c)  $\Theta(n \log n)$
- (d)  $\Theta(n^2)$





$n^k$

$\log_b k$

$k \cdot \log n$

For Micro Notes by the  
Student



$O(d(n+b))$

$O(dn)$

$O(n \cdot \log n)$

14. If we use Radix Sort to sort  $n$  integers in the range  $(n^{k/2}, n^k]$ , for some  $k > 0$  which is independent of  $n$ , the time taken would be
- (a)  $\Theta(n)$       (b)  $\Theta(kn)$   
 (c)  $\Theta(n \log n)$       (d)  $\Theta(n^2)$
15. The worst case running times of Insertion sort, Merge sort and Quick sort, respectively are:
- (a)  $\Theta(n \log n)$ ,  $\Theta(n \log n)$  and  $\Theta(n^2)$   
(b)  $\Theta(n^2)$ ,  $\Theta(n^2)$  and  $\Theta(n \log n)$   
(c)  $\Theta(n^2)$ ,  $\Theta(n \log n)$  and  $\Theta(n \log n)$   
(d)  $\Theta(n^2)$ ,  $\Theta(n \log n)$  and  $\Theta(n^2)$
16. Let  $P$  be a quick sort program to sort numbers in ascending order.  
Let  $t_1$  and  $t_2$  be the time taken by the program for the inputs  $[1\ 2\ 3\ 4]$  and  $[5\ 4\ 3\ 2\ 1]$ , respectively. Which of the following holds?
- (a)  $t_1 = t_2$       (b)  $t_1 > t$       (c)  $t_1 < t_2$       (d)  $t_1 = t_2 = 5 \log 5$
17. Let  $P$  be a Quick Sort Program to sort numbers in ascending order suing the first element as pivot. Let  $t_1$  and  $t_2$  be the number of comparisons made by

## Engineering Academy

19. Following algorithm(s) can be used to sort n in the range  $[1..n^3]$  in  $O(n)$  time
- (a) Heap sort
  - (b) Quick sort
  - (c) Merge sort
  - (d) Radix sort
20. For merging two sorted lists of sizes m and n into a sorted list of size  $m+n$ , we require comparisons of
- (a)  $O(m)$
  - (b)  $O(n)$
  - (c)  $O(m+n)$
  - (d)  $O(\log m + \log n)$
21. Give the correct matching for the following pairs:
- |   |  |
|---|--|
| (A) $O(\log n)$                                       | <input checked="" type="checkbox"/> (P) Selection sort |
| <input checked="" type="checkbox"/> (B) $O(n)$        | (Q) Insertion sort                                     |
| <input checked="" type="checkbox"/> (C) $O(n \log n)$ | (R) Binary search                                      |
| <input checked="" type="checkbox"/> (D) $O(n^2)$      | <input checked="" type="checkbox"/> (S) Merge sort     |
22. Assume that the algorithms considered here sort the input sequences in ascending order. If the input is already in ascending order, which of the following are **TRUE** ?
- I. Quicksort runs in  $\Theta(n^2)$  time
  - II. Bubblesort runs in  $\Theta(n^2)$  time
  - III. Mergesort runs in  $\Theta(n)$  time
  - IV. Insertion sort runs in  $\Theta(n)$  time
- (a) I and II only
  - (b) I and III only
  - (c) II and IV only
  - (d) I and IV only

For Micro Notes by the Student



$$d = \log_b n^3$$

$$O(d \cdot n) = 3 \cdot \log_b n$$

$$O(n \cdot \log_b n) = O(n)$$

$$O(n) \quad \text{if } b = n$$

following are TRUE ?



23. An array of 25 distinct elements is to be sorted using quicksort. Assume that the pivot element is chosen uniformly at random. The probability that the pivot element gets placed in the worst possible location in the first round of partitioning (rounded off to 2 decimal places) is 0.08

$$\frac{1}{25} + \frac{1}{25}$$

# Long Integer Multiplication (LIM): Divide & Conquer

Given Two integers  $u$  &  $v$ ;  $u, v$ : integer

$$a) u + v = O(1)$$

$$b) u * v = O(1)$$

S/W Solution to repr. Long integers?

Repr. them in array

int a[1000], b[1000], c[1000];

$$1) a + b = c$$

$$\left\{ \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ c[i] = a[i] + b[i] \end{array} \right\} O(n)$$

$$2) a * b = c \quad \begin{array}{l} \text{for } i \leftarrow 1 \text{ to } n \\ \text{for } j \leftarrow 1 \text{ to } n \\ c[i] = a[i] * b[j] \end{array} \Rightarrow O(n^2)$$

✓ integers :  $\frac{2B}{1} | 4B$   
 ↳ Max: 32767

✓ Long int :  $\frac{4B}{1} | 8B$  5 digit  
 ↳ (8 digit)

(H/W Soln is not viable)

a[i], b[i], c[i]: store a digit of long int

(123458)

a [ 1 | 2 | 3 | 4 | 5 | 8 ]  
 1 2 3 4 5 6

$\left\{ \begin{array}{l} 1234 \\ 9856 \\ \hline 7404 \\ +6170 \\ \hline 69104 \end{array} \right\}$  (School Method Non-DC)

→ Given two long integers of  $n$ - digits each, say  $u$  & ' $v$ ', then the Time Complexity to multiply them using School/Nom-Dc method is  $O(n^2)$

$\underline{O(n^2)}$

DandC Method

$\underline{O(n^2)}$

$$u = \underline{\underline{12}} \ 34 \\ = \underline{\underline{12}} \times 10^2 + \underline{\underline{34}}$$

$$v = \underline{\underline{56}} \ 78 \\ = \underline{\underline{56}} \times 10^2 + \underline{\underline{78}}$$

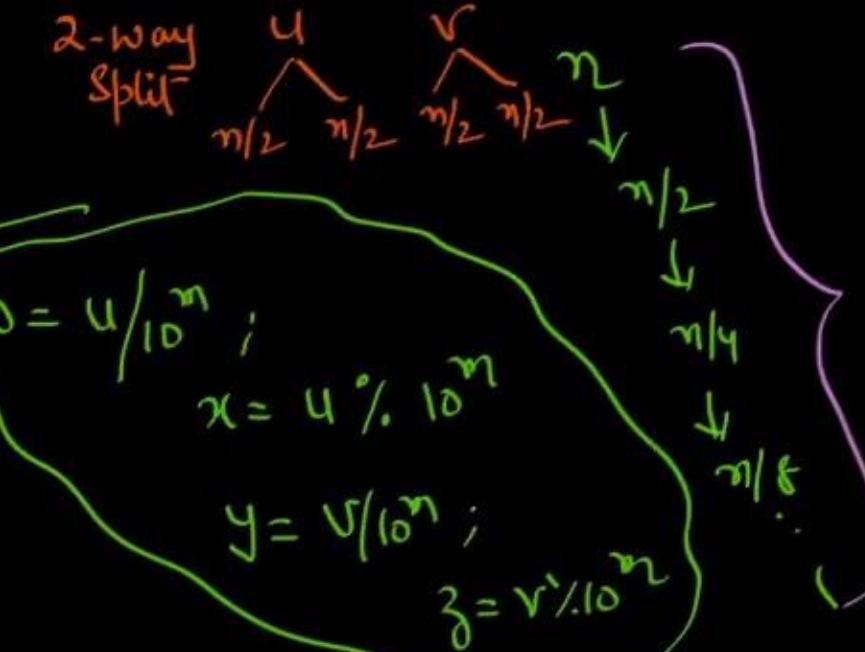
$$\begin{aligned} u, v: & 'n' - \text{digit integers}; \quad m = \lfloor \frac{n}{2} \rfloor; \\ w, x, y, z: & (\frac{n}{2}) \text{ " " } \end{aligned}$$

$$u = w \times 10^m + x \quad v = y \times 10^m + z$$

$$u \cdot v = (w \times 10^m + x) \cdot (y \times 10^m + z)$$

$$= w \cdot y \times 10^{2m} + (wz + xy) \cdot 10^m + xz$$

$$\text{Space: } O(\log n)$$



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

$$= 4 \cdot T(n/2) + bn, \quad n > 1$$

$$\text{bn is if } O(n^{2-\epsilon}) \quad \epsilon=1$$

$$\therefore T(n) = O(n^2)$$

$$u \cdot v = (w \times 10^m + x) \cdot (y \times 10^m + z)$$

$$= w \cdot y \times 10^{2m} + \underline{\underline{(wz + xy) \times 10^m}} + x \cdot z$$

$u, v$ : n-digit integers  
 $t, w, x, y, z$ :  $n/2$  "

(Anatoli Karatsuba) (AK)

$$\text{Let } t = (w+x) \cdot (y+z)$$

$$= wy + (wz + xy) + xz$$

$$\underline{wz + xy} = \underline{t} - (wy + xz)$$

$$\text{Let } \begin{cases} wy = Pr_1 \\ xz = Pr_2 \\ t = Pr_3 \end{cases}$$

$$u \cdot v = Pr_1 \times 10^{2m} + \left( Pr_3 - (Pr_1 + Pr_2) \right) \times 10^m + Pr_2 \quad \text{--- (2)}$$

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

$$= 3 \cdot T(n/2) + bn \quad n > 1$$

$$\therefore bn \in O(n^{1.58-\epsilon})$$

$$\log_2 3 = 1.58$$

$$\therefore T(n) \in O(n^{1.58}) \quad \checkmark$$

Joom & Cook's ( $T_C$ ):

3-way Split: (Break up the  $n$ -digit integer into  $n/3$  digits)

1) DandC: 3-way Split:

$$T(n) = 9 \cdot T(n/3) + bn \Rightarrow O(n^2)$$

AK optimization

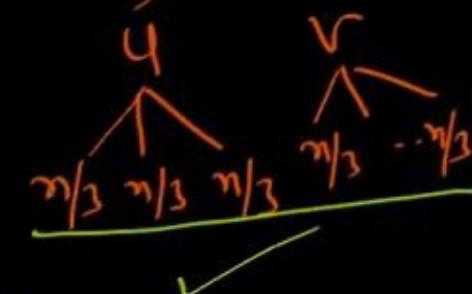
$$\hookrightarrow T(n) = 8 \cdot T(n/3) + bn$$

$$O(n^{\log_3 8}) = O(n^{1.89})$$

$$T(n) = x \cdot T(n/3) + bn$$

$$n^{\log_3 x} < n^{1.58}$$

$$\Rightarrow \log_3 x < 1.58 \Rightarrow x = 5$$



$$T(n) = 5 \cdot T(n/3) + bn$$

$$O(n^{\log_3 5}) = O(n^{1.46})$$

4-way Split:

Damd C:  $T(n) = 16 \cdot T(n/4) + bn \Rightarrow O(n^2)$

AK:  $T(n) = 15T(n/4) + bn \Rightarrow O(n^{\log_4 15}) = O(n^{1.9})$

$T_C: T(n) = \underbrace{x \cdot T(n/4)}_{n^{\log_4 x}} + bn \Rightarrow < n^{1.46}$

$$n^{\log_4 x} < n^{1.46} \Rightarrow \log_4 x < 1.46$$

$$\Rightarrow x = \underline{\underline{7}}$$

$$T(n) = 7 \cdot T(n/4) + bn \hookrightarrow O(n^{\log_4 7}) = O(n^{1.40}) \underline{\underline{<} O(n^{1.46})}$$

# K-way Split :

1) DandC :  $T(n) = K^2 \cdot T(n/K) + bn \Rightarrow O(n^{\log_K K^2}) = O(n^2)$

2) A.K :  $T(n) = (K-1) \cdot T(n/K) + bn \Rightarrow O(n^{\log_K (K-1)})$

3) T<sub>C</sub> :  $T(n) = (2K-1) \cdot T(n/K) + bn$   
 $\Rightarrow O(n^{\log_K (2K-1)})$

(2K-1)

2-way  $\Rightarrow$  3 Sub

3-way  $\Rightarrow$  5

4-way  $\Rightarrow$  7



## Backtracking vs Branch-Bound

→ D.H. Lehmer (1950's)

↳ Walker

(constitute feasibility criteria + objectivity)

Backtracking :  $\{D.F.S\}$  + Bounding functions

Branch-Bound :  $\{B.F.S\}$  + Bounding functions

↓  
 Kill the portion of  
 the State Space that  
 is not feasible (optimal)

(State Space  
tree)

→

{Search Space}

Feasible Solns  
Objective fn.  
Optimal Soln

## Terminology

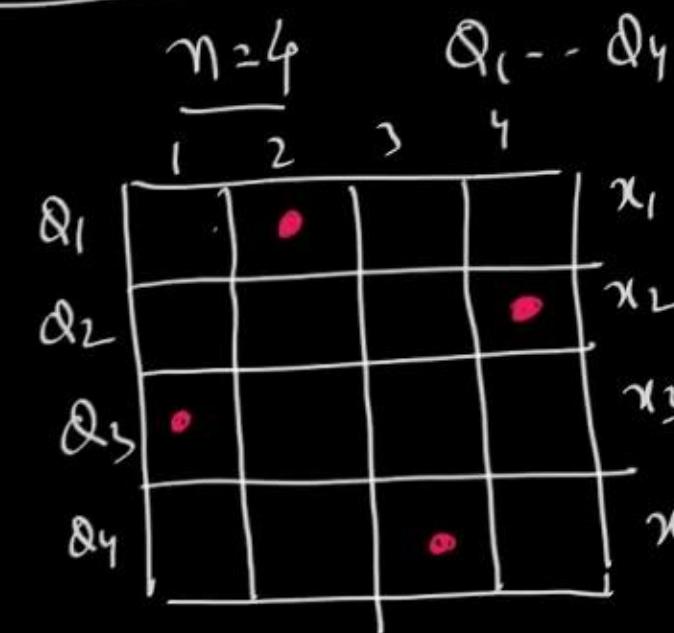
→ Problem defn

→ Constraints

Implicit implies

Local  
Feasibility



$n$ -Queens :

$\leftarrow x_4$  CHTBD  
 $Soln Space = [n]$

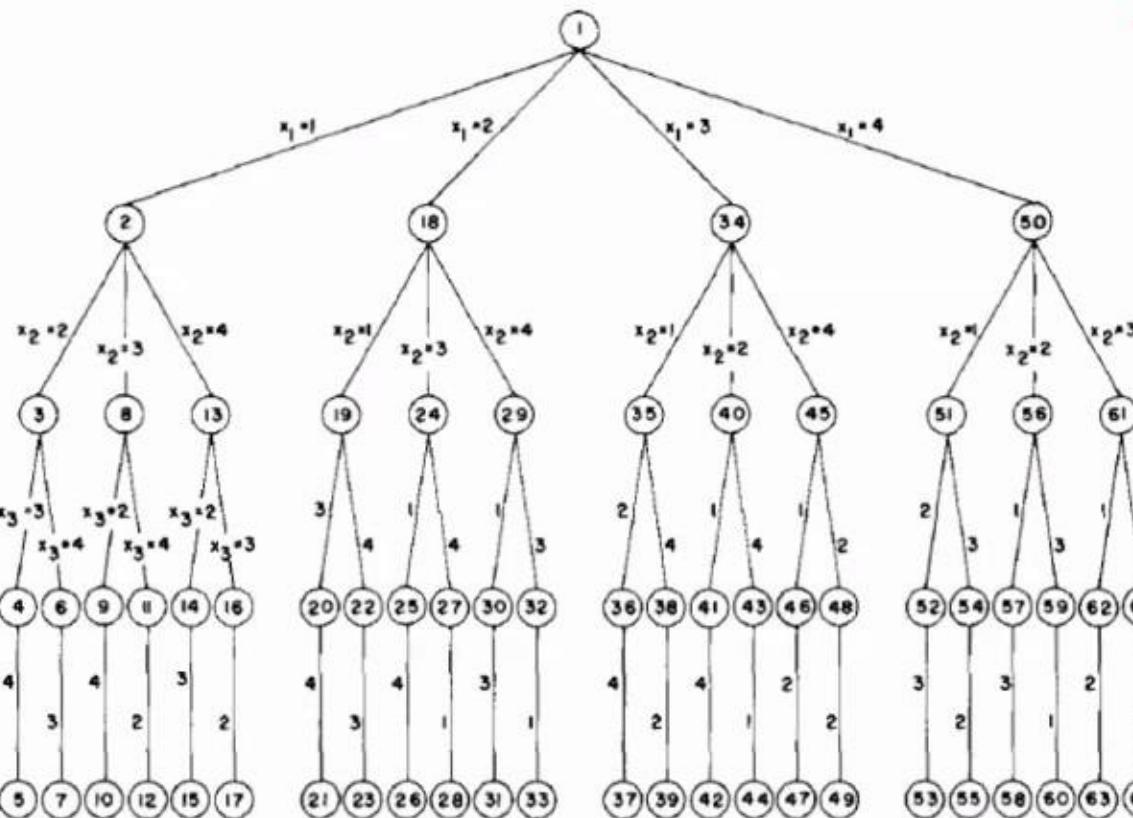
$\langle x_1 \ x_2 \ \dots \ x_n \rangle$   $\hookrightarrow n\text{-tuple}$

free structure

$x(i)$  = denote the position of  $Q_i$  placed in row  $i$

$1 \leq x_i \leq n$

$\langle 2 \ 4 \ 1 \ 3 \rangle$   
 $\langle 3 \ 1 \ 4 \ 2 \rangle$



Solution  
( $x_1 \dots x_n$ )

State-Space Tree organization  
of 4-Queens problem

$$\text{Soln Space: } 14 = 2^4$$

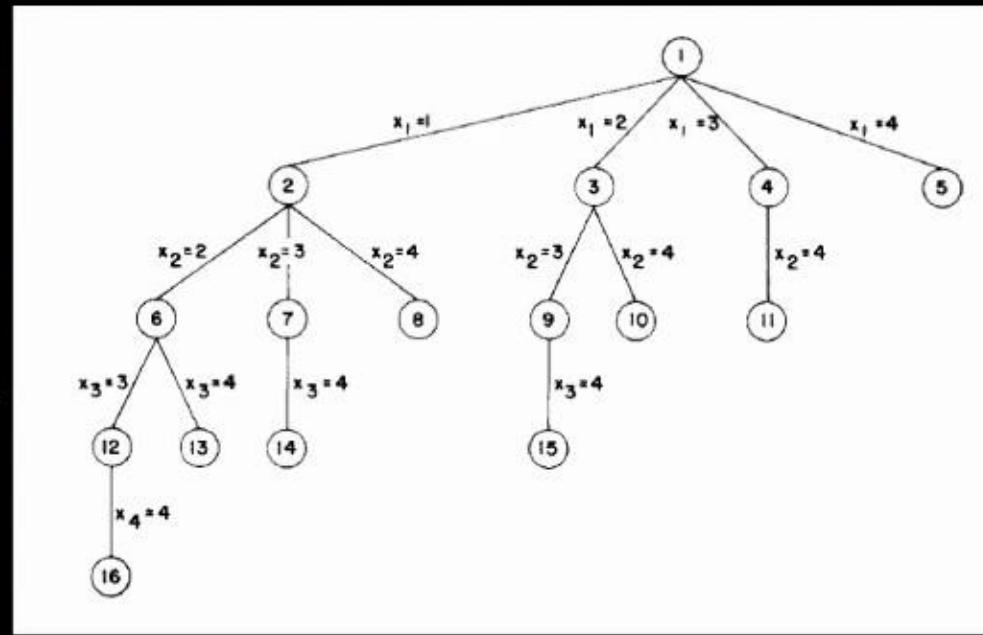
Any-Node : Problem State

4

{ Solution States }

↙  
Satisfy Implicit Constraints  
(Answer State)

State-Space Tree  
(Fixed Tuple Size)  
(Variable tuple size)



State Space Tree (Variable Tuple Size)  
for Subset-Sum (SS) Problem

$$n = 4 \quad A[1 \dots 4];$$

$$\begin{aligned} \text{No. of Subsets} &= \{2^4\} \\ &= \underline{\underline{16}} \end{aligned}$$

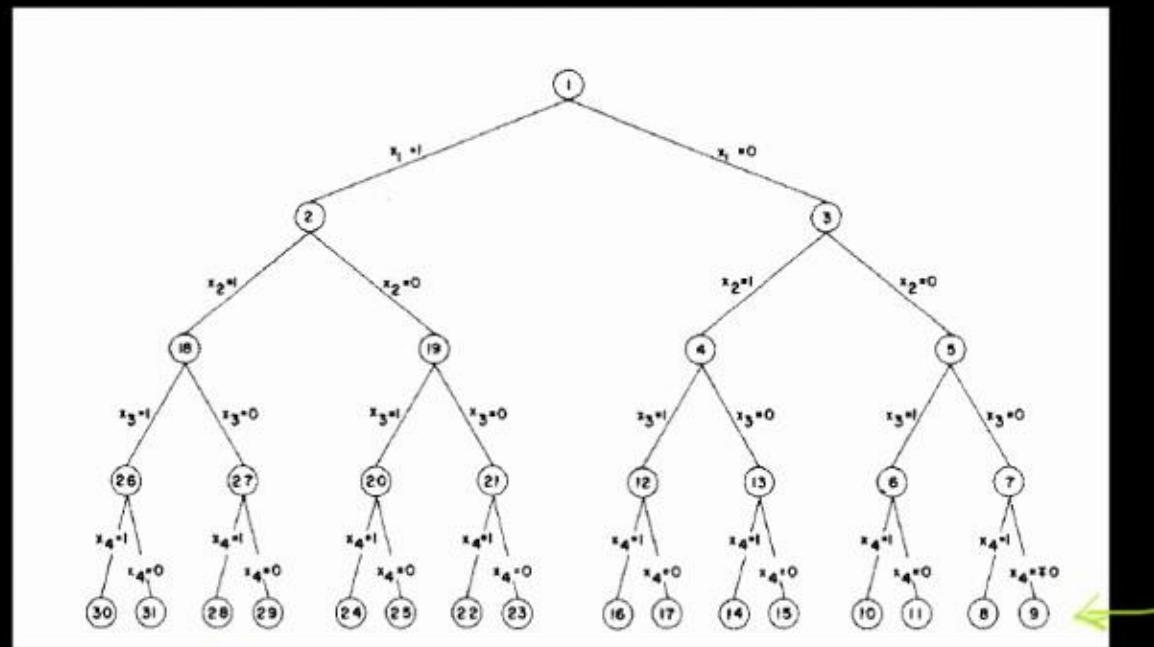
Each Node  
represents a Subset

Answer Nodes

Feasibility  
Boundary fm

fixed Tuple size  
 $\langle x_1, x_2, x_3, x_4 \rangle$

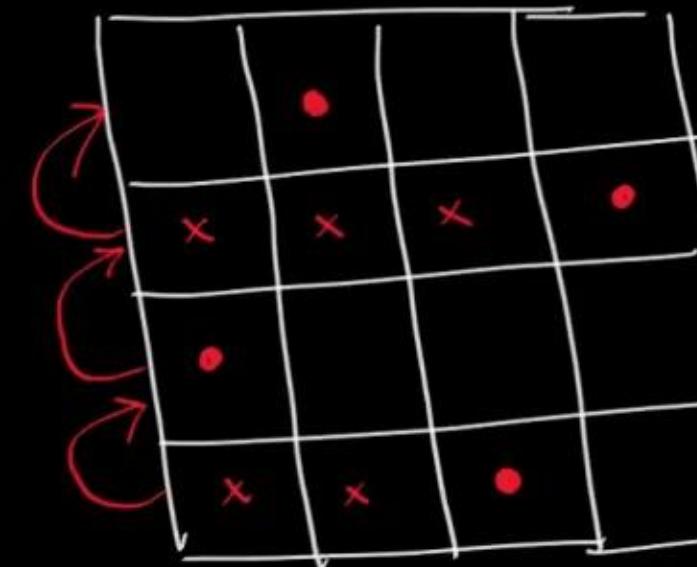
- ① =  $\emptyset \rightarrow \langle 0, 0, 0, 0 \rangle$
- ② =  $\{1\} \rightarrow \langle 1, 0, 0, 0 \rangle$
- ③ =  $\{2, 3\} \rightarrow \langle 0, 1, 1, 0 \rangle$
- ④ =  $\{1, 3, 4\} \Rightarrow \langle 1, 0, 1, 1 \rangle$
- ⑤ =  $\{1, 2, 3, 4\}$



fixed tuple size formula  
of SoS ( $n=4$ )

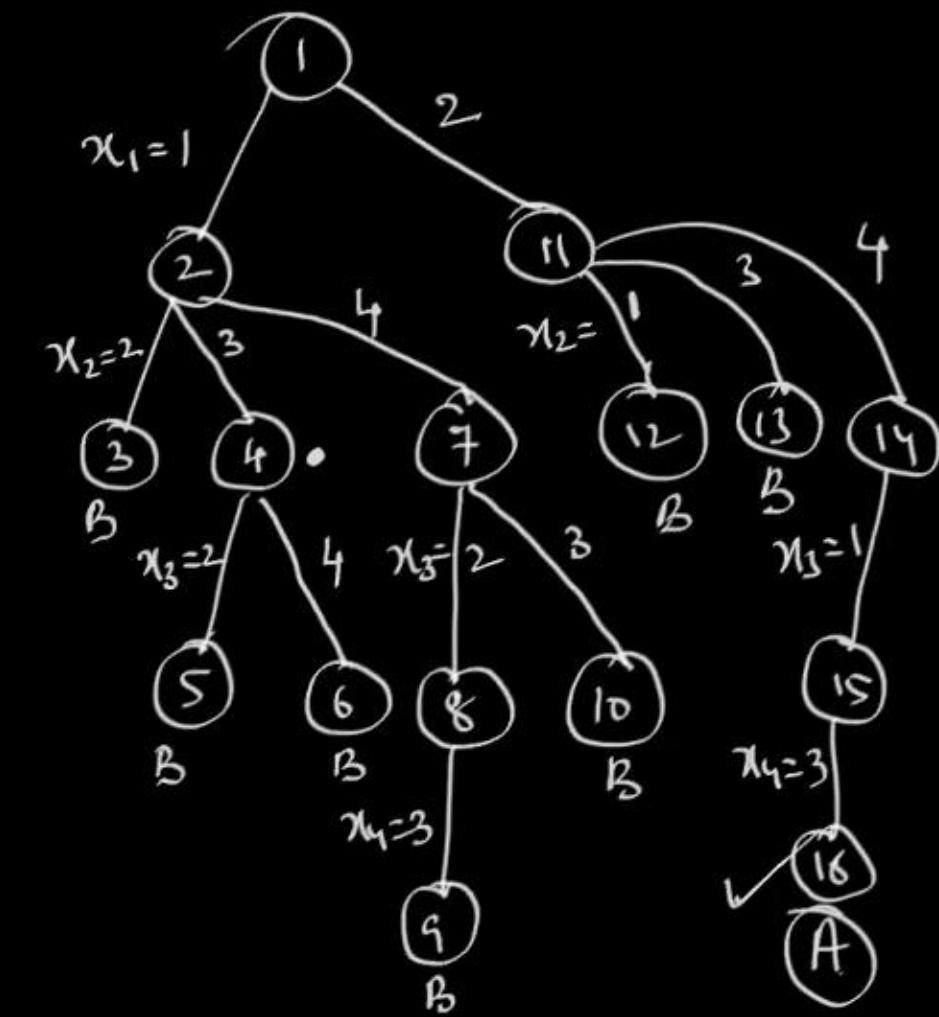
# Working Principle of Backtracking :

8-Queens:



$\langle 2, 4, 1, 3 \rangle$

$\{DFS + BF\}$



```
procedure BACKTRACK(n)
  //This is a program schema which describes the backtracking process.//
  //All solutions are generated in  $X(1:n)$  and printed as soon as they are//
  //determined.  $T(X(1), \dots, X(k - 1))$  gives all possible values of//
  // $X(k)$  given that  $X(1), \dots, X(k - 1)$  have already been chosen.//
  //The predicates  $B_k(X(1), \dots, X(k))$  determine those//
  //elements  $X(k)$  which satisfy the implicit constraints.//
  integer k, n; local  $X(1:n)$ 
  k = 1
  while k > 0 do
    if there remains an untried  $X(k)$  such that
       $X(k) \in T(X(1), \dots, X(k - 1))$  and  $B_k(X(1), \dots, X(k)) = \text{true}$ 
      then if  $(X(1), \dots, X(k))$  is a path to an answer node
        then print  $(X(1), \dots, X(k))$  endif
        k = k + 1 //consider the next set//
      else k = k - 1 //backtrack to previous set//
        endif
      repeat
    end BACKTRACK
```

## 2) Graph Coloring : (K-colorability Problem)

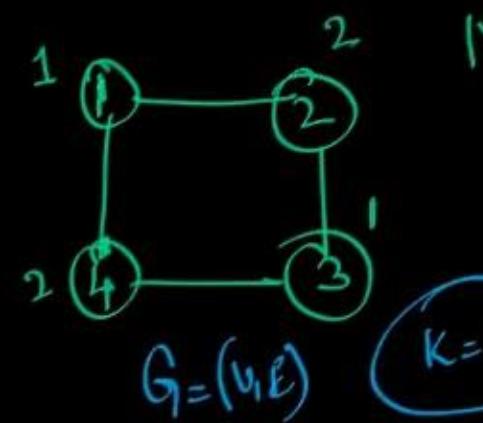
chromatic = optimization  
no. Problem

decision problem

Is it possible to color the given graph with K-colors?  
If so enumerate the orderings.

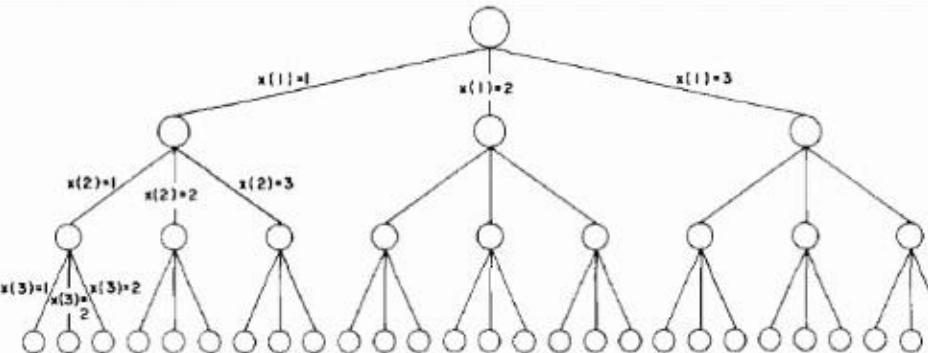
$$G = (V, E) \quad |V| = n; \quad |E| = e$$

No two adjacent vertices must have the same color;

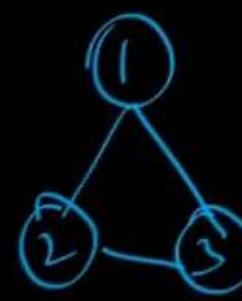


Min = 2 color  
(Chromatic No.)

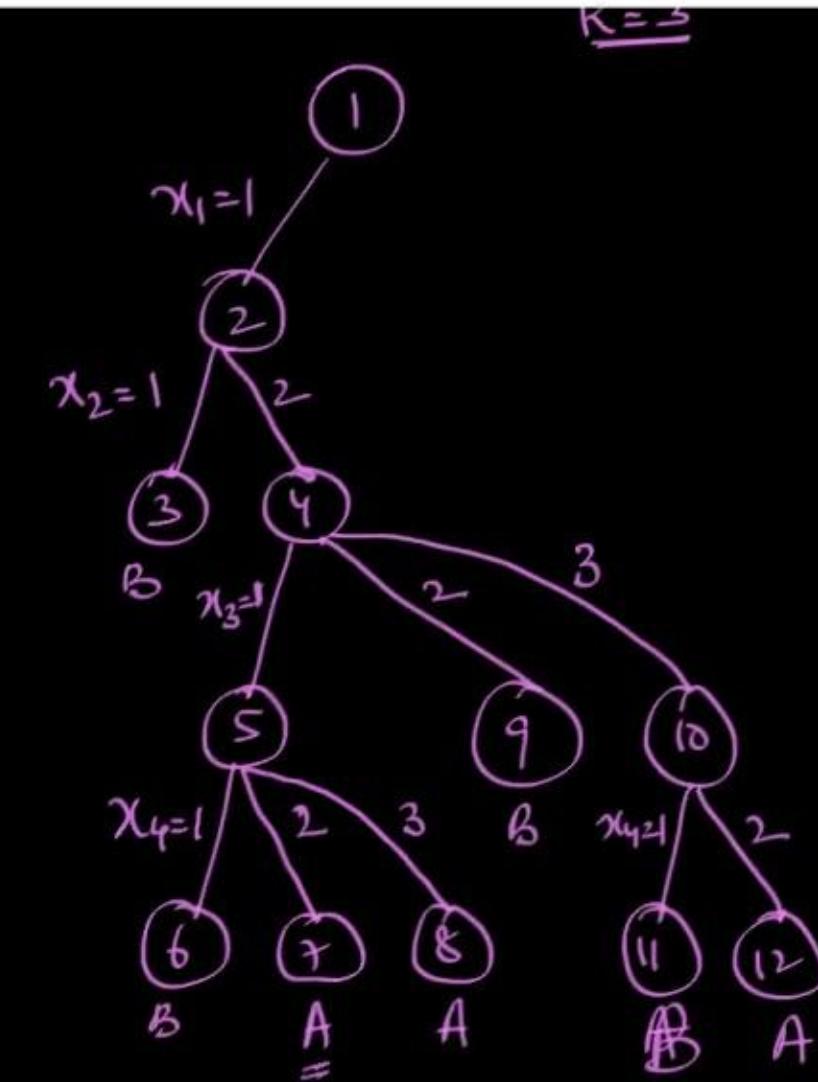
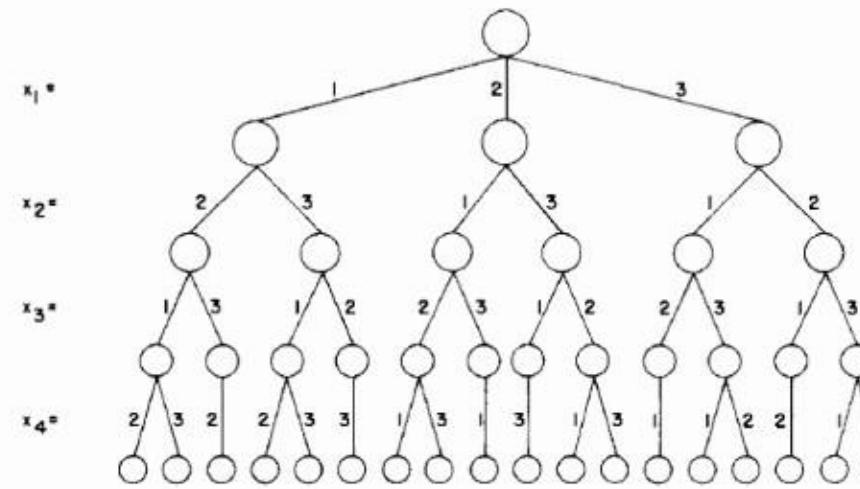
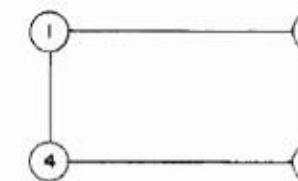
K=3



Full State Space Tree  
for a graph having  
'3' vertices

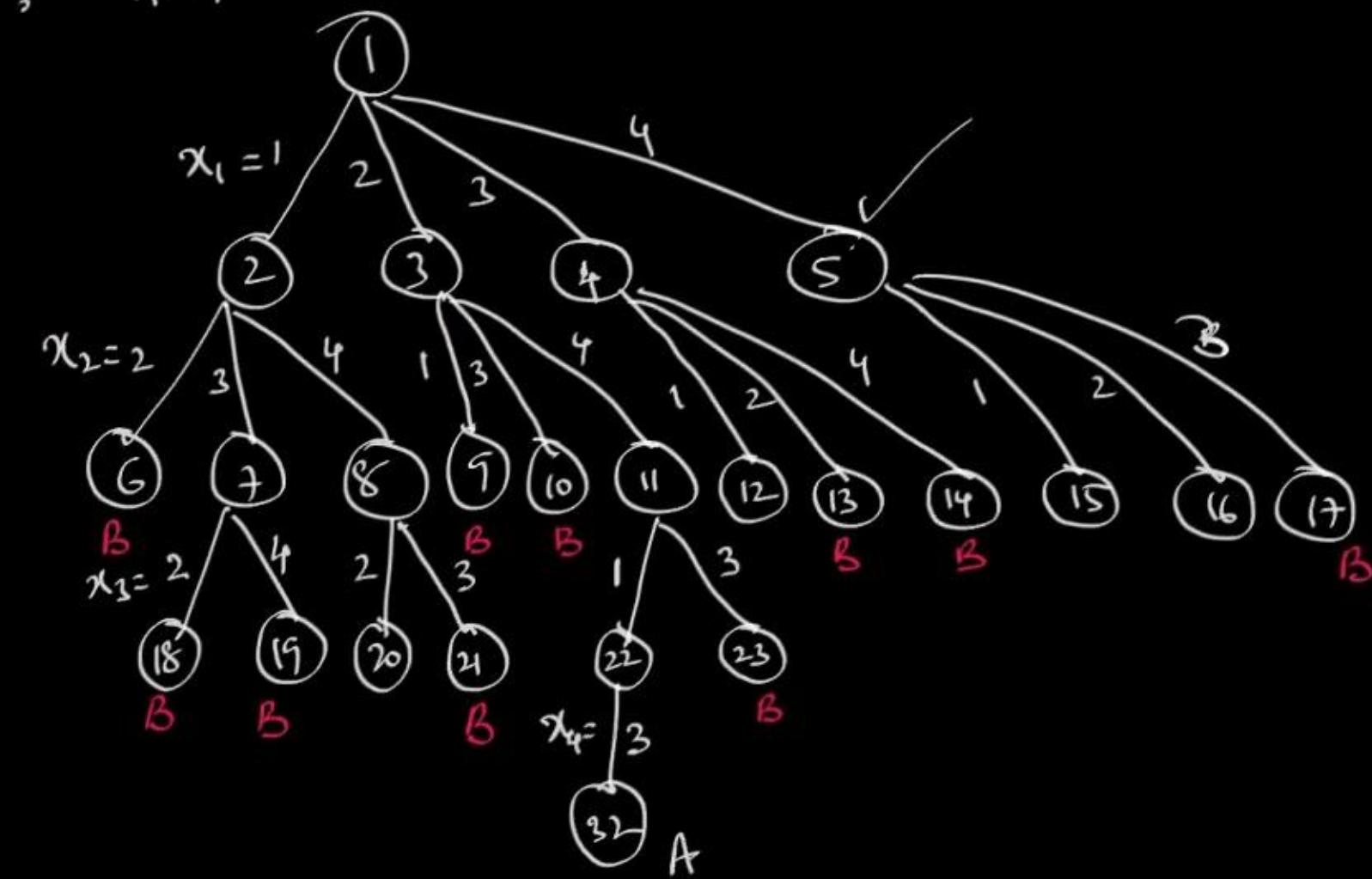


$K = 3$   
 $\times \langle 1 \dots 3 \rangle$   
 $\times [i] = \text{Color assigned}$   
 $\text{to vertex } i$

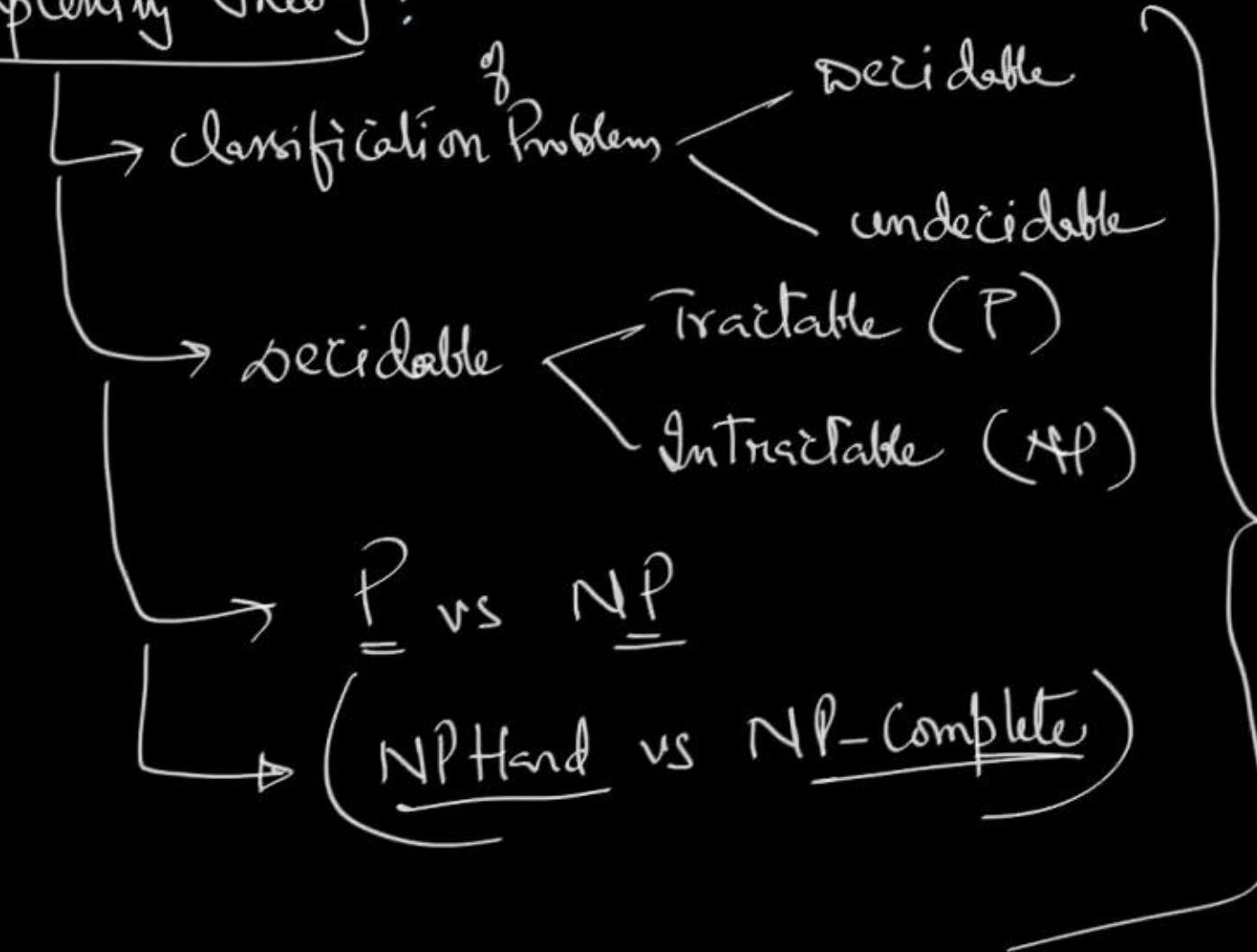


# Branch - Bound (BFS + B-FS)

n - Queens ;  $n = 4$



## Complexity Theory:





Asymptotic Apriori Analysis (ASN)  
 Space Complexity  
 Recursive Framework:  
 → loop complexity

Part - II:

D and C:  
 G.M.:

D.P.: (practise Problems)

M.C.P; LCS      OBST + B.F

Part - III:

→ Graphs  
 → Heaps  
 → Sorting Methods

$$\overline{M.S} : T(n) = \overline{2T(n/2)} + an \Rightarrow O(n \lg n)$$

$$\text{SMMN: } F(n) = \overline{2F(n/2)} + bn^2 \Rightarrow O(n^2)$$

{ After GATE }

→ Programming | Coding (Leetcode | HackerRank)

one/two PL  
C/C++

{ 3 Months }

OS CP  
5-9  
Deadline: 11<sup>th</sup> Oct

→ Basic

D-S : L-L + Arrays + Stacks + Queues  
Fund Tech : (Search + Sorting + Hash Tables)  
Trees : (B-T + AVL + Heaps + BST)  
Graphs : Traversals + Graph (Gm + DP)  
Competitive Prog : (Gm + D-P + Backtracking)



Cormen  
Sahni  
Tamatia  
das Grubba