AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Tumkur-Koratagere Road, Tumkur-572106.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Module 2 Notes for

"Design and Analysis of Algorithm"
[BCS401]

Prepared by: -
Ms.Trupthi V
Mrs.Ashwini Singh.S
Mrs.Keerthishree PV
Assistant Professors,
Department of CSE.
Akshaya Institute of Technology, Tumakuru

AKSHAYA INSTITUTE OF TECHNOLOGY

Lingapura, Obalapura Post, Koratagere Road, Tumakuru - 572106

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VISION

To empower the students to be technically competent, innovative and self-motivated with human values and contribute significantly towards betterment of society and to respond swiftly to the challenges of the changing world.

MISSION

M1: To achieve academic excellence by imparting in-depth and competitive knowledge to the students through effective teaching pedagogies and hands on experience on cutting edge technologies.

M2: To collaborate with industry and academia for achieving quality technical education and knowledge transfer through active participation of all the stake holders.

M3: To prepare students to be life-long learners and to upgrade their skills through Centre of Excellence in the thrust areas of Computer Science and Engineering.

Program Specific Outcomes (PSOs)

After Successful Completion of Computer Science and Engineering Program Students will be able to

* Apply fundamental knowledge for professional software development as well as to acquire new skills.
* Implement disciplinary knowledge in problem solving, analyzing and decision-making abilities through different domains like database management, networking, algorithms, and programming as well as research and development.
* Make use of modern computer tools for creating innovative career paths, to become an entrepreneur or desire for higher studies.

Program Educational Objectives (PEOs)

PEO1: Graduates expose strong skills and abilities to work in industries and research organizations.

PEO3: Graduates engage in team work to function as responsible professional with good ethical behavior and leadership skills.

PEO3: Graduates engage in life-long learning and innovations in multi disciplinary areas.

Analysis & Design of Algorithms		Semester	4
Course Code	**BCS401**	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	3:0:0:0	SEE Marks	50
Total Hours of Pedagogy	40	Total Marks	100
Credits	03	Exam Hours	03
Examination type (SEE)	Theory		

Course objectives:

- To learn the methods for analyzing algorithms and evaluating their performance.
- To demonstrate the efficiency of algorithms using asymptotic notations.
- To solve problems using various algorithm design methods, including brute force, greedy, divide and conquer, decrease and conquer, transform and conquer, dynamic programming, backtracking, and branch and bound.
- To learn the concepts of P and NP complexity classes.

Teaching-Learning Process (General Instructions)

These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes.

1. Lecturer method (L) does not mean only the traditional lecture method, but different types of teaching methods may be adopted to achieve the outcomes.
2. Utilize video/animation films to illustrate the functioning of various concepts.
3. Promote collaborative learning (Group Learning) in the class.
4. Pose at least three HOT (Higher Order Thinking) questions in the class to stimulate critical thinking.
5. Incorporate Problem-Based Learning (PBL) to foster students' analytical skills and develop their ability to evaluate, generalize, and analyze information rather than merely recalling it.
6. Introduce topics through multiple representations.
7. Demonstrate various ways to solve the same problem and encourage students to devise their own creative solutions.
8. Discuss the real-world applications of every concept to enhance students' comprehension.

Module-1

INTRODUCTION: What is an Algorithm?, Fundamentals of Algorithmic Problem Solving.

FUNDAMENTALS OF THE ANALYSIS OF ALGORITHM EFFICIENCY: Analysis Framework, Asymptotic Notations and Basic Efficiency Classes, Mathematical Analysis of Non recursive Algorithms, Mathematical Analysis of Recursive Algorithms.

BRUTE FORCE APPROACHES: Selection Sort and Bubble Sort, Sequential Search and Brute Force String Matching.

Chapter 1 (Sections 1.1,1.2), Chapter 2(Sections 2.1,2.2,2.3,2.4), Chapter 3(Section 3.1,3.2)

Module-2

BRUTE FORCE APPROACHES (contd..): Exhaustive Search (Travelling Salesman probem and Knapsack Problem).

DECREASE-AND-CONQUER: Insertion Sort, Topological Sorting.

DIVIDE AND CONQUER: Merge Sort, Quick Sort, Binary Tree Traversals, Multiplication of Large Integers and Strassen's Matrix Multiplication.

Chapter 3(Section 3.4), Chapter 4 (Sections 4.1,4.2), Chapter 5 (Section 5.1,5.2,5.3, 5.4)

Module-3

TRANSFORM-AND-CONQUER: Balanced Search Trees, Heaps and Heapsort.
SPACE-TIME TRADEOFFS: Sorting by Counting: Comparison counting sort, Input Enhancement in String Matching: Horspool's Algorithm.
Chapter 6 (Sections 6.3,6.4), Chapter 7 (Sections 7.1,7.2)

Module-4

DYNAMIC PROGRAMMING: Three basic examples, The Knapsack Problem and Memory Functions, Warshall's and Floyd's Algorithms.
THE GREEDY METHOD: Prim's Algorithm, Kruskal's Algorithm, Dijkstra's Algorithm, Huffman Trees and Codes.
Chapter 8 (Sections 8.1,8.2,8.4), Chapter 9 (Sections 9.1,9.2,9.3,9.4)

Module-5

LIMITATIONS OF ALGORITHMIC POWER: Decision Trees, P, NP, and NP-Complete Problems.
COPING WITH LIMITATIONS OF ALGORITHMIC POWER: Backtracking (n-Queens problem, Subset-sum problem), Branch-and-Bound (Knapsack problem), Approximation algorithms for NP-Hard problems (Knapsack problem).
Chapter 11 (Section 11.2, 11.3), Chapter 12 (Sections 12.1,12.2,12.3)

Course outcome (Course Skill Set)

At the end of the course, the student will be able to:

1. Apply asymptotic notational method to analyze the performance of the algorithms in terms of time complexity.
2. Demonstrate divide & conquer approaches and decrease & conquer approaches to solve computational problems.
3. Make use of transform & conquer and dynamic programming design approaches to solve the given real world or complex computational problems.
4. Apply greedy and input enhancement methods to solve graph & string based computational problems.
5. Analyse various classes (P,NP and NP Complete) of problems
6. Illustrate backtracking, branch & bound and approximation methods.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

Continuous Internal Evaluation:

- For the Assignment component of the CIE, there are 25 marks and for the Internal Assessment Test component, there are 25 marks.
- The first test will be administered after 40-50% of the syllabus has been covered, and the second test will be administered after 85-90% of the syllabus has been covered
- Any two assignment methods mentioned in the 22OB2.4, if an assignment is project-based then only one assignment for the course shall be planned. The teacher should not conduct two assignments at the end of the semester if two assignments are planned.
- For the course, CIE marks will be based on a scaled-down sum of two tests and other methods of assessment.

Internal Assessment Test question paper is designed to attain the different levels of Bloom's taxonomy as per the outcome defined for the course.

Semester-End Examination:

Theory SEE will be conducted by the University as per the scheduled timetable, with common question papers for the course (**duration 03 hours).**

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scored shall be proportionally **reduced to 50 marks**

Suggested Learning Resources:

Textbooks

1. Introduction to the Design and Analysis of Algorithms, By Anany Levitin, 3rd Edition (Indian), 2017, Pearson.

Reference books

1. Computer Algorithms/C++, Ellis Horowitz, SatrajSahni and Rajasekaran, 2nd Edition, 2014, Universities Press.
2. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, Clifford Stein, 3rd Edition, PHI.
3. Design and Analysis of Algorithms, S. Sridhar, Oxford (Higher Education)

Web links and Video Lectures (e-Resources):

- Design and Analysis of Algorithms: https://nptel.ac.in/courses/106/101/106101060/

Activity Based Learning (Suggested Activities in Class)/ Practical Based learning

- Promote real-world problem-solving and competitive problem solving through group discussions to engage students actively in the learning process.
- Encourage students to enhance their problem-solving skills by implementing algorithms and solutions through programming exercises, fostering practical application of theoretical concepts.

Assessment Methods -

1. Problem Solving Assignments (Hacker Rank/ Hacker Earth / Leadcode)
2. Gate Based Aptitude Test

Module : 2

Brute Force Approaches (contd...)

Exchaustive Search :

* the brute force approach used to solve combinatorial problem is called Exchaustive search.
* the solution to the combinatorial problems consume too much time.
* this searching technique generates all the possible solutions by satisfying the constraints given in the problem
* finally the desired solution which maximizes the profit is selected.
* the goal of the Exchaustive search method is to search all possible solutions and obtain an optimal solution.

Some of the problem that involve Exchaustive search are

- Travelling Salesman problem
- knapsack problem.

Travelling Salesman Problem (TSP)

* Given set of 'n' cities that visits each city exactly once before returning to the city where it started.

* the problem can be conveniently modeled by a weighted graph, with the graph's vertices representing the cities and the edge weights specifying the distance.

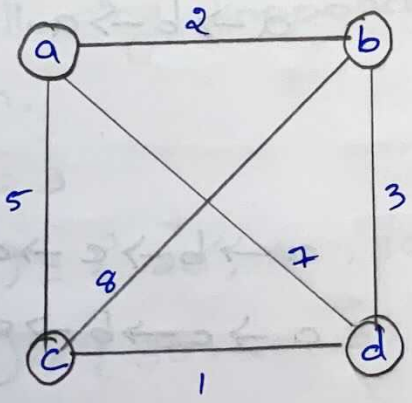
* then the problem can be stated as the problem of finding the shortest Hamiltonian Circuit of the graph.

* Hamiltonian Circuit is defined as a cycle that passes through all the vertices of the graph exactly once.

Steps

⇒ Calculating the total number of routes
⇒ Drawing and all possible routes
⇒ calculating the distance traveled in each route
⇒ choosing the shortest route, which is the optimal solution.

Example :



	a	b	c	d
a	0	2	5	7
b	2	0	8	3
c	5	8	0	1
d	7	3	1	0

adjacency matrix

Tour	length	
$a \to b \to c \to d \to a$	$2+8+1+7=18$	
$a \to b \to d \to c \to a$	$2+3+1+5=11$	optimal
$a \to c \to b \to d \to a$	$5+8+3+7=23$	
$a \to c \to d \to b \to a$	$5+1+3+2=11$	optimal
$a \to d \to b \to c \to a$	$7+3+8+5=23$	
$a \to d \to c \to b \to a$	$7+1+8+2=18$	

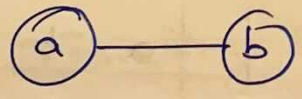
Analysis

Let us see the various routes that can be used by a travelling salesperson when he visits each and every city Exactly once and returns back to the city from where he has started.

Let us find out the possible routes taken by the salesperson by assuming he has started from the city 'a'.

Let 'n' denote the number of cities to visit

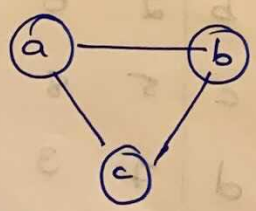
$n=2$



$a \to b \to a$ no of routes $= 1$

$(2-1)!$

$n=3$



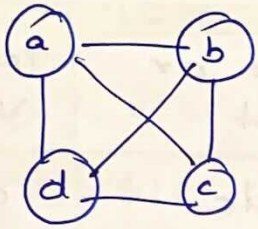
$a \to b \to c \to a$

$a \to c \to b \to a$

no of routes $= 2$

$(3-1)!$

n = 4



a → b → c → d → a
a → b → d → c → a
a → c → b → d → a
a → c → d → b → a
a → d → c → b → a
a → d → b → c → a

no of routes = 6
(4-1)!

For n cities no of routes = (n-1)! i.e f(n) = (n-1)!
time complexity is given by $f(n) \in O(n!)$

Knapsack Problem

⇒ A knapsack of capacity M and 'n' objects of weights $w_1, w_2, w_3 \ldots\ldots w_n$ with profit $P_1\ P_2 \ldots\ldots P_n$.

⇒ Let $x_1, x_2 \ldots\ldots x_n$ be the fractions of the objects that are supposed to be added into the knapsack.

⇒ the main objective is to place the objects into the knapsack so that maximum profit is obtained and the weights of objects chosen should not exceed the capacity of knapsack.

Example :

① Solve the following knapsack problem using bruteforce method given.

M = 40 , n = 3

$(w_1, w_2, w_3) = \{20, 25, 10\}$

$(P_1, P_2, P_3) = \{30, 40, 35\}$

Subset	Total weight	Feasible or not	Total Value
ϕ	0	Feasible	0
{1}	20	Feasible	30
{2}	25	Feasible	40
{3}	10	Feasible	35
{1,2}	20 + 25 = 45	Not Feasible	—
{1,3}	20 + 10 = 30	Feasible	65
{2,3}	25 + 10 = 35	Feasible	75
{1,2,3}	20 + 25 + 10 = 55	Not Feasible	—

* The Combination of objects {1,2} and {1,2,3} are not feasible because, the total weight of the objects selected will exceed the capacity of the knapsack 40 and rest are all feasible solutions.

* A feasible solution is the one that satisfies the given constraint. out of the feasible solutions, we have to select the subset which leads to maximum profit which in this case is 75 by selecting the objects {2,3}

(2) $M = 10$, $n = 4$

$w_1 = 7$, $w_2 = 3$, $w_3 = 4$, $w_4 = 5$

$V_1 = \$42$, $V_2 = \$12$, $V_3 = \$40$, $V_4 = \$25$

Subset	Total weight	Feasible or Not	Total value
ϕ	0	Feasible	$0
{1}	7	Feasible	$42
{2}	3	Feasible	$12
{3}	4	Feasible	$40
{4}	5	Feasible	$25
{1,2}	10	Feasible	$36
{1,3}	11	Not feasible	—
{1,4}	12	Not Feasible	—
{2,3}	7	Feasible	$52
{2,4}	8	Feasible	$37
{3,4}	9	Feasible	$65
{1,2,3}	14	Not Feasible	—
{1,2,4}	15	Not Feasible	—
{1,3,4}	16	Not Feasible	—
{2,3,4}	12	Not feasible	—
{1,2,3,4}	19	Not Feasible	—

A feasible solution is the one that satisfies the given Constriant. out of the feasible solutions, we have to select the subset which leads to maximum profit which in this case $65 by selecting the objects {3, 4}

Analysis:

The total number of subsets obtained for 3 elements is 8.

In general, given 'n' objects, the total number of subsets obtained will be 2^n.

Even, in the best case and worst case, the total number of subsets generated will be 2^n

the Time complexity of knapsack is

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

Decrease and Conquer

* Decrease and conquer is an approach for solving a problem by.

1. change an instance into one smaller instance of the problem
2. Solve the smaller instance
3. Convert the solution of the smaller instance into a solution for the larger instance.

* In decrease and conquer method the problem can be solved using Top down (Recursive) solution (or) using Bottom-up (Iterative (or) non recursive) solution.

Variations of Decrease and Conquer

These are three major variations of decrease & conquer

1. Decrease by Constant
2. Decrease by a Constant factor
3. Variable size decrease.

1. Decrease by Constant :

* In this method the size of the instance is Reduced by same constant on each iteration of the algorithm.

* Generally this constant is equal to one

Example : To compute a^{10} we can write

$$a^{10} = a^9 . a^1$$

If we formulize this Example then we can write

$$a^n = a^{n-1} . a$$

∴ If the function $f(n) = a^n$

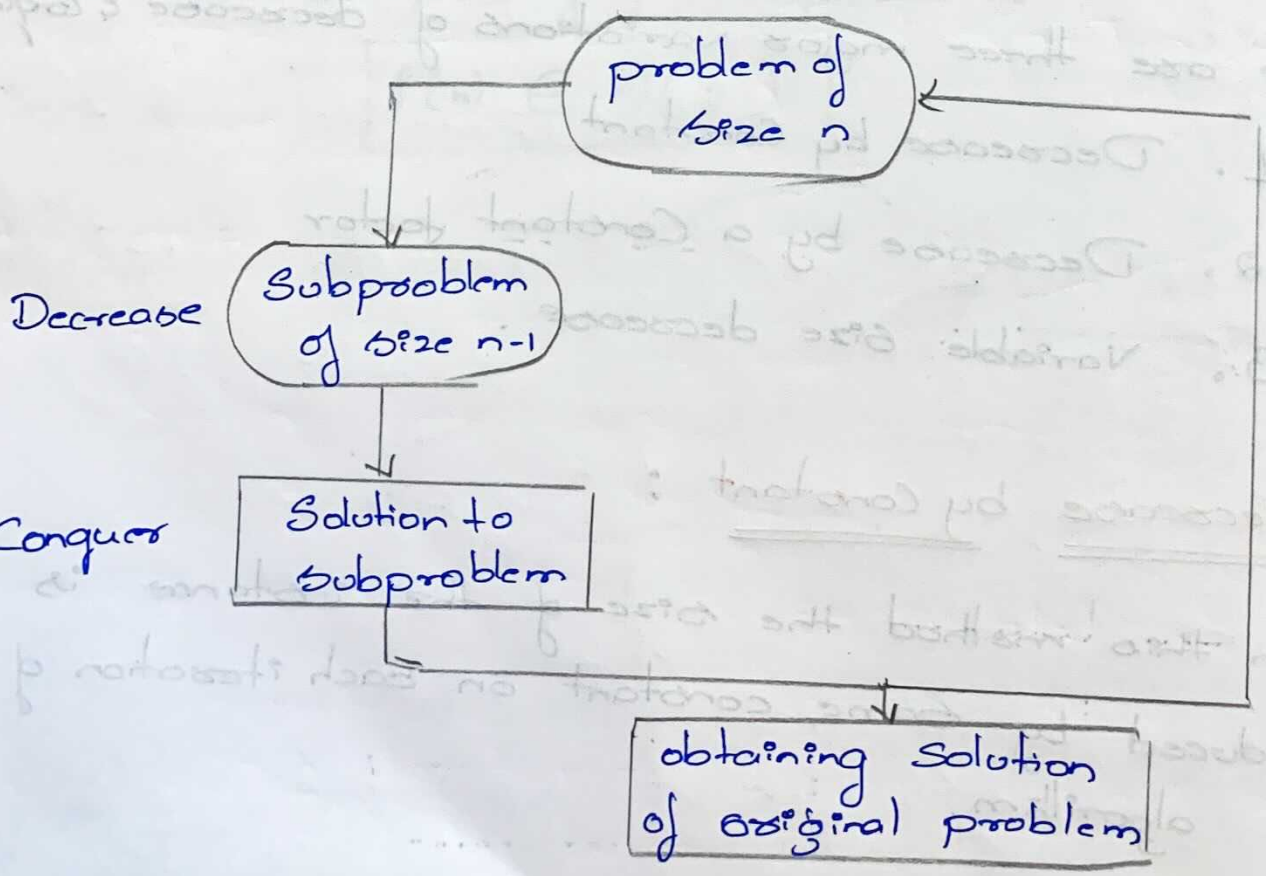
then using recursive approach we can write,

$$f(n) = f(n-1) . a \quad \text{if } n > 1$$

and $f(n) = a \quad \text{if } n = 1$

Whereas using iterative approach we multiply 'a' by n-1 times of a

$$f(n) = a * (n-1) \text{ time } a$$



Application of decrease by Constant :

1. Insertion sort
2. Graph searching Algorithm
   * Depth first search (DFS)
   * Breadth first search (BFS)
   * Topological sorting

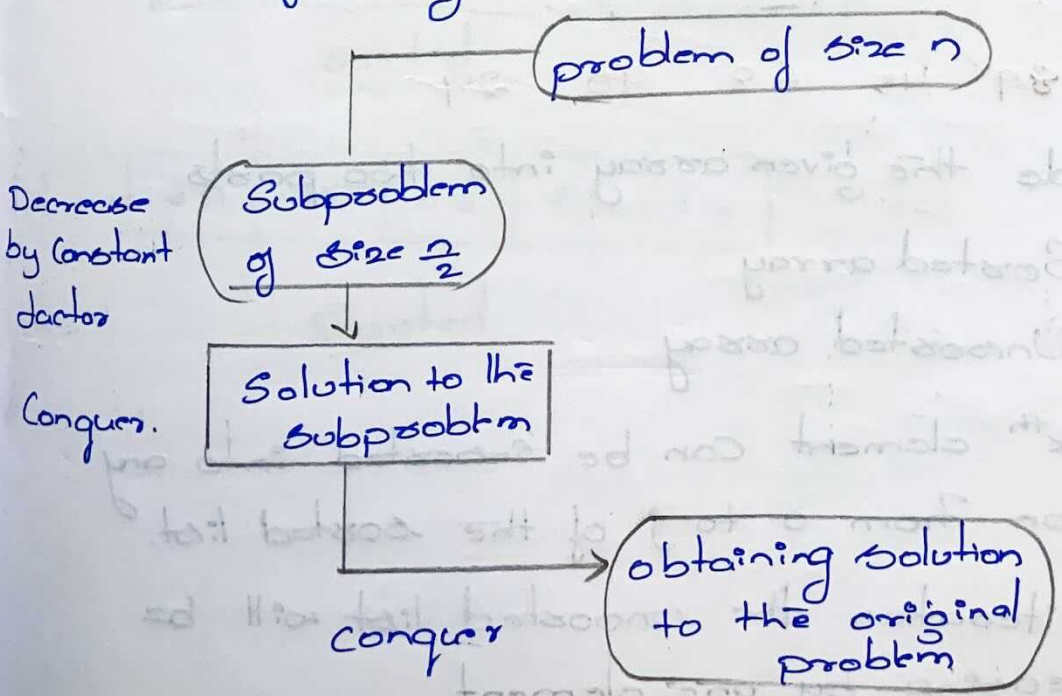
2. Decrease by a Constant factor :

* Decrease by a constant factor decreases the instant size by half or by some other fraction

Ex: $a^{10} = a^5 \cdot a^5$

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

the Efficiency is $O(\log n)$



3. Variable Size decrease method :

In Variable size decrease method the size reduction pattern varies from one iteration of an algorithm to another

Ex: Finding GCD of two numbers using Euclid's Algorithm.

$$gcd(m,n) = gcd(n, m \bmod n)$$

Insertion Sort

* Insertion sort is one of the simplest sorting algorithm for the reason that it sorts a single element at a particular instance.

* the scanning the sorted subarray from right to left until the first element smaller than (or) Equal to A[n-1] is Encountered to Insert A[n-1] right after that element

* Example : 89 45 68 90 29

Step 1 : Divide the given array into two parts

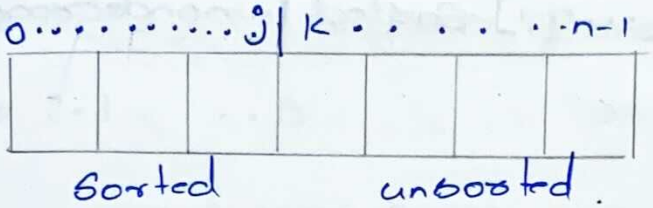
* Sorted array
* Unsorted array

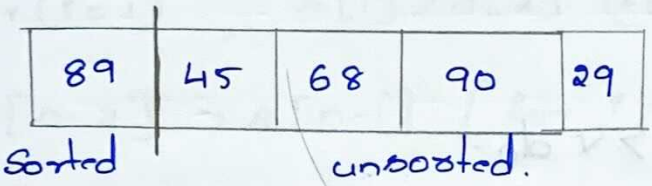
Step 2 : the k^{th} element can be inserted into any position from 0 to j of the sorted list.

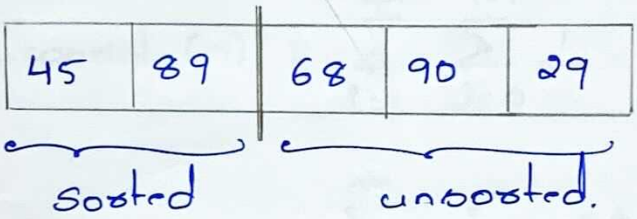
Step 3 : Each iteration the unsorted list will be decreasing by one element.

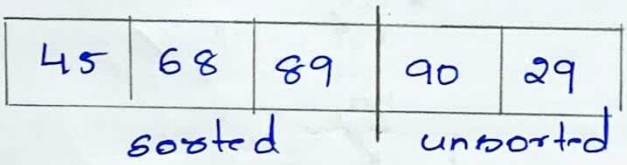
Step 4: After n-1 iterations all elements will be in sorted array.

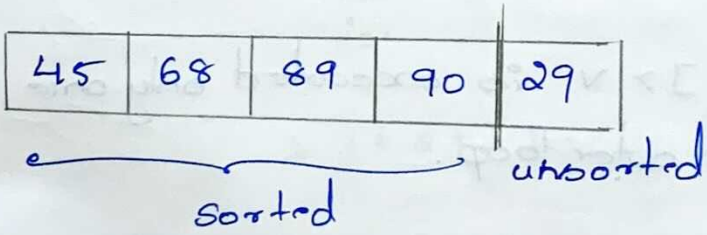
Example 89 45 68 90 29

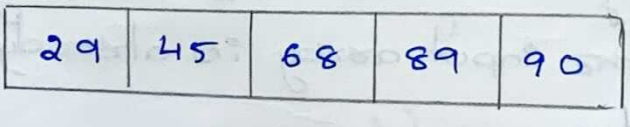












Sorted

Algorithm InsertionSort (A[0.... n-1])

// Sorts a given array by insertion sort

// Input : An array A[0.... n-1] of 'n' orderable elements

// output : An array A[0.... n-1] sorted in nondecreasing order

```
for i ← 1 to n-1 do
    v ← A[i]
    j ← i-1
    while j ≥ 0 and A[j] > v do
        A[j+1] ← A[j]
        j ← j - 1
    A[j+1] ← v
```

Analysis

Best case:

* the comparison $A[j] > v$ is executed only once on every iteration of the outer loop.

* It happens if and only if $A[i-1] \leq A[i]$ for every $i = 1 \ldots\ldots n-1$. ie., if the input array is already sorted in ascending order.

$$C_{best}(n) = \sum_{i=1}^{n-1} 1$$

$$= n-1-1+1$$

$$= n-1$$

$$C_{best}(n) = \theta(n)$$

Worst case

* $A[j] > v$ is executed the largest number of times

  i.e., $j = i-1 \ldots\ldots 0$.

* $v = A[i]$, it happens if and only if $A[j] > A[i]$ for $j = i-1 \ldots 0$

* the worst case input, we get $A[0] > A[1]$ (for $i=1$), $A[1] > A[2]$ (for $i=2$) ..... $A[n-2] > A[n-1]$ (for $i = n-1$).

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} i-1-0+1$$

$$= \sum_{i=1}^{n-1} i$$

$$= 1 + 2 + 3 + \ldots\ldots (n-2) + (n-1)$$

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

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

Average Case :

* It is based on investigating the no of element pairs out of order
* It makes on average half as many comparisons as on decreasing arrays.

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

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

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

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

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

$$= \frac{n^2}{4} - \frac{n}{4} + \frac{n}{2} - \frac{1}{2}$$

$$= \frac{n^2}{4} - \frac{n}{2} - \frac{1}{2}$$

$$\boxed{f(n) = \Theta(n^2)}$$

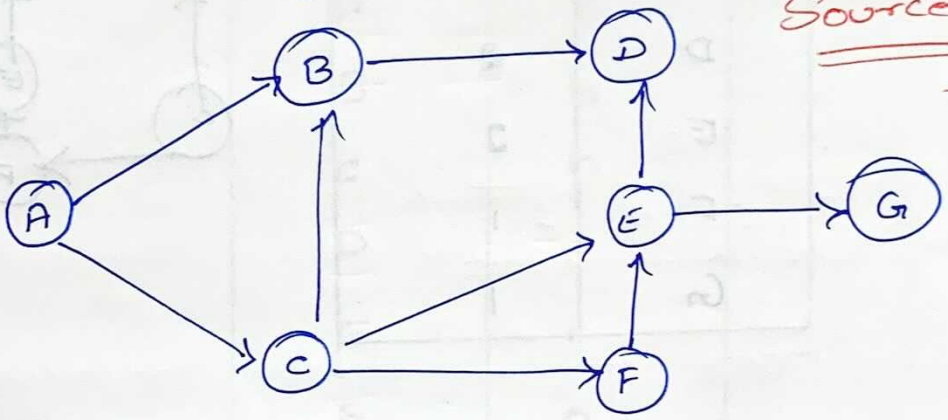
Topological Sorting

* Topological sort is a linear ordering of the vertices in such a way that if there is an edge in Directed Acyclic graph (DAG) from vertex 'u' to vertex 'v' then 'u' comes before 'v' in the ordering

* Topological sorting is possible if and only if the graph is a DAG

* the ordering of the nodes in the array is called a topological ordering.

* Source Removal method



Step 1 : Write the In-degree of each vertex

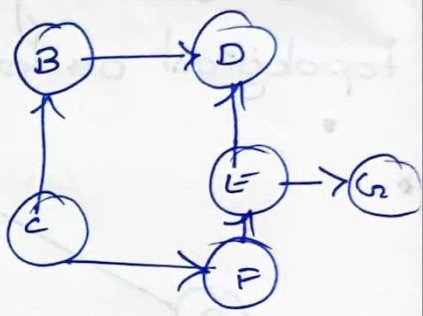
In-degree -> the edges which are coming towards the vertex

Vertices	In-degree
A	0
B	2
C	1
D	2
E	2
F	1
G	1

Step 2: Vertex 'A' with no incoming edges (choose indegree is 0) is selected and deleted along with the outgoing edges.

Now, update the in-degree of other vertices. Add node 'A' to array.

Vertices	In-degree
A	0
B	1
C	0
D	2
E	2
F	1
G	1



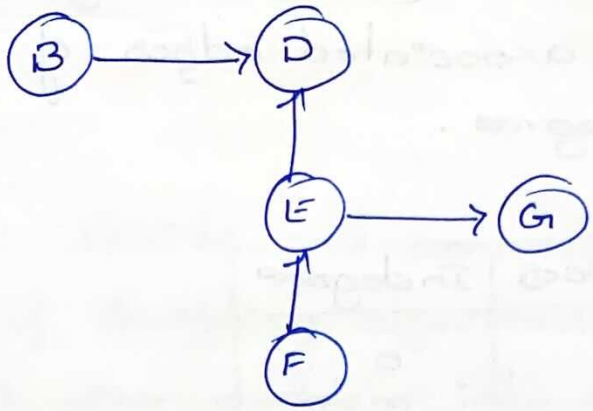
Node 'A' is added to the array

A	

Step 3: Next select the vertex choose indegree is 0 that is Vertex 'c' and remove the associated edges of c and update in-degree

Vertices	In-degree
A	0
B	0
C	0
D	2
E	1
F	0
G	1

A	C	

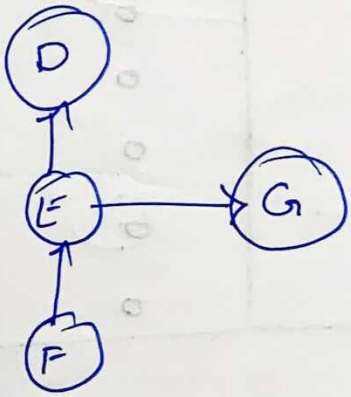


Step 4 : Select Vertex 'B' (choose indegree is 0) and remove the associated edges of 'B' and update in-degree

Vertices	In-degree
A	0
B	0
C	0
D	1
E	1
F	0
G	1

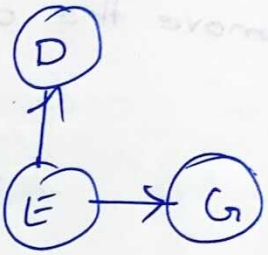
Step 5 :

A	C	B		



Step 5 : Select Vertex 'F' (choose in degree is 0) and remove the associated edges of 'F' and update in-degree.

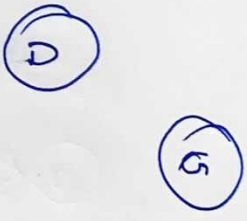
Vertices	In degree
A	0
B	0
C	0
D	1
E	0
F	0
G	1



A	C	B	F

Step 6 : Select vertex 'E' and remove the associated edges of 'E' and update in-degree.

Vertices	In-degree
A	0
B	0
C	0
D	0
E	0
F	0
G	0



A	C	B	E	F	D	G

Source Removal Method : this is the direct implementation of decrease and Conquer method

Step 1 :

From a given graph find a vertex with no incoming edges.

Step 2 : Delete it along with all the edges outgoing from it.

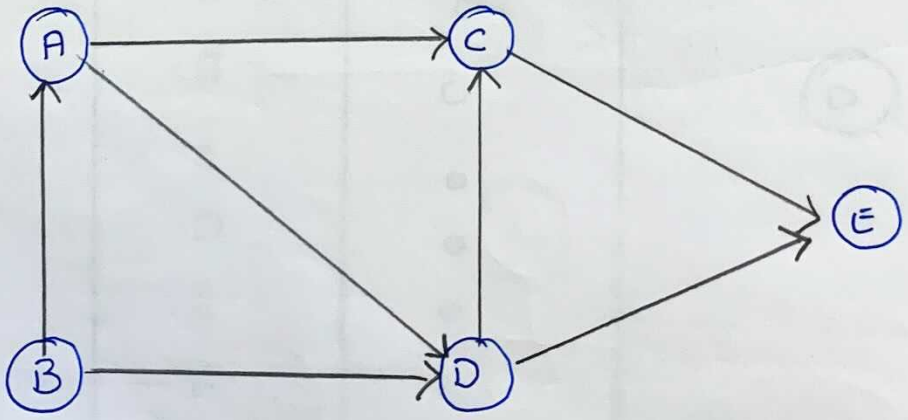
Step 3 : Note the vertices that are deleted.

Step 4 : All these Recorded vertices give topologically sorted list.

DFS Based Algorithm

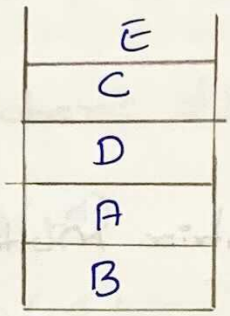
Topological sort is a process of assigning a linear ordering to the vertices of a DAG.

Ex : Sort the diagraph for topological sort using DFS Based Algorithm.



Sol^{n}: As the graph contains no cycle i.e. the graph is a DAG, the ~~todoo~~ topological sorting is possible.

Step 1: First find the Depth First Search and push the visited vertices in the stack thus Creates a DFS traversal stack.



Step 2: Now pop off the contents of the stack

E, C, D, A, B

Step 3: Reverse the popped contents.

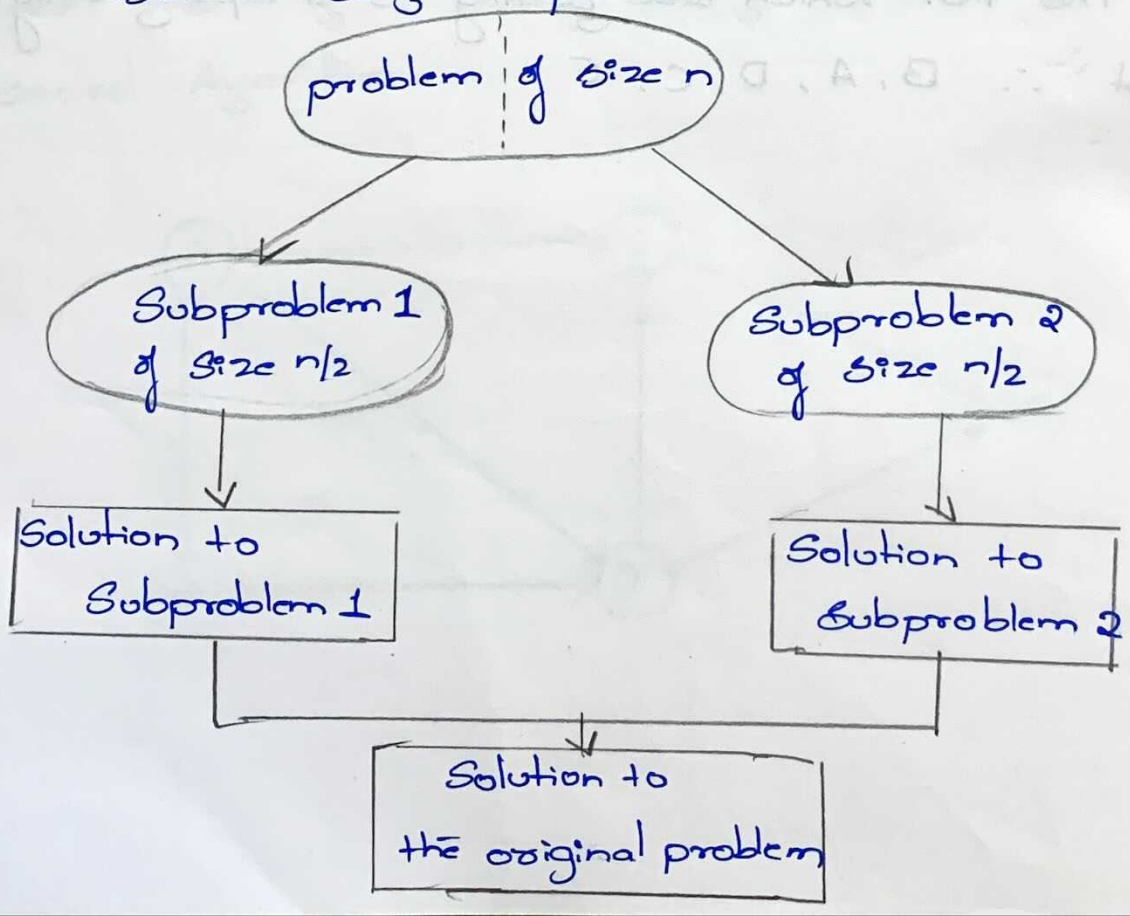
* the list which are getting is a topologically sorted list $\therefore$ B, A, D, C, E

Divide and Conquer

Def": Divide and Conquer is a top-down approach of designing algorithms in which the technique divides the given problem into smaller sub-problems and find sdution, then Combine their solutions to get the solution of the original problem.

Example :
* Merge Sort
* Quick Sort
* Binary search
* Strassen's Matrix Multiplication.

1. Divide : Given problem is divided into several subpro -blems of same type of equal size.
2. Conquer : Solve subproblems recursively.
3. Combine : Combine the sdutions of subproblems to get a solution original problem.



General Recurrence relation for the running time $T(n)$

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

* $f(n)$ → Time spent to dividing instance of size 'n' into n/b and combining their solutions.
* Solution order of growth $T(n)$ depends on constant values a and b & $f(n)$

If $f(n) \in \Theta(n^d)$ where $d \geq 0$ in recurrence equation then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

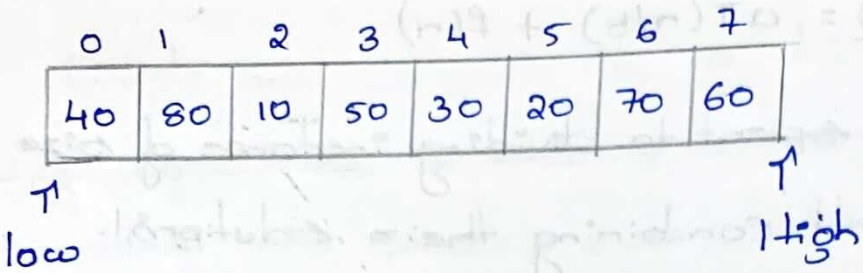
Merge Sort

* Important Sorting technique in Divide and Conquer strategy.
* Divide input elements according to their position in the array.

STEPS

1. Divide: Divide given array $A[0 \ldots n-1]$ into two halves $A[0 \ldots \frac{n}{2}-1]$ and $A[\frac{n}{2} \ldots n-1]$
2. Conquer: Sorting each half recursively
3. Combine: Merging the two smaller sorted array into a single sorted one.

Example : 40 80 10 50 30 20 70 60.



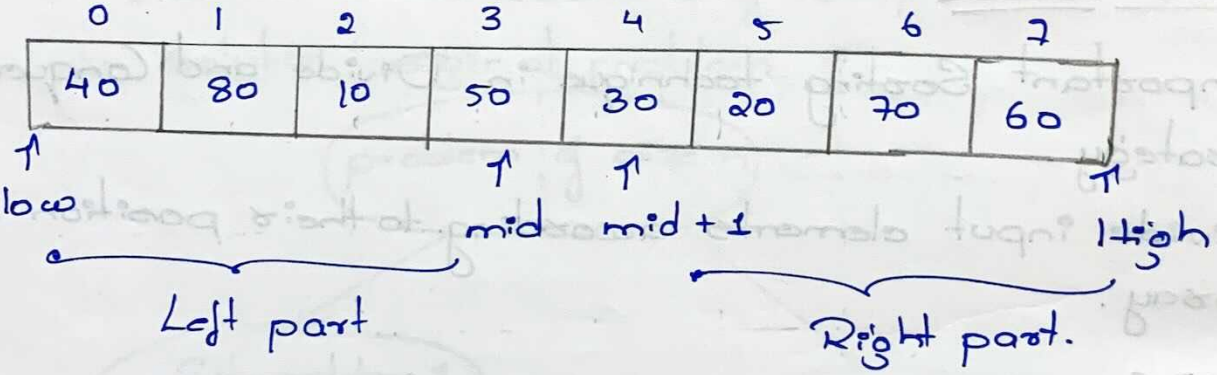
if (low > high) return

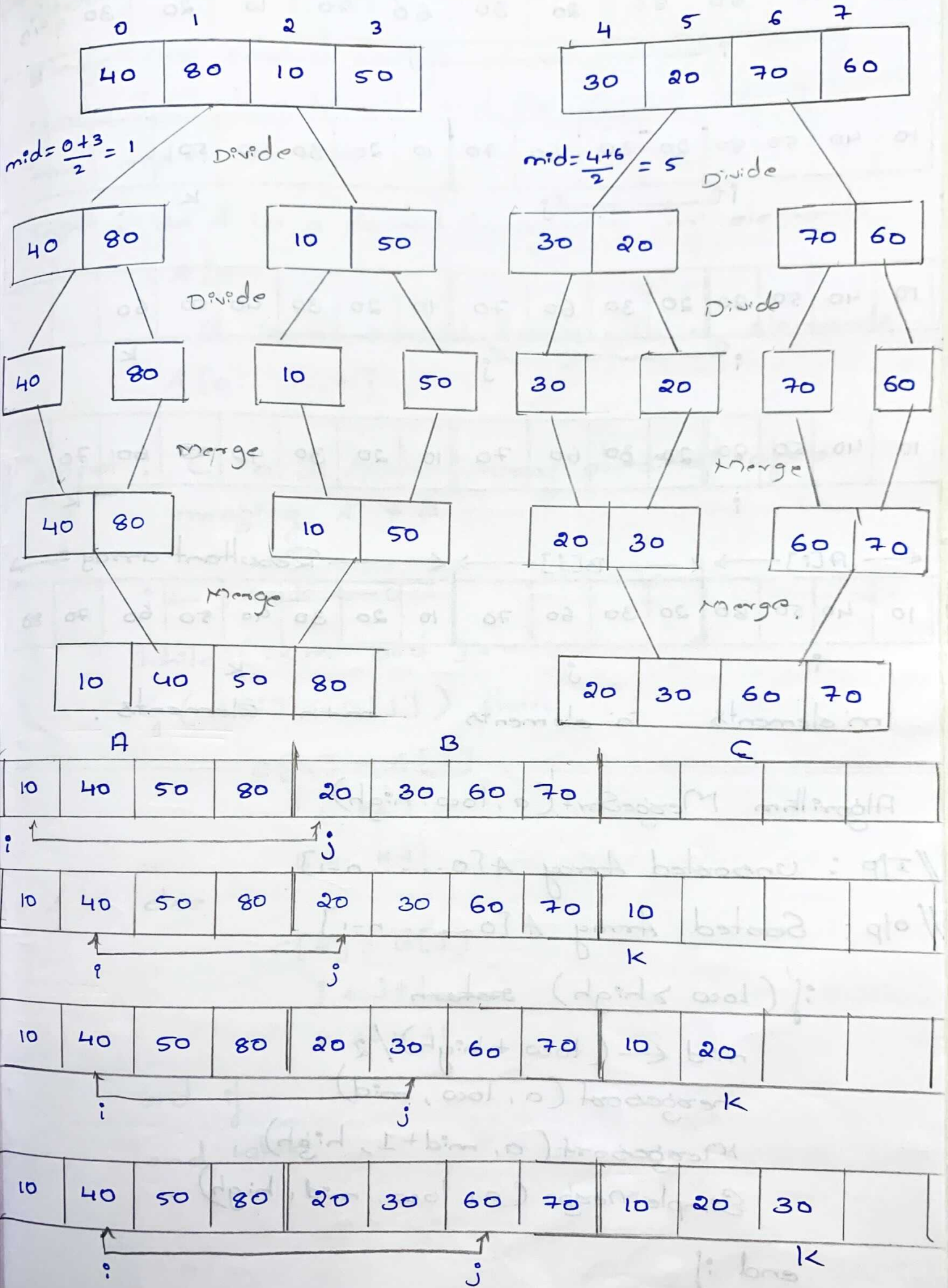
Calculate mid element

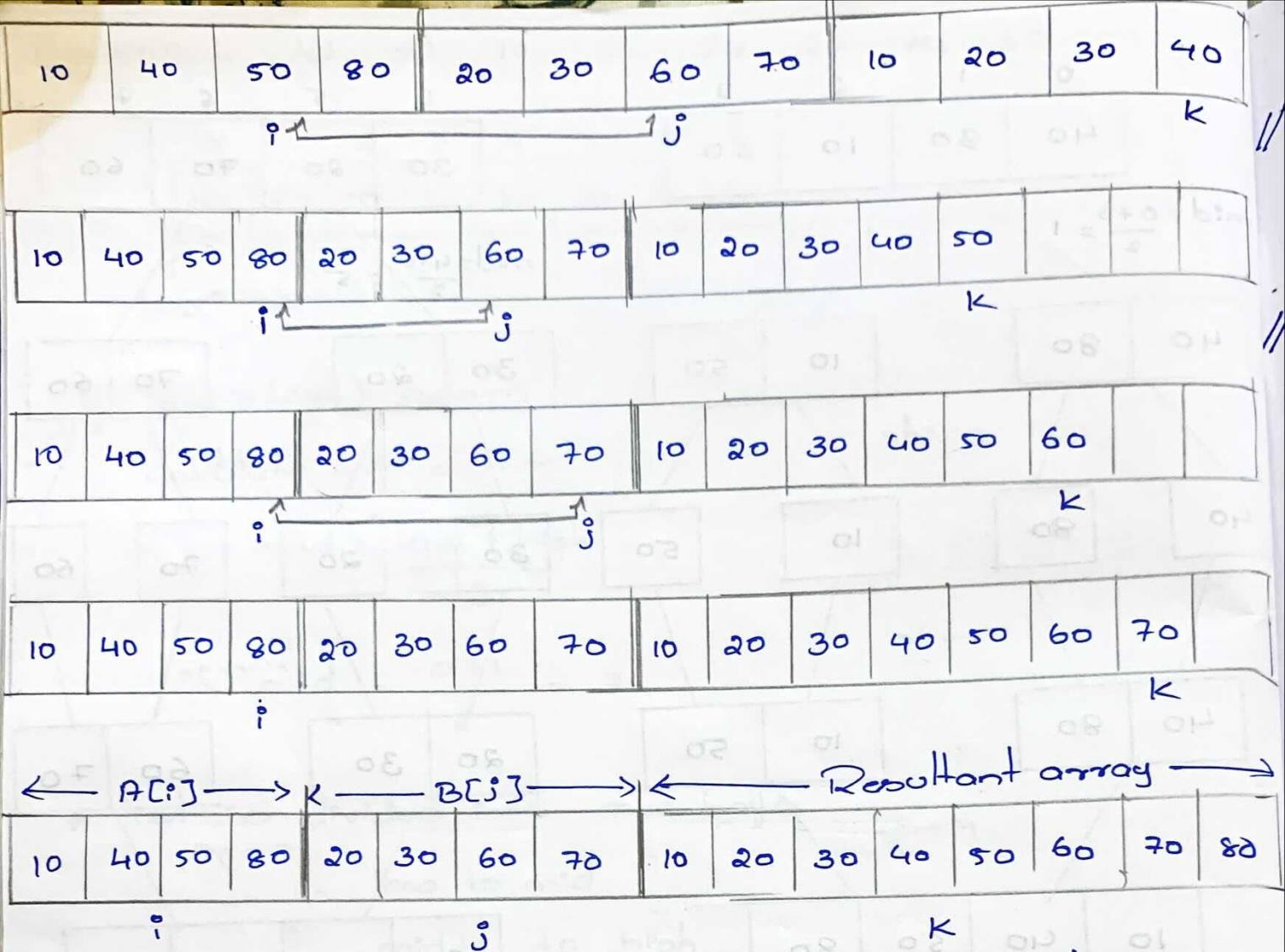
$$mid = \frac{low + high}{2}$$

$$mid = \frac{0+1}{2} = 3.$$

* list is divided into two halfs
  1. low to mid
  2. mid+1 to high.




0 1 2 3
40 80 10 50
4 5 6 7
30 20 70 60
mid = 0+3/2 = 1
Divide
mid = 4+6/2 = 5
Divide
40 80
10 50
30 20
70 60
Divide
Divide
40
80
10
50
30
20
70
60
Merge
Merge
40 80
10 50
20 30
60 70
Merge
Merge
10 40 50 80
20 30 60 70
A
B
C
10 40 50 80 20 30 60 70
i
j
10 40 50 80 20 30 60 70 10
i
j
k
10 40 50 80 20 30 60 70 10 20
i
j
k
10 40 50 80 20 30 60 70 10 20 30
i
j
k



Algorithm MergeSort(a, low, high)

// I/p : Unsorted Array A[0.....n-1]

// o/p : Sorted Array A[0.....n-1]

```
if (low > high) return
    mid ← (low + high)/2
    mergesort(a, low, mid)
    Mergesort(a, mid+1, high)
    SimpleMerge(a, low, mid, high)
end if
```

Algorithm SimpleMerge (A, B, C, m, n)

// Merge two sorted arrays where the first array starts from loco to mid and the second array starts from mid+1 to high

// Input : A is a sorted Array with m elements

A [0 . . . . . m]

B is a sorted Array with n elements

A [0 . . . . . n]

// output : C is a sorted array obtained after merging A + B

```
i ← j ← k ← 0
while (i < m and j < n)
if ( A[i] < B[j] ) then
        c[k] = A[i]
        i ← i+1
        k ← k+1
else
        c[k] = B[j]
        j ← j+1
        k ← k+1
end if
end while.
```

```
while (i < m)
    C[k] ← A[i]
    i ← i+1
    k ← k+1
end while.

while (j < n)
    C[k] ← B[j]
    j ← j+1
    k ← k+1
end while.
```

Time complexity using substitution method.

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

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

Replacing n by $\frac{n}{2}$

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

Substitute (2) in (1)

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

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

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

Replacing n by $\frac{n}{2^2}$ in Equation (1)

$$T\left[\frac{n}{2^2}\right) = 2T\left[\frac{2n}{2^3}\right) + \frac{n}{2^2} \quad \text{---- (4)}$$

Substitute Equation (4) in (3)

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

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

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

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

$$\vdots$$

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

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

To get the initial condition $T(1) = 0$, let $2^i = n$ ---- (5)

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

$$= n \cdot T(1) + n \cdot i$$

$$= n \cdot 0 + n \cdot i$$

$$= n \cdot i \quad \text{---- (6)}$$

From equation (5) we have $2^i = n$

Taking log on both sides we have,

$$\therefore \log_2 2^i = \log_2 n$$

$$i = \log_2 n$$

Substituting this in equation (6) we have,

$$T(n) = n \log_2 n$$

$\therefore$ Time complexity of merge sort is given by

$$\boxed{T(n) = \theta(n \log_2 n)}$$

Advantages

⇒ Merge sort algorithm is stable algorithm.

⇒ It can be applied to files of any size.

⇒ Since I/O is largely sequential, tapes can be used.

⇒ The time complexity is n.logn which is very efficient when compared to other algorithms

⇒ Most popular ~~der~~ algorithm for all types of sorting

Disadvantages

⇒ The algorithm uses extra space proportional to N. So, the algorithm is not in-place

⇒ It uses more memory on the stack because of recursion.

Quick Sort

Step 1 : Select an element as pivot element which divides the array into two subarrays (first Element)

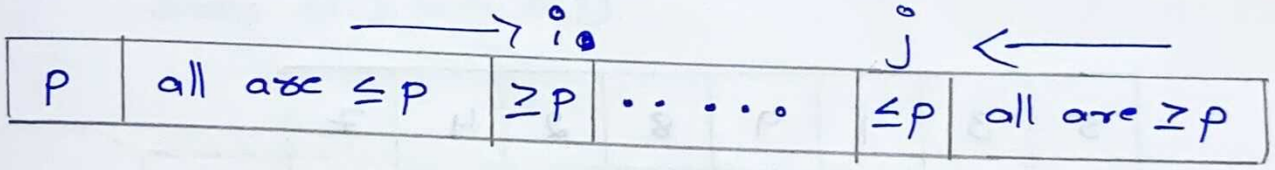
Step 2 : Two Scans of the subarray.

Step 2a: Left-to-Right Scan

* Starts with second element & Array index is 'i'
* Scan and identify the first element greater than or equal to the pivot ($p \geq i$)

Step 2b : Right-to-Left Scan

* Starts with last element & Array index is 'j'
* Scan and identify the first element smaller than or equal to the pivot ($p \leq j$)



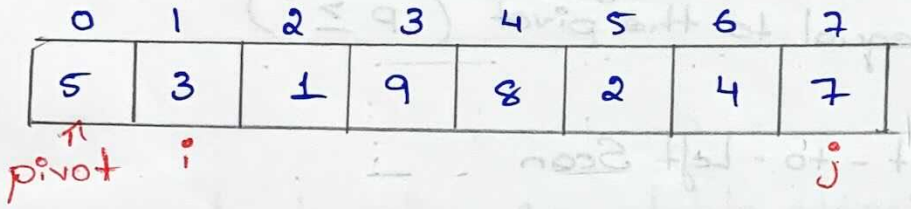
Step 3 : After both the scan stops three situations may arise

1. If Scanning indices i and j have not Crossed i.e. $i < j$ simply exchange a[i] and a[j] and resume the scans by incrementing 'i' and decrementing 'j'.

2. If the Scanning indices have Crossed over. i.e. $i > j$. Exchange the pivot with a[j] and partition the array.

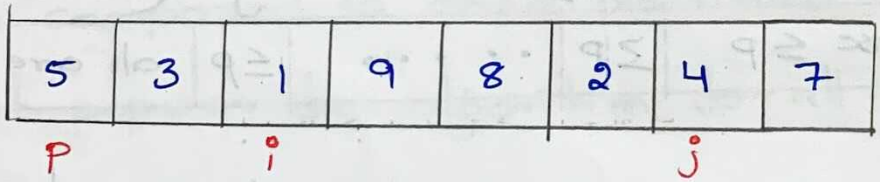
3. If the Scanning indices stops while pointing to the same element i.e. $i = j$, the value pointing must be equal to p. So the array partitioned with the split position $s = i = j$

Example 5 3 1 9 8 2 4 7



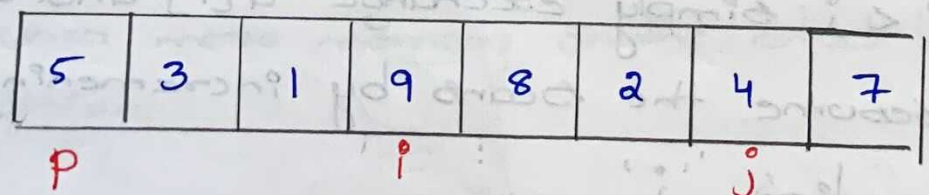
$\Rightarrow p \geq A[i]$ $(5 > 3)$ we will condition is True, we will increment i

$p \leq A[j]$ $(5 < 7)$ is True, we will decrement j



$p \geq A[i]$ $(5 \geq 1)$ is ~~false~~, we will ~~stop~~ increment i

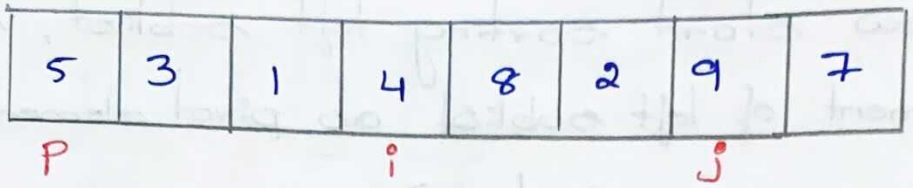
$p \leq A[j]$ $(5 \leq 4)$ is false, we will stop decrement



$p \geq A[i]$ $(5 \geq 9)$ is False, we will stop increment 'i'

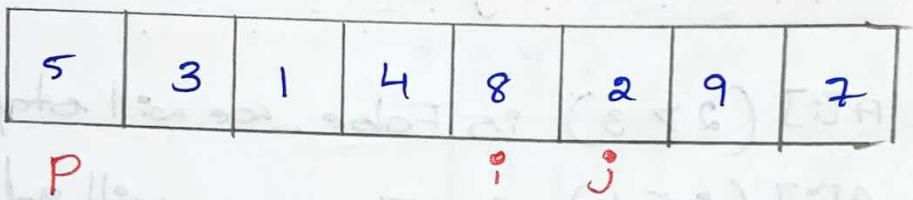
$p \leq A[j]$ $(5 \leq 4)$ is False, we will stop decrement 'j'

Both conditions are false then swap A[i] and A[j]



$p \geq A[i]$ $(5 \geq 4)$ is True, we will increment 'i'

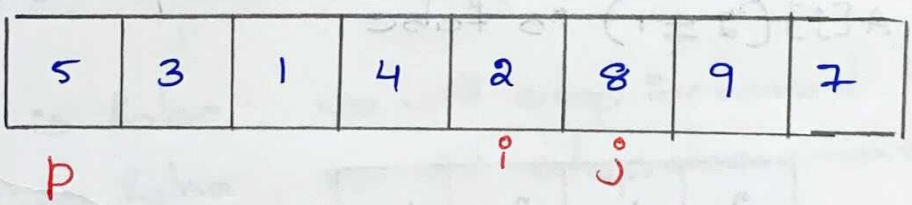
$p \leq A[j]$ $(5 \leq 9)$ is True, we will increment 'j'



$p \geq A[i]$ $(5 \geq 8)$ is false, we will stop increment

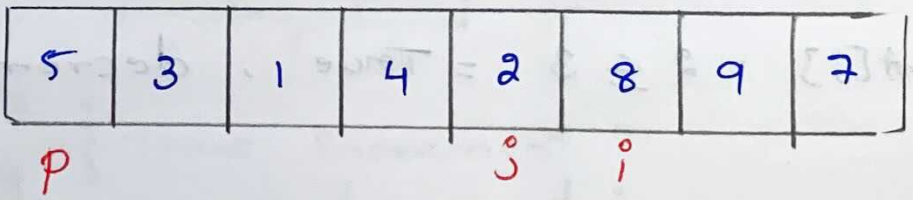
$p \leq A[j]$ $(5 \leq 2)$ is ~~false~~, we will stop ~~increment~~

Swap A[i] and A[j]

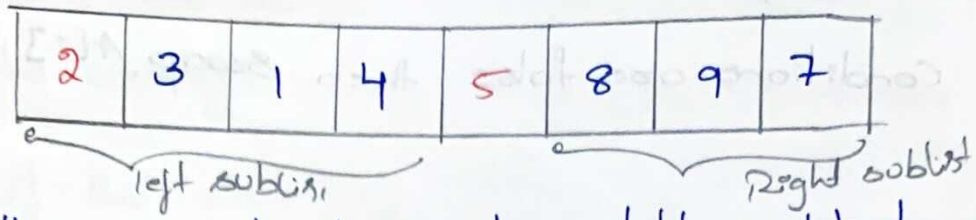


$p \geq A[i]$ $(5 \geq 2)$, is True, we will ~~stop~~ increment

$p \leq A[j]$ $(5 \leq 8)$ is True, we will decrement.

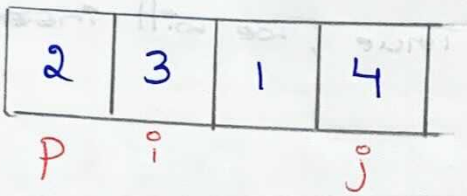


j Crossed : i.e., $j < i$ we will swap A[p] and A[j]



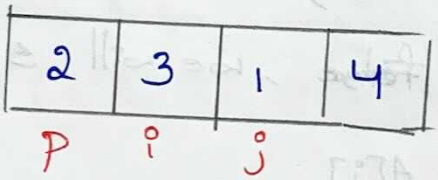
we will now start sorting left sublist, Assuming the first element of left sublist as pivot element.

Thus, now new pivot = 2.



~~A[i]~~ $P \geq A[i]$ $(2 \geq 3)$ is False, we will stop incrementing i

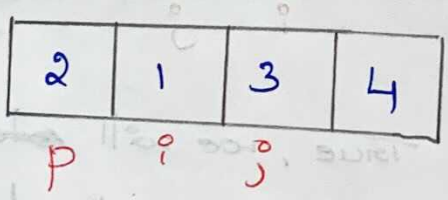
$P \leq A[j]$ $(2 \leq 4)$ is True, we will decrement j



$P \geq A[i]$ $(2 \geq 3)$ is false

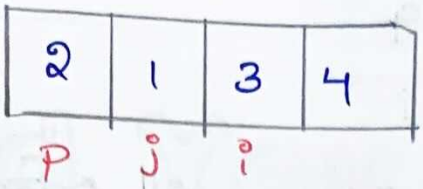
$P \leq A[j]$ $(2 \leq 1)$ is false

Swap A[i] and A[j]

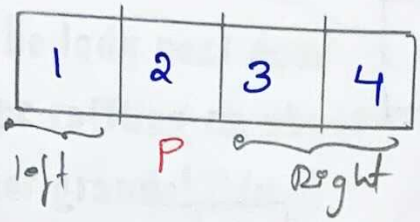


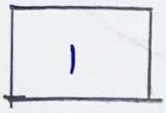
$P \geq A[i]$ = $2 \geq 1$ = True, increment i

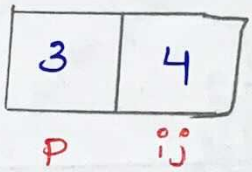
$P \leq A[j]$ = $2 \leq 3$ = True, decrement j



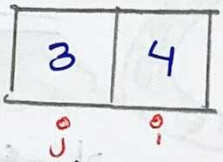
j Crossed i. i.e, j < i we will swap p[0] and A[j]

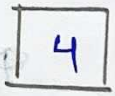


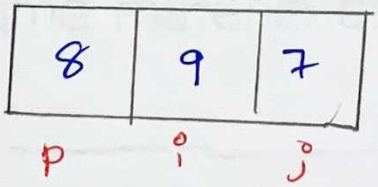




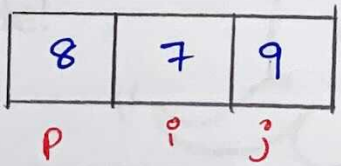
$3 \geq 4$
$3 \leq 4$ } True



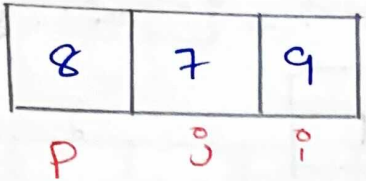




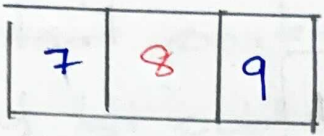
$8 \geq 9$ is false, we will stop increment i
$8 \leq 7$ is false, we will stop decrement j
Swap A[i] and A[j]

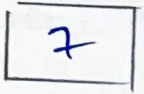


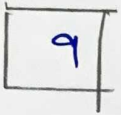
$8 \geq 7$
$8 \leq 9$ } True increment i
decrement j

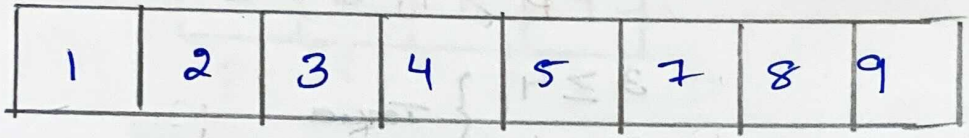


j Crossed i i.e, j < i we will scoop p[0] and A[j]









Sorted array

Algorithm Quick_Sort(A, low, high)

// purpose : Sort the element of the array using Quick sort

// Input : low: the position of the first element of array 'a'

: high: the position of the last element of array 'a'

a: It is an array consisting of unsorted elements

// output : a: It is an array consisting of sorted elements

```
if (low < high)
    k ← partition(a, low, high)
    QuickSort(a, low, k-1)
    QuickSork(a, k+1, high)
end if
```

Algorithm partition(A, low, high)

// purpose : Divide the array into parts such such that elements towards left of pivot element are ≤ pivot element and elements towards right of keys are ≥ pivot element.

// Input : low : the position of the first element of array 'a'

high : the position of the last element of array 'a'

a : It is an array consisting of unsorted elements

// output : a : partitioned array such that elements towards left of pivot element are ≤ pivot element and elements towards right of key are ≥ pivot elements.

```
pivot ← a[low]
i ← low
j ← high + 1

while (i <= j)
        do i ← i+1   while(pivot >= a[i])
        do j ← j-1   while(pivot <= a[j])
        if (i < j) exchange(a[i], a[j])
end while
exchange (a[j], a[low])
return j.
```

Time Complexity using substitution method

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

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

Replacing n by $\frac{n}{2}$

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

Substitute (2) in (1).

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

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

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

Replacing n by $\frac{n}{2^2}$ in Equation (1)

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2} \quad \text{---- (4)}$$

Substitute equation (4) in (3)

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

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

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

111'4

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

$$\vdots$$

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

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

To get the initial condition $T(1) = 0$, Let $2^i = n$ ---- (5)

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

$$= n \cdot T(1) + n \cdot i$$

$$= n \cdot 0 + n \cdot i$$

$$T(n) = n i \quad \text{---- (6)}$$

From equation (5) we have $2^i = n$

Taking log on both sides we have,

$$i \cdot \log_2 2^i = \log n$$

$$i = \log n$$

Substituting this in equation (6) we have,

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

∴ Time complexity of Quick sort is given by

$$\boxed{T(n) = \Theta(n \log n)}$$

Advantages

⇒ Time complexity : Quick sort requires time complexity of $O(n \log n)$ in the best case and average case to sort 'n' items

⇒ It has an extremely short inner loop

⇒ A very precise statement can be made about performance issues

Disadvantages

⇒ It is not stable

⇒ The time complexity of quick sort is quadratic i.e., $O(n^2)$ time in the worst case

Binary Tree Traversal

Binary Tree : A binary tree is a directed tree in which outdegree of each node is less than or equal to two i.e., each node in the tree can have 0, 1, or 2 children. In other words, a binary tree is a tree which can be empty or partitioned into 3 subgroups.

1] Root

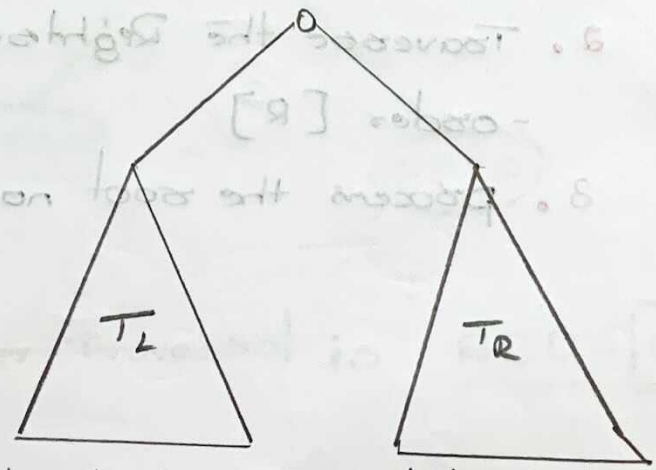
2] left Subtree

3] Right Subtree.

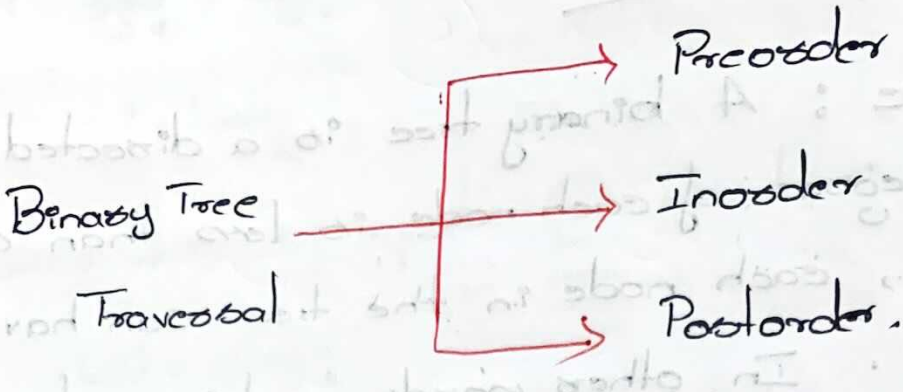
Root : the first node in the tree which does not have a parent is called root node.

Left Subtree : It is a tree which is connected to the left of root. Since this tree comes under root, it is called left subtree.

Right Subtree : It is a tree which is connected to the right of root. Since this tree comes under root, it is called Right subtree.

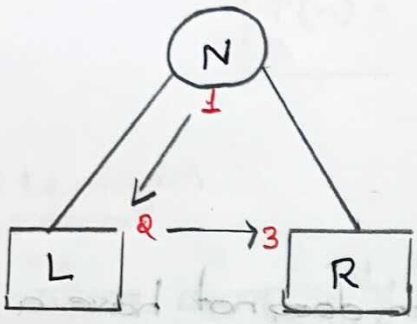


Standart representation of a binary tree



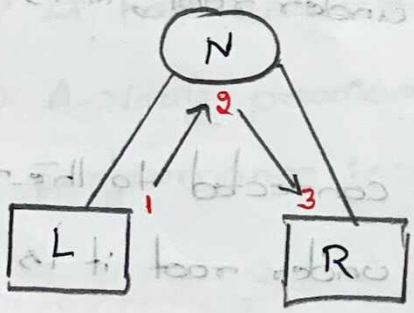
Preorder Traversal :

1. Process the root Node [N]
2. Traverse the left subtree in preorder [L]
3. Traverse the Right subtree preorder [R]



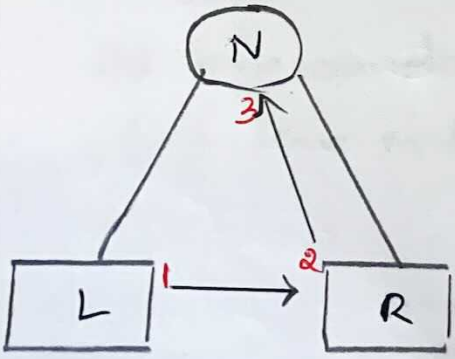
InOrder Traversal :

1. Traverse the left subtree inorder [L]
2. Process the root node [N]
3. Traverse the right subtree inorder [R]

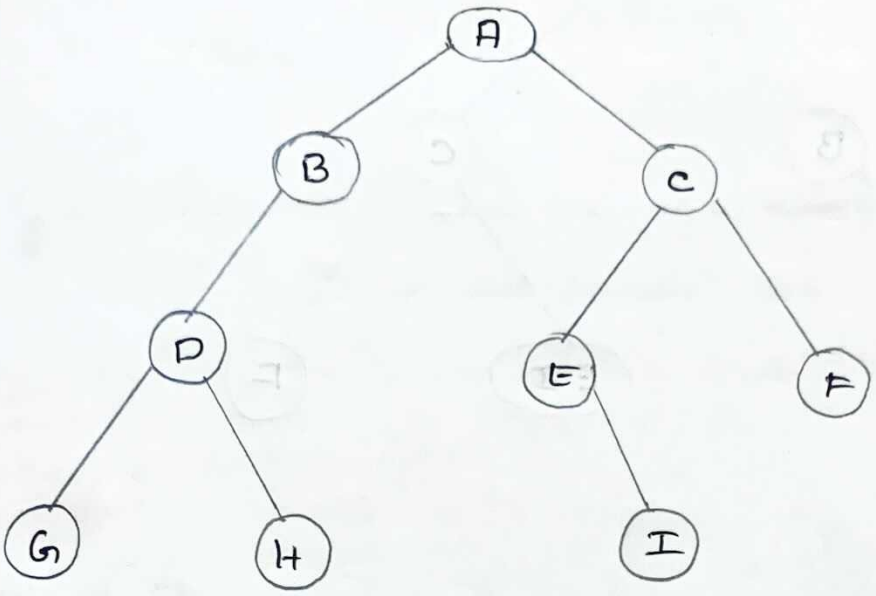


postorder Traversal :

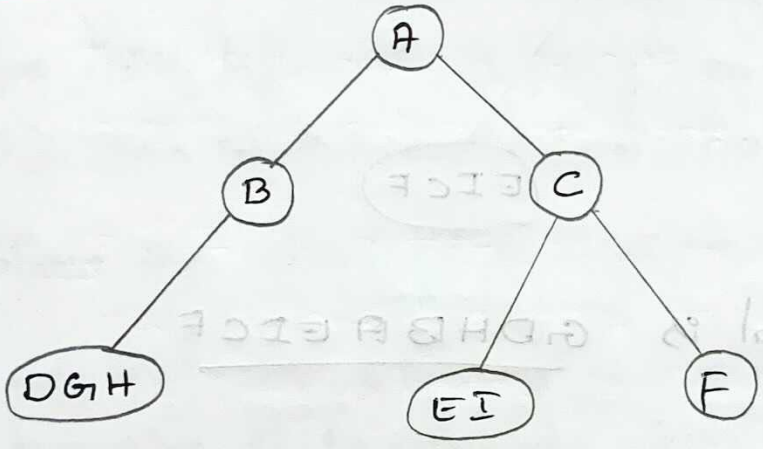
1. Traverse the Left subtree postorder [L]
2. Traverse the Right subtree post-order [R]
3. process the root node [N]

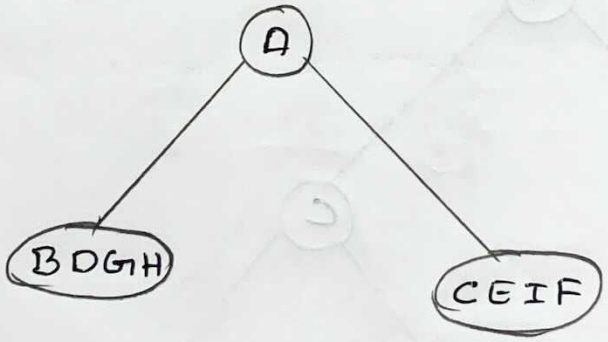


Example: Traverse the following tree in preorder, Inorder and postorder.



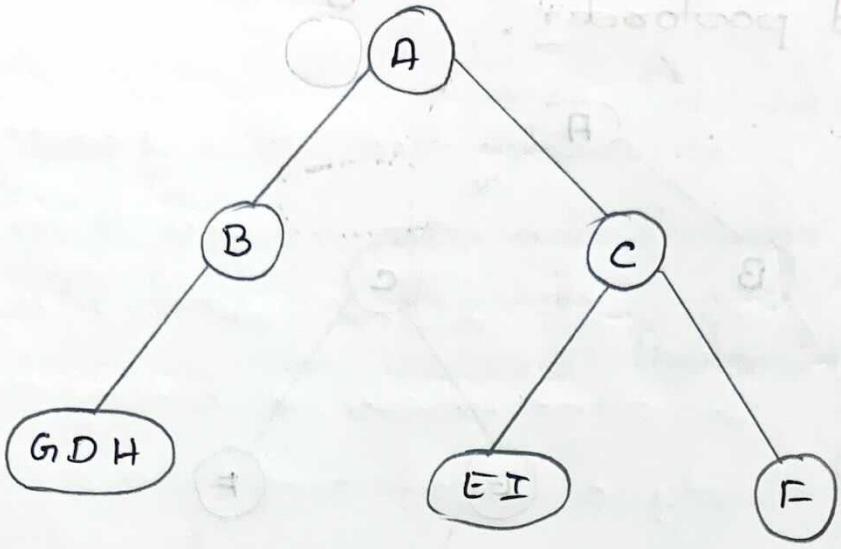
preorder :

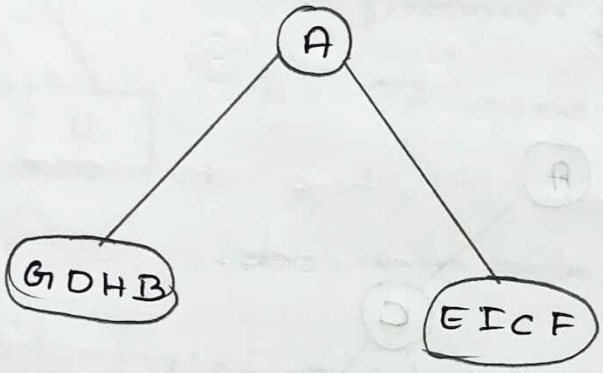




preorder Traversal is ABDGHCEIF

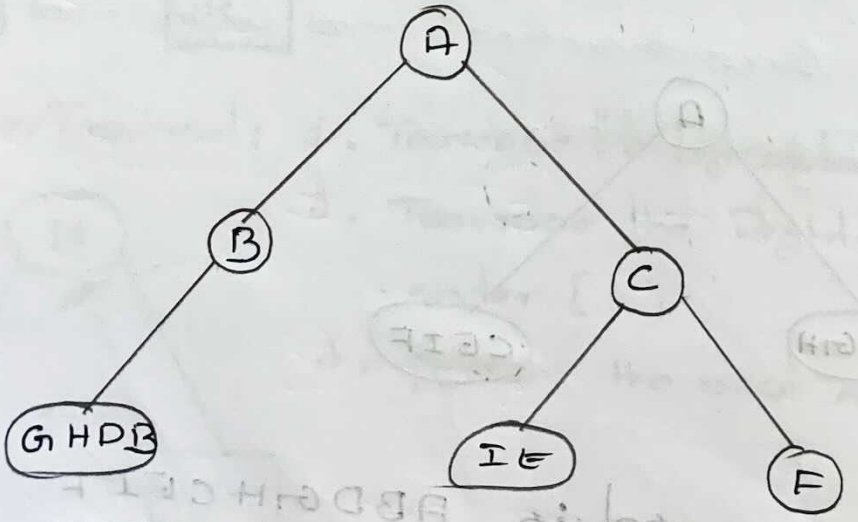
Inorder.

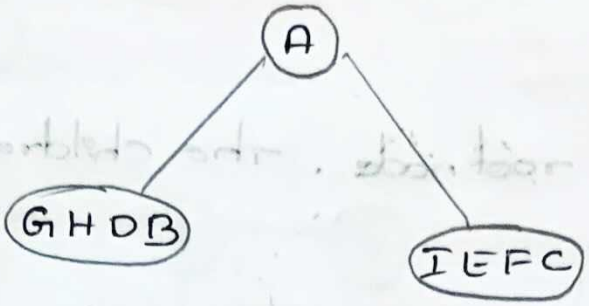




InOrder Traversal is GDHBAEICF

postorder:



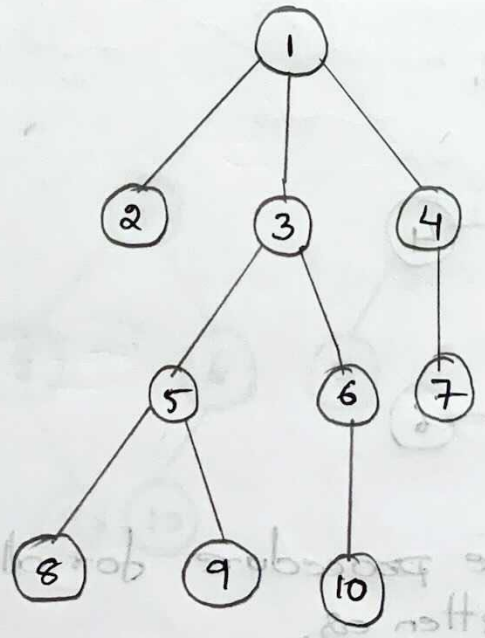


postorder traversal is GHDBIEFCA.

Convert general tree to binary tree (first child - next sibling representation)

* Start from the root node
* Connect the root to its leftmost offspring
* For this leftmost offspring, connect all the siblings which are in the same level from left to right.
* Follow the above steps for each subtree.

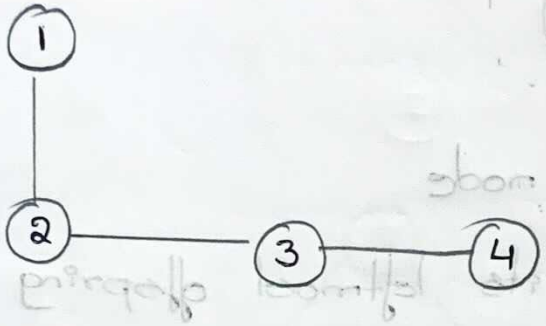
Ex: Convert the following tree into binary tree and traverse it in preorder, Inorder, postorder



Soln

Make node 1 as the root node. the children of node 1 are 2, 3 and 4

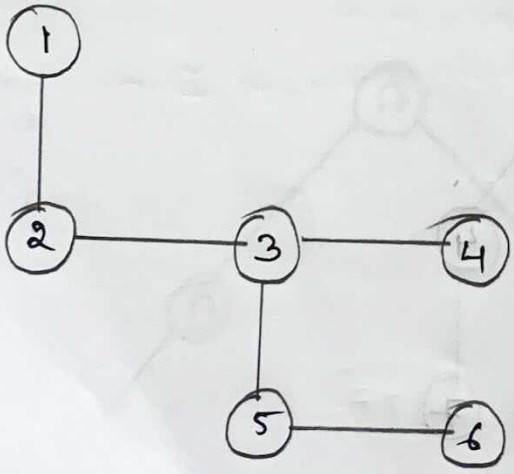
connect leftmost child 2 to node 1 and remaining siblings 3 and 4 are connected to 2 from left to right



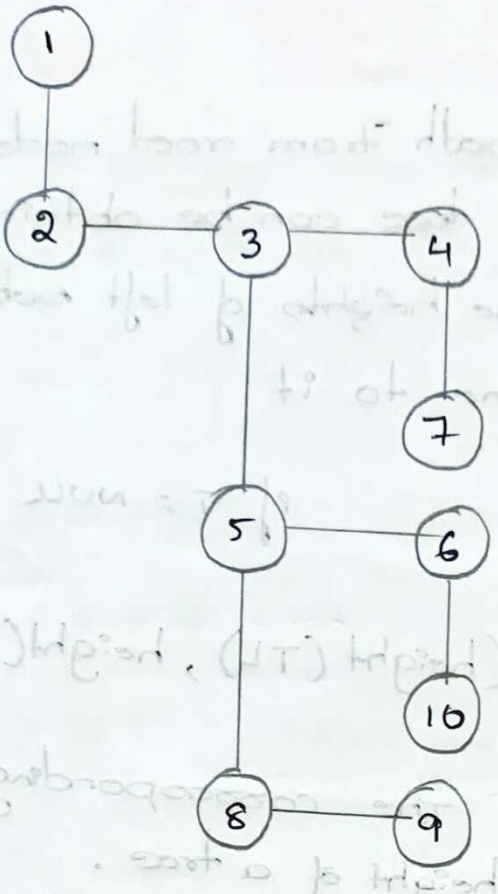
From the subtree whose root node is 3, the children are 5 and 6.

Connect leftmost child 5 to parent.

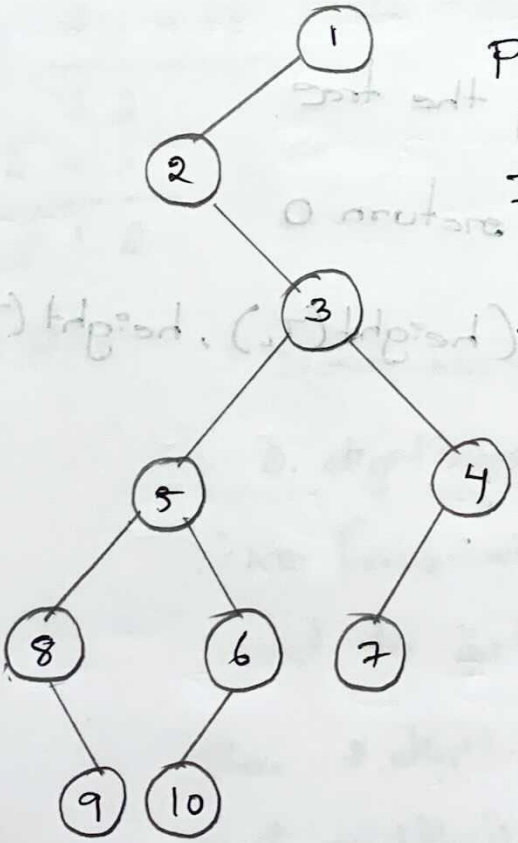
the sibling 6 is connected to 5 from left to Right.



By repeating the same procedure for all the subtrees, the final tree can be written as,



the above tree can be represented as binary tree by Considering the Vertical lines as left children and horizont -al lines as the right children.



preorder: 1, 2, 3, 5, 8, 9, 6, 10
4, 7

Inorder: 2, 8, 9, 5, 10, 6, 3
7, 4, 1

postorder: 9, 8, 10, 6, 5, 7
4, 3, 2, 1

Height of a binary tree

The length of the longest path from root node to a leaf node. The height of the tree can be obtained by finding the maximum of the heights of left subtree to right subtree and adding one to it

$$\text{Height}(T) = \begin{cases} 0 & \text{if } T = \text{NULL} \\ 1 + \max(\text{height}(T_L), \text{height}(T_R)) & \text{otherwise} \end{cases}$$

the corresponding algorithm to find the height of a tree.

```
Algorithm height (T)
// purpose : to compute the height of the tree
// Input   : A binary tree
// output  : the height of the tree
    if (T == NULL) return 0
    return 1 + max(height(T_L), height(T_R))
```

Multiplication of large Integers

* Let a & b be two integers, the product of a & b can be expressed as $\boxed{c = a \times b}$

* If $a = 5$, $b = 7$

$$c = a \times b$$

$$c = 5 \times 7$$

$$c = 35$$

Here we have performed 1 multiplication operation i.e. one digit no

* If $a = 24$, $b = 13$

$$c = a \times b$$

$$c = 24 \times 13$$

$$\begin{array}{r} 72 \\ 24\times \\ \hline 312 \end{array}$$

$$c = 312$$

Here we have multiplied '3' with 2 digits and same way to get 24 we have multiplied '1' with two digits and to get an answer 312

So. 2 digit no 2 times, 2 digit no & no of multiplication we are doing is 4

* In Brute force method, if 'n' denotes the size of 2 integers (a & b) the no of multiplication done is n^2

for $n = 1$, we got 1

for $n = 2$, we got 4

for $n = 3$, we got 9

* By using divide and conquer, we decrease the time complexity to $n^{1.59}$

procedure

1. Let a & b be two integers. Size of a & b be 'n'
2. Represent $a = a_1 a_0$

   $b = b_1 b_0$ where a_1, a_0, b_1, b_0 are digits of size $\frac{n}{2}$

* the product c is calculated as,

Step 1 : $c_0 = a_0 \times b_0$

Step 2 : $c_2 = a_1 \times b_1$

Step 3 : $c_1 = (a_0 + a_1) * (b_0 + b_1) - (c_2 + c_0)$

Step 4 : $c = c_2 10^n + c_1 10^{n/2} + c_0$

Example :

1] $a = 26, \quad b = 45$

$n = 2$

$a_1 = 2$

$a_0 = 6$

$b_1 = 4$

$b_0 = 5$

Step 1 : $C_0 = a_0 \times b_0$

$C_0 = 26 \times 5 = 30$

Step 2 : $C_2 = a_1 \times b_1$

$C_2 = 2 \times 4 = 8$

Step 3 : $C_1 = (a_0 + a_1) * (b_0 + b_1) - (C_2 + C_0)$

$C_1 = (6+2) * (5+4) - (8+30)$

$C_1 = 8 * 9 - 38$

$C_1 = 34$

Step 4 : $C = C_2 10^n + C_1 10^{n/2} + C_0$

$C = 8 \times 10^2 + C_1 10 + 30$

$C = 800 + 340 + 30$

$C = 1170$

2] a = 1212 b = 322

To make equal size just add '0' at proceeders of 'b' because 'a' is having 4 digit no 'b' is having 3 digit no so make it b = 0322.

$a_1 = 12$ $n = 4$

$a_0 = 12$

$b_1 = 03$

$b_0 = 22$

Step 1 : $C_0 = a_0 \times b_0$

$C_0 = 12 \times 22$

$C_0 = 264$

Step 2 : $C_2 = a_1 \times b_1$

$C_2 = 12 \times 03$

$C_2 = 36$

Step 3 : $C_1 = (a_0 + a_1) * (b_0 + b_1) - (C_2 + C_0)$

$C_1 = (12 + 12) * (22 + 03) - (36 + 264)$

$C_1 = 300$

Step 4 : $C = C_2 10^n + C_1 10^{n/2} + C_0$

$C = 36 \times 10^4 + 300 \times 10^2 + 264$

$C = 390204$

Analysis

$$T(n) = 3 \cdot T\left[\frac{n}{2}\right] \quad \text{——— (1)}$$

Replacing n by $\frac{n}{2}$ in above equation, we have

$$T\left[\frac{n}{2}\right] = 3T\left[\frac{n}{2^2}\right] \quad \text{——— (2)}$$

Subtitute equation (2) in (1)

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

$$= 3^2 T\left[\frac{n}{2^2}\right]$$

$$= 3^3 T\left[\frac{n}{2^3}\right]$$

$$= 3^4 T\left[\frac{n}{2^4}\right]$$

$$\vdots$$

$$= 3^i T\left[\frac{n}{2^i}\right] \quad \text{——— (3)}$$

To get the terminal condition $T(1) = 1$,

Let $n = 2^i$ ——— (4)

Subtitute equ (4) in (3)

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

$$= 3^i T(1)$$

$$T(n) = 3^i \quad \text{——— (5)}$$

$2^i = n$

Taking log on both sides we get

$\therefore \log 2^i = \log n$

$\therefore i = \log n$

Substituting i in Equation (5)

$T(n) = 3^i$

$T(n) = 3^{\log n}$

$T(n) = n^{\log 3}$

$T(n) = n^{1.585}$

$\therefore$ Time complexity of multiply two integers using divide and conquer is given by

$$\boxed{T(n) = \Theta(n^{1.585})}$$

Strassen's Matrix Multiplication

* Strassen's matrix multiplication is usually done by brute force approach, and by many approaches.
* Bruteforce approach, which takes 8 multiplications & 4 additions for 2×2 matrix
* the divide and conquer approach can be used for implementing strassen's matrix multiplication

Divide : Divide the matrices into submatrices $A_0, A_1 \ldots\ldots$

Conquer : Use a group of matrix multiplication Equations

Combine : Recursively multiply submatrices and get find result of multiplication after performing required addition or subtraction.

Algorithm mat_mul (A, B, C, n)

this is accomplished by using the following formulas

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$S_2 = (A_{21} + A_{22}) \times B_{11}$$

$$S_3 = A_{11} \times (B_{12} - B_{22})$$

$$S_4 = A_{22} \times (B_{21} - B_{11})$$

$$S_5 = (A_{11} + A_{12}) \times B_{22}$$

$$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$C_{11} = S_1 + S_4 - S_5 + S_7$$

$$C_{12} = S_3 + S_5$$

$$C_{21} = S_2 + S_4$$

$$C_{22} = S_1 + S_3 - S_2 + S_6$$

Example:

1] $\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \begin{bmatrix} 8 & 7 \\ 1 & 2 \end{bmatrix}$

Sol^n

$$A = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 8 & 7 \\ 1 & 2 \end{bmatrix}$$

$A_{11} = 1$
$A_{12} = 2$
$A_{21} = 5$
$A_{22} = 6$

$B_{11} = 8$
$B_{12} = 7$
$B_{21} = 1$
$B_{22} = 2$

$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$S_1 = (1+6)(8+2)$

$S_1 = 7 \times 10$

$\boxed{S_1 = 70}$

$S_2 = (A_{21} + A_{22}) \times B_{11}$

$S_2 = (5+6) \times 8$

$S_2 = 11 \times 8$

$\boxed{S_2 = 88}$

$S_3 = A_{11} \times (B_{12} - B_{22})$

$S_3 = 1 \times (7-2)$

$S_3 = 1 \times 5$

$\boxed{S_3 = 5}$

$S_4 = A_{22} \times (B_{21} - B_{11})$

$S_4 = 6 \times (1-8)$

$S_4 = 6(-7)$

$\boxed{S_4 = -42}$

$S_5 = (A_{11} + A_{12}) \times B_{22}$

$S_5 = (1+2) \times 2$

$S_5 = 3 \times 2$

$\boxed{S_5 = 6}$

$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$

$S_6 = (5-1) \times (8+7)$

$S_6 = 4 \times 15$

$\boxed{S_6 = 60}$

$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$

$S_7 = (2-6) \times (1+2)$

$S_7 = (-4) \times 3$

$\boxed{S_7 = -12.}$

$C_{11} = S_1 + S_4 - S_5 + S_7$

$C_{11} = 70 + (-42) - 6 + (-12)$

$C_{11} = 70 - 42 - 6 - 12$

$\boxed{C_{11} = 10}$

$C_{12} = S_3 + S_5$

$C_{12} = 5 + 6$

$\boxed{C_{12} = 11}$

$C_{21} = S_2 + S_4$

$C_{21} = 88 + (-42)$

$\boxed{C_{21} = 46}$

$C_{22} = S_1 + S_3 - S_2 + S_6$

$C_{22} = 70 + 5 - 88 + 60$

$\boxed{C_{22} = 47}$

The final matrix is

$$C = \begin{bmatrix} 10 & 11 \\ 46 & 47 \end{bmatrix}$$

2]

$$A = \begin{bmatrix} 5 & 3 & 0 & 2 \\ 4 & 3 & 2 & 6 \\ 7 & 8 & 1 & 4 \\ 9 & 4 & 6 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 3 & 2 & 4 & 7 \\ 2 & 5 & 2 & 9 \\ 3 & 9 & 0 & 3 \\ 7 & 6 & 2 & 1 \end{bmatrix}$$

Let, $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$$A = \begin{bmatrix} \begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 2 & 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 \\ 9 & 4 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \end{bmatrix} \qquad B = \begin{bmatrix} \begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 4 & 7 \\ 2 & 9 \end{bmatrix} \\ \begin{bmatrix} 3 & 9 \\ 7 & 6 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \end{bmatrix}$$

$S_1 = (A_{11} + A_{22})(B_{11} + B_{22})$

$$= \left[\begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \right] \left[\begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} + \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \right]$$

$$= \begin{bmatrix} \begin{bmatrix} 6 & 7 \\ 10 & 10 \end{bmatrix} \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} 18+28 & 30+42 \\ 30+40 & 50+60 \end{bmatrix}$$

$$S_1 = \begin{bmatrix} 46 & 72 \\ 70 & 110 \end{bmatrix}$$

$$S_2 = (A_{21} + A_{22})\, B_{11}$$

$$= \left[\begin{bmatrix} 7 & 8 \\ 9 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} \right]$$

$$= \left[\begin{bmatrix} 8 & 12 \\ 15 & 11 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 24+24 & 16+60 \\ 45+22 & 30+55 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 48 & 76 \\ 67 & 85 \end{bmatrix}$$

$$S_3 = A_{11} \times (B_{12} - B_{22})$$

$$= \left[\begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} \times \begin{bmatrix} 4 & 7 \\ 2 & 9 \end{bmatrix} - \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \right]$$

$$= \left[\begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} \times \begin{bmatrix} 4 & 4 \\ 0 & 8 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 20+0 & 20+24 \\ 16+0 & 16+24 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} 20 & 44 \\ 16 & 40 \end{bmatrix}$$

$$S_4 = A_{22} \times (B_{21} - B_{11})$$

$$= \left[\begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 3 & 9 \\ 7 & 6 \end{bmatrix} - \begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} \right]$$

$$= \left[\begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 0 & 7 \\ 5 & 1 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 0+20 & 7+4 \\ 0+35 & 42+7 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} 20 & 11 \\ 35 & 49 \end{bmatrix}$$

$$S_5 = (A_{11} + A_{12}) \times B_{22}$$

$$= \left[\begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 0 & 2 \\ 2 & 6 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \right]$$

$$= \left[\begin{bmatrix} 5 & 5 \\ 6 & 9 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 0+10 & 15+5 \\ 0+18 & 18+9 \end{bmatrix}$$

$$S_5 = \begin{bmatrix} 10 & 20 \\ 18 & 27 \end{bmatrix}$$

$$S_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})$$

$$= \left[\begin{bmatrix} 7 & 8 \\ 9 & 4 \end{bmatrix} - \begin{bmatrix} 5 & 3 \\ 4 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 2 \\ 2 & 5 \end{bmatrix} + \begin{bmatrix} 4 & 7 \\ 2 & 9 \end{bmatrix} \right]$$

$$= \left[\begin{bmatrix} 2 & 5 \\ 5 & 1 \end{bmatrix} \times \begin{bmatrix} 7 & 9 \\ 4 & 14 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 14+20 & 18+70 \\ 35+4 & 45+14 \end{bmatrix}$$

$$S_6 = \begin{bmatrix} 34 & 88 \\ 39 & 59 \end{bmatrix}$$

$$S_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})$$

$$= \left[\begin{bmatrix} 0 & 2 \\ 2 & 6 \end{bmatrix} - \begin{bmatrix} 1 & 4 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 3 & 9 \\ 7 & 6 \end{bmatrix} + \begin{bmatrix} 0 & 3 \\ 2 & 1 \end{bmatrix} \right]$$

$$= \begin{bmatrix} -1 & -2 \\ -4 & -1 \end{bmatrix} \times \begin{bmatrix} 3 & 12 \\ 9 & 7 \end{bmatrix}$$

$$= \begin{bmatrix} -3-18 & -12-14 \\ -12-9 & -48-7 \end{bmatrix}$$

$$S_7 = \begin{bmatrix} -21 & -26 \\ -21 & -55 \end{bmatrix}$$

$$C_{11} = S_1 + S_4 - S_5 + S_7$$

$$= \begin{bmatrix} 46 & 72 \\ 70 & 110 \end{bmatrix} + \begin{bmatrix} 20 & 11 \\ 35 & 49 \end{bmatrix} - \begin{bmatrix} 10 & 20 \\ 18 & 27 \end{bmatrix} + \begin{bmatrix} -21 & -26 \\ -21 & -55 \end{bmatrix}$$

$$= \begin{bmatrix} 66 & 83 \\ 105 & 159 \end{bmatrix} - \begin{bmatrix} 10 & 26 \\ 18 & 27 \end{bmatrix} - \begin{bmatrix} -21 & -26 \\ -21 & -55 \end{bmatrix}$$

$$C_{11} = \begin{bmatrix} 77 & 89 \\ 108 & 187 \end{bmatrix} \cdot \begin{bmatrix} 66 & 83 \\ 105 & 159 \end{bmatrix} - \begin{bmatrix} 31 & 46 \\ 39 & 82 \end{bmatrix}$$

$$C_{11} = \begin{bmatrix} 35 & 37 \\ 66 & 72 \end{bmatrix}$$

$$C_{12} = S_3 + S_5$$

$$= \begin{bmatrix} 20 & 44 \\ 16 & 40 \end{bmatrix} + \begin{bmatrix} 10 & 20 \\ 18 & 27 \end{bmatrix}$$

$$C_{12} = \begin{bmatrix} 30 & 64 \\ 34 & 67 \end{bmatrix}$$

$$C_{21} = S_2 + S_4$$

$$= \begin{bmatrix} 48 & 76 \\ 67 & 85 \end{bmatrix} + \begin{bmatrix} 20 & 11 \\ 35 & 49 \end{bmatrix}$$

$$C_{21} = \begin{bmatrix} 68 & 87 \\ 102 & 134 \end{bmatrix}$$

$$C_{22} = S_1 + S_3 - S_2 + S_6$$

$$= \begin{bmatrix} 46 & 72 \\ 70 & 110 \end{bmatrix} + \begin{bmatrix} 20 & 44 \\ 16 & 40 \end{bmatrix} - \begin{bmatrix} 48 & 76 \\ 67 & 85 \end{bmatrix} + \begin{bmatrix} 34 & 88 \\ 39 & 59 \end{bmatrix}$$

$$= \begin{bmatrix} 66 & 116 \\ 86 & 150 \end{bmatrix} - \begin{bmatrix} 48 & 76 \\ 67 & 85 \end{bmatrix} + \begin{bmatrix} 34 & 88 \\ 39 & 59 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 77 & -48 \\ -20 & 12 \end{bmatrix}$$

$$C_{22} = \begin{bmatrix} 66 & 116 \\ 86 & 150 \end{bmatrix} - \begin{bmatrix} 14 & -12 \\ 28 & 26 \end{bmatrix}$$

$$C_2 = \begin{bmatrix} 52 & 128 \\ 58 & 124 \end{bmatrix}$$

thus the final Product matrix C will be

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$= \begin{bmatrix} 35 & 37 & 30 & 64 \\ 66 & 77 & 34 & 67 \\ 68 & 87 & 52 & 128 \\ 102 & 134 & 58 & 124 \end{bmatrix}$$

Analysis

$$T(n) = 7 \cdot T\left[\frac{n}{2}\right] \quad \text{---(1)}$$

Replacing n by $\frac{n}{2}$ in above equation (1)

$$T\left[\frac{n}{2}\right] = 7 \cdot T\left[\frac{n}{2^2}\right] \quad \text{---(2)}$$

Substitute equation (2) in (1)

$$T(n) = 7 \cdot 7\, T\left[\frac{n}{2^2}\right]$$

$$= 7^2 T\left[\frac{n}{2^2}\right]$$

$$= 7^3 T\left[\frac{n}{2^3}\right]$$

$$= 7^4 T\left[\frac{n}{2^4}\right]$$

$$\vdots$$

$$= 7^{i} T\left[\frac{n}{2^i}\right]$$

To get the terminal condition $T(1) = 1$, Let $n = 2^i$ — (3)

$$T(n) = 7^{i} T\left[\frac{n}{n}\right]$$

$$T(n) = 7^{i} T(1)$$

$$T(n) = 7^{i} \quad \text{— (4)}$$

From Equation (3) we have $2^i = n$. So, taking log on both side we get.

~~$i \log 2$~~

$$\log_2 2^i = \log_2 n$$

$$i \log_2 2 = \log_2 n$$

$$i = \log_2 n \quad \text{— (5)}$$

Substituting Equation (5) in (4)

$$T(n) = 7^{\log_2 n}$$

$$T(n) = n^{\log_2 7}$$

$$T(n) = n^{2.807}$$

$$\boxed{T(n) = \Theta\left(n^{2.807}\right)}$$