Data Science and Artificial Intelligence

Algorithms

Revision Class

(One Shot)

By- Dr. Khaleel Khan sir

Topics to be Covered

- Topic — Asymptotic Analysis : (Time & Space Complexities)
- Topic — Divide And Conquer : M.S + Q.S
- Topic — Shortest Path Problem (G.M + D.P)
- Topic — Graph Algorithms
- Topic — Sorting Methods

ABOUT ME

Hello, I'm Dr. Khaleel Ur Rahman Khan.

1. Ph.D. in Computer Science.
2. Professor in Computer Science.
3. Has more than 29 Years of Experience in Teaching at Engineering Colleges.
4. Published more than 75 Journal articles in the area of Wireless Networks.
5. Nine(9) candidates have been awarded Ph.D. Under his Supervision.
6. Has more than 23 years of Educating and Mentoring the GATE Aspirants.

By- Dr. Khaleel Khan sir

Topic : Syllabus

I. **Search Algorithms:**
   - Linear Search
   - Binary Search

II. **Basic Sorting Algorithms:**
   - Selection Sort
   - Bubble Sort
   - Insertion Sort

III. Divide and Conquer:
   - Merge Sort
   - Quick Sort

IV. Introduction to Graph Theory

V. Basic Graph Algorithms:
   - Traversals
   - Shortest path

Topic : Lecture Schedule

I. Analysis of Algorithms

1.1 Algorithm Concept and Lifecycle
1.2 Analysis of Algorithms
    1.2.1 Need for Analysis
    1.2.2 Performance Analysis: Time and Space Complexities
1.3 Methodology of Analysis
    1.3.1 Aposteriori Analysis
    1.3.2 Apriori Analysis
1.4 Types of Analysis
1.5 Asymptotic Notations
1.6 Properties of Asymptotic Notations

1.6 Properties of Asymptotic Notations
1.7 Framework for Analysing Recursive Algorithms
1.8 Apriori analysis of Non-Recursive Algorithms
1.9 Analysing Loops
1.10 Space Complexity
1.10.1 Components of Space Complexity
1.10.2 Examples
1.11 Mathematical Background
1.12 Problem Solving

Pre Requisite

1) Math Foundations

→ Logarithms
→ Exponents
→ Summation Series
→ Probability
→ Combinatorics
→ Diff. & Integration

II. Design Strategies

2. Divide & Conquer
   - 2.1 General Method
   - 2.2 Max-Min Problem
   - 2.3 Merge Sort
   - 2.4 Binary Search
   - 2.5 Quick Sort
   - 2.6 Matrix Multiplication
     - 2.6.1 D and C Method
     - 2.6.2 Strassen's Method
   - 2.7 Long Integer Multiplication (LIM)
   - 2.8 Master Method for D and C Recurrences
   - 2.9 Recursion Tree
   - 2.10 Problem Solving

3. Sorting Algorithms

- 3.1 Basic terminology
- 3.2 Methods
  - 3.2.1 Bubble Sort
  - 3.2.2 Selection Sort
  - 3.2.3 Insertion Sort
  - 3.2.4 Radix Sort
- 3.3 Problem Solving

4. Graph Algorithms

4.1 Representation of Graphs

4.2 Graph Traversals

DFS

4.2.1 Undirected Connected Graphs

4.2.2 Undirected Disjoint Graphs: DFT

4.2.3 Directed Graphs & Types of Edges

4.2.4 DAG

BFS

4.2.5 FIFO BFS

4.2.6 LIFO BFS

4.2.7 LC BFS

4.3 Parenthesization Theorem

4.4 Problem Solving

1. ASYMPTOTIC ANALYSIS

Topic : Introduction of Course

Algorithms

Analysis of Algorithms

- Algorithm concept and life cycle
- Needs for analysis
- Methodology of analysis
- Types of analysis
- Asymptotic notation
- Framework for analysis of recursive and non recursive Algorithms
- Analysing loop
- Space complexities

Design Strategies

- Divide and Conquer
- Greedy method
- Dynamic programming

Graph Algorithms

- Graph Algorithms
- ~~Heap Algorithms~~
- ~~Hashing~~
- ~~Sets~~
- Sorting Algorithms

Algorithm Concept and Life Cycle Steps

Basic opn's

$x \leftarrow y + z$

for $i \leftarrow 1$ to n

$c = c + 1;$

$f();$

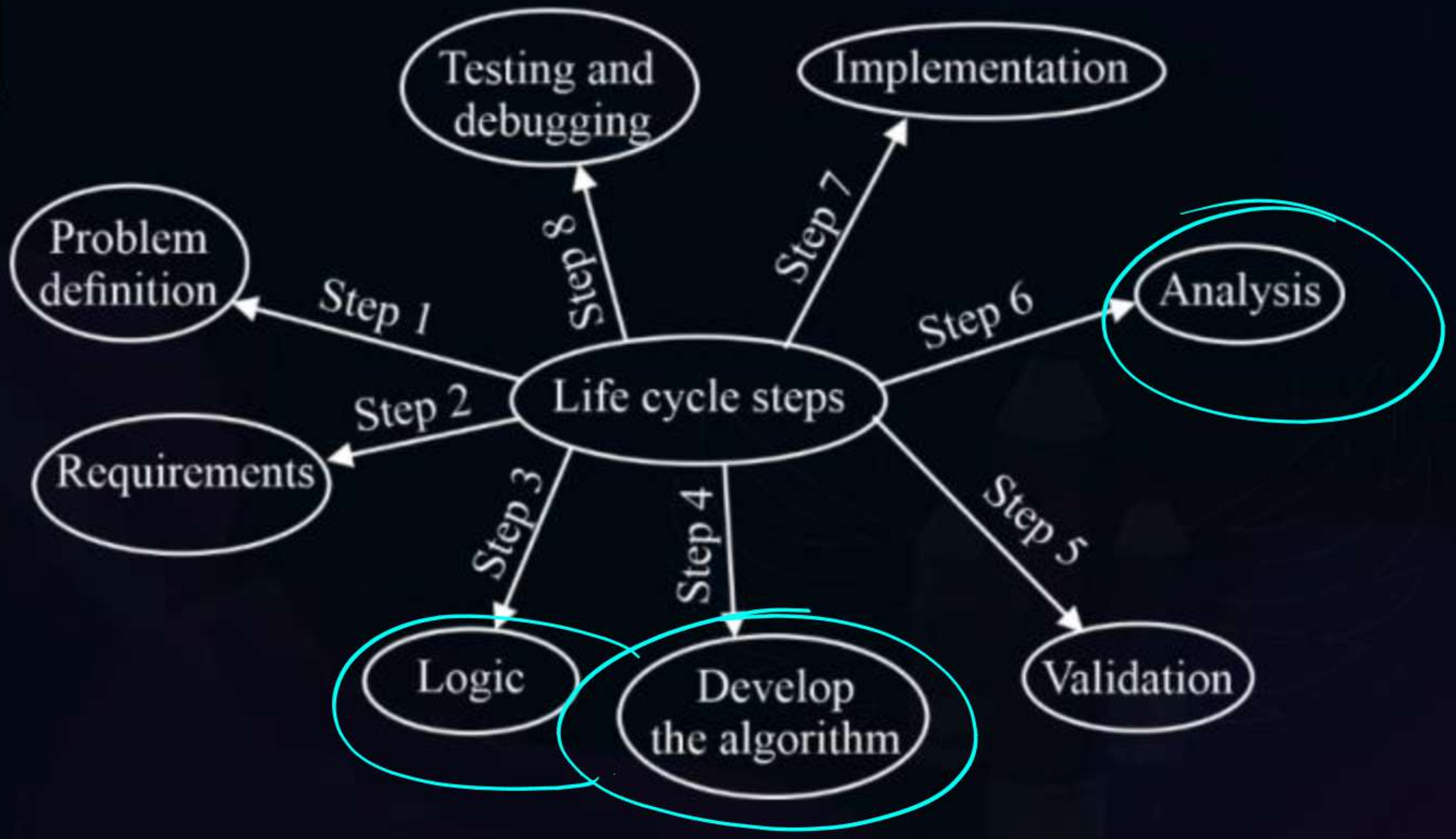
Algorithm

Statement

- An Algorithm consists finite number of steps to solve any problem.
- Every step involves some operations and each operation must be definite and effective.

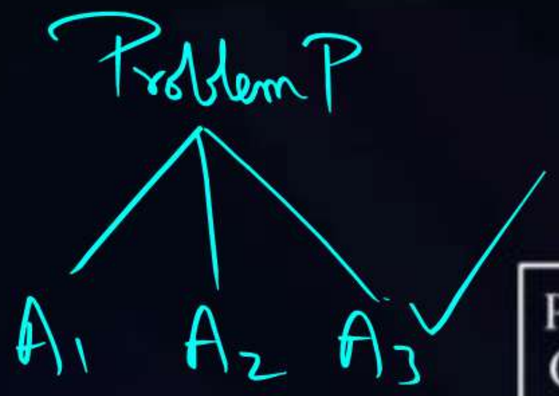
Algorithm Concept and Life Cycle Steps

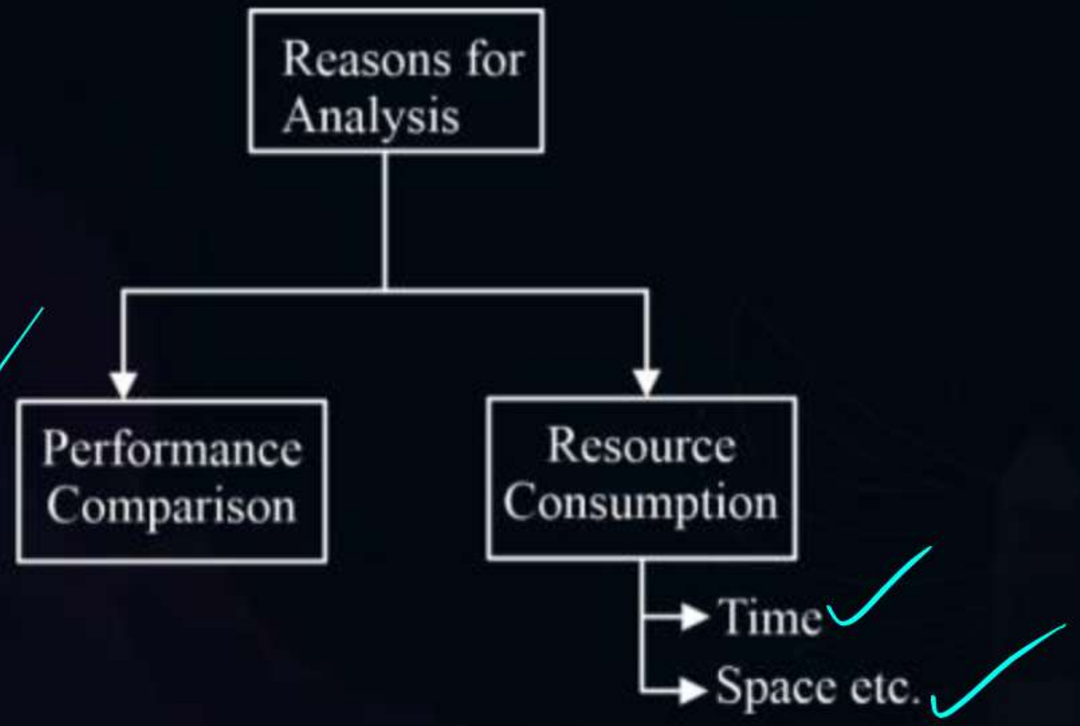
Life Cycle Steps



Need of Analysis

- In performance comparison comparing different algorithms for optimal solution.





Need of Analysis

Time Complexity

- Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the input size.

Space Complexity

- Space complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of input size.

NOTE:

- To find the time complexity of an algorithm, find the loops and also consider larger loops.
- Space complexity is depends on two things input size and some extra space (stack space link, space list etc.)

Methodology of Analysis

How

PL + O.S

H/W S/W

Platform Independent

Experiment.

Platform Dep. Analysis

Pen + Paper

RAM Model

Methodology of analysis

- Aposteriori Analysis
  - Advantage
    - Always gives exact value of time complexity and space complexity
  - Disadvantage
    - System to system result varies
- Apriori Analysis
  - Advantage
    - Always gives same result
  - Disadvantage
    - Analysis gives approximate value of time and space complexity

Types of Analysis

Behaviour

Worst Case

The input class for which the algorithm does maximum work and hence, take maximum time.

Best Case

The input class for which the algorithm does minimum work hence, take minimum time.

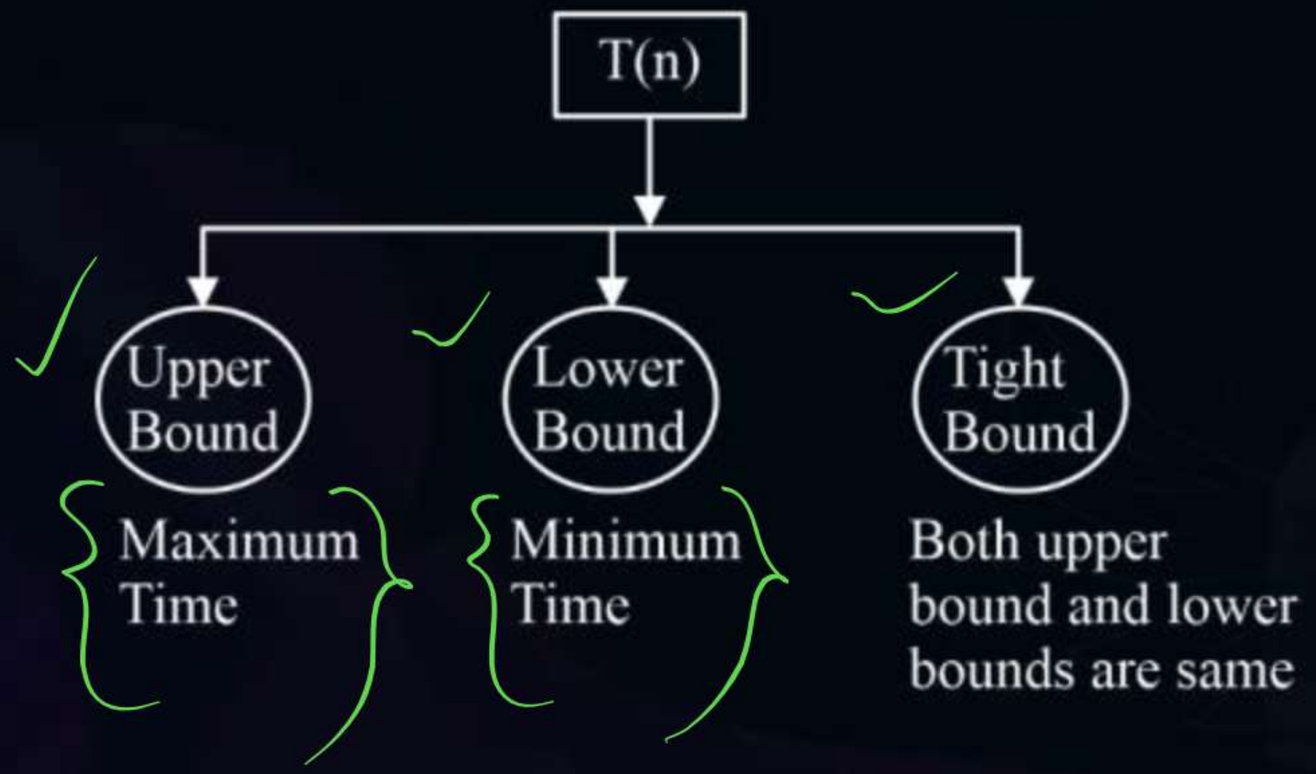
Average Case

* Average case can be calculated form best case to worst case.

$$\sum_{i=1}^{k}$$

Asymptotic Notations (ASN)

Suppose, T(n) be a function of time for any algorithm.



Types of Asymptotic Notations

Asymptotic Notation

Big Notations

- Big oh (O): Upper bound
- Big omega (Ω): Lower bound
- Theta (θ): Tight bound

Small Notations

- Small oh (o): Proper upper bound
- Small omega (ω): Proper lower bound

Big O – Notation

$f(n)$: Time $\rightarrow g(n)$

$$\boxed{f(n) \le c \cdot g(n)}$$ for some $c > 0$ where $n > k$ $[k > 0]$

Two Functions f(n), g(n)

f(n) = O(g(n))

When the growth of g(n) is same or higher than f(n) like a ≤ b

$$\boxed{f(n) \in g(n)}$$

Example:

n^2

f(n) = 3n + 10, g(n) = n2 + 2n + 5

f(n) = O(g(n))

$f(n)$'s

(Less rate of growth)

Polynomial

(n^x) (x is a const.)

n, n^2, n^3

Logarithms

$(\log n)$

$(n \log n)$

Higher rate of Growth

Exponential

a^n $(a > 1)$

$2^n, 3^n, 4^n, n^n$

$(\log n)^n$

Ω - Notation

$F(n) = \Omega(f(n))$

$\therefore f(n) \geq C \cdot g(n) \qquad (a \geq b)$

Example: $3^n = \Omega(2^n)$

$3^n > 2^n$

θ - Notation

If $f(n) \le c1.g(n)$

And

$f(n) \ge c2.g(n)$

$f(n) = g(n)$

$\therefore$ $f(n) = \theta(g(n))$

Example: Big Small

$f(n) = 2n^2, g(n) = n+10$

$f(n) > g(n)$ here

so, $\boxed{f(n) = \Omega(g(n)) \text{ or } g(n)=O(f(n))}$

$f(n) = n^2$

$\Theta(n^2)$

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

g g

$f(n) \le c_1 \cdot n^2$ $f(n) \ge c_2 \cdot n^2$

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

Small Notations

- o — Proper U.B
- ω — Proper L.B

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

for all c

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

for all

$f(n) = n^2 < O(n^2), O(n^3)$

$\rightarrow o(n^3)$ ✓

$\rightarrow o(n^2)$ ✗

Properties with respect to asymptotic notations

	Reflexive	Symmetric	Transitive	Transpose symmetric
Big oh (O)	✓	×	✓	✓
Big omega (Ω)	✓	×	✓	✓
Theta (θ)	✓	✓	✓	×
Small oh (o)	×	×	✓	✓
Small omega (ω)	×	×	✓	✓

Example 1

#Q. Consider the following function $f(n) = \sum_{p=1}^{n} p^3$ Which of the following is/are true of f(n)

$\left[\frac{n(n+1)}{2}\right]^2 = cn^4$

A. $\theta(n^4)$ ✓

B. $\theta(n^5)$ ✗

C. $O(n^5)$ ✓

D. $\Omega(n^3)$ ✓

$\langle A + C + D \rangle$ ✓

Solution: (a, c, d)

$$f(n) = \sum_{p=1}^{n} p^3$$

$$= 1^3 + 2^3 + 3^3 + 4^3 + \ldots\ldots + n^3$$

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

$$= O(n^4) \text{ or } \Omega(n^4) = \theta(n^4)$$

Example 2

#Q. Consider the following function $f(n) = \sum_{p=1}^{n} p^{\frac{1}{2}} = q$

Find the value of q in terms of asymptotic notation.

$O(n\sqrt{n})$

$$\int_1^n x^x = \frac{x^{x+1}}{x+1}$$

$$= \left[\frac{p^{1/2+1}}{1/2+1}\right]_1^n$$

$$= \frac{n^{3/2}}{c} = n^{1.5}$$

$$= n\sqrt{n}$$

Example 2

#Q. Consider the following function f(n) = $\sum_{p=1}^{n} p^{\frac{1}{2}} = q$

Find the value of q in terms of asymptotic notation.

Solution: f(n) = $\sum_{p=1}^{n} p^{\frac{1}{2}}$

$= 1 + (2)^{1/2} + (3)^{1/2} + \ldots..$

$= \frac{2}{3}\left[n^{3/2} - 1\right]$

$= O(n^{1.5})$

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

$n!$ vs n^n

Example 3

#Q. Arrange the following functions in increasing order.

$f_1 = n\log n$, $f_2 = \sqrt{n}$, $f_3 = 2^n$, $f_4 = 3^n$, $f_5 = n!$, $f_6 = n^n$, $f_7 = \sqrt{\log n}$, $f_8 = 100n\log n$

P P E E E E P P

$$\boxed{f_7 < f_2 < \underbrace{(f_1 = f_8)}_{A.E}} < f_3 < f_4 < f_5 < f_6$$

Example 3

Arrange the following functions in increasing order.

f_1=nlogn, $f_2=\sqrt{n}$, $f_3 = 2^n$, $f_4 = 3^n$, $f_5 = n!$, $f_6 = n^n$, $f_7 = \sqrt{logn}$, f_8 = 100nlogn

$\rightarrow f_7 < f_2 < f_1 = f_8 < f_3 < f_4 < f_5 < f_6$

Example 4

#Q. Arrange the following functions in increasing order.

$f_1 = 10$, $f_2 = \sqrt{n}$, $f_3 = \log\log n$, $f_4 = (\log n)^2$, $f_5 = n^2$, $f_6 = n\log n$, $f_7 = n!$, $f_8 = 2^n$,

$f_9 = n^n$, $f_{10} = n^2\log n$

Example 4

Arrange the following functions in increasing order.

$f_1 = 10$, $f_2 = \sqrt{n}$, $f_3 = \log\log n$, $f_4 = (\log n)^2$, $f_5 = n^2$, $f_6 = n\log n$, $f_7 = n!$, $f_8 = 2^n$,

$f_9 = n^n$, $f_{10} = n^2\log n$

$\rightarrow \boxed{f_1 < f_3 < f_4 < f_2 < f_6 < f_5 < f_{10} < f_8 < f_7 < f_9}$

$\log(\sqrt{n})$ $\qquad$ $\log((\log n)^2)$

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

Example 5

H/W

#Q. Arrange the following functions in increasing order.

$f_1 = \log \log n$

$f_2 = \log n$

$f_3 = (\log n)2$

$f_4 = \sqrt{\log n}$

$f_5 = n^{1/10}$

$f_6 = n$

$f_7 = n2$

$f_8 = n\log n$

$f_9 = n \log \log n$

$f_{10} = n^2 \log n$

$f_{11} = n^3$

$f_{12} = 2^n$

$f_{13} = e^n$

$f_{14} = n!$

$f_{15} = n^n$

$f_{16} = n^{3/2}$

Example 5

Arrange the following functions in increasing order.

$f_1 = \log \log n$	$f_9 = n \log \log n$
$f_2 = \log n$	$f_{10} = n^2 \log n$
$f_3 = (\log n)2$	$f_{11} = n^3$
$f_4 = \sqrt{\log n}$	$f_{12} = 2^n$
$f_5 = n^{1/10}$	$f_{13} = e^n$
$f_6 = n$	$f_{14} = n!$
$f_7 = n2$	$f_{15} = n^n$
$f_8 = n\log n$	$f_{16} = n^{3/2}$

$f_1 < f_4 < f_2 < f_3 < f_5 < f_6 < f_9 < f_8 < f_{16} < f_7 < f_{10} < f_{11} < f_{12} < f_{13} < f_{14} < f_{15}$

$a^{\log_b c} = c^{\log_b a}$

$\therefore 2^{\log_2 n} \Leftrightarrow n^{\log_2 2} = n$

Example 6

#Q. Arrange the following functions in increasing order.

$f_1 = n!, f_2 = n^n$

$f_1 = n \times (n - 1)(n - 2) \times \ldots \times 3 \times 2 \times 1$

$f_2 = n \times n \times n \times n \times \ldots \times n \times n \times n$

$f_2 > f_1$

$\therefore f_1 = O(f_2)$

$\boxed{2^n < 3^n < 4^n < n! < n^n}$

[n * Log n]

#Q. Let F(n)=(log n)n Then which of the following statements is/are correct?

A. F(n)=Ω(n(log log n)) ✓

B. F(n)=θ(n2) ✗

C. F(n)=O(nlog n) ✓

D. None of the above

#Q. You are given that for all n, $f(n) \geq 0$ and $g(n) \geq 0$

(1) If $f(n) = O(g(n))$, then $g(n) = O(f(n))$; Not always

(2) $f(n) + g(n) = O(\max(f(n), g(n)))$

(3) If $f(n) = \Omega(g(n))$, then $g(n) = O(f(n))$

Which of the above are TRUE?

A. Only 2

B. Both 1 & 2

C. Both 2 and 3

D. All the above

#Q. Let f(n) and g(n) be asymptotically positive functions, then consider the following statements:

(a) $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$ ✗

(b) $f(n) = O((f(n))2)$ ✗

$(f(n))^2$

$f(n) = n$ $\quad (f(n))^2 = n^2$ ✓

$f(n) = 1/n > 1/n^2$

1) $f(n) = n$; $g(n) = n^2$

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

2) $f(n) = 2n$; $g(n) = n$

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

(4^n)

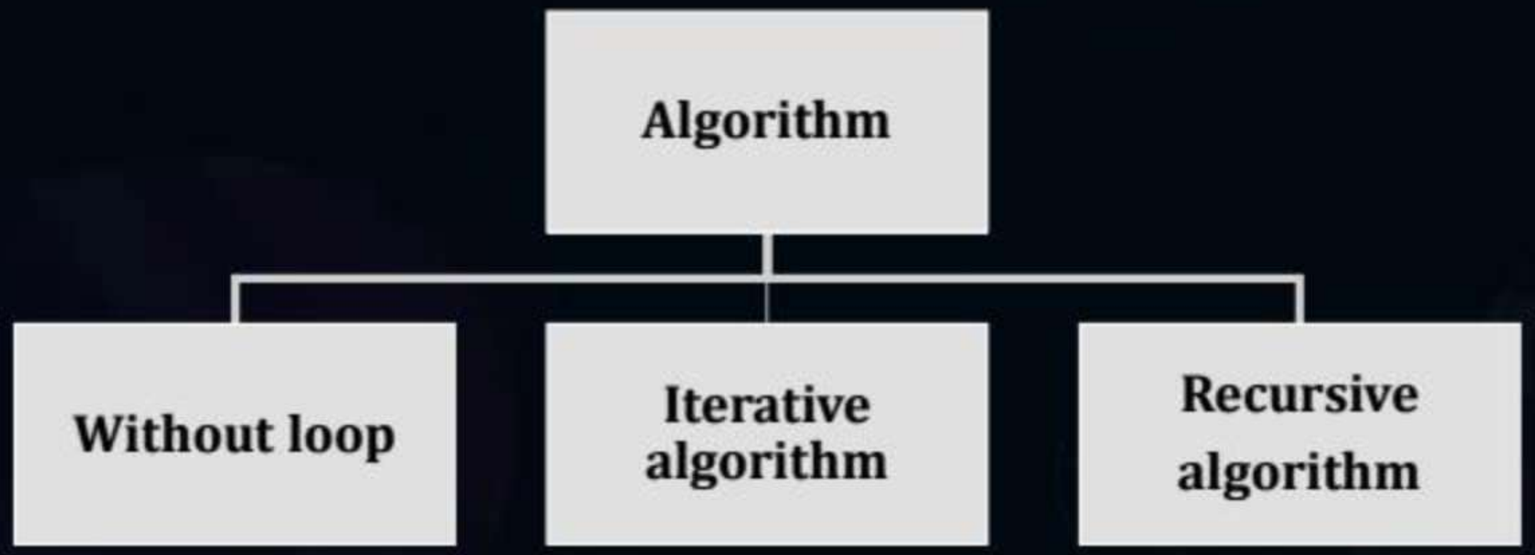
A a is false ✓

B b is false ✓

C a, b are true

D a is ture

Analysis of an Algorithm



Without loop

Example:

```
int fun (int n)
{
    return n*(n+1)/2;
}
```

c

Time: O(1)

1) loop complexities
2) Non-Recursive
3) Recursive Algo

Iterative Algorithm Analysis

Example 1:

```
for (i =1; i ≤ n; i=i*2)
printf("Sushil")
```

1 2 3 4 5 . . . K

$i = 1, 2, 4, 8, 16, \ldots$

$= 2^0, 2^1, 2^2, 2^3, 2^4 \quad 2^K$

$O(K) \therefore O(\log_2 n)$

$2^K = n$

$K = \log_2 n$

for i ← 1 to n

for $i = 1, n$

i = 1;

i ≤ n;

i++;

Iterative Algorithm Analysis

Example 1:

```
for (i =1; i ≤ n; i=i*2)
printf("Sushil")
```

Solution:

$i=1, 2, 2^2, 2^3 \ldots, 2^k$

$\rightarrow 2k \leq n$

$k \log 2 \leq \log n$

$k \leq \frac{\log n}{\log 2}$

$\therefore k \leq \log_2 n$

$k = \lfloor \log_2 n \rfloor$

So, this will execute $\lfloor \log_2 n \rfloor + 1$ time and Complexity $O(\log_2 n)$

$i = i*a \rightarrow \log_a n$ ✓

Example 2:

```
For (i=1; i ≤ n; i=i*3)
    printf("Aaveg");
```

$\log_3 n$

Example 2:

```
For (i=1; i ≤ n; i=i*3)
   printf("Aaveg");
```

Solution:

So, this will execute $\lfloor log_3 n \rfloor$ +1 time and complexity $O(\log_3 n)$

Example 3:

```
for (i = 1; i ≤ n; i++)   : n
{
        for (j=1; j ≤ 10; j++)   : 10
        {
                printf("Dhananjay");
        }
}
```

$10 * n = O(n)$

#Q. Consider the following C program:

```
j = n;
while (j > 1)
{
    for (i = 1; i ≤ j; ++i)
        printf("GATE Wallah");
    j = j/2;
}
```

$: n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots + \frac{n}{2^k}$

$n\left[\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k}\right]$

$n\left[\sum_{i=1}^{k} 1/2^i\right] = n\left(1 - \frac{1}{2^k}\right) = (n-1)$

$= \theta(n)$

Express number of time "GATE Wallah" is printed using θ notation in terms of n

A. θ(n)

B. θ(logn)

C. $\theta(n^2)$

D. θ(nlongn)

#Q. Consider the following ~~C program~~ Snipplet

```
Physics Wallah( )
{
        int n = 2^(2^k)
        for (i = 1, i ≤ n; i ++)
        {
            j = 2;
            while (j < = n)
            {
                j = j^2;
                printf("GATE CSIT")
            }
        }
}
```

K+1

$n = 2^{2^k}$

$k = \log\log n$

K

$k=1$	$k=2$	$k=3$
$n=4$	$n=16$	$n=256$
**	***	****
$n(2)$	$n(3)$	$n(4)$

(K+1)

Total $(n * (K+1))$

Asymptotic time complexity in big-oh is given as:

A. nlogn

B. nloglogn

C. n^2logn

D. $\sqrt{n}$ log n

Recursive Algorithm Analysis

Example 1:

```
void fun (int n)        T(n)
{
    if (n > 0)    1 compare; C1 time
    {
        printf ("% d", n);  ← C2 time
        fun (n - 1);  ←  T(n-1)
    }
}
```

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

Recursive framework

↓

Time ~ Recurrence

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

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

$$O(n)$$

Example 2:

```
void fun (int n)          T (n)
{
        if (n > 0)                C1 time
        {
                for (i = 1; i <= n; i + 1)        n time
                 printf("Hello");
                fun (n - 1);        T (n - 1)
        }
}
```

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

$= a + n + b + T(n-1) = O(n^2)$

Example 3:

```
void fun (int n)        ←——   T(n)
{
        if (n > 0)      ←——   C1
        {
                for (i = 1; i < = n; i = i*2)   ←——   ⌊log2 n⌋
                        printf("Divyajyoti");
                fun (n - 1);    ←——   T (n - 1)
        }
}
```

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

$\Rightarrow O(n \cdot \log n)$

Topic: Solving Recurrence Relation

Substitution Method

Example: (1) $T(n) = T(n - 1) + C$

$T(1) = C$

Topic: Solving Recurrence Relation

Substitution Method

Example: (1) $T(n) = T(n-1) + C$

$T(1) = C$

n size on problem n – 1 size x1 convert them

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

$[T(n-2) + C] + C$

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

$= T(n-3) + 3C$

$T(n) = T(n-k) + kC$

$\therefore n - k = 1$

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

$= C + (n-1)C$

$T(n) = O(n)$

Example: (2) $T(n) = T(n-1) + a \cdot n$

$T(1) = C$

Example: (2) $T(n) = T(n-1) + a \cdot n$

$T(1) = C$

Solution:

$$\therefore T(n) = T(n-1) + a \cdot n$$
$$= [T(n-2) + a \cdot (n-1)] + a \cdot n$$
$$= [T(n-3) + a \cdot (n-2)] + a(n-1) + a \cdot n$$
$$= T(n-3) + (n-2) \cdot a + (n-1) \cdot a + n \cdot a$$
$$= T(n-k) + a(n-k+1) + a(n-k+2) + \ldots + a(n-k+k)$$

$\therefore n - k = 1$

$$T(n) = T(1) + T(2) + a(3) + a(4) + \ldots + a(n-1) + a(n)$$
$$= C + a(2) + (3)a + 4(a) + \ldots + (n-1)a + (n) \cdot a$$
$$= a[1 + 2 + 3 + \ldots + n]$$
$$= O(n^2)$$

Example: (3) $T(n) = T(n/2) + C$

$T(1) = 1$

Example: (3) $T(n) = T(n/2) + C$

$T(1) = 1$

Solution:

$$
\begin{aligned}
T(n) &= T(n/2) + C \\
&= [T(n/2^2) + C] + C \\
&= T(n/4) + 2C \\
&= T(n/2^3) + 3C \\
T(n) &= T(n/2^k) + kC \\
&= (n/2^k) = 2 \\
T(n) &= T(2) + (\log_2 n - 1)\, C \\
&= 1 + (\log_2 n - 1)\, C \\
&= O(\log n)
\end{aligned}
$$

Example: (4) $T(1) = 1$

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

Example: (4) $T(1) = 1$

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

Solution:

$$T(n) = 2\left[2T\left(\frac{n}{2^2}\right) + C\right] + C$$

$$= 2^2T\left(\frac{n}{2^2}\right) + 2^2C + C$$

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + C\right] + 2C + C$$

$$= 2^3T\left(\frac{n}{2^3}\right) + 2^2C + 2C + C$$

$$= 2^kT\left(\frac{n}{2^k}\right) + 2^{k-1}C + 2^{k-2}C + \cdots + 21 \cdot C + C$$

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

$$\rightarrow T(n) = nT(1) + 2^{k-1}.C + 2^{k-2}.C + \ldots + 2C + C$$

$$= 2^k + 2^{k-1}.C + 2^{k-2} + \ldots + 2C + C$$

$$= 2^k + C(2^{k-1} + 2^{k-2} + \ldots + 2^1 + 2^0)$$

$$= 2^k + C$$

$$= 2^k + C(2^k - 1)$$

$$= 2^k + 2^k.C - C$$

$$= n.C$$

$$= O(n)$$

Master Method

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta\left(n^k(\log n)^P\right)$$

$a \geq 1, b > 1, k \geq 0, p$ = real number

If $a > b^k$ or $\log_b a > k$

$T(n) = \Theta\left(n\log_b a\right)$

Case I : O

Case II : Θ

Case III : Ω

Question 1: $T(n) = 2T\left(\frac{n}{2}\right) + (n)^0 \log n$

Solution: $a = 2, b = 2, k = 0$

$a > b^k; 2 > 2^0; 2 > 1$

$T(n) = \Theta(n)$

Question 2: $T(n) = 2T\left(\frac{n}{2}\right)+n$: n is it $O(n^{1-\epsilon})$ $\epsilon > 0$ ✗

Solution: [redacted]

n is it $\Theta(n(\log n)^k)$ $k=0$

$\therefore T(n)$ is $\Theta(n \cdot \log n)$

Question 3: $T(n)=2T\left(\frac{n}{2}\right) + n\log n$

Solution: [redacted]

$n \log n$ is it $O(n^{1-\epsilon})$ ✗

$n \log n$ is it $\Theta(n \cdot (\log n)^k)$ ✓

$k=1$

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

#Q. What will be the time complexity for the following recurrence relation?

$$T(n) = \begin{cases} 3T\left(\frac{n}{4}\right) + n^{1/2} & \text{for } n > 0 \\ 0 & \text{otherwise} \end{cases}$$

$n^{1/2}$ is it $O(n^{\log_4 3 - \epsilon})$ ✓

$\epsilon > 0$

A. $\theta(n^{\log_4 3})$ ✓

B. $\theta(n^{\log_2 5})$

C. $\theta(n^{\log_3 5})$

D. None

#Q. Let the recurrence relation

$$T(n) = \sqrt{2}\, T\left(\frac{n}{2}\right) + \log n$$

And the worst case time complexity is $O(n^r)$. Minimum possible value for r is _____.

$\Theta(n^{1/2}) \Rightarrow r = 1/2$

$\log_2 \sqrt{2} = 1/2 \log_2 2 = 1/2$

$\log n$ is $O(n^{1/2 - \epsilon})$ $\epsilon > 0$

$\therefore T(n)$ is $\Theta(n^{1/2})$

#Q. What is the time complexity of the following code snippet? Assume x is a global variable and "statement" takes O(n) time?

```
int x= θ;
int A (int n)
{
    if (n ==1)
    return 1;
    else
    {
        statement;
        x = 8*A(n/2)+ n^4;
        return x;
    }
}
```

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

$= a + n + T(n/2) + b$

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

A. $\theta(n)$

B. $\theta(n^2)$

C. $\theta(n^3)$

D. None

#Q. The running time of an algorithm T(n) for an input of size n.

$T(n) = 3T(n-1) + 1$ for $n \geq 2$

$T(1) = 1$

A. $\theta(3^n)$

B. $\theta(3\log n)$

C. $\theta(n^3)$

D. $\theta(n^2 \log n)$

Recursion Tree Approach

(1) $T(1) = 2T\left(\frac{n}{2}\right) + C$

$T(1) = C$

$\frac{n}{2^k} = 1 \rightarrow n = 2^k \rightarrow k = \log_2 n$

Total Work done $= C + 2C + 2^2C + 2^3C + \ldots + 2^kC$

$= C(1 + 2 + 2^2 + \ldots + 2^k)$

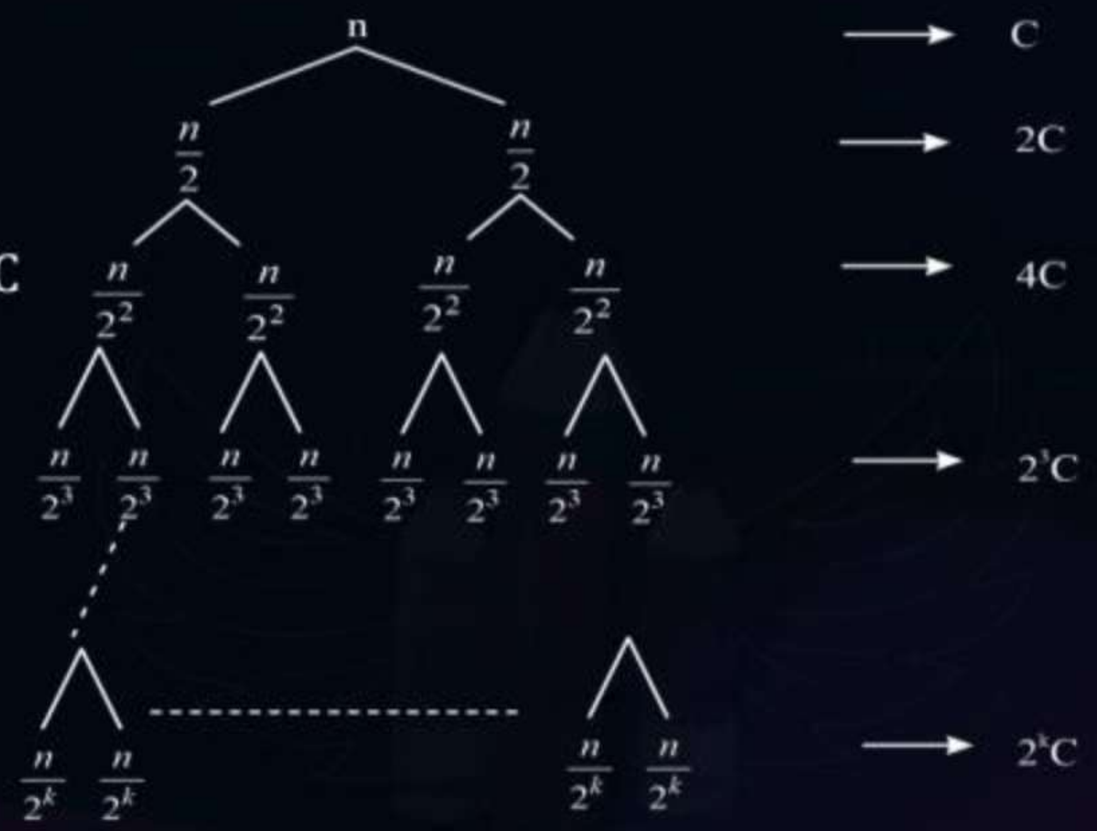
$= C\left(\frac{2^{k+1}-1}{2-1}\right)$

$= C(2^{k+1} - 1)$

$= C(2 \cdot 2^k - 1)$

$= C(2n - 1)$

$= O(n)$



(2) $T(1) = 2T\left(\frac{n}{2}\right) + C$

$T(1) = C$

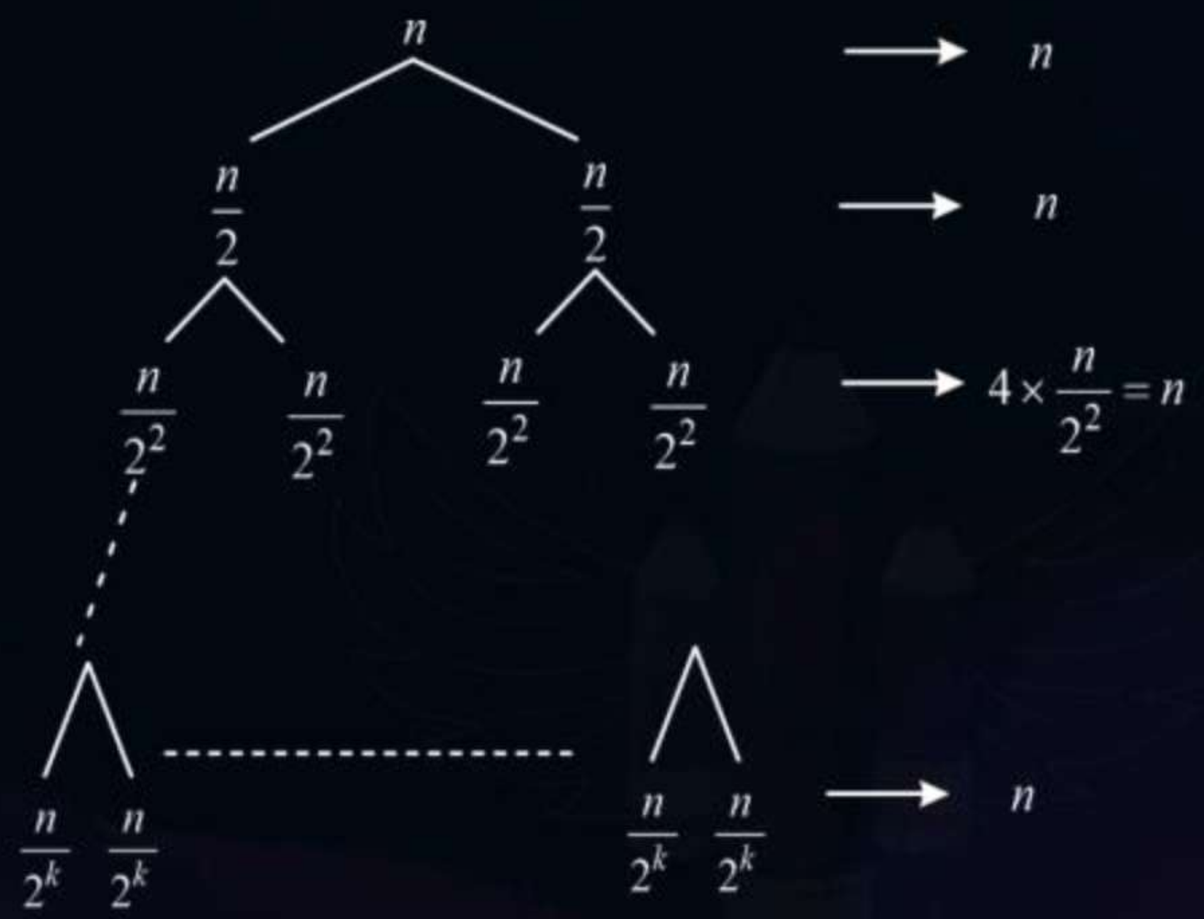
$\frac{n}{2^k} = 1;\ n = 2^k\ ;\ k - \log_2 n$

$\therefore n + n + n + \ldots + n$

$= k\,n$

$= n\ n\log_2 n$

$= O(n\log_2 n)$



(3) $T(1) = 4T\left(\frac{n}{2}\right) + C$

$n = 2^k \,;\, k = \log_2 n$

$= n + 4\left(\frac{n}{2}\right) + 4^2\left(\frac{n}{2}\right) + \cdots + 4^k\left(\frac{n}{2}\right)$

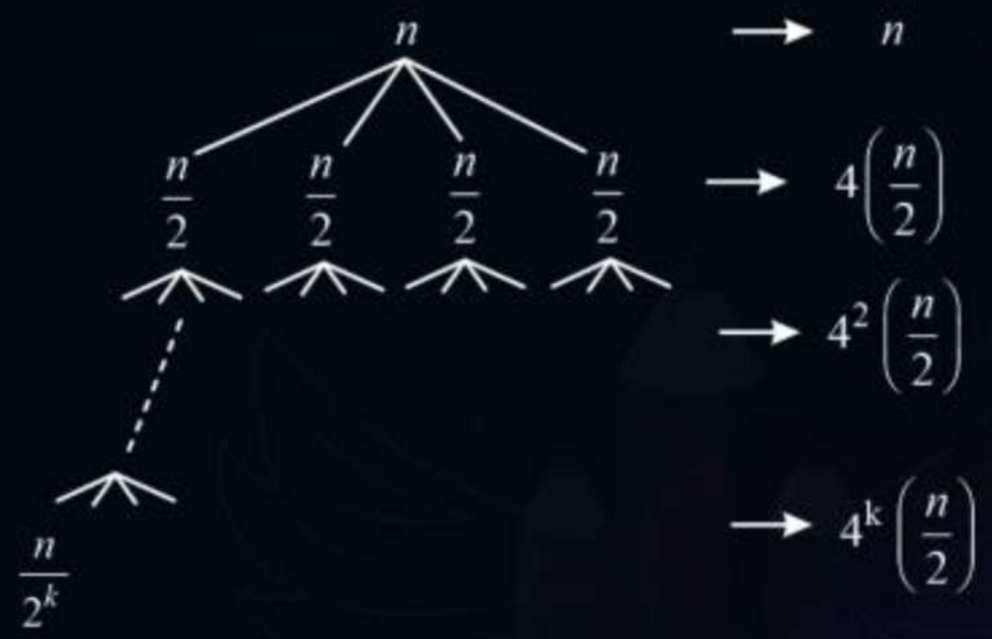
$= n\,[1 + 2 + 2^2 + 2^3 + .. + 2^k]$

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

$= n(2 \cdot 2^{k-1})$

$= n(2n) - 1$

$= O(n^2)$



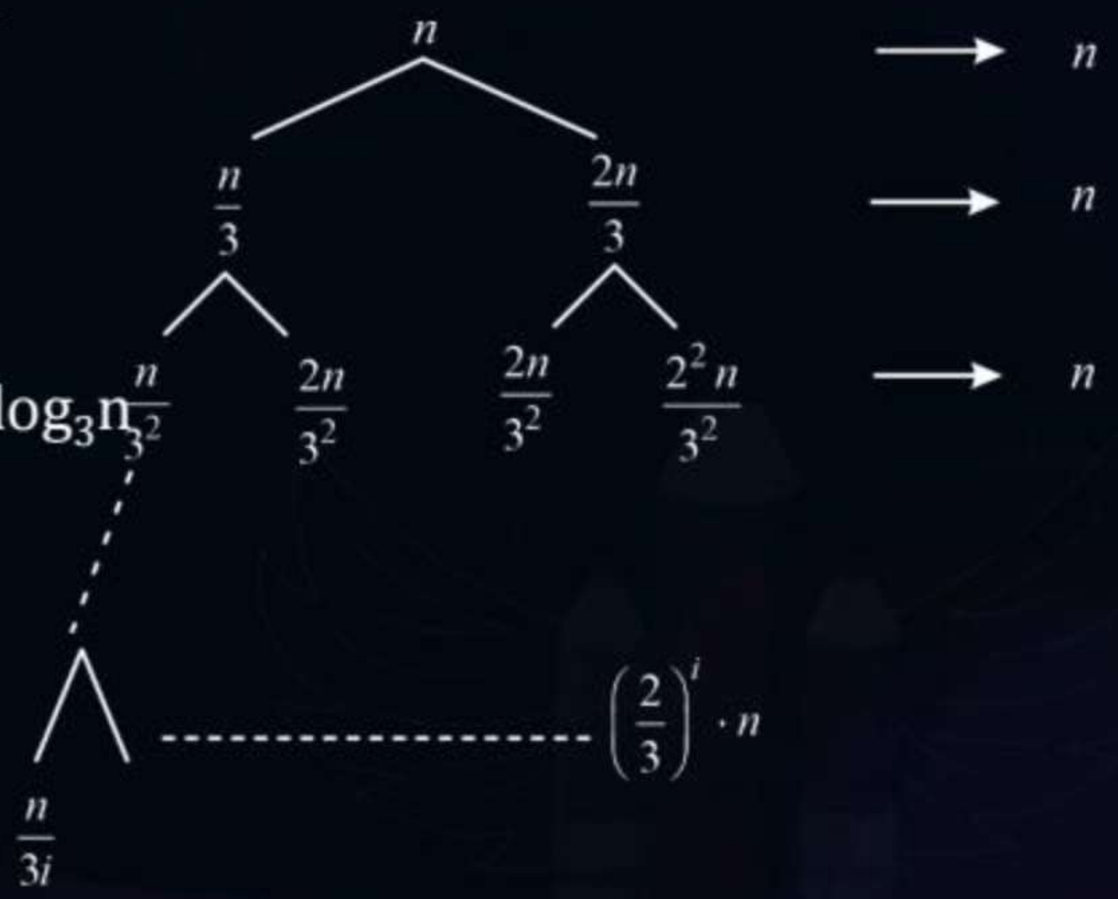
(4) $T(1) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n\,T(1)=1$

$\frac{n}{3i}$ -1; n - 3i; i - $\log_3 n$

$= n + n + \ldots + \log_3 n$

$= (n + n + n + \ldots + \log_3 n) \geq n + \ldots + \log_3 n$

$\Omega(n\log_{3/2} n)$



Topic: Recurrence Relations and their Time Complexity

$T(n) = C; n = 2$ $T(n) = 2\,T(\sqrt{n}) + C; n > 2$	$O(\log n)$
$T(n) = C; n = 2$ $T(n) = T(n-1) + C; n > 2$	$O(n)$
$T(n) = C; n = 1$ $T(n) = T(n-1) + n + C; n > 2$	$O(n^2)$
$T(n) = C; n = 1$ $T(n) = 2T(n-1) + C; n > 1$	$O(2^n)$
$T(n) = C; n = 1$ $T(n) = 2T\left(\frac{n}{2}\right) + C; n > 1$	$\theta(n)$
$T(n) = C; n = 1$ $T(n) = 2T\left(\frac{n}{2}\right) + n; n > 1$	$\theta(n\log n)$
$T(n) = C; n = 1$ $T(n) = T\left(\frac{n}{2}\right) + C; n > 1$	$\theta(\log n)$
$T(n) = 1; n = 2$ $T(n) = T(\sqrt{n}) + C; n > 2$	$\theta(\log\log n)$

Topic: Space Complexities

Space Complexity S(P)

C

Input space

Input space

+

Sp

Work space

Auxiliary Space

Additional space for computation steps like stack, link list etc.

$$S(P) = C + Sp$$

```
int n, A[n];
Algorithm Rsum(A, n)
{
    if (n = 1) return (A(1));
    else
    return (A[n] + RSum(A,(n−1));
}
```

Time: $T(n) = T(n-1) + c$

Space: Recursion Stack

$= O(n)$

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

Recursion Stack $= O(k)$

$k = \log n$

$O(\log n)$

Space Complexity

Space complexity will depend on number of activation record pushed into the stack

Suppose, n = 16

A (1)	For n = 2K we are pushing the 'K' activation record
A (2)	
A (4)	
A (8)	
A (16)	

$\therefore$ Space Complexity

$$n = 2K$$

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

$$K = \log_2 n$$

Space Complexity = O(logn)

Example 3:

```
Algorithm A(n)
{
    if (n = 2) return;
    else
    return (A(√n));
}
```

Solution:

$T(n) = 1; n = 2$

$T(n) = T(\sqrt{n}) + C; n > 2$

Time Complexity =

Practice Problems

#Q. Consider functions Function 1 and Function 2 expressed in pseudocode as follows:

```
Function_1
while n > 1, do
    for i = 1 to n do
        x = x + 1;
    end for
    n ← n/2
end while
```

$O(n)$

U.F

$n + n/2 + n/4$

```
Function_2
    for i = 1 to 100 * n do
        x = x + 1;
    end for
```

$O(n)$

Let $f_1(n)$ and $f_2(n)$ denote the number of times the statement "x = x + 1" is executed in Function 1 and Function 2, respectively.
Which of the following statements is/are TRUE?

A. $f_1(n) \in \Theta(f_2(n))$

B. $f_1(n) \in o(f_2(n))$

C. $f_1(n) \in w(f_2(n))$

D. $f_1(n) \in O(n)$

#Q. Which one of the following statement is TRUE for all positive functions $f(n)$?

A. $f(n^2) = \theta\,(f(n)^2)$, where $f(n)$ is a polynomial ✓ n^2

B. $f(n^2) = o\,(f(n)^2)$ ✗ $f(n) = n$

C. $f(n^2) = O\,(f(n)^2)$, where $f(n)$ is an exponential function ✗ e^n

D. $f(n^2) = \Omega\,(f(n)^2)$ ✗

#Q. Consider the following C function

```
int fun (int n) {
   int i, j;
     for (i = 1; i < = n; i ++) {
          for (j = 1; j < n; j + = i) {
               printf ("%d  %d", i, j);
               }
          }
     }
```

Asymptotic Notation of fun in terms of 0 notation is

A. $\theta(n\sqrt{3})$

B. $\theta(n^2)$

C. $\theta(n\log n)$ ✓

D. $\theta(n^2\log n)$

U.F :

$$\frac{n}{1}+\frac{n}{2}+\frac{n}{3}+\frac{n}{4}+\cdots+\frac{n}{n}$$

$$n\left(\sum_{x=1}^{n} 1/x\right)\log n$$

$$\Rightarrow O(n\cdot\log n)$$

#Q. Consider the following C function: —-

```
int fun1(int n) {
    int i, j, k, p, q = 0;
    for (i=l; i<n; ++i) {
        p=0;
        for (j = n; j > 1 ; j = j/2)
            ++p;
        for (k=l; k<p; k=k*2) ++q;
    }
    return q;
}
```

n

Log n

Log Log n

Which one of the following most closely approximates: the return value of the function fun1

A. n^3

B. $n(\log n)^2$

C. $n\log n$

D. $n\log(\log n)$

c is it $O(n^{\log_2 3 - \epsilon})$ ✓

#Q. The solution to the recurrence equation $T(n) = 3.T(n/2) + 1$, $T(1) = 1$ is-

A $O(2^{\log_2 n})$

B $O(3^{\log_2 n + 1} - 2)$

C $O(3^{\log_2 n})$ ✓

$n^{\log_2 3}$

D $O(2^{\log 3} . \log n)$

#Q. $f(n) = \sum_{i=1}^{n} i^{\frac{1}{2}}$ is____.

A $\theta(n)$

B θ (nlogn)

C $\theta\left(n\sqrt{n}\right)$

D $\theta\left(\sqrt{n}\right)$

#Q. Let (n) and A(n) respectively denote the best case and average case running time of an algorithm of input size 'n' which of the following is always false?

B(n)

A. $B(n) = \Omega(A(n))$

B. $B(n) = \theta(A(n))$

C. $B(n) = \omega(A(n))$

D. $B(n) = O(A(n))$

$B(n) \le A(n)$

2. DIVIDE AND CONQUER

Topic: Finding Maximum-Minimum element

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

a → no. of Subproblems

b → Size of Subproblems

Recurrence Relation:

$$T(n) = \begin{cases} 1 \; if \;\; n = 2 \\ 2T\left(\frac{n}{2}\right) + 2; n > 2 \end{cases}$$

$O(n)$

Time Complexity:

T(n) = O(n)

Time complexity is same for every case (Best case/Worst case).

Space Complexity:

(Space Complexity)

- Input space O(n)
- Work space O(log n) (Stack Space)

Space Complexity = O(logn)

Number of comparisons to find maximum / minimum element on an given array of n elements:

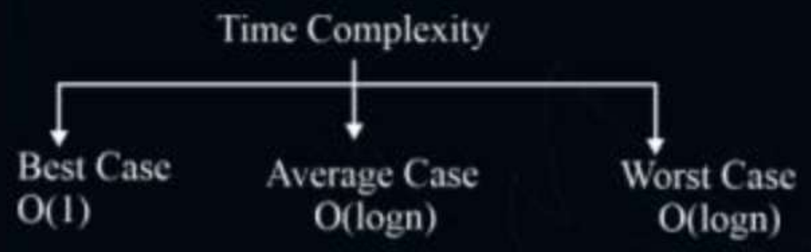
Comparison = $\frac{3n}{2} - 2$ $= O(n)$

Topic: Binary Search

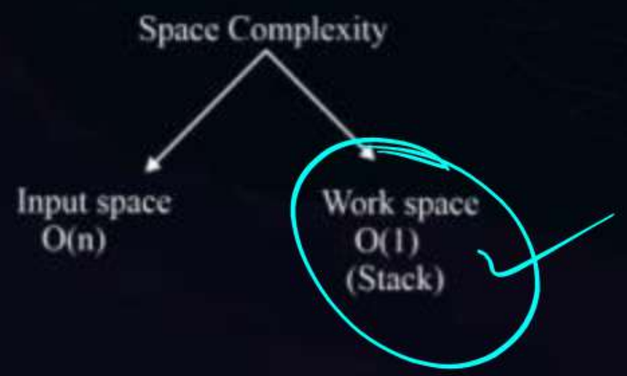
Given a sorted array and an element x, need to return the index of element x if it is present then 1, otherwise – 1.

Recurrence relation: $T(n) = \begin{cases} 1; & n = 1 \\ T\left(\frac{n}{2}\right) + C; & n > 1 \end{cases}$

Time Complexity:



Space Complexity:



Space complexity = O(1)

Topic: Merge Algorithm

→ Combine | Conquer

$L_1 \langle 2, 3, 4 \rangle$ $L_2 \langle 6, 10, 12, 15 \rangle$

$\langle 2, 3, \cdots 4, 6, 10, \cdots 15 \rangle$

Merging two sorted sub arrays of input size m,n.

Number of comparisons to merge two sorted sub arrays of size m,n.

$n_1 \le n_2$

Comparisons = m + n – 1 (worst case)

Number of moves =m + n (Outplace Algorithm)

$L_1(n_1)$ $L_2(n_2)$

Time Complexity

- Best Case O(m+n)
- Average Case O(m+n)
- Worst Case O(m+n)

$Min(n_1)$

$Max(n_1+n_2-1)$ $L(n_1+n_2)$

Number of comparisons in best case of merging two sorted subarrays of size m, n.

comparisons = min (m, n)

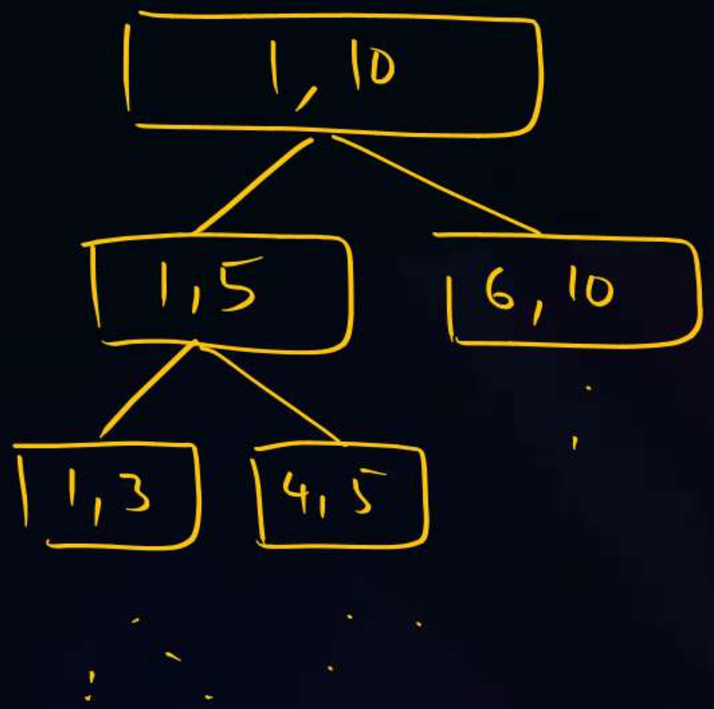
$O(n_1+n_2)$

Moves = m + n (Always)

Note:

Best Case comes in comparisons no effect on moves.

Topic: Merge Sort



Merge

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

$$\Theta(n \cdot \log n)$$

Merge Sort Algorithm

Outplace

- Time Complexity
  $T(n) = O(n\log n)$
  Every case (BC, WC, AC)
- Space Complexity
  - Input Space: n
  - Work Space: n + (logn)

  $S_p = O(n)$: Additional Array
- Recurrence relation

$$T(n) = \begin{cases} 1; & n = 1 \\ 2T\left(\frac{n}{2}\right) + n; & n > 1 \end{cases}$$

Inplace

- Time Complexity
  $T(n) = \Theta(n^2)$
- Recurrence relation

$$T(n) = \begin{cases} 1; & n = 1 \\ 2T\left(\frac{n}{2}\right) + c; & n > 1 \end{cases}$$

NOTE:

- In GATE exam if merge sort given then always consider outplace.
- If array size very large, merge sort preferable.
- If array size very small, then prefer insertion sort.
- Merge sort is stable sorting technique.

Topic: Quick Sort Algorithm

Partition-Exchange Sort

→ Pivoting

Quick Sort Algorithm

Best Case / Average Case

→ Recurrence Relation

$$T(n) = \begin{cases} 1; & n = 1 \\ 2T\left(\frac{n}{2}\right) + n + C; & n > 1 \end{cases}$$

→ Time complexity

T(n) = O(nlogn)

Worst Case

→ Recurrence Relation

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

→ Time Complexity

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

Example 1:

In Quick for sorting n elements, the smallest element is selected as pivot. what is the worst-case time Complexity?

$$T(n) = T(n-1) + a + O(n)$$

$$O(n^2)$$

Partitioning

Example 2:

The median of n elements can be found in O (n) time then, what is the time complexity of quick sort algo in which median selected as pivot?

$$T(n) = \underbrace{O(n)}_{\text{Median}} + 2 \cdot T(n/2) + \underbrace{O(n)}_{\text{Partitioning}}$$

$O(n \log n)$

Example 2:

The median of n elements can be found in O (n) time then, what is the time complexity of quick sort algo in which median selected as pivot?

Solution: $T(n) = \underbrace{O(n)}_{\text{Find median}} + \underbrace{C}_{\text{Swap median with last}} + \underbrace{O(n)}_{\text{Partition algo}} + T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$

$$= 2T\left(\frac{n}{2}\right) + C \cdot n$$

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

#Q. Suppose that the splits at very level of quick sort are in the proportion $1-\alpha$ to α, where $0<\alpha\leq\frac{1}{2}$ is a constant. Minimum depth of a leaf in the recursion tree is approximately:

$\alpha = 1/3 \quad (1-\alpha) = 2/3$

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

A. $\frac{\log(n)}{\log(\alpha)}$

B. $\text{Log}(n\alpha)$

C. $n \log n$

D. $-\frac{\log(n)}{\log(\alpha)}$

$T((\alpha)^K \cdot n)$

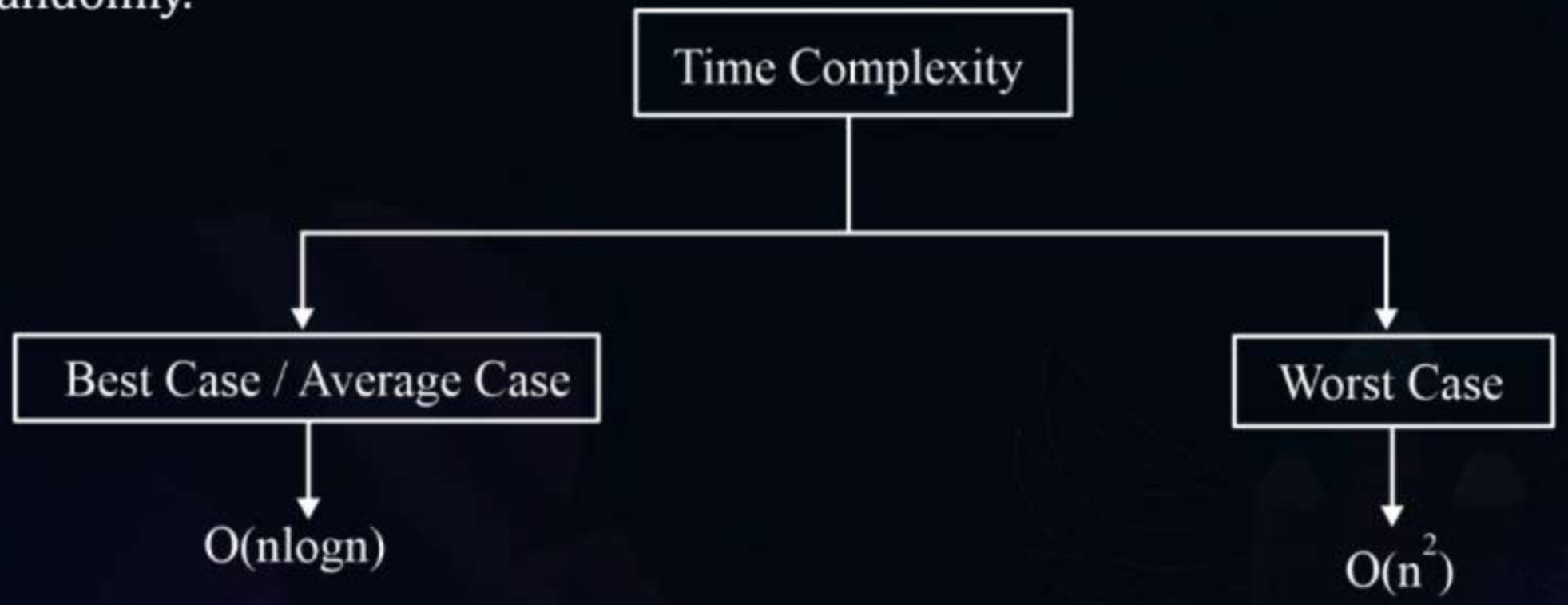
$(\alpha)^K \cdot n = 1$

$(\alpha)^K = 1/n = n^{-1}$

$K \cdot \log\alpha = -1 \cdot \log n$

$K = -\frac{\log n}{\log\alpha}$

- In Randomized quick sort algorithm selection of pivot element can be taken randomly.



Topic: Strassen's Matrix Multiplication

Without Divide and Conquer

- Matrix Addition: $O(n^2)$
- Matrix multiplication: $O(n^3)$

According to Strassen's

- Time Complexity: $O(n^{2.81})$
- Recurrence relation: $7T\left(\frac{n}{2}\right)+n^2$

Practice Problems

#Q. Which of the following correctly describes the Recurrence relation for the standard Binary Search Algorithm on a sorted array of n numbers where c is a constant.

A. $T(n) = 2*T(n/2) + c$

B. $T(n) = T(n/2)$

C. $T(n) = T(n-1) + c$

D. $T(n) = T(n/2) + c$

#Q. Consider the following program.

```
int func (int n)
{
    if(n<=l)
        return n;
    else
        return (3*func(n-3) - 3*func(n-2));
    end
}
```

The running time of the above function is:

A. $\Theta(n)$

B. $\Theta(n^2)$

C. $\Theta(3^n)$

D. $\Theta(2^n)$

c_1 c_2 of n

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

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

$= T(n-3) + T(n-2) + d$

$T(n) < 2 \cdot T(n-2) + d$

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

$O(2^n)$

#Q. The time complexity to multiply two long integers of size n- digit represented in an array of size n, using divide & conquer strategy with 3-way split is-

A. $O(n)$

B. $O(n\log n)$

C. $O(n^{1.32})$

D. $O(n^2)$

#Q. Match the pairs

A	$O(\log n)$	(P)	Merge sort
B	$O(n)$	(Q)	Binary Search
C	$O(n\log n)$	(R)	Insertion sort
D	$O(n^2)$	(S)	Partitioning / pivoting

#Q. 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_____.

$$\left[\frac{1}{25} + \frac{1}{25}\right] = 2/25 = 0.08$$

#Q. Which one of the following is the recurrence equation for the worst-case time complexity of the Quicksort algorithm for sorting n (≥2) numbers? In the recurrence equations given in the options below, c is a constant.

A. $T(n) = 2T(n/2) + cn$

B. $T(n) = T(n-1) + T(1) + cn$

C. $T(n) = 2T(n-1) + cn$

D. $T(n) = 2(n/2) + cn$

#Q. Let P be a quick sort program to sort numbers in ascending order using the first element as the pivot. Let t1 and t2 be the number of comparisons made by P for the inputs [1 2 3 4 5] and [4 1 5 3 2] respectively. Which one of the following holds?

W.C t_1 > B.C t_2

A $t_1 = 5$

B $t_1 < t_2$

C $t_1 > t_2$

D $t_1 = t_2$

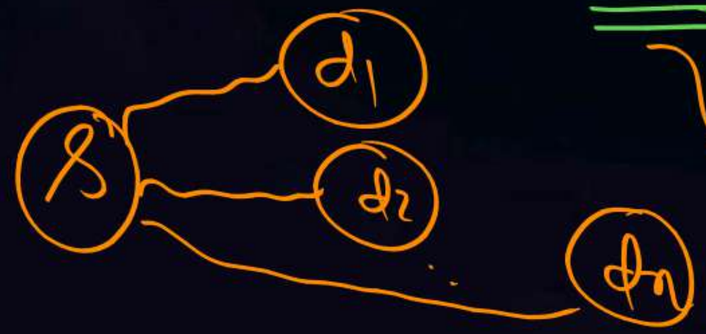
3. SHORTEST PATH PROBLEM

Graph based Problem:

- Directed / undirected
- weighted
  - weights of edges can be +ve / -ve;

(i) Single pair Shortest Path

(ii) Single Source Shortest Paths



Broadcasting

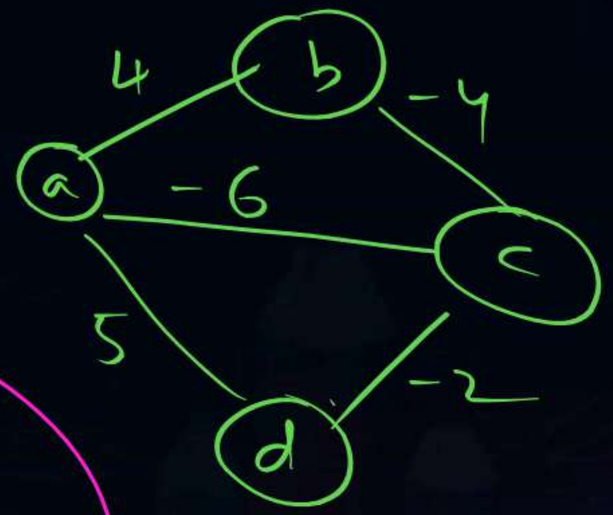
- Dijkstra's Algo (Greedy Method)
- Bellman Ford (Dynamic Prog.)

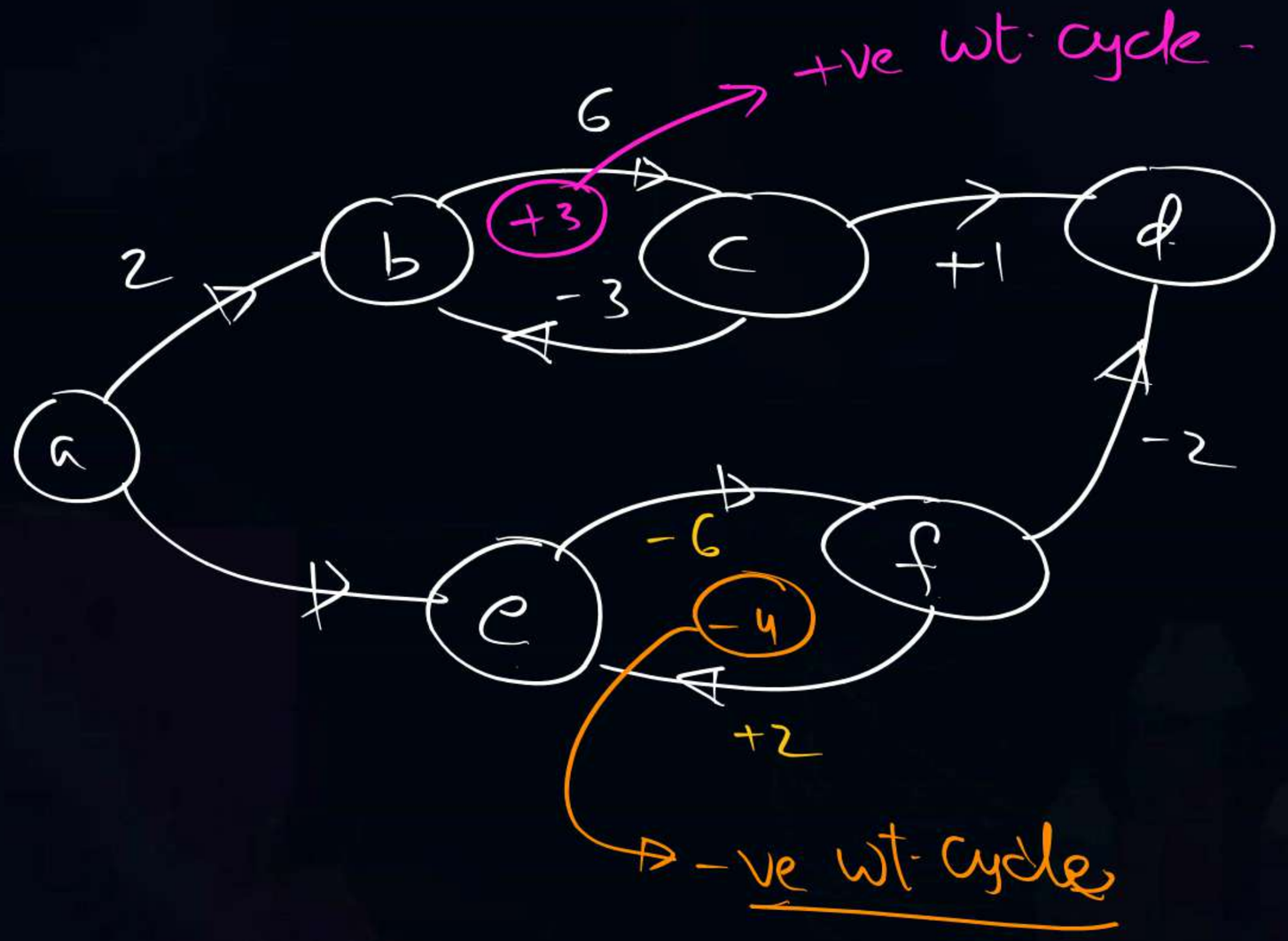
→ Limitation of Dijkstra's Algo →

May Not work with -ve wt edges

→ Go for Bellman-Ford [D.P]

Always work for graphs having -ve wt edges, but No -ve wt cycle




+ve wt. cycle
6
+3
2
b
c
-3
+1
d
a
-2
-6
f
e
-4
+2
-ve wt. cycle

3) All-pairs Shortest Paths → Flyod-warshalls

$i \rightarrow j$

for all $i = 1, n$

for all $j = 1, n$

Dynamic Programming

Topic: Single Source Shortest Path

Dijkstra Algorithm

- Using min heap & adjacency list = $O(E + V)\log V$
- Using adjacency Matrix & Min heap = $O(V^2 E\log V)$
- Using adjacency list & Unsorted Array = $O(V^2)$
- Using adjacency list & Sorted Doubly Linked List = $O(EV)$

Bellman-Ford

- Time Complexity = $O(EV)$
- If negative edge weight cycle then for some vertices Incorrect answer.

#Q. The Floyd-Warshall algorithm for all-pair shortest paths computation is based on

A. Greedy paradigm

B. Divide-and-Conquerparadigm.

C. Dynamic Programing paradigm.

D. Neither Greedy nor Divide-and-Conquer nor Dynamic Programming paradigm.

4. GRAPH ALGORITHMS

D.F.S

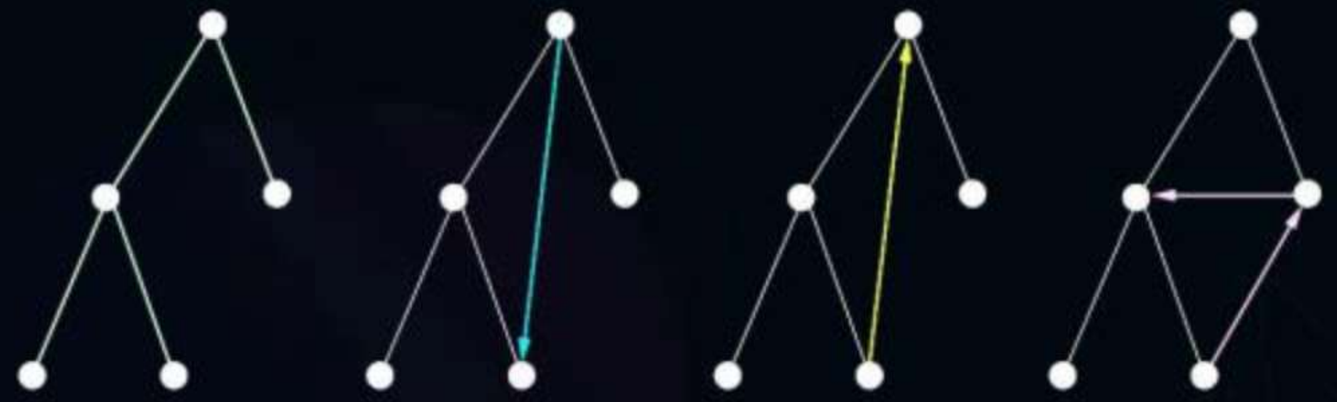
B.F.S

Topic : Graph Techniques

Edge Classification for DFS

<Directed Graph>

Every edge is either

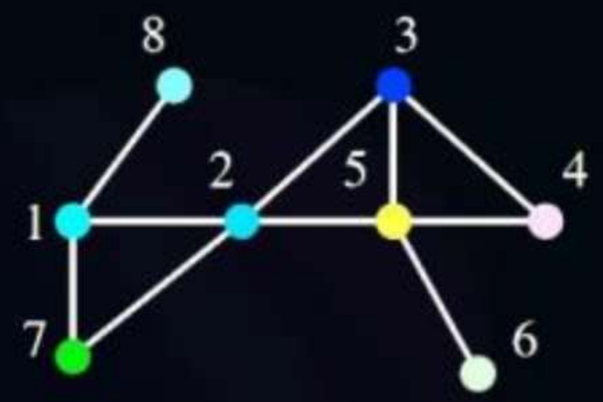


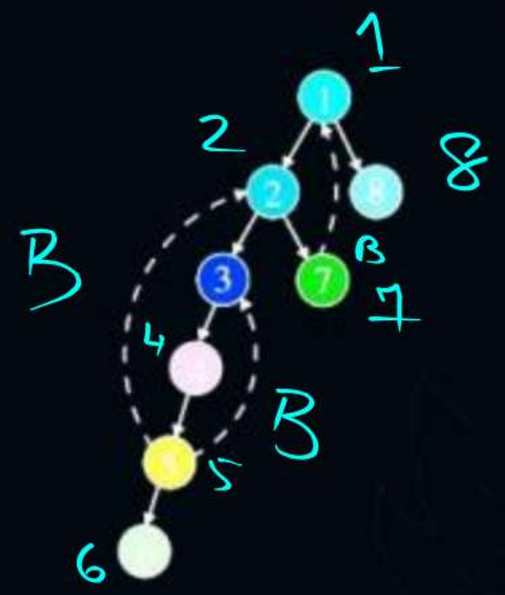
(i) Tree edge
(ii) Forward Edge
(iii) Back Edge
(iv) Cross Edge

On any particular DFS or BFS of a directed or undirected graph, each edge gets classified as one of the above.

Topic : Graph Techniques

The key Idea with DFS





A depth-first search of a graph organizes the edges of the graph in a precise way. In a DFS of an undirected graph, we assign a direction to each edge, from the vertex which discover it:

DFS: Tree Edges and Back Edges Only

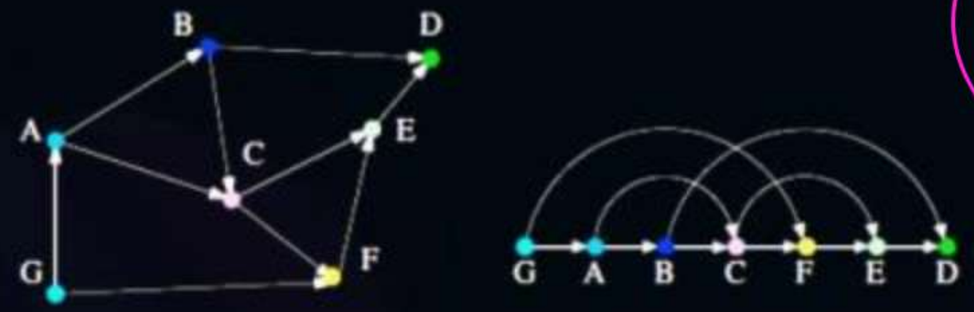
The reason DFS is so important is that it defines a very nice ordering to the edges of the graph.

In a DFS of an undirected graph, every edge is either a tree edge or a back edge. Why? Suppose we have a forward edge. We would have encountered (4,1) when expanding 4, so this is a back edge.

Topological Sorting [Appl. of D.F.S] : D.A.G : Linear order

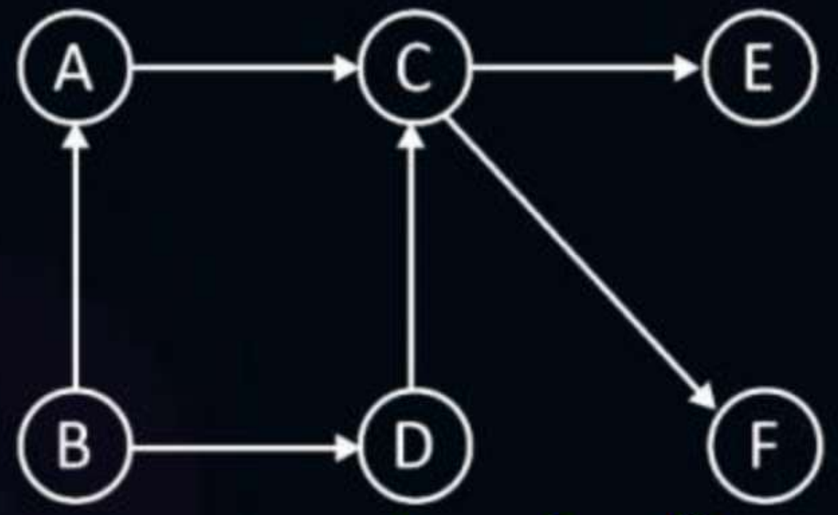
A directed, acyclic graph has no directed cycles.

IMPORTANT TOPIC



A topological sort of a graph is an ordering on the vertices so that all edges go from left to right.

DAGs (and only DAGs) has at least one topological sort (here G,A,B,C,F,E,D).



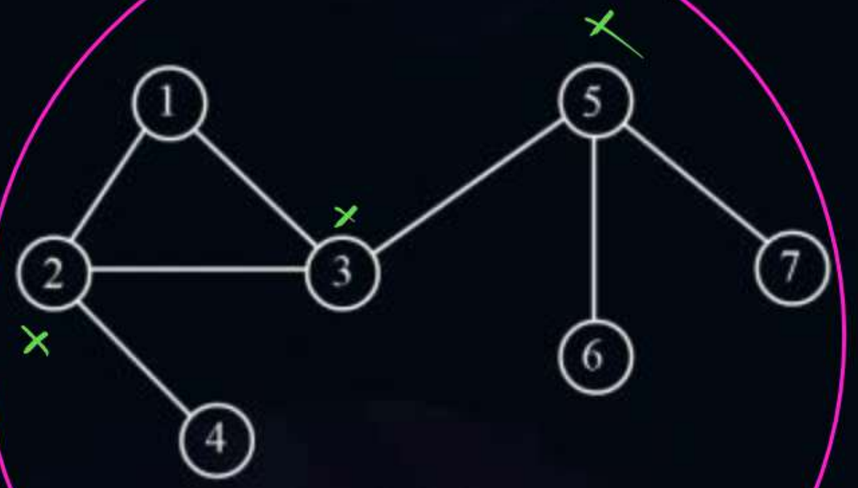
B — A — D — C — E — F — 1

B — A — D — C — F — E — 2

B — D — A — C — E — F — 3

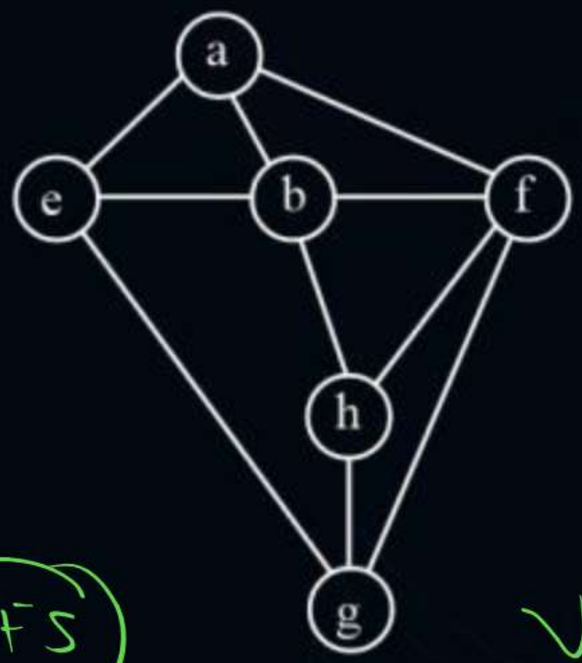
B — D — A — C — F — E — 4

#Q.



No. of Articulation Pts.

3



Valid

DFS

A a b e g h f

B a b f e h g

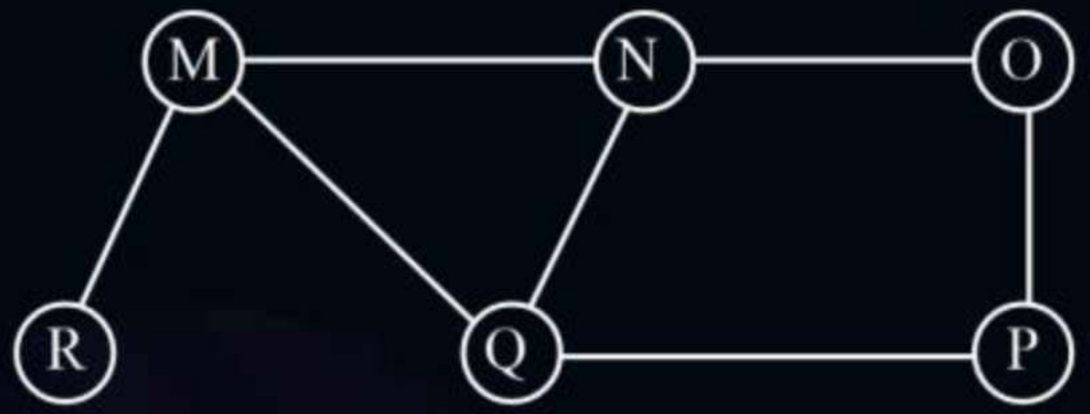
C A b f h g e

D a f g h b e

DFS

#Q.

BFS



Valid

QMO

A M N O P Q R

B N Q M P O R

C Q M N P R O

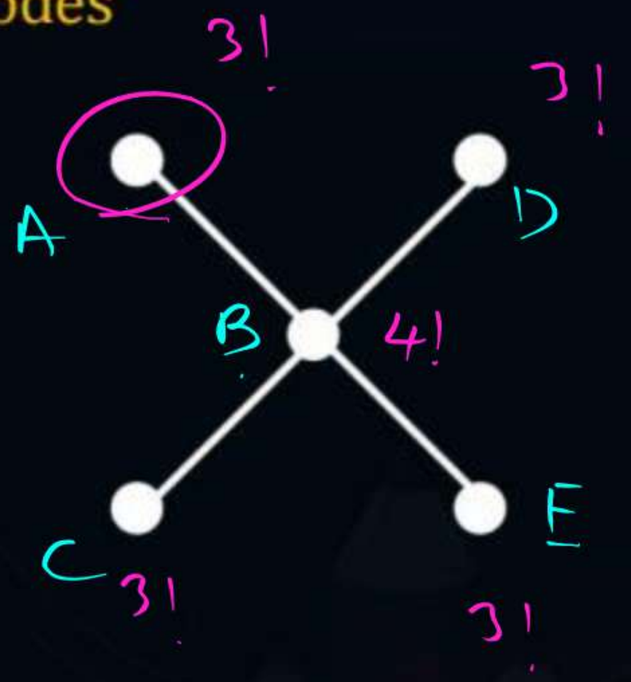
D Q M N P O R

MNPR

MNP

#Q. Consider the following undirected graph on 5 nodes

Assume you are performing Breadth First Search on this graph using a Queue Data Structure. How many unique Breadth First orderings are possible on this graph?



AB (3!)

$4 * 3! + 4! = 48$

A 9

B 24

C 48

D 120

t||w

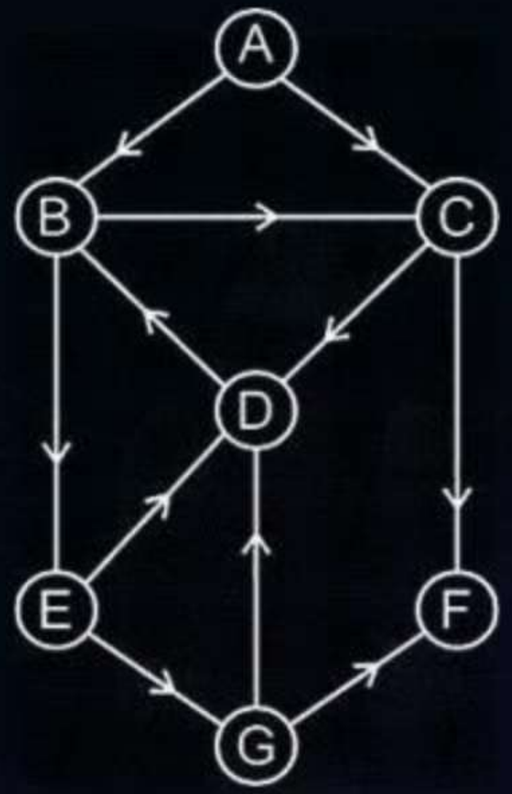
#Q. Consider the given graph. If DFS algorithm runs from vertex A and adjacency are chosen based on lexicographic order of vertex labels. Then the number of possible tree edges a, back edges b, cross edges f and forward edges g are:

A. a = 6, b = 2, g = 3, f = 4

B. a = 6, b = 1, f = 1, g = 4

C. a = 6, b = 1, f = 3, g = 1

D. a = 3, f = 2, g = 1, b = 2



#Q. Number of topological sort possible form the given graph is _____.

$2! \times 2! \times 2! = 8$

#Q. Which of the following option(s) is/are correct? MSQ

(A+C+D)

A. Time complexity of Breadth first search and Depth first search are same as O(V+E) in case of adjacency list.

B. Time complexity of Breadth first traversal and Depth first traversal are same as O(V) in case of square matrix.

C. Space complexity of Breadth first search and Depth first search are same as O(V).

D. Time complexity of ~~Breadth~~ Depth first search and Breadth First traversal are same as $O(v^2)$ in case of adjacency matrix.

5. SORTING METHODS

Topic : Sorting

- Sorting is the basic building block that many other algorithms are built around. By understanding sorting, we obtain an amazing amount of power to solve other problems.
- Most of the interesting ideas used in the design of algorithms appear in the context of sorting, such as divide-and-conquer, data structures, and randomized algorithms.
- Computers have historically spent more time sorting than doing anything else. A quarter of all mainframe cycles were spent sorting data [Knu98]. Sorting remains the most ubiquitous combinatorial algorithm problem in practice.
- Sorting is the most thoroughly studied problem in computer science. Literally dozens of different algorithms are known, most of which possess some advantage over all other algorithms in certain situations.

Applications of Sorting

- **Searching:** Binary search whether an item is in a dictionary in $O(\log n)$ time, provided the keys are all sorted. Search preprocessing is perhaps the single most important application of sorting.

- **Closest pair:** Given a set of n numbers, how do you find the pair of numbers that have the smallest difference between them? Once the numbers are sorted, the closest pair of numbers must lie next to each other somewhere in sorted order. Thus, a linear-time scan through them completes the job, for a total of $O(n \log n)$ time including the sorting.

Applications of Sorting

- **Element uniqueness:** Are there any duplicates in a given set of *n* items? This is a special case of the closest-pair problem above, where we ask if there is a pair separated by a gap of zero. The most efficient algorithm sorts the numbers and then does a linear scan through checking all adjacent pairs.

- **Frequency distribution:** Given a set of *n* items, which element occurs the largest number of times in the set? If the items are sorted, we can sweep from left to right and count them, since all identical items will be lumped together during sorting.

Topic : Sorting

Applications of Sorting

- **Selection:** What is the kth largest item in an array? If the keys are placed in sorted order, the kth largest can be found in constant time by simply looking at the kth position of the array. In particular, the median element appears in the (n/2)nd position in sorted order.
- **Convex hulls:** What is the polygon of smallest area that contains a given set of *n* points in two dimensions? The convex hull is like a rubber band stretched over the points in the plane and then released. It compresses to just cover the points. The convex hull gives a nice representation of the shape of the points and is an important building block for more sophisticated geometric algorithms, as discussed in the catalog in

Take-Home Lesson: Sorting lies at the heart of many algorithms. Sorting the data is one of the first things any algorithm designer should try in the quest for efficiency.

Time Complexity of Comp. based Sort

: [No. of Comp's + No. of Swaps]

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

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

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

(iv) $n\log n + n^2 = O(n^2)$

Topic : Sorting

```
Algorithm BUBBLE_SORT (A, n)
{
    1.  for pass ← n down to 1
        {
            2.  for i ← 1 to (pass –1)
                {
                    if (A[i] > A[i + 1])
                    {
                        swap(A[i], A[i + 1]);
                    }
                }
        }
}
```

Sorting Tech.

- Comparison
  - B.S
  - S.S
  - I.S
  - M.S
  - Q.S
  - H.S
- Non-Comparison
  - (Radix Sort)

Topic: Sorting

1. Find the smallest element in the list.
2. Swap it with the value in the current with position.
3. Repeat this process for all elements until the array is sorted.

```
Algorithm SELECTION_SORT (A, n)
{
        1. for i ← 1 to n – 1 do
           {
               min ← i;
               for j ← i + 1 to n
               {
                   if (A[j] < A[min]) then
                   {
                       min ← j;
                   }
               }
                swap(A[min], A[i];
           }
}
```

Topic: Sorting

```
Algorithm INSERTION_SORT (A, n)
{
   1.   for j ← 2 to n
        {
            key ← A[j];
            i ← j –1;
            while (i > 0 and A[i] > key) {
                    A[i + 1] ← A[i];
                    i ← i – 1;
            }
           A[i + 1] ← key;
        }
}
```

Topic: Comparison Based Sorting Algorithms

Sorting Algorithm	Basic logic of sorting Algo	BC	AC	WC	Stable sorting	Inplace sorting
Quick sort	Choose pivot element place in correct position	o(nlogn)	o(nlogn)	o(n2)	No	Yes
Merge sort	Divide to equal parts recursively sort each sub part & marge them	o(nlogn)	o(nlogn) = n logn	o(nlogn) = n log n	Yes	No
Heap sort	Build heap(max) delete max place	o(nlogn)	o(nlogn)	o(nlogn)	No	Yes
Bubble sort	Compare exchange	o(n)	o(n2)	o(n2)	Yes	Yes
Selection sort	Find position of min element from [1 to n]	o(n2)	o(n2)	o(n2)	No	Yes
Insertion sort	Insert a [i + 1] into correct position	o(n)	o(n2)	o(n2)	Yes	Yes

#Q. A sorting technique is called stable if

A. it takes O(n log n) time

B. it maintains the relative order of occurrence of non-distinct elements

C. it uses divide and conquer paradigm

D. it takes 0(n) space

#Q. Assume that the algorithms consider 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. Merge sort 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

#Q. The worst-case running times of Insertion sort, Merge sort and Quick sort respectively are

n^2 $n\log n$ n^2

A. $\Theta(n\log n), \Theta(n^2), \Theta(n^2)$

B. $\Theta(n^2), \Theta(n\log n), \Theta(n\log n)$

C. $\Theta(n^2), \Theta(n\log n), \Theta(n^2)$

D. $\Theta(n^2), \Theta(n^2), \Theta(n^2)$

#Q. Consider the following array.

23	32	45	69	72	73	89	98

Which algorithm out of the following options uses the least number of comparisons (among the array elements) to sort the above array in ascending order? :

A. Selection sort n^2

B. Merges sort $n\log n$

C. Insertion sort n

D. Quicksort using the last element as pivot n^2

#Q. Choose the correct alternatives (more than one may be correct) and write the corresponding letters only: Following algorithm(s) can be used to sort n in the range [I... n^3] in O(n) time

A. Heap sort

B. Quick sort

C. Merge Sort

D. Radix sort

#Q. Let the numbers 794, 332, 561, 342, 200, 607 and 893 be sorted using RADIX SORT. What will be the sixth number in the sequence of numbers after sorting the second digit?

A. 561

B. 794

C. 893

D. 332

: 200, 561, 332, 342, 893, 794, 607

: 200, 607, 332, 342, 561, 794, 893

0	1	2	3	4	5	6	7	8	9
607 200			332	342		561			893 794

2 Min Summary

THANK - YOU