What is a Greedy Algorithm?

- In Greedy Algorithm a set of resources are recursively divided based on the maximum, immediate availability of that resource at any given stage of execution.

- It follows locally optimal solution. Decisions are made based on the current available information.

- Greedy algorithms have some advantages and disadvantages:

- It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem. **It is simple and easy to program.**

- Analyzing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer). For the Divide and conquer technique, it is not clear whether the technique is fast or slow. This is because at each level of recursion the size of gets smaller and the number of sub-problems increases. It is quick means take less time to run the problem.

- The difficult part is that for greedy algorithms you have to work much harder to understand correctness issues. Even with the correct algorithm, it is hard to prove why it is correct. Proving that a greedy algorithm is correct is more of an art than a science. It involves a lot of creativity.

Architecture of the Greedy approach

- STEP 1)
- Scan the list of activity costs, starting with index 0 as the considered Index.

- STEP 2)
- When more activities can be finished by the time, the considered activity finishes, start searching for one or more remaining activities.

- STEP 3)
- If there are no more remaining activities, the current remaining activity becomes the next considered activity. Repeat step 1 and step 2, with the new considered activity. If there are no remaining activities left, go to step 4.

- STEP 4)
- Return the union of considered indices. These are the activity indices that will be used to maximize throughput.

Fractional Knapsack Problem

- The fractional knapsack problem is also one of the techniques which are used to solve the knapsack problem. In fractional knapsack, the items are broken in order to maximize the profit. The problem in which we break the item is known as a Fractional knapsack problem.

- In Greedy approach, we calculate the ratio of profit/ weight, and accordingly, we will select the item. The item with the highest ratio would be selected first.

Fractional Knapsack Problem Approaches

- There are basically three approaches to solve the problem:

- The first approach is to select the item based on the maximum profit.

- The second approach is to select the item based on the minimum weight.

- The third approach is to calculate the ratio of profit/weight.

Fractional Knapsack Problem First Approach

- Objects: 1 2 3 4 5 6 7
- Profit (P): 10 15 7 8 9 4
- Weight(w): 1 3 5 4 1 3 2
- W (Weight of the knapsack): 15. n (no of items): 7

First approach:

Object	Profit	Weight	Remaining weight
3	15	5	15 - 5 = 10
2	10	3	10 - 3 = 7
6	9	3	7 - 3 = 4
5	8	1	4 - 1 = 3
7	7 * ¾ = 5.25	3	3 - 3 = 0

The total profit would be equal to (15 + 10 + 9 + 8 + 5.25) = 47.25

Fractional Knapsack Problem Second Approach

Second approach:

The second approach is to select the item based on the minimum weight.

Object	Profit	Weight	Remaining weight
1	5	1	15 - 1 = 14
5	7	1	14 - 1 = 13
7	4	2	13 - 2 = 11
2	10	3	11 - 3 = 8
6	9	3	8 - 3 = 5
4	7	4	5 - 4 = 1
3	15 * 1/5 = 3	1	1 - 1 = 0

In this case, the total profit would be equal to (5 + 7 + 4 + 10 + 9 + 7 + 3) = 46

Fractional Knapsack Problem Third Approach

In the third approach, we will calculate the ratio of profit/weight.

Objects: 1 2 3 4 5 6 7

Profit (P): 5 10 15 7 8 9 4

Weight(w): 1 3 5 4 1 3 2

In this case, we first calculate the profit/weight ratio.

Object 1: 5/1 = 5

Object 2: 10/3 = 3. 33

Object 3: 15/5 = 3

Object 4: 7/4 = 1.7

Object 5: 8/1 = 8

Object 6: 9/3 = 3

Object 7: 4/2 = 2

P:w: 5 3.3 3 1.7 8 3 2

Fractional Knapsack Problem Third Approach

Object	Profit	Weight	Remaining weight
5	8	1	15 - 1 = 14
1	5	1	14 - 1 = 13
2	10	3	13 - 3 = 10
3	15	5	10 - 5 = 5
6	9	3	5 - 3 = 2
7	4	2	2 - 2 = 0

- Total profit would be equal to (8 + 5 + 10 + 15 + 9 + 4), i.e., 51.

Fractional Knapsack Problem Algorithm

```
Fractional Knapsack (Array w, Array p, int M)
for i in range(1,n):
{
    calculate p/w

}

    Sort objects in descending order of p/w ratio


for i in range(1,n):
{
    if M>0 :
      If w[i]<=M:

           M = M-w[i]
            p = p + p[i]

      else:

        p = p + p[i](M/w[i])
}
```

Time Complexity:
O(n)+O(nlogn)+O(n)
=O(nlogn)