Greedy Algorithm:

The greedy algorithm is a problem solving technique that builds a solution incrementally by selecting the best available option at each step, ensuring that the choice is best.

It aims to find a solution that is good enough and in some cases optimal by choosing the locally optimal solution at each step.

Some Algorithms under the greedy method is given below:—

- Fractional Knapsack problem
- Huffman coding
- Travelling salesman problem
- Dijkstra's Shortest Path algorithm
- Prim's and Kruskal's Algorithm for finding ~~so~~ The Minimum Spanning Tree in graph.
- Job Scheduling

Fractional knapsack Problem:

The fractional knapsack Problem is a variant of knapsack problem where we can take fractions of items instead of having to decide whether to take an entire item on not. It can be solved more efficiently with greedy algorithm.

Algorithm:

1. Calculate the value to weight ratio for each item.
2. Sort the items by the value to weight ratio in descending order.
3. Initialize the total profit and weight to zero.
4. Loop through the sorted items:
   - If the item fits completely add it
   - If it does not fit fully, take the fraction of it that fits.
5. Return the total profit obtained.

Time Complexity Calculation:

Calculation of the value-to-weight ratio (P_i/w_i) for each item. This requires one pass through n items.

Time Complexity T_1 : $O(n)$

Sorting the items by their value-to-weight ratio: Sorting n items typically takes $O(n \log n)$ time using a efficient algorithm like merge sort or quick sort.

Time Complexity T_2 : $O(n \log n)$

Iterating through the sorted list to fill the knapsack; this take $O(n)$ times

Time Complexity T_3 : $O(n)$

Hence total time complexity : $O(n) + O(n \log n) + O(n)$

$= O(n \log n)$

In Best case, Average case and Worst case time complexity remains same that is $O(n \log n)$

Job Scheduling Problem:

The job scheduling problem is also known as sequencing algorithm that can be solved using greedy technique. It involves scheduling jobs to maximize a specific objective, such as profit under given constrains like deadline on time slots.

Algorithm:

Step-1: Sort all jobs in decending order of profit

Step-2: Initialize an array to keep track of free time slots

Step-3: For each job in the sorted list:

- Find the latest available time slot
- If a slot is available, assign the job to that slot and its profit to the total profit.

Time Complexity Calculation:-

Annange all jobs in decneasing ondere of profit using sorting algonithm like quick sont menge sont.

So, Time Complexity $T_1 = O(n \log n)$

For each job (m_1), we have to linean seanch to find particulare slot in annay size of (n) where

n = maximum deadline
m = total jobs

Time Complexity $T_2 = O(n \cdot m)$

In wonst case $m = n$, so $O(n^2)$

Overcall Complexity : $O(n \log n + n) \approx O(n^n)$

	Time Complexity
Best Case	$O(n \log n)$
Avenage Case	$O(n \log n)$
Wonst Case	$O(n^2)$

Travelling Salesman Problem

The travelling salesman problem is a combinational optimization problem where the goal is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Algorithm:

step-1: Start at any city

step-2: Visit the nearest unvisited city

- Find the city with the smallest distance from the current city that hasn't been visited yet.
- Mark the city as visited

step-3: Repeat untill all the cities are visited

step-4: Return to the starting city to complete the tour.

Huffman Coding

Huffman Coding is lossless data compression Algorithm that assigns variable-length binary codes to input characters, with shorter codes assigned to more frequent characters. It uses greedy approach to ensure optimal compression.

Algorithm:

Step-1: Input characters and their frequencies

Step-2: Build Priority Queue: Insert all characters as a node in a priority queue (min heap) based on frequency.

Step-3: Construct Tree:

- Extract two nodes with smallest frequencies.
- Creat a new node with these two nodes as children and frequency equal to their sum.
- Insert the new node back into the priority queue.

Step-4: Repeat until there is only one node left in the priority queue.

Step-5 : Generate codes by the tree.

Time Complexity Calculation

Building the priority queue:—

- Insertion of n elements into a Heap : O(nlogn)

Constructing the tree :

- Extracting the smallest element n-1 time : O(nlogn)
- Inserting combined node back into the heap: O(nlogn)

Hence total time complexity: O(nlogn)

	Time Complexity
Best Case	O(nlogn)
Avenage Case	O(nlogn)
Wonst Case	O(nlogn)

Applications:

1. Fractional Knapsack:
   - Resource allocation
   - Investment Optimization

2. Job Scheduling:
   - CPU scheduling
   - Manufacturing ~~ig~~:

3. Huffman Coding:
   - Data Communication
   - Data Transmission

4. Travelling Salesman Problem:
   - logistics and Transportation
   - Manufacturing and Robotics
   - Data Packet Routing
   - Sales Route planning

5. Minimum Spanning Tree:
   - Network design
   - Transportation System
   - Circuit Design
   - Road Construction
   - Networking

etc.