Implement Greedy search algorithm for any of the following application:

I. Selection Sort

II. Minimum Spanning Tree

III. Single-Source Shortest Path Problem

IV. Job Scheduling Problem

V. Prim's Minimal Spanning Tree Algorithm

VI. Kruskal's Minimal Spanning Tree Algorithm

VII. Dijkstra's Minimal Spanning Tree Algorithm

**I. Selection Sort using Greedy Algorithm – Theory**

**🔷 Problem Statement:**

**Sort an array of n elements in ascending order using the Greedy method.**

**🔷 Why Greedy?**

**Selection Sort uses a greedy approach because at each step it selects the minimum element from the unsorted part and places it at the correct position in the sorted part. It makes the best choice at each step, assuming it leads to the overall best result.**

**📌 Key Idea:**

**"Find the smallest element from the unsorted part and swap it with the element at the current position."**

**🔢 Formula / Concept:**

**At each iteration i, we:**

* **Find min(A[i...n-1])**
* **Swap A[i] with the found minimum**

**🧠 Algorithm: Selection Sort (Greedy Approach)**

**Step-by-step:**

1. **Start from index i = 0 to n-1**
2. **Find the minimum element in sub-array A[i...n-1]**
3. **Swap it with A[i]**
4. **Repeat until the array is sorted**

**✏️ Pseudocode:**

**SelectionSort(A[1..n]):**

**for i = 0 to n-2:**

**min\_index = i**

**for j = i+1 to n-1:**

**if A[j] < A[min\_index]:**

**min\_index = j**

**Swap A[i] and A[min\_index]**

**⏱️ Time Complexity:**

* **Best, Worst, Average: O(n²)**
* **Space Complexity: O(1) (in-place)**

**Minimum Spanning Tree (MST) – Using Greedy Algorithm**

**🔷 What is a Minimum Spanning Tree?**

* **A Spanning Tree is a subgraph of an undirected, connected, weighted graph that:**
  + **Connects all the vertices**
  + **Has no cycles**
  + **Has exactly V-1 edges (where V = number of vertices)**
* **A Minimum Spanning Tree is the spanning tree with the smallest total edge weight.**

**🔷 Why Greedy Algorithm?**

**MST is solved using Greedy Algorithms because:**

* **At each step, we make the locally optimal choice (minimum weight edge).**
* **If the graph has the greedy-choice property and optimal substructure, the greedy algorithm gives the global optimal solution.**

**✅ Greedy Algorithms for MST:**

**Two main greedy algorithms are used:**

**🟩 1. Prim’s Algorithm – (Node-Based Greedy Approach)**

**🔷 Algorithm Steps:**

1. **Start with any vertex.**
2. **Mark it as visited.**
3. **Among all edges from visited nodes to unvisited ones, choose the minimum weight edge.**
4. **Add the corresponding node to the MST.**
5. **Repeat until all vertices are included.**

**📌 Key Formula:**

**Choose edge (u,v) such that w(u,v) is minimum and v∉MST\text{Choose edge } (u, v) \text{ such that } w(u, v) \text{ is minimum and } v \notin \text{MST}Choose edge (u,v) such that w(u,v) is minimum and v∈/MST**

**⏱️ Time Complexity:**

* **Using Priority Queue: O(E log V)**

**🟦 2. Kruskal’s Algorithm – (Edge-Based Greedy Approach)**

**🔷 Algorithm Steps:**

1. **Sort all edges in increasing order of weight.**
2. **Initialize MST as empty.**
3. **For each edge, check if it forms a cycle using Union-Find.**
   * **If it doesn’t, include it in MST.**
4. **Repeat until MST has V - 1 edges.**

**📌 Key Formula:**

**Include edge (u,v) if it connects disjoint sets (no cycle)\text{Include edge } (u, v) \text{ if it connects disjoint sets (no cycle)}Include edge (u,v) if it connects disjoint sets (no cycle)**

**⏱️ Time Complexity:**

* **Sorting edges: O(E log E)**

**🧠 Example Use Case:**

* **Designing efficient electrical wiring or road networks**
* **Reducing cost in network layout problems**

**Single-Source Shortest Path (SSSP) – Using Greedy Algorithm**

**🔷 What is the Single-Source Shortest Path Problem?**

**Given a weighted graph and a source vertex, the goal is to find the shortest path (minimum total weight) from the source to every other vertex in the graph.**

**🔷 Why Greedy Algorithm?**

**Dijkstra’s Algorithm uses a greedy strategy:**

* **It always picks the unvisited node with the smallest known distance.**
* **It updates the distances to its neighboring vertices.**

**This greedy choice is locally optimal and leads to a globally optimal solution, if there are no negative weights.**

**🔢 Distance Formula:**

**dist(v)=min(dist(v),dist(u)+w(u,v))**

**Where:**

* **dist(v) is the shortest distance to vertex v**
* **u is a vertex already processed**
* **w(u, v) is the weight of the edge between u and v**

**✅ Dijkstra’s Algorithm – Steps**

1. **Initialize dist[source] = 0, and all others as ∞.**
2. **Create a priority queue or min-heap to store vertices with their distance.**
3. **While the queue is not empty:**
   * **Remove the vertex u with the smallest distance.**
4. **Repeat until all vertices are processed.**

**⏱️ Time Complexity:**

* **Using min-heap: O((V + E) log V)**
* **Where V = number of vertices, E = number of edges**

**🧠 Conditions:**

* **Works only with non-negative weights.**
* **Fails with negative-weight edges (use Bellman-Ford in such cases).**

**📌 Example Use Case:**

* **GPS Navigation (shortest path from one location to all others)**
* **Network routing**
* **Game AI pathfinding**

**Job Scheduling Problem – Using Greedy Algorithm**

**🔷 What is the Job Scheduling Problem?**

**You are given a set of n jobs, where each job has:**

* **A deadline**
* **A profit if the job is completed before the deadline**
* **Each job takes 1 unit of time**

**The goal is to schedule jobs such that the total profit is maximized, and no two jobs overlap.**

**🔷 Why Greedy?**

**The greedy approach is used because:**

* **We always want to choose the job with the highest profit first.**
* **We try to schedule it as late as possible before its deadline to leave room for other jobs.**

**✅ Greedy Algorithm – Steps**

**Input:**

* **n jobs with id, deadline, and profit**

**Output:**

* **Sequence of jobs that maximizes profit**

**Algorithm:**

1. **Sort all jobs in descending order of profit.**
2. **Initialize an array of time slots of size equal to the maximum deadline.**
3. **For each job:**
   * **Find a free slot from min(deadline, max\_slot) to 1.**
   * **If a free slot is found, assign the job to that slot.**
4. **Return the scheduled job sequence and total profit.**

**⏱️ Time Complexity:**

* **Sorting: O(n log n)**
* **Slot checking: O(n)**
* **Total: O(n²) or O(n log n) with efficient slot checking**

**Prim’s Minimal Spanning Tree Algorithm – Using Greedy Algorithm**

**🔷 What is a Minimal Spanning Tree (MST)?**

**A Minimal Spanning Tree (MST) of a connected, undirected, weighted graph is a tree that:**

* **Contains all the vertices of the graph.**
* **Has exactly V - 1 edges (where V is the number of vertices).**
* **The sum of the edge weights is minimized (the total weight is the smallest possible).**

**🔷 Why Greedy Algorithm?**

**Prim's Algorithm is based on a greedy approach because it grows the MST one edge at a time, always selecting the minimum weight edge that connects a vertex in the MST to a vertex outside the MST.**

**🧠 Prim's Algorithm Steps:**

1. **Start with any vertex and mark it as visited.**
2. **Initialize a priority queue or min-heap to store edges connected to the MST.**
3. **For the current vertex u, examine its neighbors and add edges (u, v) to the priority queue with their weights.**
4. **Repeatedly choose the minimum weight edge that connects a visited vertex to an unvisited vertex.**
5. **Add the selected edge to the MST and mark the new vertex as visited.**
6. **Continue until all vertices are included in the MST.**

**⏱️ Time Complexity:**

* **Using a priority queue (min-heap), the time complexity is:**
  + **O((V + E) log V), where:**
    - **V is the number of vertices,**
    - **E is the number of edges.**

**Kruskal’s Minimal Spanning Tree Algorithm – Using Greedy Algorithm**

**🔷 What is a Minimal Spanning Tree (MST)?**

**A Minimal Spanning Tree (MST) of a connected, undirected, weighted graph is a tree that:**

* **Contains all the vertices of the graph.**
* **Has exactly V - 1 edges (where V is the number of vertices).**
* **The sum of the edge weights is minimized (the total weight is the smallest possible).**

**🔷 Why Greedy Algorithm?**

**Kruskal's Algorithm is based on a greedy approach because it:**

* **Selects the minimum weight edge in each step.**
* **Always picks the edge that connects two different components of the MST (ensuring no cycles are formed).**

**By doing this repeatedly, the algorithm guarantees that the resulting tree is the minimum spanning tree.**

**🧠 Kruskal’s Algorithm Steps:**

1. **Sort all the edges in increasing order of weight.**
2. **Initialize an empty set for the MST.**
3. **For each edge (u, v) in sorted order:**
   * **If u and v belong to different sets (using Union-Find data structure to track connected components):**
     + **Add edge (u, v) to the MST.**
     + **Merge the two sets containing u and v.**
4. **Stop when the MST has V - 1 edges.**

**⏱️ Time Complexity:**

* **Sorting edges takes O(E log E).**
* **Union-Find operations (with path compression and union by rank) take O(E α(V)), where α is the inverse Ackermann function (almost constant for practical purposes).**
* **Total time complexity: O(E log E).**

**Dijkstra’s Algorithm for Single-Source Shortest Path (SSSP)**

**Dijkstra's algorithm is focused on finding the shortest paths from a single starting vertex to all other vertices in a weighted graph. It works as a greedy algorithm, picking the nearest unvisited vertex, updating distances to its neighboring vertices, and repeating this process until all vertices are visited.**

**🔷 What is the Single-Source Shortest Path Problem?**

**The Single-Source Shortest Path problem involves finding the shortest path from a given source vertex to all other vertices in a graph. Dijkstra’s algorithm uses a greedy strategy to solve this by repeatedly selecting the vertex with the minimum known distance from the source and updating the distances of its neighboring vertices.**

**🔷 Why Not MST?**

* **Prim’s and Kruskal’s algorithms are used for Minimal Spanning Tree problems, where we aim to connect all vertices in the graph with the least total edge weight.**
* **Dijkstra’s algorithm is used for shortest path problems, not for spanning trees. It focuses on finding the shortest path from a given source to all other nodes rather than connecting all nodes directly in a spanning tree.**

**🔷 Dijkstra’s Algorithm – Steps for Single-Source Shortest Path**

1. **Initialization:**
   * **Set the distance to the source vertex as 0 and all other vertex distances to infinity (∞).**
   * **Create a priority queue (min-heap) to track vertices by their current shortest distance.**
2. **Process Each Vertex:**
   * **Extract the vertex u with the smallest distance from the priority queue.**
   * **For each neighboring vertex v of u, check if the distance through u offers a shorter path than the current known distance for v. If so, update the distance for v.**
3. **Repeat:**
   * **Continue extracting the vertex with the smallest distance from the priority queue and updating the distances until all vertices have been processed.**
4. **Result:**
   * **The distances to all vertices from the source will be minimized.**

**⏱️ Time Complexity:**

* **Using a min-heap for the priority queue:**
  + **O((V + E) log V), where:**
    - **V is the number of vertices.**
    - **E is the number of edges.**
* **E operations (for relaxing edges) and V operations (for extracting the minimum distance vertex) lead to the complexity.**