Muhammad Hamza
01-131232-057
BSE-4A

66 Design and Analysis of Algorithm >>

Home Work :-

GCD Algorithms

1. Start with two number (A, B).
2. Divide A by B and find the Reminder (A mod B)
3. Replace A with B and B with the Reminder.
4. Repeat Step 2 and 3 until Reminder become Zero.
5. The last non-zero reminder is the GCD.

Step by Step Solution :-

We apply Eculids Algorithms (60, 24)

Step 1. $60 \div 24 = 2$ → quotient

$|60 - |2 \times 24|| = 12$

Step 2.

$24 \div 12 = 2$ → quotient

$|24 - |2 \times 12|| = 0$

Step 3. $B = 0$

So we stop here because The Reminder is zero.

The Last non-zero Reminder is 12

$$\boxed{GCD(60, 24) = 12}$$ Ans

Euclidean Algorithm:-

→ Using for the computing the Greatest common divisor GCD.

→ also used for HCF.

We are finding the GCD through two Methods:

First Method:-

1.

- First Find Divisor
- Second Find the common Divisor
- Third Find the largest from the common divisor.

GCD (12, 33)

12 : 1, 2, 3, 4, 6, 12 → Divisor

33 = 1, 3, 11, 33 → Divisor

→ Now we find the common Divisor.

There are 2 common divisor 1, 3.

→ Now we find the Greatest From Common it is 3

GCD = 3.

2 GCD(60, 24)

Divisor :-

60 = 1, 2, 3, 4, 5, 6, 10, 12, 15, 20 30, 60

24 = 1, 2, 3, 4, 6, 8, 12, 24

Common Divisor :-

1, 2, 3 4, 6, 12

Greatest Divisor :-

12

GCD(60, 24) = 12 Ans

2nd Method :-

12, 33

Q	A	B	R
2	33	12	9
1	12	9	3
3	9	3	0
X	3	0	X

GCD(12, 33) = 3 Ans

ii) GCD (750, 900)

Q	A	B	R
1	900	750	150
5	750	150	0
X	150	0	X

```
1
150 | 900
      750
      ---
      150
        5
150 | 750
      750
      ---
        0
```

The GCD (750, 900) is 150.

Time Effeciency :-

That how much time take the algorithms is called time.

Depends :-

1) NO of input / Size.
2) Unit measuring Running Time - Basic Operations.

→ Basic Operation is called dominant Operation.

Framework of Analysis :-

1) Basic Operation.
2) input of size n.

O - notation :-

prove that :-

$100n + 5 \in O(n^2)$

Sol

Comparing this with standard

$t(n) \in O(g(n)) \quad \Rightarrow \quad 100n + 5 \in O(n^2)$

$t(n) = 100n + 5$

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

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

$100n + 5 \leq c \cdot n^2$

For what value of c, and n0 this relations holds true

$C = 105$ Locked value

n	$t(n)$ $100n+5$	n^2	$C \cdot g(n)$
1	105	1	105
2	205	4	420
3	305	9	945
4	405	16	1680

→ We multiply the n^2 to locked value 105 & find the $C \cdot g(n)$ values.

$C \cdot g(n)$

$105 \times 1 = 105$

$105 \times 4 = 420$

$105 \times 9 = 945$

$105 \times 16 = 1680$

H/W :→

Lower Bound (Big-Omega) :-

$t(n) \geq C_2 g(n)$ $\quad \frac{1}{2}n(n-1) \in \Theta(n^2)$

$t(n) \geq \frac{1}{2}n(n-1)$

$t(n) \geq \frac{1}{2}(n^2-n)$

n^2 term dominate the $-n$ term.

$n^2 - n \approx n^2$

$t(n) \simeq \frac{1}{2}n^2$ ignoring small term.

$t(n) \geq \frac{1}{4}n^2$ Ans ★

Analysis types :-

1) prior (pre-Analysis) :-

The intial analysis performed before runing an algorithm.

- Algorithm Design
- Theoritical Analysis

→ The goal of prior analysis is to predict how efficient the algorithm is based on input size.

2) posterior (post-Analysis) :-

Analysis done before running the algorithms on a specific set of input

- Emperical data :- runing algo on real set of input to check it's performance.
- Run time meaursement That how much the time the algorithm take.

Assignment : 01

Name : Muhammad Hamza
Enrollment : 01-131232-057
Subject : DAA
Date : 03/10/2025

Q1.

Prove The following polynomials $\theta(n^3)$.

$$t(n) = 2n^3 - 10n^2 + 100n - 5$$

a) prove $t(n)$ is $O(n^3)$ Finding Constant C_2 and n_0

$t(n) \leq C_1 n^3$ For all $n \geq n_0$

SoL :-

To Find Const C_2 and n_0

$t(n) \leq C_1 n^3$ $\quad n \geq n_0$

Step1 :-

Analyze The polynomial

$$t(n) = 2n^3 - 10n^2 + 100n - 50$$

For Large n, The dominant term is $2n^3$ as The other term $-10n^2$, $100n$, -50.

M. Hamza
01-131232-057

Step 2:-

Find C_1

We want to bound $t(n)$ above by $C_1 n^3$. It's upper bound we chose it above than 2 it's 3.

$C_1 = 3.$

$t(n) = 2n^3 - 10n^2 + 100n - 50 \leq 3n^3$

Step 3:-

Solve for n_0

We find n_0 such that for all $n \geq n_0$, the inequality holds

$2n^3 - 10n^2 + 100n - 50 \leq 3n^3$

$-10n^2 + 100n - 50 \leq 3n^3 - 2n^3$

$-10n^2 + 100n - 50 \leq n^3 \quad , n = 6$

$190 \leq 216$

For this when I put $n \geq 6$ the polynomial is hold true. We show that $t(n) \leq 3n^3$ for all $n_0 \geq 10$.

Therefore $t(n) = O(n^3)$ with $C_1 = 3$ and $n_0 = 10$.

b) prove $t(n)$ is $\Omega(n^3)$ by finding positive const c_2 and n_0 such that

$$t(n) \geq c_2 n^3 \qquad n \geq n_0$$

Sol:

$$t(n) \geq c_2 n^3 \qquad n \geq n_0$$

Step 1:-

Analyze polynomial

$$t(n) = 2n^3 - 10n^2 + 100n - 50$$

The dominant term is $2n^3$.

Step 2:-

Find c_2

we want to bound $t(n)$ below by $c_2 n^3$. Let chose $c_2 = 1$

$$t(n) = 2n^3 - 10n^2 + 100n - 50 \geq n^3$$

M. Hamza
01-131232-057

Step 3:- To find n_0 such that for all $n \geq n_0$, inequality holds.

$$2n^3 - 10n^2 + 100n - 50 \geq n^3$$

Rearranging

$$2n^3 - n^3 - 10n^2 + 100n - 50 \geq 0$$

$$n^3 - 10n^3 + 100n - 50 \geq 0$$

(i) $-10 + 100 - 50 \geq 0$
$-10 + 50 \geq 0$
Hence

For $n \geq 1$ The polynamial true.

For $n \geq 11$ The left hand-side

Final result:-

Since $t(n) = O(n^3)$ and $t(n) = \Omega(n^3)$ we conclude that
$t(n) = \Theta(n^3)$.

Q.2:-

Find exact number of times (in term of n) The innermost Statement ($x = x+1$) is excuted in The Following code Running time in term of O(), Ω() or Θ() as appropriate.

```
X ← 0                                   n = input
For K ← 1 to n  → outer loop
  For j ← 0 to n-K → inner loop.
    X ← X+1
```

Sol:-

We Count The number of times The innermost Statement ($x = x+1$) excuting by both Loops.

Outer Loop (K Loop)

$j = 0$ to $j = n-K$ inner Loop

The innermost statement $x = x+1$ is excuted once for each iteration of The inner loop.

M. Hamza
01-131232-057

Step 2: Calculating number of iteration

Total $x = x+1$ ✓

$$\sum_{K=1}^{n} (n-K+1)$$

$$\sum_{K=1}^{n} (n+1-K)$$

$$\sum_{K=1}^{n} (n+1) - \sum_{K=1}^{n} K \quad \checkmark$$

$$\sum_{K=1}^{n} K = \frac{n(n-1)}{2} \quad \checkmark$$

$$= x = n(n+1) - \frac{n(n+1)}{2} \quad \checkmark$$

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

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

Total no of time $x = x+1$ excute

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

Step 3:- Running time Analysis

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

$x = \theta(n^2)$

The total running time is $\theta(n^2)$.

— X — X — X — X —

$\sum_{K=1}^{n} (n+1-K)$

$\sum_{K=1}^{n} (n+1) - \sum_{K=1}^{n} K$

$x \leftarrow x+1$

$x - n-K+1$

Unique Element Algorithms :-

```
For i ← 0 to n-2 do
    For j ← i+1 to n-1
    if A[i] = A[j] return false
    return true
```

→ In the above algorithms the basic operation is comparision because they used mostly time.

Summation Formulas :-

1. $\sum_{i=l}^{u} 1 = u - l + 1$

2. $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$

02/03/2025

Home Work :-

$$\sum_{i=0}^{n-2} (n-1-i)$$

$$\sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$(n-1) \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$(n-1)\left[(n-2)-0+1\right] - \frac{(n-2)(n-2+1)}{2}$$

$$(n-1)(n-1) - \frac{(n-2)(n-1)}{2}$$

$$\frac{2 \times (n-1)(n-1) - (n-2)(n-1)}{2}$$

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

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

$$\frac{2n^2 - 4n + 2 - n^2 + 3n - 2}{2}$$

$$\frac{n^2-n}{2} = \frac{n(n-1)}{2} \quad \text{Ans}$$

$$C_{worst}(n) = \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2)$$

prove that

$\frac{1}{2}n^2 \in \Theta(n^2)$?

Solve:-

Compare it $g(n) = n^2$

c_1 and c_2

$$c_1 n^2 \leq \frac{1}{2}n^2 \leq c_2 n^2$$

$\div$ by n^2

$$c_1 \leq \frac{1}{2} \leq c_2$$

$c_1 = \frac{1}{4}$ and $c_2 = \frac{1}{2}$

$$\frac{1}{4}n^2 \leq \frac{1}{2}n^2 \leq \frac{1}{2}n^2$$

for all $n \geq n_0$

n_0 is positive number

$\frac{1}{2}n^2 \in \Theta(n^2)$.

Ans

(O, Ω, Θ)

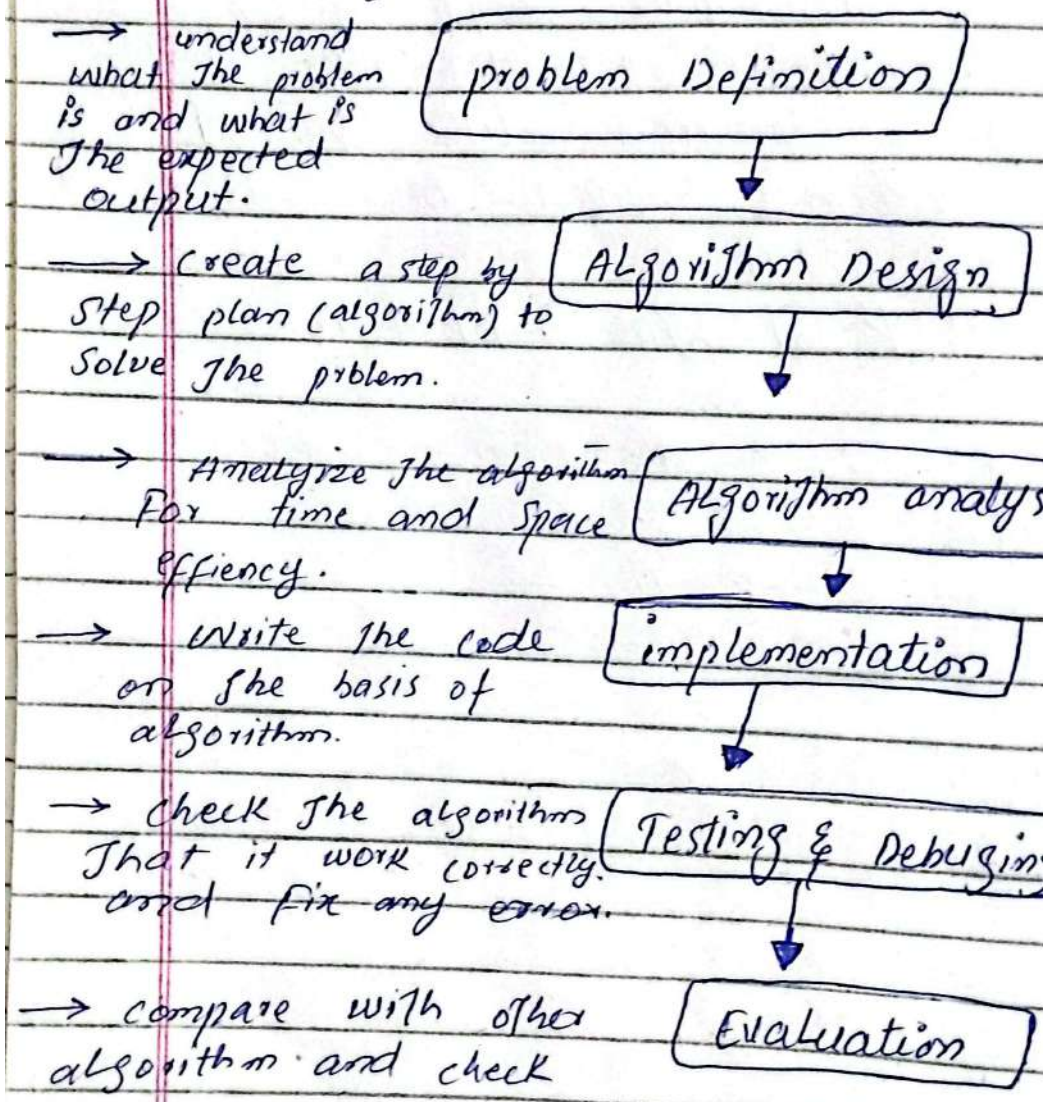
Mid term:-

1. Upto Brute Force.
2. ALL 4 Question carry equall marks.
3. Question like Quiz #1
4. Time / Space Complexity.
5. Definitions and understanding.
6. 2 Question like Quiz #1
7. Length must be ~~same~~ dont matter
8. Read The Question.

chapter 2 → Algorithm & Analysis
chapter 3 → Brute Force.
chapter 2 → O, Ω, Θ

-: PRACTICE QUESTION

1. Diagram The process of algorithm analysis. and Explain.



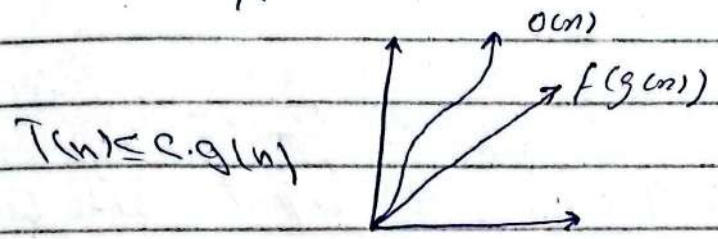
2.

Asymptotic Notation :-

They help us to describe the growth of time or space with respect to input size (n).

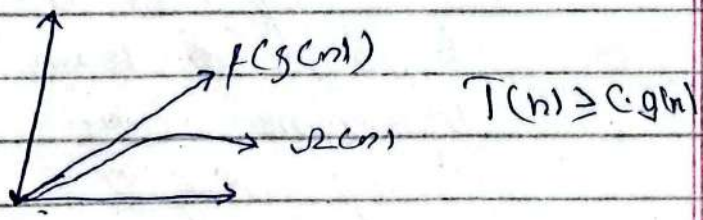
Big O :-

worst case
upper Bound



Big Ω :-

Best case time Complexity
Lower Bound.



Theta (Θ) :-

Average case
Tight Bound.

3. Basic Class EFFICIENCY :-

class	Name	Description
$O(1)$	const	Doesn't grow
$O(\log n)$	logrithmic	slowly grow
$O(n)$	linear	linearly grow
$O(n \log n)$	log-linear	or ( )
$O(2^n)$	exponential	Fastly grow
$O(n!)$	Factorial	Grow extremly
$O(n^2)$	Quadratic	Nested loop.

4. Non-Recursive :-

1. Identify the input size (n).
2. Basic Operation
3. Loop structure.
4. Count the total no of basic Operation.
5. Express in term of n.
6. Determine time complexity using Big-O

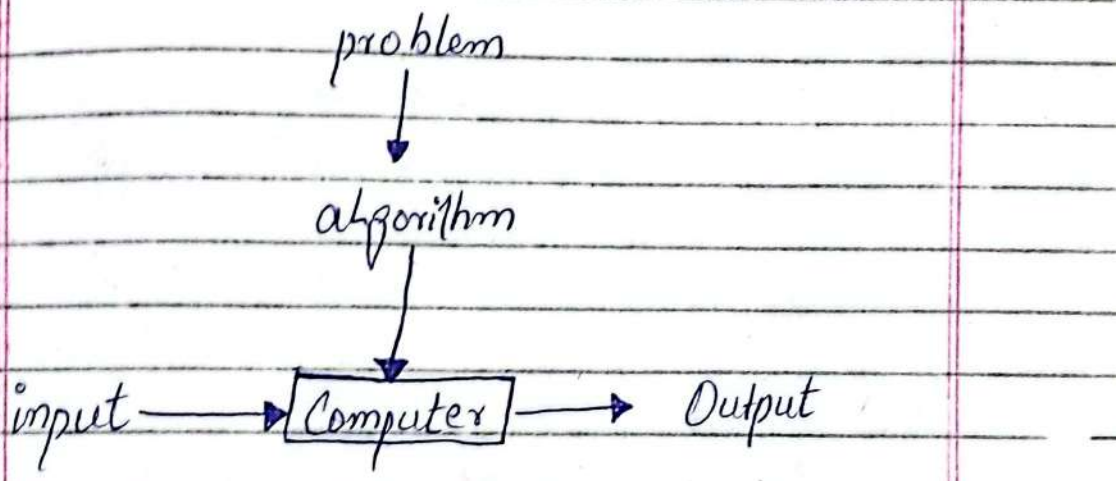
5. Recursive :-

1. Identify Base case and its time.
2. Write recurence relation for recurs call.
3. Solve recurence :-
   i) Recursion tree
   ii) Master Theorm.

4. Find time complexity using Big-O notation.

6.

An algorithm is a sequence of unambiguous instructions for solving problem.



7.

An Algorithm is finite set of well-defined instruction for specific problem or specific task.

Characteristic :-

- Input
- Output
- Definitness
- Finiteness
- Effectiveness

→ No, algorithm language are not specific it has written in the following as :-

- plan language
- pseudocode
- flowcharts
- java, c++ etc.

→ The word algorithim is derived from persian mathemi Muhammad bin Musa al Khwarzmi.

8.

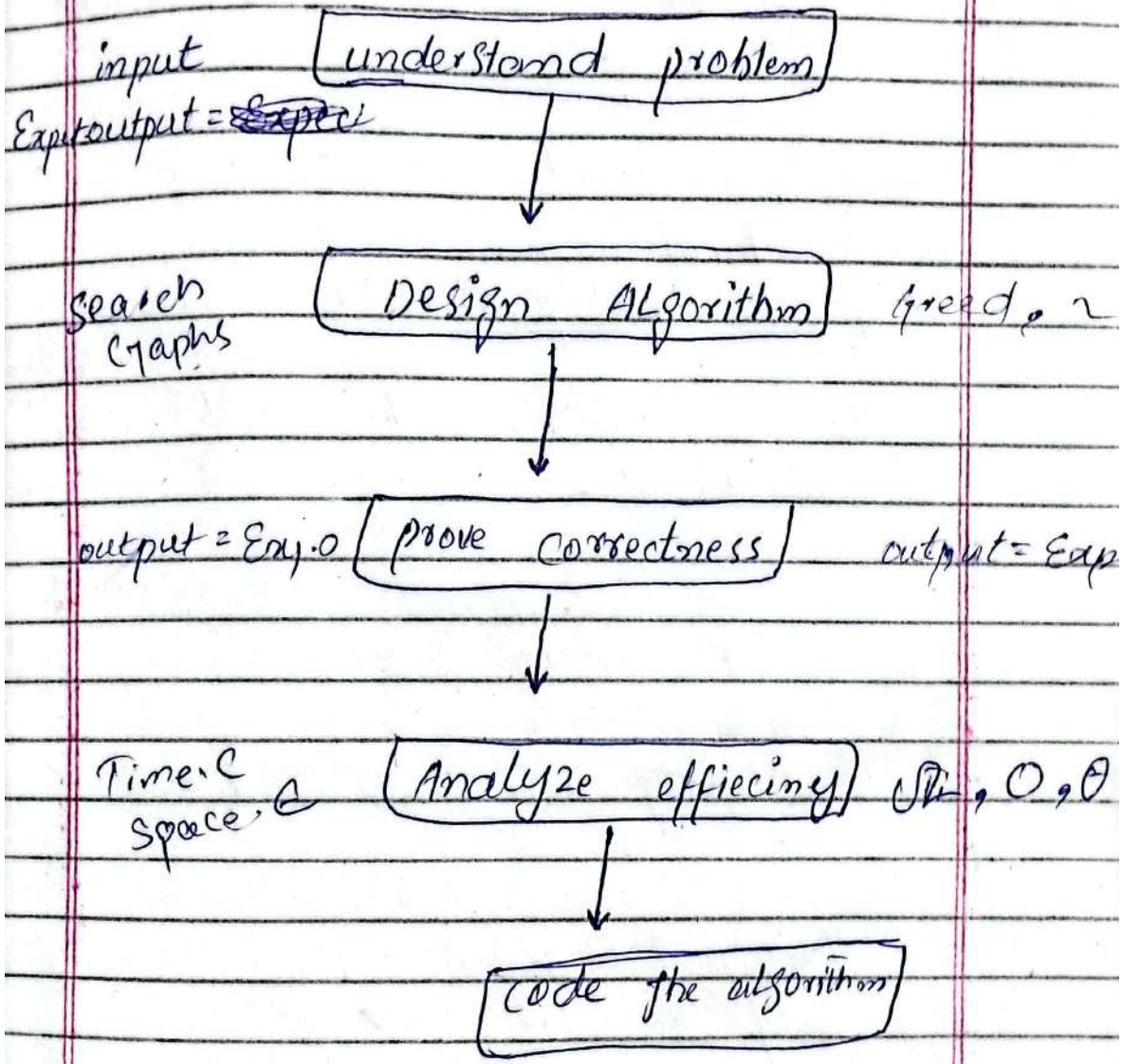
Sorting Number in Asscending Order.

unsorted = [23, 45, 12, 37, 11, 56]
Sorted = [11, 12, 23, 37, 45, 56]

```
for i = 0 to n-2
if element[i] > element[i+1]
    swap element[i] and element[i
end.
```

9.

Algorithm Design and Analysis :-



10.

1. understand problem :-

clearly Figure Out the problem

- input
- Output.

2. Ascertaining the capabilites of the Computational Device :-

Decide what tools you can use.

- Basic Operation (+, -, x, ÷
- Hardware (Gpus)

3. Chosing Exact vs Appropria Solutions :-

The perfect answer but it may be slow → Exact.

Good enough but fast ↳ Appropriate.

4. Algorithm Design techniques : pick strategy to solve problem.

- Brute Force
- Divide and Conquer
- Greedy

5. Designing an Algorithms and Data Structure:-

Designing Algorithms and using DSA Concept.

6. Method of Specifying Algorithms:-

How to describe it

- Pseudocode
- Flow chart
- Math Formula
- programming language

7. Proving Algorithm Correctness:-

Ensure it alway work.

- Try Small input
- output correct.

8. Analyze Algorithm :-

Meaurse efficiency

- Time Complexity
- Space Complexity.

9. Coding Algorithm :-

- Translate it into real code

11.

Sorting :-

Arrange data in Order is called Sorting

- Bubble Sort
- Merge Sort
- Quick Sort

Searching :-

Find item in data set.

- Linear Search
- Binary Search.

Graphs :-

Solve problems about networks (Nodes + edges)

- BFS
- Dijiska Algorithm.
- TSP

12.

Analysis of Algorithms :-

Analysis in algorithm design refers to evaluating how good an algorithm is. Focusing on

- Time Complexity
- Space Complexity

13.

Basic Operation :-

The Operation Contributing The most in the running algorithms is called Basic Operations.

→ also called dominant Operation

14.

Assuming That

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

how much algorithm will run if we double the input size?

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

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

$\therefore \frac{1}{2} n$ is small we don't use

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

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

$$= \frac{\frac{1}{2}(4n^2)}{\frac{1}{2} n} = 4n$$

→ time is about 4 time longer.

15.

General plan for non-recursive Algorithms :-

1. Chose input size
2. Identify Basic Operation
3. Count How often it run
4. Simplify count.
5. Determine growth rate.
   (Big O)

Non Recursive :-

1. No of input (n)
2. Basic operation
3. Find the ~~number~~ number of time the basic operation run.
4. Express in the time Complexity

$$\sum_{i=l}^{u} 1 = u - l + 1 \rightarrow$$

for 0 to i+1

$$\sum_{0}^{i+1} 1 \quad i+1-0+1$$

$$i+2$$

Recursive :-

1. input (n)
2. Base case
3. Recursive case.
4. Correctness.

Question 02:-

a) Binary Search Step by Step Solution:-

Solution:-

Given a Sorted Array A of n elements and a Key K. implement Search to find the index of K in Array A.
If not found return -1.

Example:-

A = [1, 3, 5, 7, 9, 11] K = 5
output : 2 (index of 5)

Step 1:-

Decrease and Conquere
Reduce the problem size by half at each Step by discarding one half of the array.

- Compare K with the middle element
- If equall return the index.
- If K is Smaller, Search the left half. Otherwise the right.

Step 2:-

pseudocode:-

```
i ← 0          // left Bound
r ← length(A) - 1   // Right Bound

while  l ≤ r
   m ← Floor ((l + r)/2)
   if  K = A[m]
      return m;
   else  if  K < A[m]
      r ← m - 1
   else
      l ← m + 1
      return -1.   // not found.
```

Step 3:-

Time Complexity Analysis

1. Basic Operation :- 'Comparison (K = A[m])

2. Worst Case Scenario :-
   Key not present at the end.
   each step the array size halves → $\log_2 n$ steps.

3. Complexity :-

Time :- $O(\log n)$ → logrithmic time

Space :- $O(1)$ → use Const Space.

Step 4 :-

Advantages & Disadvantages :-

Binary Search	Linear Search
• Fast $O(\log n)$	• Slow $O(n)$
• Require Sorted input	• work on unsorted data.
• Efficient for large data.	• Simple to implement.

Testing with Example :-

[1, 3, 5, 7, 9, 11]

Key = 5

output = 2 because 5 is in index 2.

1 iteration.

Q 2 :-

part b) :-

Insertion Sort :-

A = [12, 11, 13, 5, 6]
output = [5, 6, 11, 12, 13]

Step 1 :-

- Start with Single element. (trivally Sorted)
- Decrease The problem Size.
- Divide The array into Sub-arrays
- For each element A[i] Compare it with the previous element. into it position which is already Sorted.

Step 2 :-

pseudocode :-

For i ← From 1 to length(n-1)
    Key ← A[i]
    j ← i-1

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$ (5)

While $j \geq 0$ and $A[j] > Key$
$A[j+1] \leftarrow A[j]$ // shift right
$j \leftarrow j-1$

~~while $j \geq 0$ And $A[j] <$~~
$A[j+1] \leftarrow Key$

return A.

Example :-

$A = [12, 11, 13, 5, 6]$

12 First element

1st Iteration :-

12 Compare with 11
$12 > 11$ Swap.
$12 < 13$ No Swap
$12 > 5$ Swap
$12 > 6$ Swap

→ Throy j They Compare with at the end of loop.

Output :- $[5, 6, 11, 12, 13]$.

Time Complexity :-

Basic Operation :- Comparision

A[j] > Key → Shift

Total operation $1+2+\cdots\cdots+n-1 = n(n-1)/2$

worst Case :-

when array is reverse Sorted

A = [6, 5, 4, 3, 2]

$n(n-1)/2$ runs.

Complexity :-

Time = $O(n^2)$ quadratic time.

Space :- $O(1)$

Best Case :-

When array is already sorted.

Advantage & DisAdvantages:

Insertion Sort	Merge Sort
• Simple to implement	• Faster ($O(n \log n)$)
• Efficient for small data.	• Require extra memory

Question 03:-

Traveling Salesman problem (TSP).

Example 4 cities

Cities : A, B, C, D

Distance Matrix.

A → B	10
A → C	15
A → D	20
B → C	35
B → D	25
C → D	30

1. Brute Force Approach:-

- All possible permutations of routes to find the shortest one.
- Genarate all (n-1) possible routes (For n cities).
- Calculate total distance for each routes.
- Track minimum distance found.

pseudocode :-

```
tsp_BruteForce (distance, n)

    min_distance ← ∞
    best_routes ← [ ]
    cities ← [0, 1, 2 ..... n-1] // city indices

    Current_distance ← distance[0][routes[0]]
    For i from 0 to length (route)-2.

Current_distance ← current_distance + distance[route[i]]
                                      [route[i+1]]

Current_distance ← current_distance + distance[route[-1][0]
                   // Returns to Origin

if Current_distance < min_distance
    min_distance ← current_distance
    best_route ← [0] + route + [0]

    return best_route, min_distance.
```

Complexity Analysis :-

Time :- $O((n-1)!)$

factorial time → Impractical for $n > 10$

Space :- $O(n)$ store the current permutation

Example :-

4 cities

permutation :-

[B, C, D]
[B, D, C]
[C, B, D] etc.

Optimal routes :-

A → B → D → C → A

2. Decrease and Conquer :-

Divide :-
Divide cities into smaller subsets.

Conquer :-
Solve TSP for subsets. recursively.

Combine :-
Merge the result to find the global optimal.

Pseudocode :-

```
tsp-Decrease-Conquer (distance, n)

    memo ← {}   // memoization table.
function dp (mask, current_city)
  if mask = (1 << n)-1 // All cities visited.
  return distance [current_city[0]] // return Orig

if (mask, current_city) in memo:

    return memo [(mask, current_city)]

    min_distance ← ∞

for next_city from 0 to n-1:
```

```
if not (mask & (1 << next_city))
    new_mask ← mask | (1 << next_city)

distance ← distance[current_city][next_city]
           + dp[new_mask, next_city]

if distance < min_distance
    min_distance ← distance.

memo[(mask, current_city)] ← min_distance

return min_distance.

return dp(1, 0)  // start with
// city 0 with mask = 0001)
```

Complexity Analysis :-

Time :- $O(n^2 * 2^n)$

→ exponential but faster than Brute Force.

Space :- $O(n * 2^n)$

Example :-

A → B → C → D → A

Distance 80 unit

Comparison :-

Metric	Brute force	Decrease & Conquer.
Time Complexity	$O((n-1)!)$	$O(n^2 * 2^n)$
Space Complexity	$O(n)$	$O(n^2 * 2^n)$
Advantage	Simple execution exact solution	Faster than Brute force
DisAdvantage	infesiable $n > 10$	exponential for large n.

Question 02 :-

Time efficiency class of your algorithms?

Solution :-

Time Complexity Analysis :

- n = number of term
- m = number of matches
- $m = \frac{n(n-1)}{2}$

Worst Case :-

1) Graph Construction from match result = $O(n^2)$

Because we might check results for every pair of teams.

2) Topological Sort (DFS or BFS) = $O(m+n)$

Since $m = O(n^2)$ in a round robin tournment. Then

- Graph construction = $O(n^2)$
- Topological Sort = $O(n+n^2) = O(n^2)$

So

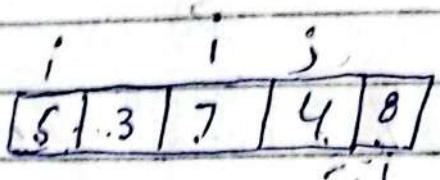
Best case = $\theta(n^2)$
Average case = $\theta(n^2)$
Worst case = $\theta(n^2)$

Final answer :-

Time efficiency of your algorithm is $\theta(n^2)$, since building the graph and sorting it involves checking all team pairs.

Divide and Conquer :-

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



i = 0 // i = 5

i = increment and j = decrement

i & < pivot.p

| 4 | 3 | 7 | 5 | 8 |

A

5 | 3 | 1 | 9 | 8 | 2 | 4 | 7

5 | 3 | 1 | 4 | 8 | 2 | 9 | 7

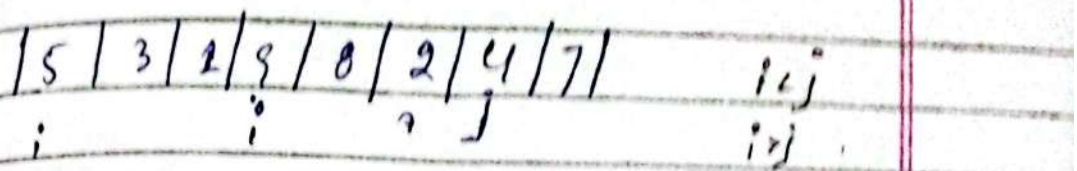
5 | 3 | 1 | 4 | 2 | 8 | 9 | 7

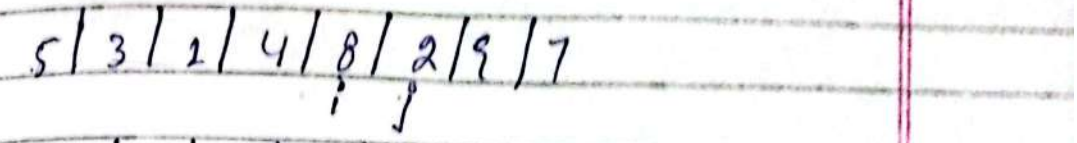
2 | 3 | 1 | 4 | 5 | 8 | 9 | 7

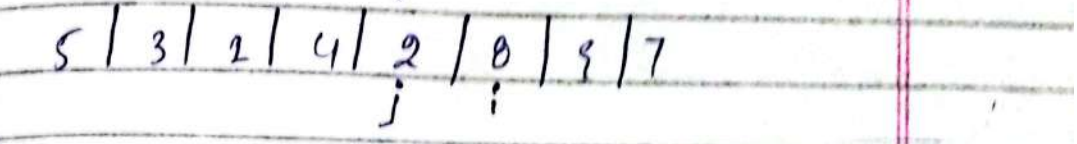
2 | 1 | 3 | 4 | 5 | 8 | 9 | 7

2 | 2 | 3 | 4 | 5 | 8 | 9 | 7

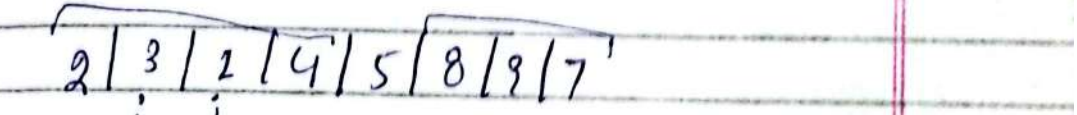
QUICK Sort:-



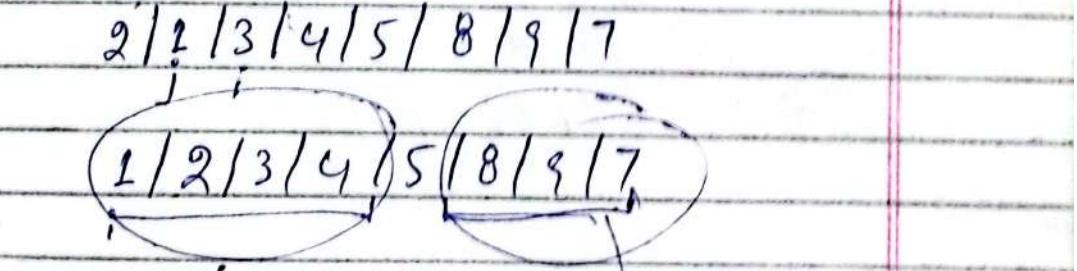


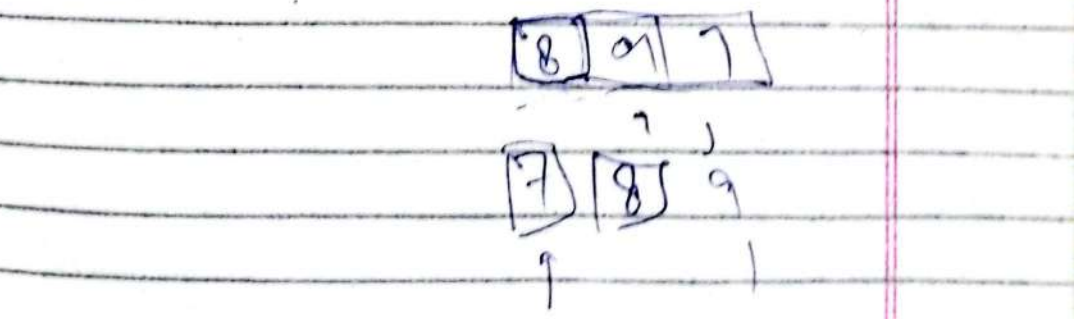


when $i > j$ swap piviot point with j



The new piviot point is 2





if a pivot point i++ j--
i > j SWAP

Practice Questions:-

1.

7	8	2	9	11	3

Dry Run:-

7	8	2	9	11	3
@	i				j

SWAP

7	3	2	9	11	8
	j		i		

j SWAP with piviot

2	3	7	9	11	8
		p		i	

Divided into Two Sub-Array

2	3	7

9	8	11
	j	i

8	9	11

Now we Merge it

2	3	7	8	9	11

Ans

a = 1
i[0] = a

1

1 = 1

Array:-

0	1	2	3	4	5
1	2	3	5	6	7

Find the missing one?

```
For(int i = 0, i<n, i++)
    i = a,          // a == 0

if (a == i++)
    return True;        [i]
else
    retur false;    i
```

a[i] = (1)

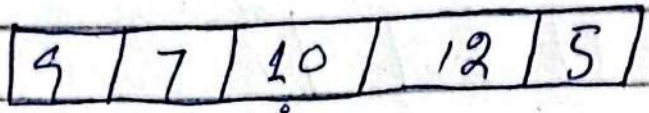
i+1 = (1)

a[i] = (1)

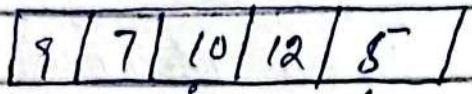
(1)

i+1

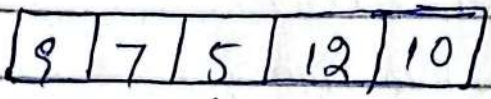
2.

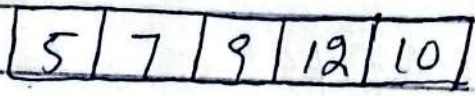


pivot = 5



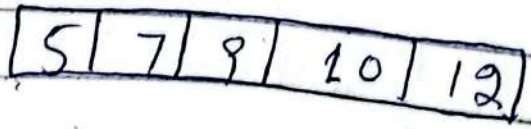
SWAP i, j





∴ Two Sub-Array

SWAP



Ans

(2), 3, 4, 2

check uniqness in Array :-

For i ← 0 to n-2 do

if A[i] = A[i+1]

Return false

else

Return True // ~~no repeat~~ de uniqness

Question No 1 :-

(part g)

Cost Optimization (Divide and Conquer with Dynamic programing)

Objective :-

1. All dependencies are satisfied.
2. No time overlaps (Task can't run simultaneously) But Back-to-Back scheduling is allowed.

Pseudocode :-

```
FUNCTION MinCostSubset(task, memo):
IF task is empty
    Return 0, []
// Sort task by end time
```

```
Sorted_tasks = Sort(task.By end_time)

// Base case: Single task
n = LENGTH (Sorted_tasks)
  if n == 1:
    task = Sorted_task[0]
      dep_cost = 0
      dep_Subset = []
  For each dep IN task-dependencies:
d_cost, d_Subset = memo[dep] If
dep IN memo.

ELse Mincost subset ([dep], memo).
  dep_cost += d_cost
  APPEND d_Subset TO dep_subset
  If no_overlap (task, task_dep):
  total_cost = task.cost + dep_cost;
   Subset = [task] + Sub_set
  memo[task] = total_cost, subset.
   Return total cost, Subset.
     Return 0, [];

// Divide
    mid = n//n
left_tasks = sorted_task[0: mid]
right_tasks = Sorted_task[0: mid, n]
```

// Conquer

left_cost, left_subset = MinCostSubset
(left_tasks, memo)

right_cost, right_subset = MinCostSubset
(right_tasks, memo)

// Combine

IF left_cost < right_cost:
    min_cost = left_cost.
    Optimal_subset = right_subset

// Try to combining right tasks with left subset.

For each task t IN right_tasks:
    can_add = TRUE
    For each s IN optimal subset.
        IF Overlaps(t, s)
        can_add = False
            BREAK;
        dep_cost = 0
        dep_subset = [].

For each dep IN t.dependencies:
IF dep NOT IN optimal_subset
AND dep NOT in right subset

```
can_add = false
    BREAK
d_cost, d_subset = memo[dep]
dep_cost += d_cost
Append d_subset TO dep_subset
  IF can_add
  new_cost = min_cost + t.cost + dep_cost
  If new_cost < min_cost OR min_cost == 0
  min_cost = new_cost.
  Optimal_subset = optimal_subset
  + [t] + dep_subset
  memo[sorted_tasks] = min_cost, optimal_subset
  Return min_cost, Optimal_subset.

Function Overlaps(task1, task2)
Return task1.start_time < task2.end_time
AND task2.start_time < task1.end_time

// Divide Code

  memo = Empty_HASHMAP
  total_cost, selected_tasks =
  MinCostSubset([T1, T2, T3, T4, T5, T6], memo)

OUTPUT: "Minimum cost:", Total_cost,
"selected_tasks
```

Execution Steps:-

1. Sorted by end time
   $T_1(4)$, $T_2(5)$, $T_3(6)$ $T_4(7)$ .... $T_6(9)$

2. Divide
   left : T_1, T_2, T_3
   Right : T_4, T_5, T_6

3. Conquer

   left (T_1, T_2, T_3)
   Right (T_4, T_5, T_6)

4. Combine

   left$[T_1]$, cost = 10
   Right $[T_3, T_6]$, cost = 70
   Try adding T_1 overlap T_3
   So can't combine

Results:-

Minimum Cost : 70
Subset : $[T_3, T_6]$

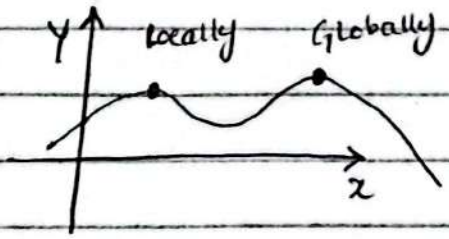
Greedy Algorithms :-

It's used in optimization problem.

It's works in phases :-

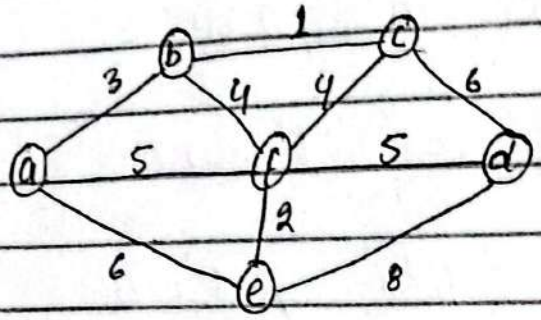
Global optimal Solution :-

Where are no other fessiable Solution is called Global fessiable Solution.



Prims Algorithm :-

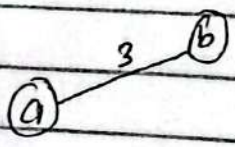
To find Minimum Spaning tree.



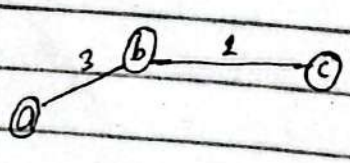
Sol :-

starting from a

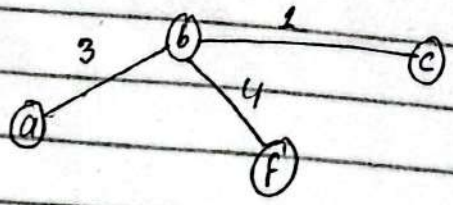
1.



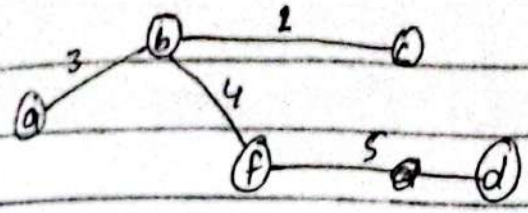
2.



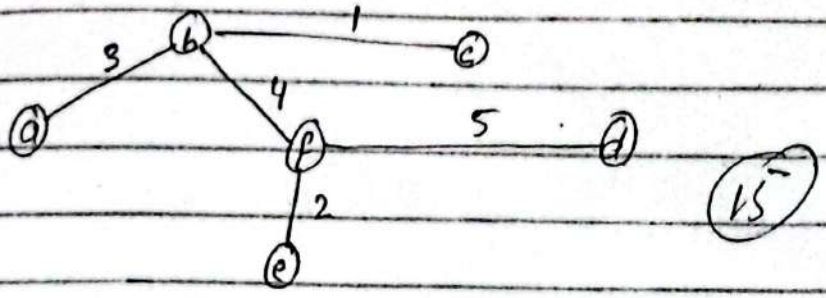
3.



4.

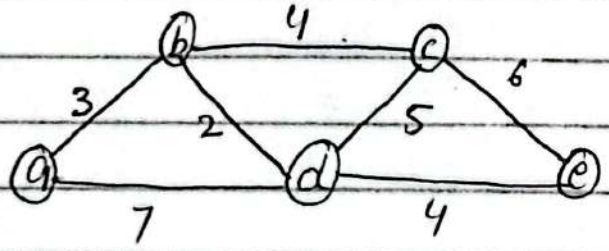


5.

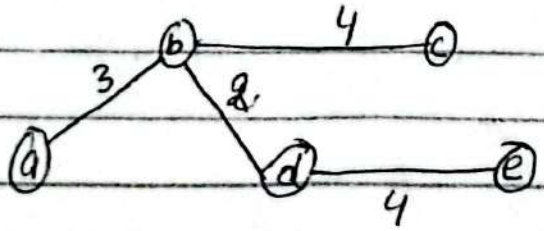


Ans

To find the shortest path through Dijistra's Algorithm:



Sol :-



Final Exam :-

(a) Course Covered :-

(1) Comprehensive exam
(2) All course included.

(b) paper pattern :-

Q1. Decrease & Conquer
Q2. Divide & conquer
Q3. Transform & Conque
Q4
Q5. } → Greedy Techniques.

→ Each carry equall marks.

Questions :-
Answers :-

(1) Definitions
(2) Analysis
(3) Time Complexity
(4) Application of Them.

Decrease and Conquer :-

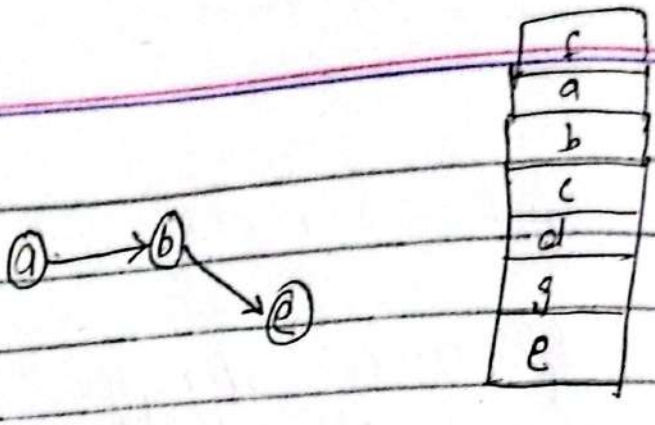
It's is algorithmic strategy where you reduce the problem into smaller version and solve it and build up the solution is called divide and conquer.

Step of Decrease and conquer :-

1) Decrease :- Reduce the problem size
2) Conquer :- Solve the smaller problem
3) Extend :- use the smaller solution to solve the orignal problem.

Types of Decrease & conquer :-

1) Decrease by 1 :- n to $n-1$
2) Decrease by constant factor ($n \rightarrow n/2$)
3) Decrease by variable size.



a b e

Divide and Conquer :-

It's is an algorithms Design Strategy that Divide a big problem into Smaller subproblem Solve them recursively and then Combine the results to get final answer.

Three Main Steps :-

① Divide :-
Break problem into 2 or more sub-problem. (usually the same type and size).

② Conquer :-
Solve each problem recursively. If it is small enough.

(3) Merge :-

Merge and Combine the results of the sub-problem to form final results

~~f(n) = a f(n)~~

$f(n) = a f(n/b) + f(n)$ → Support

↓

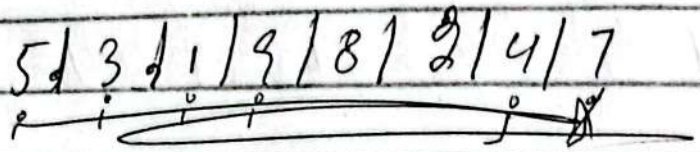
No of Sub problems

Quick sort:-

0 1 2 3 4 5 6 7

5, 3, 1, 9, 8, 2, 4, 7

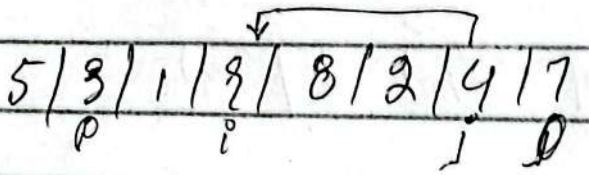
p i j



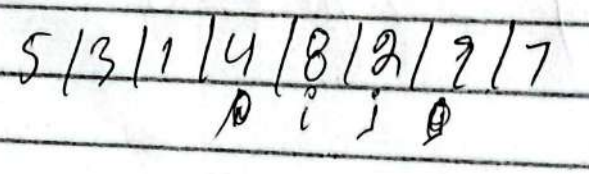
i < j

i > j

j--



pivot p = 5



~~5 3 1 4 8 9 7~~