Imperial College of Engineering, Khulna
Affiliated By Rajshahi University.

Name: Teresa Jency Bala
ID: 193852011 3

Part : II, Semester: Even.

Course Code: CSE 2221

Course Title : Design & Analysis of Algorithm

Topic : Algorithms, purpose of algorithms, backtracking, NP-completeness, Convex hull greedy Algorithm.

Submitted to,
Md. Rajib Hossen
Lecturer, ICE

Date of Submission: 25-01-22
Day of Submission: Tuesday.

1. Write down the algorithm of all sorts- Quick Sort, merge sort, insertion sort, heap sort, Radix sort,

Quick Sort:

Step 1: Start

Step 2: Take input for array list [1,...,n]

Step 3: Consider the first element of the list as pivot.

pivot = list [0]

Step 4: Define two variables left and right. Set left and right to first and last ~~em~~element of the array respectively

Step 5: Decrement right until pivot < list [right] & then stop

Step 6: Increment left until pivot > list [left] & then stop

Step 7: If left < right then swap {list [left], list [right]}

Step 8: Repeat step 5, 6, & 7 until left > right

Step 9: Exchange the pivot element with list [right] element

Step 10: Pass the partitioned elements in the quick sort function till each element gets separated.

Step 11: End.

Merge Sort:

1. Set array [1,...,n], l = 0, r = n, send values to the Merge Sort (array, l, r) function.

2. If r > l

a. Find middle point to divide the array into two halves:

middle, m = l + (r-l)/2

b. Call mergesort for first half:
call mergeSort (array, l, m)

c. Call merge sort for second half:
call mergeSort (array, m+1, r)

3. Merge the two halves sorted in b & c.
call merge(arr, l, m, r)

Insertion Sort:

Step-1: Start
Step-2: Initialize array a[1, ... n]
Step-3: Set first element as sorted portion & all remaining as unsorted portion.
Step-4: Select the first elements in unsorted portion and compare it will all elements in sorted sub-list. Shift all values in sorted sublist that are greater then the value to be sorted.
Step-5: Insert that value in sorted list.
Step-6: Repeat ~~un~~ step 4 & 5 untill all the elements from unsorted portion are moved into the sorted list.

#Step-7: End.

Heap Sort:

Step-1: Start
Step-2: Construct a binary tree with given list of elements.
Step-3: Transform the Binary tree into Max heap (for ascending order).

Step-4: Delete the root element from Max heap using heapify method.

Step-5: Put the deleted element into sorted list.

Step-6: Repeat the same steps 2 to 5 until Max heap becomes empty.

Step-7: Display the sorted list.

Step-8: End.

Radix Sort:

Step-1: Define 10 queue each representing bucket from each digit from 0 to 9.

Step-2: Consider the least significant digit of each number in the list which is to be sorted.

Step-3: Insert each number into their respective queue based on the least significant digit.

Step-4: Group all numbers from queue 0 to queue -9 in the order they have been inserted into their respective queue.

Step-5: Repeat step 3 based on the next least significant digit.

Step-6: Repeat from 2 until all the numbers are grouped based on most significant digit

Step-7: Display the sorted data.

2. What is backtracking?

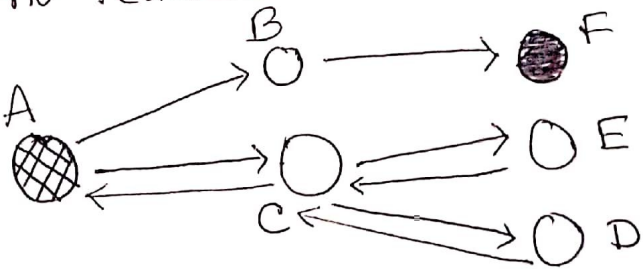
Answer: Backtracking is a technique based on algorithm to solve problem. It uses recursive calling to find the solution by building a solution step by step increasing values with time. It removes the solutions that doesn't give rise to the solution of problem based on the constaints given to solve the problem.

Backtracking algorithm is applied to some specific types of problem - ① Decision problem used to find a feasible solution of the problem.

② Optimisation problem used to find the best solution that can be applied.

③ Enumeration problem used to find the set of all feasible solution of problem.

In backtracking problem, the algorithm tries to find a sequence path to the solution which has some small checkpoints from where the problem can backtrack if no feasible solution is found for that problem.



Here,
we start with the checked box. ~~Quo~~ A. Our target is to go to F. The points D & E are not feasible solution Here, the algorithm propagates to an end to check if it is a solution or not, if it is then returns the solution otherwise backtracks to the point one step behind it to find track to the next point to find solution.

Algorithm:
Step 1: If current_position is goal, return success
Step 2: else
Step 3: if current_position is an end point, return

Some problems where backtracking is implemented:-
8N-queen, puzzle, Sudoku, crossword, graph coloring problems etc.

NP-Completeness:

P: P is a set of problems that can be solved by a deterministic Turing machine in Polynomial time.

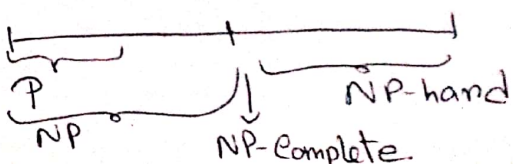
NP: NP is a set of decision problems that can be solved by a Non-deterministic Turing machine in Polynomial time.

P is a subset of NP.

A problem is called NP (non-deterministic polynomial) if its solution can be guessed and verified in polynomial time; Non-deterministic means that no particular rule is followed to make the guess.

NP-Complete: The problems that are present in both NP & NP-hard classes. If a problem is NP and all the NP problems can be reduced to that problem in polynomial time, than that problem is called NP-complete. Thus, finding an efficient algorithm for any NP-complete problem implies that an efficient algorithm can be found for all such problems, since any problem belonging to this class can be recast (modified) into any other member of the class.

NP-complete means most difficult NP Problems. If we can solve an NP-complete problem efficiently, we can solve any NP problems efficiently.



NP-Complete Problems: + Determining Hamiltonian cycle, checking satisfactory Boolean formula, traveling salesman.

NP-Hard Problems: A problem is said to be NP-Hard when an algorithm for solving NP Hard can translated to solve any NP-problem. Then we can say, this problem is at least as hard as any NP-problem, but it could be much harder or more complex.

countable

Polynomial time: The x time required to process an algorithm is polynomial time. An algorithm is said to be solvable in polynomial time if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some non-negative integer k, where n is the complexity of input. Polynomial-time algorithm are said to be fast.

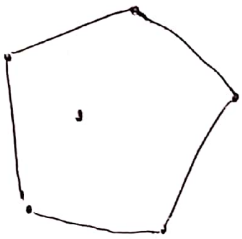
Polynomial Reduction (intuitively): A problem Q can be reduced to another problem Q' if any instance of Q can be "easily rephased" as an instance of Q', the solution to which provides a solution to the instance of Q. For example: the problem of solving linear equation in an indeterminate x reduces to the problem of solving quadratic equation. Given an instance $ax+b=0$ we transform it to $0x^2+ax+b=0$, whose solution provides a solution to $ax+b=0$. Thus, if a problem Q reduces to another problem Q', then Q is in a case, "no harder to solve" than Q'.

So, a language L_1 is a polynomial-time reducible to a language L_2, written $L_1 \leq L_2$, if there exists a polynomial-time computable function $f: \{0,1\}^* \to \{0,1\}^*$ such that for al $x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$. We call f the reduction function and a polynomial-time algorithm F that computes f is called a reduction algorithm.

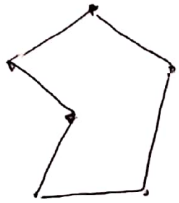
4. Convex hull & Graham's algorithm:

Convex hull: The convex hull of a set Q of points is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior. It is the smallest convex polygon containing a set of points on a grid.

* A convex hull of a set of points is defined as the smallest convex polygon, that encloses all of the points in the set. Convex means that the polygon has no corner that is bent inwards.



(a)
Aconvex polygon



(b)
A non-convex polygon

Graham's scan algorithm for finding convex hull with suitable example:

Graham's scan algorithm is a method of computing the convex hull of a finite set of points in the plane with time complexity O(nlogn). The algorithm finds all vertices of the convex hull ordered along its boundary.

The procedure in Graham's scan follows:

(1) Find the point with the lowest y coordinate. If there are two points with same y value, then the point with smaller x value is considered. Put the bottom-most point at first position.

(2) Consider the remaining n-1 points and sort them by polar angle in counterclockwise order around the points [0]. If polar angle ~~is~~ of two points is same, then put the nearest point first.

(3) Create an empty stack S and push points[0], points[1], and points[2] to S.

(4) Process remaining n-3 points one by one. Do the following for every point 'points[i]'

(a) Keep removing points from stack while orientation of following 3 points is not counter-clockwise (or they don't make a left turn).

(i) Point next to top in stack

(ii) Point at top of stack.

(iii) Points[i]

(b) Push points[i] to S.

Example:

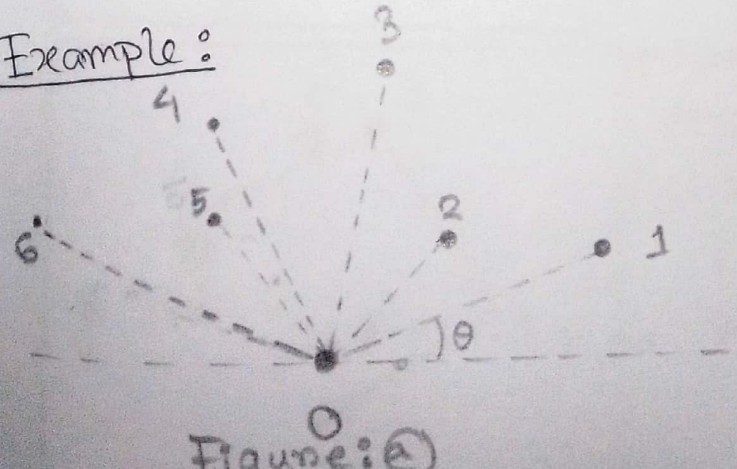
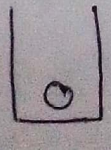


Figure: (a)

Stack:



Step-1: ~~So~~ We select the point with lowest y coordinate. Here, we select 0. Now, we sort the points by angle relative to the bottom most point and horizontal

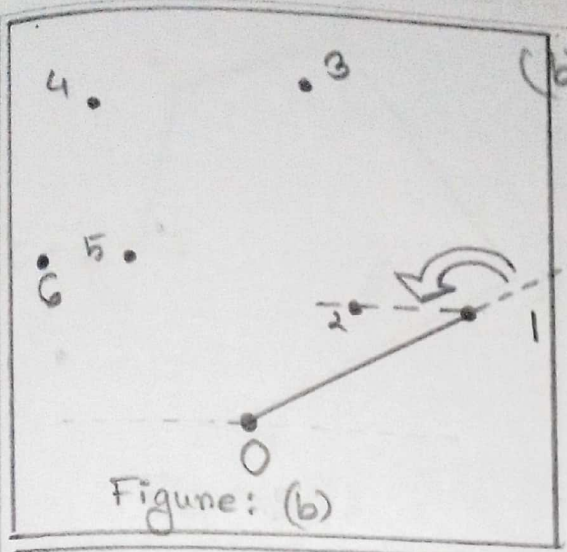


Figure: (b)

(b) Step-2: Iterate in sorted order, placing each point on the stack, but ~~by~~ only if it makes a counter-clockwise turn relative to the previous 2 points on the stack.

Stack: | 2 1 0 |

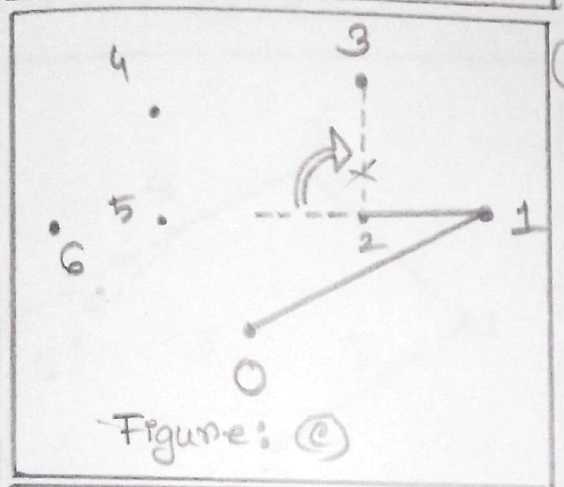


Figure: (c)

(c) Step-3: As we can see from point 2 to 3 we have to make a clockwise ~~to~~ move. We pop 2 from stack. And see if we can reach 3 from 1 ~~b~~ in counter clockwise direction.

Stack: | 1 0 |

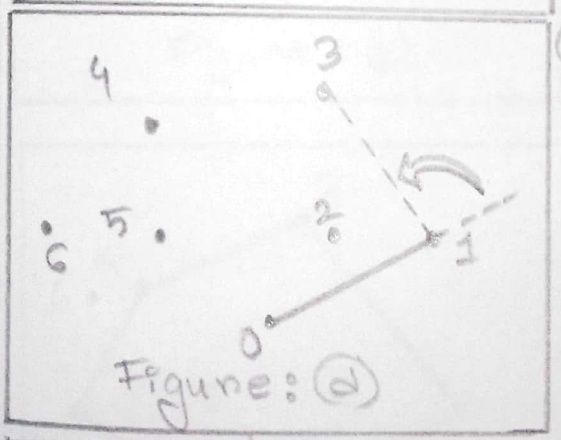


Figure: (d)

(d) Step-4: From 1 to 3 we have a counter clockwise rotation. So, ~~insert~~ push 3 onto the stack.

Stack: | 3 1 0 |

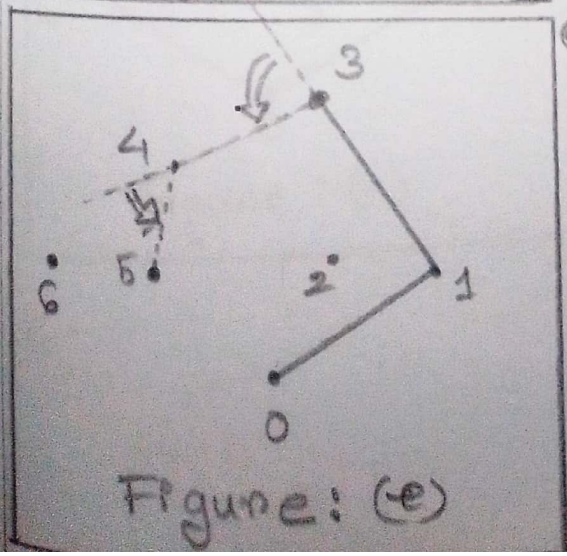


Figure: (e)

(e) Step-5: Insert 4 onto stack.

Step-6: Insert 5 onto stack as both gives counter clockwise notation

stack: | 5 4 3 1 0 |

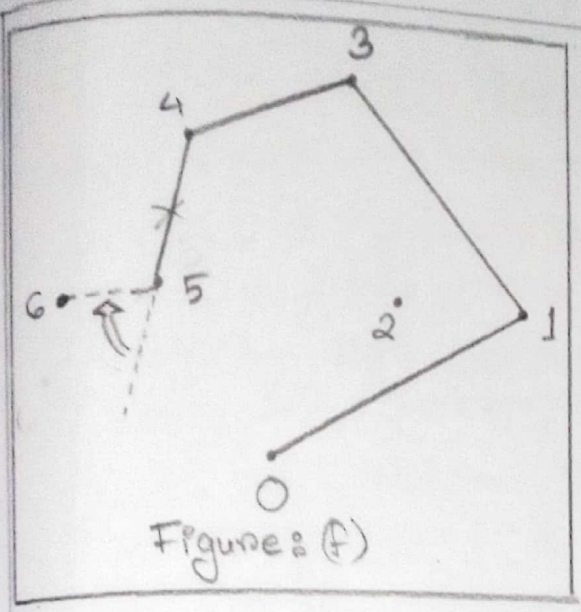


Figure: (f)

Step-7: As we can see From point 5 to 6 we make a clockwise rotation we have to remove 5 from stack. & Check if we can go to 6 from 4 in counter-clockwise rotation.

Stack:

4
3
1
0

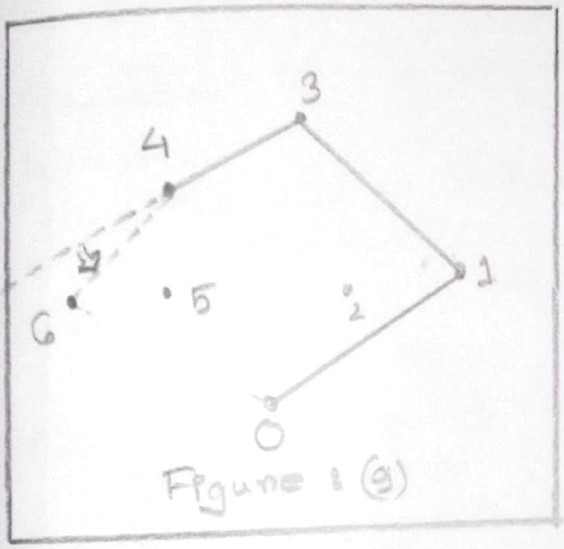


Figure: (g)

Step-8: From 4 to 6 we can go counter clockwise

Stack:

6
4
3
1
0

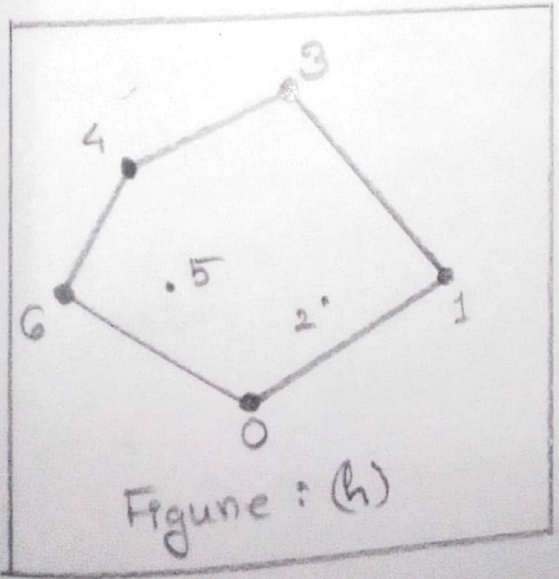


Figure: (h)

Step-9: From 6 we go to 0 & close the polygon.

stack: 0 1 3 4 6

5. Binary Search Tree: BST is a node-based binary tree data structure which has the following properties:

(a) The left subtree of a node contains only keys lesser than the node's key.

(b) The right subtree of a node contains only nodes with keys greater than the node's key

(c) The left & right subtree each must also be a binary search tree.

In short: leftsub-tree (keys) < parent (key) ≤ right sub-tree (keys)

Operations that can be performed on BST includes:

- Insertion - add an item and its key to the BST
- Search - look up an item in BST by it's key
- Remove - Delete an item/key from the BST by its key.

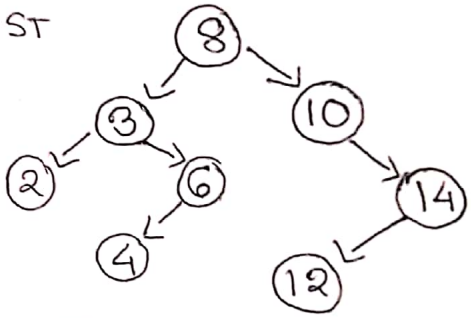


Figure: BST

Applications of BST:

1. BST is used for indexing and multi-level indexing
2. They are also helpful to implement various searching algorithm
3. It is helpful in maintaining a sorted stream of data

Purpose of BFS:

1. To find the shortest path and Minimum Spanning Tree for unweighted graph.
2. BFS is used to find all neighbour nodes in peer to peer network.

3. Breadth first search is using in copying garbage collection using Cheney's algorithm
4. To detect a cycle in unweighted graph
5. Finding all nodes within one connected component.

Purposes of DFS:

1. Detecting cycle in graph: A graph has cycle if and only if we see a back edge during DFS. We can run DFS for the graph and check for back edges.
2. Path Finding: We can specialize DFS algorithm to find a path between two given vertices u and z.
3. Topological Sorting: It is used for scheduling jobs from the given dependencies among jobs.
4. To test if graph is bipartite.
5. Finding Strongly Connected Components of a graph.
6. Solving puzzles with only one solution, such as mazes.

Purposes of Bellman Ford:

1. Helps to find the shortest path from a vertex to all other vertices of a weighted graph.
2. It works with graphs in which edges can have negative weights.

Real-life purpose: In google map the Bellman Ford for finding shortest part between two locations.
* Considering map as graph.
* locations in map as vertices.
* Roads & Road lengths as the edges & weights

*Network Routing, Robot Navigation, Uber traffic planning
Telemarketer operator scheduling.

Purposes of Dijkstra's Algorithm: (single source shortest path)

1. Used in finding shortest path.
2. Used in geographical map
3. Can solve shortest path for any non-negative weighted graph.
4. Can handle graphs consisting of cycles.
5. Used in routing protocol. This algorithm provides the shortest cost path from the source router to another routers in network.
6. Implemented in conducting of data in networking & telecommunication domains for decreasing the obstacle taken place for transmission.

Purposes of Floyd-Warshall Algorithm: (All pair shortest Path)

1. It computes the shortest paths between every pair of nodes.
2. As all nodes can be taken as source so the smallest path from every vertex can be determine.
3. Can apply dynamic programming approach
4. To find shorted path in a directed graph.
5. To find transitive closure of directed graph.
6. To find inversion of real matrices.
7. For testing whether an undired graph is bipartile.

7. What is greedy Algorithm?

Definition: The algorithm that considers the choice that looks best at the moment. It makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. This means that the algorithm picks the best solution at the moment without regard for consequence. It picks the best immediate output, but does not consider the over all possibilites, hence it is called greedy Algorithm

The problems that can be solved using greedy algorithm are -

1. Huffman encoding which is used to compress data.
2. Dijkstra's Algorithm, which is used to find the shortest path through a graph.
3. Graph Coloring is a way of coloring graph vertices such that no two adjacent vertices share the same colon. The greedy algorithm can be applied.
4. Sequencing job problem by scheduling.
5. Round-robin scheduling: In this algorithm, we assign a fixed time slot to each process. If the process doesn't execute completely in that time slot, it will complete its remaining execution in next round.
6. Shortest job first. In this algorithm, we give the highest priority to process with the lowest burst time.

Optimization Problem: The problem that requires either minimum or maximum result.

Advantages of Greedy Algorithm:

1. It is highly optimized and one of ~~me~~ the most straightforward algorithms
2. This algorithm takes lesser time as compared to others because the best solution is immediately reachable.
3. In the greedy method, multiple activities can execute in a given time frame.
4. We don't need to combine the solutions of sub-problem as it automatically reaches the optimal solution.
5. The algorithm is easy to implement

Types of methods which can be applied for optimization problem : (1) Greedy Method, (2) Dynamic Programming (3) Branch and Bound.

Greedy method Algorithm:

n=5

a	a_1	a_2	a_3	a_4	a_5
	1	2	3	4	5

```
Algorithm Greedy (a,n){
    for i=1 to n do {
        x = Select (a);
        if feasible (x) then
            solution = solution + x;
    }
}
```

Greedy ~~method~~ approach has it's own known method of selection and based on that selection method the ~~result~~ result obtained is considered best result

For example, Consider a person wants to buy a car. One thing he can do is go through all brands and all models available in world & buy the best one basing on the features.

But that will be time consuming. just ~~on~~ basing on So, the features. So, another approach could be firstly select the top brand. Then selecting the latest model or any well-known car according to that person. The picking that can that suits the person's interest. The approach adapted here is greedy method approach.

An activity ~~-selection Problem~~:

When to Use Greedy method: The problems with following two properties can be solved -

(1) Greedy choice property: A global optimum can be arrived at by selecting a local optimum

(2) Optimal Substructure: An optimal solution to the problem contains an optimal solution to subproblems

Application

1. Activity selection Problem
2. Huffman coding
3. Job sequencing problem
4. Fractional knapsack problem
5. Prim's Minimum spanning tree

8. Differences between Divide-Conquer & Dynamic programming

Divide & Conquer Method		Dynamic programming.
It ~~deals~~ involves 3 steps at each level of recursion: a) Divide the problem into a number of subproblems b) Conquer the subproblem by solving them recursively. c) Combine the solution of the subproblems into solution for original sub-problems	1. Steps	It involves the sequence of 4 steps: a) Characterize the structure of optimal solution. b) Recursively defines the value of optimal solution. c) Compute the value of optimal solutions in a Bottom-up minimum d) Construct an Optimal Solution from computed information.
It is Recursive	2. Type	It is non-recursive.
It does more work on subproblems and hence has more time consumption	3. Time Consumption	It solves ~~suppt~~ subproblems only once and then stores in the table. Requires less time.
It is a top-down approach	4. Approach type	It is a Bottom-up approach
In this subproblems are independent of each other	5. Dependence	In this subproblems are interdependent.
Merge Sort, Binary search	6. Example	Matrix-chain multiplication
Combines solutions of subproblems to obtain solution of main problem	7. Theme	Uses result of subproblems to find optimum solution of main problem.

9. Dynamic Programming Vs greedy method.

Greedy Method		Dynamic Programming.
In a greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global optimal solution	1. Feasibility	In Dynamic Programming. we make decision at each step considering current problem and solution to previously solved subproblem to calculate optimal solution.
Sometimes there is no such guarantee of getting Optimal Solution.	2. Optimality	It is guaranteed that Dynamic programming will generate an optimal solution as it generally considers all possible cases and then choose the best
Follows the problem solving heuristic (well-defined guess) (অনুসন্ধানমূলক) of making locally optimal choice at each stage	3. Recursion	An algorithmic technique which is usually based on recurrent formula that uses some previously calculated states
More efficient in terms of memory as it never looks back or revise previous choices	4 Memoiza-tion	It requires dynamic programming table for memoization & it increases it's memory complexity.
Less efficien	5. Efficiency	More efficient
Top-down	6 Approach	Bottom-up
Less reliable.	7. Reliability	Highly Reliable.
Generates single decision sequence	8. Basic	Many decision sequence may be generated.
Fractional Knapsack	9 Example	0/1 knapsack
kruskal.		

10. What are the applications & Algorithms of BFS, DFS, Prim's, Kruskal's algorithm.

BFS

Algorithm

Step 1: Set start

Step 2: Take inputs for graph G

Step 3: Set STATUS = 1 (ready state) for each node in G.

Step 4: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 5: Repeat step 6 & 7 until queue is empty.

Step 6: Dequeue a node N. Process it and set its STATUS = 3 (processed state)

Step 7: Enqueue all neighbours of N that are in ready state (whose STATUS = 1) and set their STATUS = 2. (waiting state)

[End of LOOP]

step 8: Exit.

Applications:

1. Determine shortest Path and Minimum Spanning Tree for unweighted gram.
2. In Peer to Peer Networks like BitTorrent, BFS is used to find all neighbour nodes.
3. Crawlers in search engines uses BFS for building index.
4. In social network sites, we find people within a given distanke using BFS.
5. In GPS Navigation system BFS is used to find all neighboring location
6. In networks, a broadcasted packet follows BFS to reach all nodes.
7. To determine cycle in Undirected graph.
8. To test if graph is Bipartile.
9. Find paths between two vertices.

DFS

Algorithm:

step 1: Start

step 2: Take inputs of Graph G.

Step 3: SET STATUS=1 (ready state) for each node in G.

Step 4: Push the starting node A on stack and set its STATUS=2 (waiting state)

Step 5: Repeat steps 6 & 7 until STACK is empty.

step 6: Pop the top node N. Process it and set it's STATUS = 3 (processed state).

step 7: Push on the stack all the Neighbours of N that are in ready state (whose STATUS=1) and set their Status = 2 (waiting state).

[end of loop].

Step 8: Exit

Applications:

1. To detect cycle in graph. A graph has cycle if and only if we see a back edge during DFS.
2. We can specialize DFS algorithm to find a path between two given vertices.
3. Topological sorting is used for scheduling jobs from the give dependencies among jobs. We can apply DFS for instruction scheduling, logic synthesis, determining order of compilation tasks, etc.
4. To test a graph is Bipartite. We can discover new vertex & color it opposited to its parent, and for each other edge, it doesn't link two vertices of same color.
5. Finding Strongly Connected components of a graph. A directed graph is strongly connected if there is a path from each vertex in the graph to every other vertex.
6. Solving puzzles with only one solution, such as mazes. (DFS can be adapted to find all solutions to a maze by only including nodes on the current path in visited set).

Prim's Algorithm

Step 1: Select a ~~starting~~ starting vertex.

Step 2: Repeat Step 3 & 4 until there are ~~border/edge~~ fringe vertices.

Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight.

Step 4: Add the selected edge and the vertex to the minimum spanning tree T.

[exit of loop]

Step 5: EXIT.

Applications:

1. Laying cables of electrical wiring (By minimizing total length of wire connecting customers)
2. In network design
3. To make protocols in ~~a~~ network cycles.
4. To connect several cities using highways or rail networks, ~~you~~ using this algo to find minimum length of roads/railtracks connecting all these cities.
5. Can be used to find travelling salesman problem
6. Design local area networks.

Kruskal's Algorithm

Step 1: Sort all edges from low weight to high weight.

Step 2: Take the edge with lowest weight and use it to connect the vertices of graph

Step 3: If adding an edge creates a cycle, then reject that ~~reject th~~ edge and go for next least weight edge.

Step 4: Keep adding edges by doing step 2 & 3 till all the vertices are in the minimum spanning tree with no. of edges $(v - 1)$.

Kruskal's Algorithm Application:

1. In order to layout electrical wiring.
2. In computer network (LAN connection.
3. Creating single-link cluster.
4. Handwriting recognition of mathematical expression
5. Circuit design: Implementing efficient multiple constant multiplication, as used in finite impulse response filters.
6. Creating network of pipes for drinking water or natural gas.

Prim's Algorithm		Kruskal's Algorithm.
Starts to build the Minimum Spanning tree from any vertex in the graph	1. Theme.	It starts to build the Minimum spaning tree from the vertex carring minimum weighted edge in the graph.
It traverses one node more than once to get minimum distance	2. Node traversal	It traverses one node only once.
Generally, $O(V^2)$, V is no. of vertices, Can be improved to $O(E \log V)$ using fibonacci heaps.	3. Time complexity	Kruskal's algorithm time complexity is $O(e \log v)$. V being the.
Uses ~~Lin~~ List data structure	4. Data-structure type.	Uses heap data structure
Prim's Algorithm runs faster in dense graph	5. Strong Graph	Kruskals algorithm runs faster in sparse graph.