Unit – 1

1. What is Algorithm?

- CATEGORIES
  - Search
  - Sort
  - Insert
  - Update
  - Delete

2. How to Write an Algorithm?

- **Step 1 – START ADD**
- **Step 2 – get values of a & b**
- **Step 3 – c← a + b**
- **Step 4 – display c**
- **Step 5 – Stop**

3. Use of the Algorithm?

- **Computer Science**
- **Mathematics**
- **Operations Research**
- **Artificial Intelligence**
- **Data Science**

4. Types of Algorithms.

- **Brute Force Algorithm**
- **Recursive Algorithm**
- **Backtracking Algorithm**
- **Searching Algorithm**
- **Sorting Algorithm**
- **Hashing Algorithm**
- **Divide and Conquer Algorithm**
- **Greedy Algorithm**

5. Advantages of Algorithms:

- It is easy to understand.
- An algorithm is a step-wise representation of a solution to a given problem.
- In an Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

6. Disadvantages of Algorithms:

- Writing an algorithm takes a long time so it is time-consuming.
- Understanding complex logic through algorithms can be very difficult.
- Branching and Looping statements are difficult to show in Algorithms(imp).

7. Efficiency of an Algorithm:

- Computer resources are limited that should be utilized efficiently. The efficiency of an algorithm is defined as the number of computational resources used by the algorithm.

- An algorithm must be analysed to determine its resource usage. The efficiency of an algorithm can be measured based on the usage of different resources.

- For maximum efficiency of algorithm, we wish to minimize resource usage. The important resources such as time and space complexity cannot be compared directly, so time and space complexity could be considered for an algorithmic efficiency.

8. Average ,Best and Worst case Analysis(Time Complexity)

- **Best Case Analysis:**
- **Worst Case Analysis:**
- **Average Case Analysis:**

9. Asymptotic Notations:

- **Big O Natation(O):**
- **Omega Notation(Ω):**
- **Theta Notation(Θ):**

10. Analysing Control statement:

- **Sequence logic, or sequential flow**
- **Selection logic, or conditional flow**
- **Iteration logic, or repetitive flow**

11. Sorting Algorithms and Analysis:

- **Bubble Sort**
- **Selection Sort**
- **Insertion Sort**
- **Shell Sort**
- **Heap Sort**
- **Bucket Sort**
- **Radix Sort**
- **Counting Sort**

12. Loop invariant and the correctness of the algorithm.

- **Repeat-For Structure:**
  - **This structure has the form:**

    Repeat for i = A to N by I:

    [Module]

    [End of loop]

- **Repeat-While Structure:**
  - **This Structure has the form:**

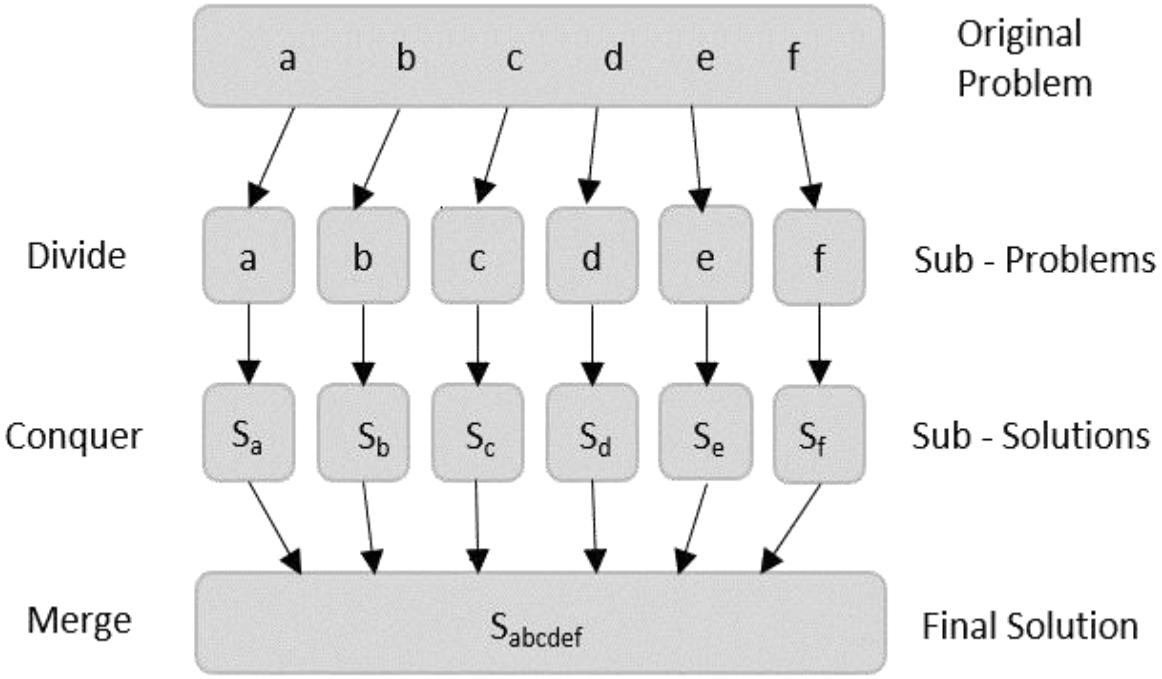
    Repeat While condition:

    [Module]

    [End of loop]

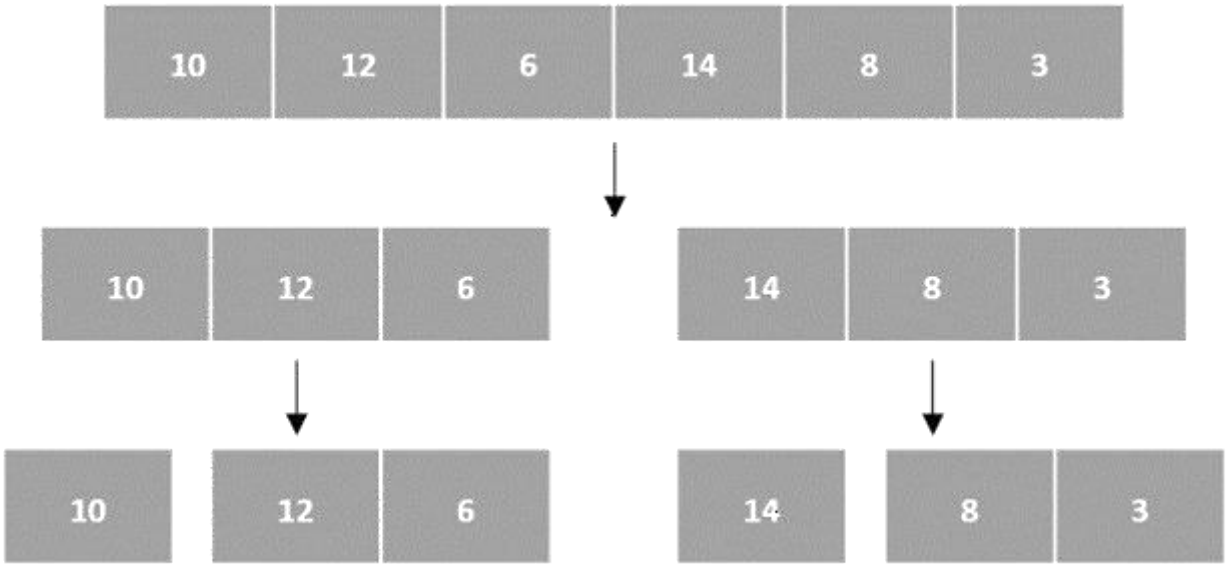
Unit - 2

1. Divide and Conquer Introduction.

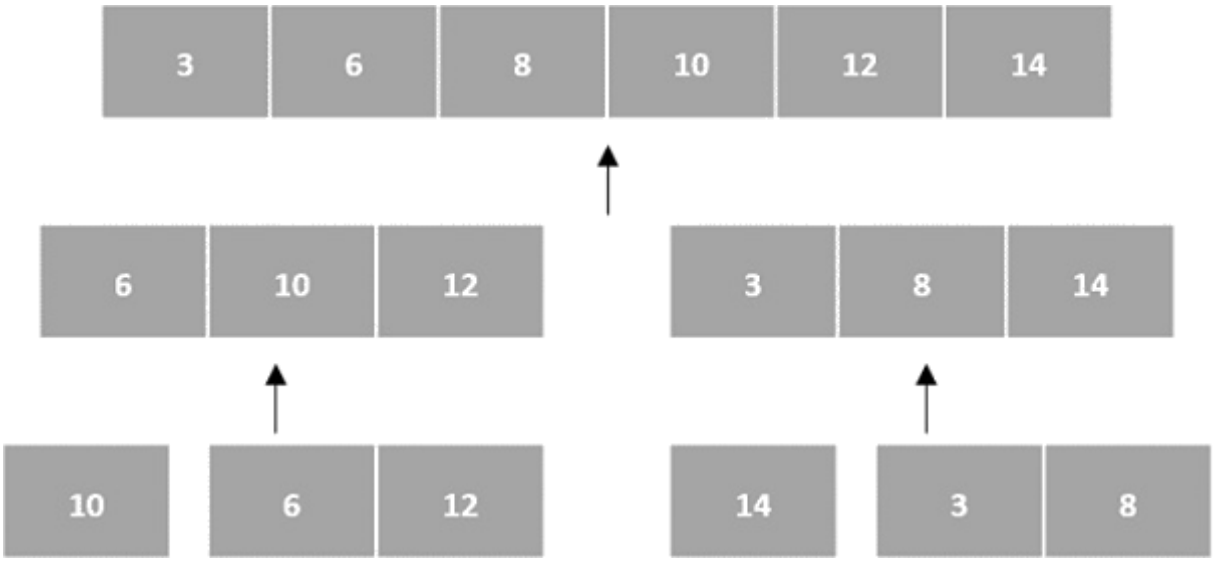


- **Divide and conquer approach in a three steps:**
  - **Divide/Break**
  - **Conquer/Solve**
  - **Merge/Combine**

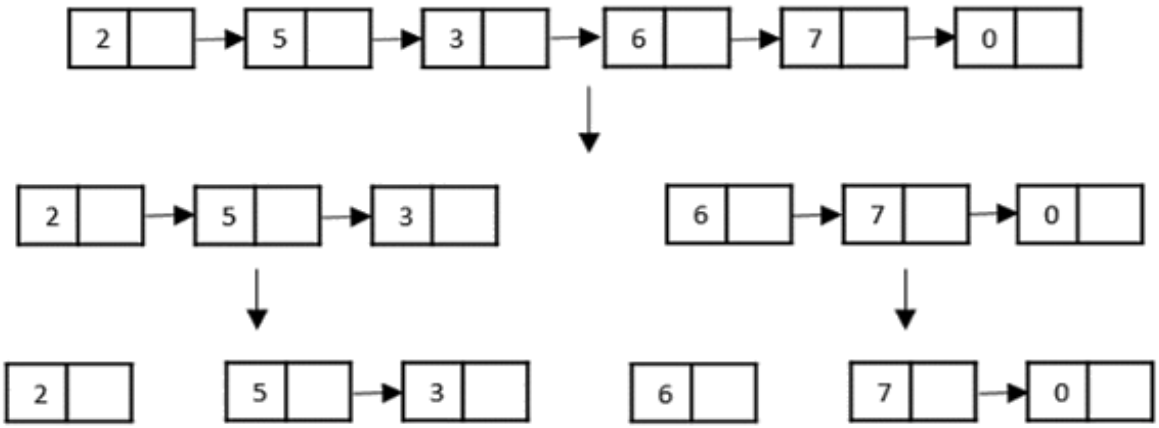
2. Arrays as Input:



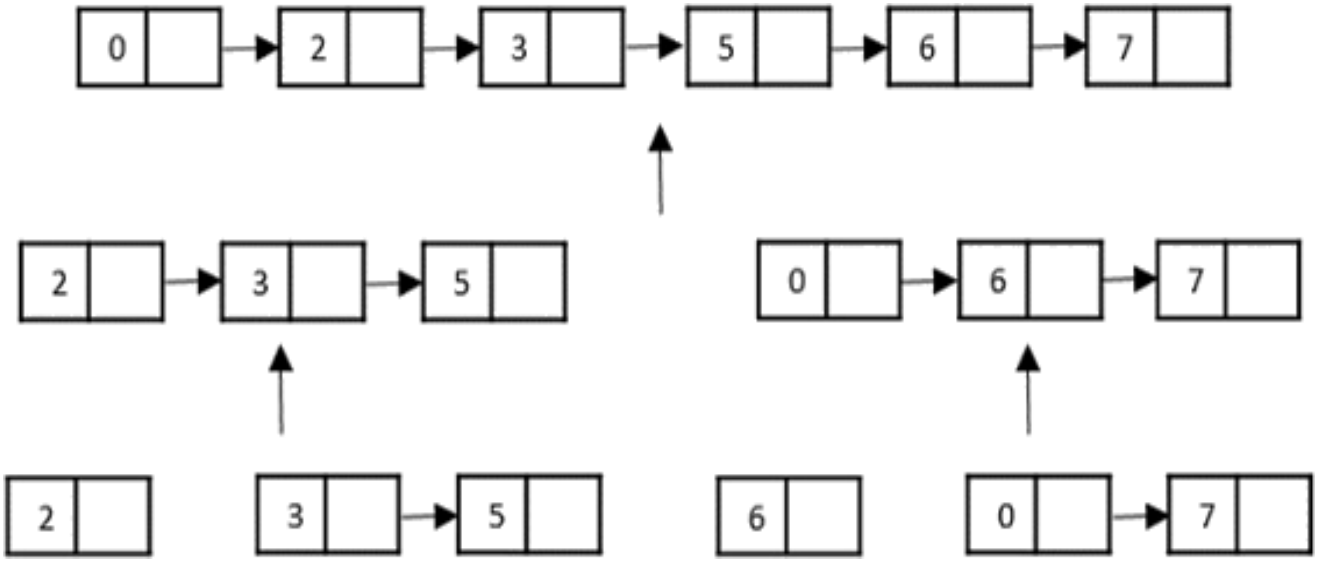
Then, the subproblems are sorted (the conquer step) and are merged to form thesolution of the original array back (the combine step).



3. Linked Lists as Input:



Then, the nodes in the list are sorted (conquered). These nodes are then combined (or merged) in recursively until the final solution is achieved



4. Examples of Divide and Conquer Approach.

- **Merge Sort**
- **Quick Sort**
- **Binary Search**
- **Strassen's Matrix Multiplication**
- **Closest pair (Points)**
- **Karatsuba**

5. Multiplying large Integers Problem.

Unit – 3

1. Dynamic Programming:

- **How does the dynamic programming approach work?**

  - It breaks down the complex problem into simpler subproblems.
  - It finds the optimal solution to these sub-problems.
  - It stores the results of sub-problems (memorization). The process of storing theresults of subproblems is known as memorization.
  - It reuses them so that same sub-problem is calculated more than once.
  - Finally, calculate the result of the complex problem.

- **Approaches of dynamic programming:**

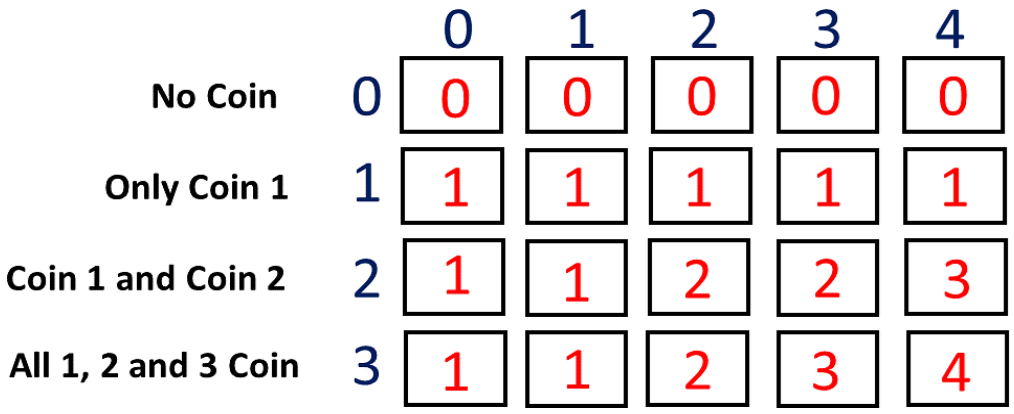
  - Top-down approach
  - Bottom-up approach

2. What is Binomial Coefficient?

- Naïve Approach for finding Binomial Coefficient:
- Optimized Approach for finding Binomial Coefficient:
- Java Code to find Binomial Coefficient:

3. Coin Change Problem:

- **Coin Change Problem Solution Using Dynamic Programming:**



4. Assembly Line Scheduling:

- Define the problem:
- Define the sub-problems:
- Define the recurrence relation:
- Solve the sub-problems:
- Trace the optimal path:

5. Knapsack Problem:

- There are two types of knapsack problems:
  - 0/1 knapsack problem
  - Fractional knapsack problem

How this problem can be solved by using the Dynamic programming approach?

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	2	2	2	2	2	2
2	0	0	0	2	3	3	3	5	5
3	0	0	0	2	3	3	3	5	5
4	0	0	0	2	3	3	4	5	5

6. Matric Chain Multiplication:

7. Longest Common Subsequence.

Unit – 4

1. Greedy Algorithm:

- Characteristic components of greedy algorithm:
  - The Feasible solution:
  - Optimal Solution:
  - Feasibility check:
  - Optimality check:
  - Optimal substructure property:

2. Activity Selection Problem:

Input Data for the Algorithm:

- act[] array containing all the activities.
- s[] array containing the starting time of all the activities.
- f[] array containing the finishing time of all the activities.

Output Data from the Algorithm:

- sol[] array referring to the solution set containing the maximum number of non- conflicting activities.

❖ Steps for Activity Selection Problem:

❖ **Step 1**: Sort the given activities in ascending order according to their finishing time. **Step 2**: Select the first activity from sorted array act[] and add it to sol[] array.

❖ **Step 3**: Repeat steps 4 and 5 for the remaining activities in act[].

❖ **Step 4**: If the start time of the currently selected activity is greater than or equal to the finish time of previously selected activity, then add it to the sol[] array.

❖

❖ **Step 5**: Select the next activity in act[] array.

❖

❖ **Step 6**: Print the sol[] array.

3. Elements of Greedy Strategy:

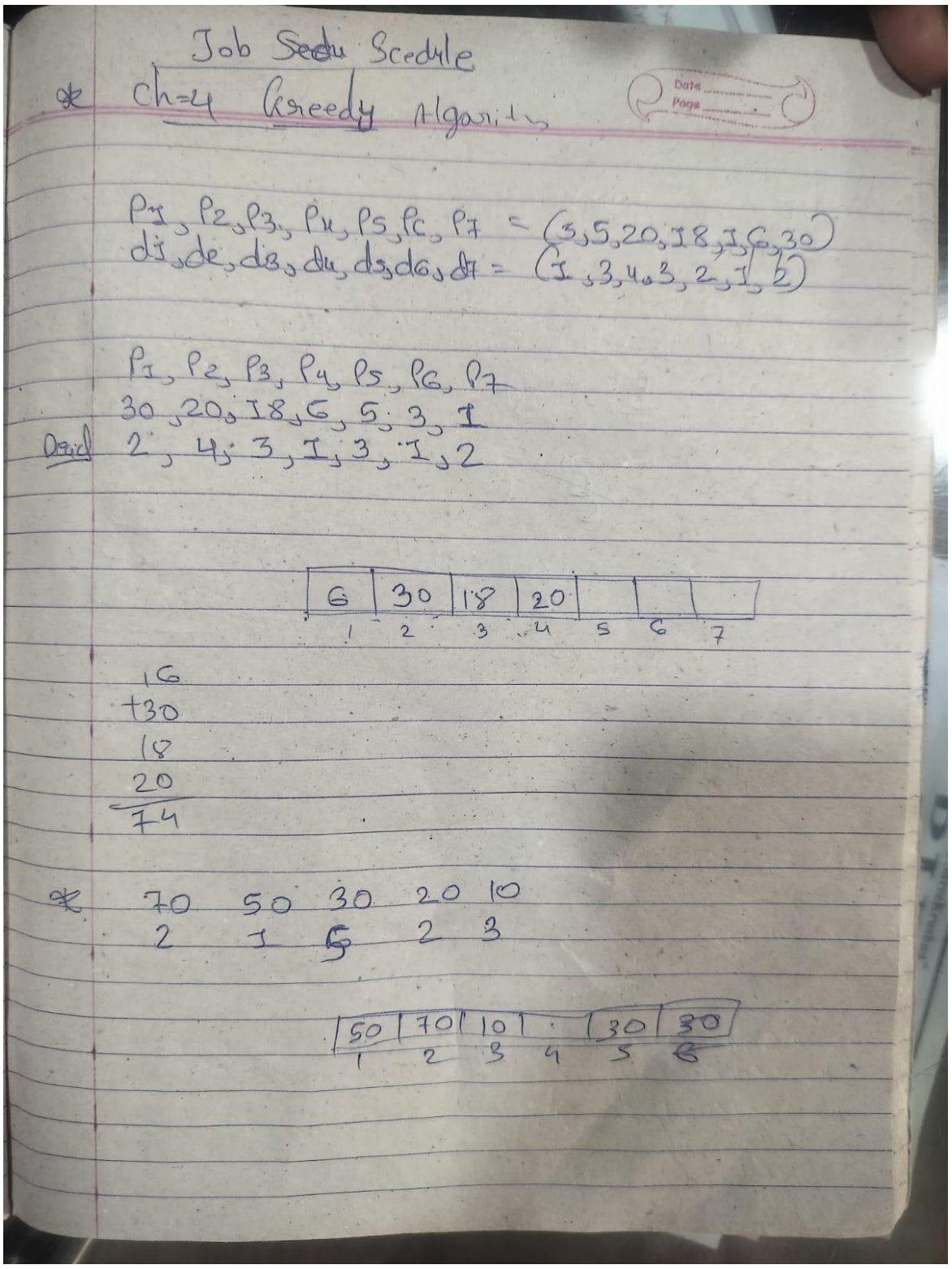
- Greedy Choice Property:
- Optimal Substructure:
- Greedy Algorithm:
- Selection of Locally Optimal Choices:
- Does Not Always Guarantee an Optimal Solution:
- No Backtracking:
- Efficiency:
- Sorting or Priority Queus:
- Proof of Correctness:

4. Kruskal's (Prim's) Algorithm (Minimun Spanning Trees):

- **What is Minimum Cost Spanning Tree?**

  - The minimum spanning tree is a spanning tree that has the smallest total edge weight. The Kruskal algorithm is an algorithm that takes the graph as input and finds the edges from the graph, which forms a tree that includes every vertex of a graph.

5. Job Scheduling using Greedy Algorithm:



Job Sedu Scedule

* Ch=4 Greedy Algorithm

Date
Page

$P_1, P_2, P_3, P_4, P_5, P_6, P_7 = (3, 5, 20, 18, 1, 6, 30)$
$d_1, d_2, d_3, d_4, d_5, d_6, d_7 = (1, 3, 4, 3, 2, 1, 2)$

$P_1, P_2, P_3, P_4, P_5, P_6, P_7$
30, 20, 18, 6, 5, 3, 1
Deid 2, 4, 3, 1, 3, 1, 2

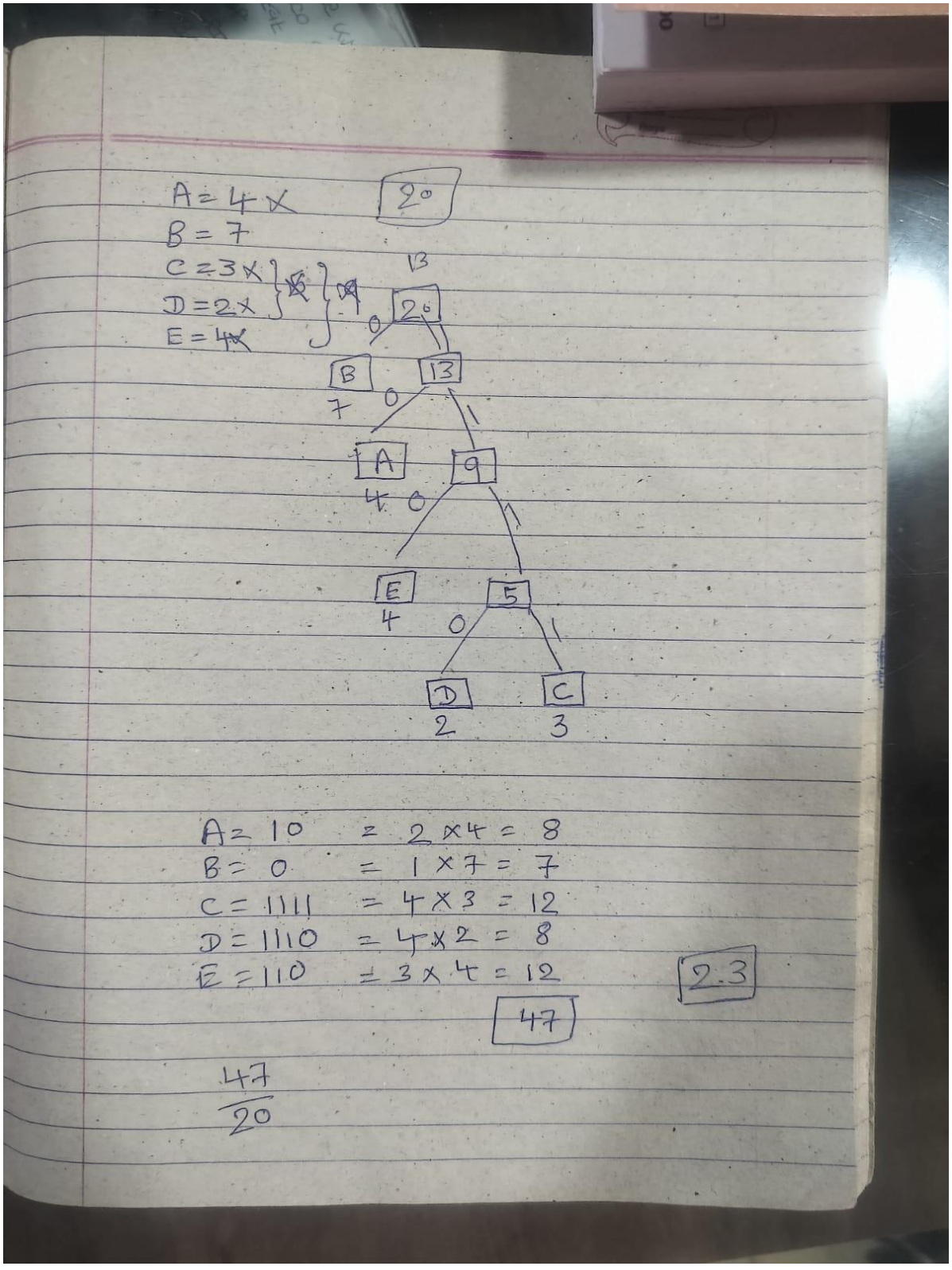
6	30	18	20			
1	2	3	4	5	6	7

16
+30
18
20
74

* 70 50 30 20 10
2 1 5 2 3

50	70	10		30	30
1	2	3	4	5	6

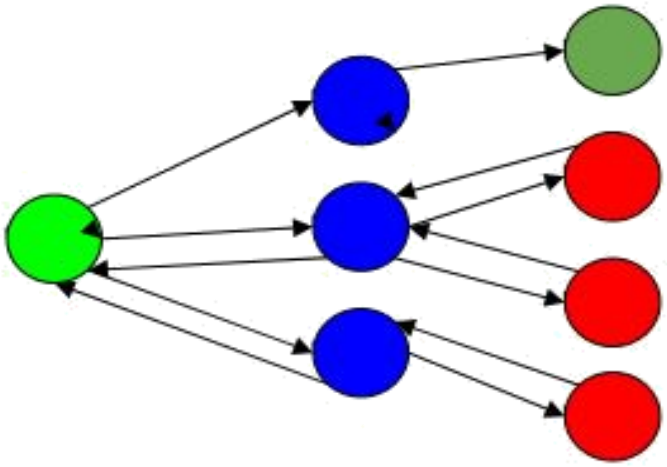
6. Huffman Coding:



Unit – 5

1. Introduction to Backtracking Algorithm:

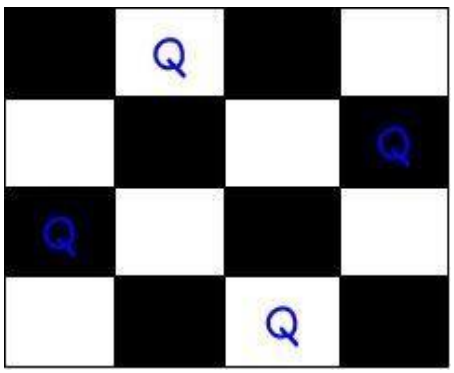
- **Backtracking** is a technique based on **algorithm** to solve problem. It uses recursivecalling to find the solution by building a solution step by step increasing values with time. It removes the solutions that doesn't give rise to the solution of the problem based on the constraints given to solve the problem.



2. The N – queen's problem:

- In N-Queen problem, we are given an NxN chessboard and we have to place nqueens on the board in such a way that no two queens attack each other.

- A queen will attack another queen if it is placed in horizontal, vertical



or diagonalpoints in its way.

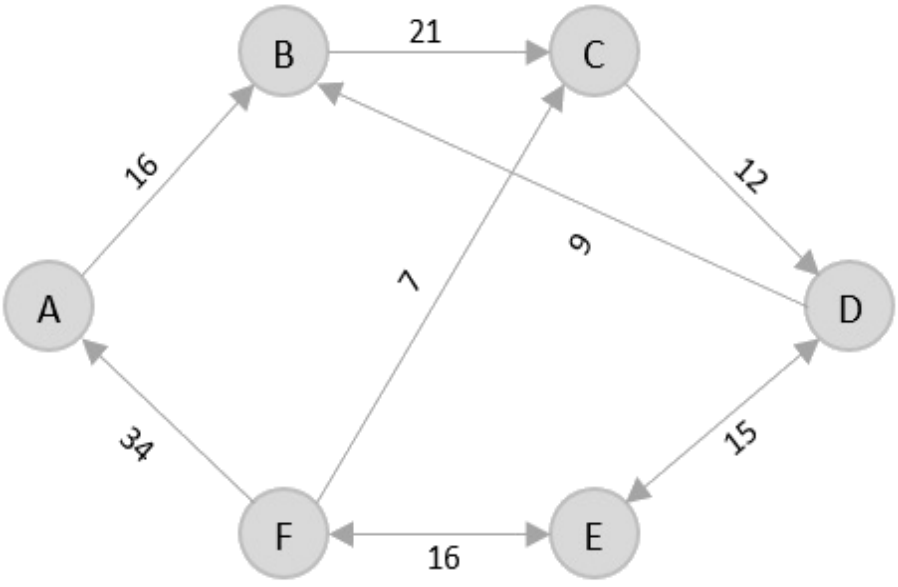
3. Travelling Salesman Problem:

➢ Travelling Salesperson Algorithm:

- As the definition for backtracking approach states, we need to find the best optimal solution locally to figure out the global optimal solution.
-
- The inputs taken by the algorithm are the graph G {V, E}, where V is the set of verticesand E is the set of edges. The shortest path of graph G starting from one vertex returning to the same vertex is obtained as the output.

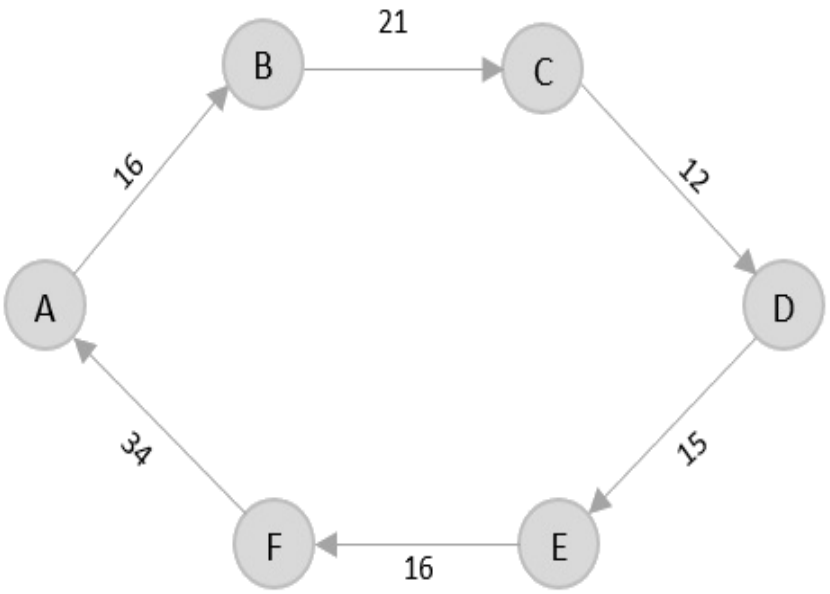
Examples:

➢ Consider the following graph with six cities and the distances between



them –

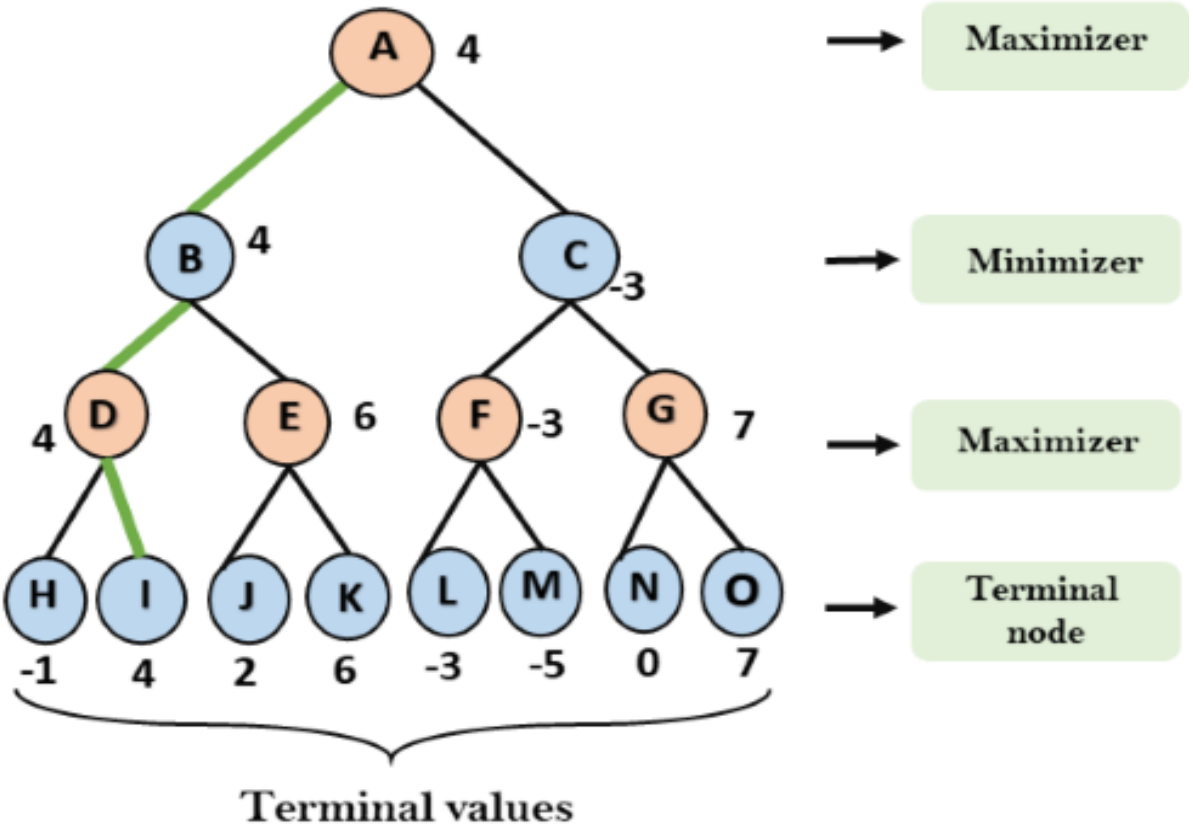
- Again, even though F → C has lower distance than F → A, F → A is added into theoutput graph in order to avoid the cycle that would form and C is already visited once.



- The shortest path that originates and ends at A is A → B → C → D → E → F → A

- The cost of the path is: 16 + 21 + 12 + 15 + 16 + 34 = **114**.

4. Mini-Max Algorithm in Games:


A
4
B
4
C
-3
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

Unit – 6

1. Introduction to Graphs:

- A **graph is an advanced data structure** that is used to organize items in an **interconnected network**. Each item in a graph is known as a **node**(or **vertex**) and these nodes are connected by **edges**.

❖ Types of Graphs:

A. Null Graph:
B. Cyclic Graph:
C. Acyclic Graph:
D. Weighted Graph:
E. Connected Graph:
F. Disconnected Graph:
G. Complete Graph:
H. Multigraph:

2. Dijkstra's Algorithm:

❖ **A Brief Introduction to Graphs:**

- **Components of a Graph:**
  - **Vertices:**
  - **Edges:**
- **Application of the Graph:**

❖ **Types of Graphs:**

A. **Undirected Graph**
B. **Directed Graph**

3. Breadth First Serach:

- **The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.**

❖ Two Categories:

I. Visited and
II. 0.

❖ **How does BFS work?**

- Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

- To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

4. Depth First Search:

- **Depth First Traversal (or DFS)** for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

❖ How does DFS work?

- Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

5. Topological Sort:

- The **topological sort** algorithm takes a directed graph and returns an array of the nodes where each node appears *before* all the nodes it points to.

6. Connected Components.

7. Traversing Graphs.

- **Breadth First Serach:**

- **The Breadth First Search (BFS) algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.**

❖ Two Categories:

III. Visited and
IV. 0.

❖ **How does BFS work?**

- Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

- To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

- **Depth First Search:**

- **Depth First Traversal (or DFS)** for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

❖ **How does DFS work?**

- Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.