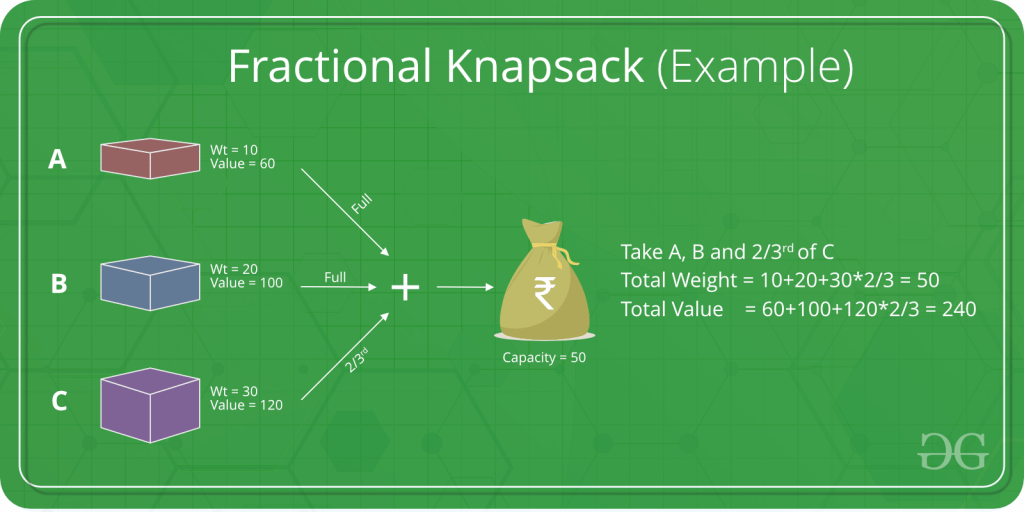
**Topic 1 : Introduction to Greedy Algorithm**

**Greedy**is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to the global optimal solution are best fit for Greedy.  
  
For example, consider the **Fractional Knapsack Problem**. The problem states that:

*Given a list of elements with specific values and weights associated with them, the task is to fill a Knapsack of weight W using these elements such that the value of knapsack is maximum possible.****Note****: You are allowed to take a fraction of an element also in order to maximize the value.*

The local optimal strategy is to choose the item that has maximum value vs weight ratio. This strategy also leads to a global optimal solution because we are allowed to take fractions of an item.  
  
  
  
In general, the **Greedy Algorithm** can be applied to solve a problem if it satisfies the below property:

*At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem.*

Let us consider one more problem known as the **Activity Selection Problem** to understand the use of Greedy Algorithms.  
  
**Problem**: You are given N activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.

**Example 1**: Consider the following 3 activities sorted by  
finish time.  
 start[] = {10, 12, 20};  
 finish[] = {20, 25, 30};  
A person can perform at most two activities. The   
maximum set of activities that can be executed   
is {0, 2} [ These are indexes in start[] and   
finish[] ]  
  
**Example 2**: Consider the following 6 activities   
sorted by finish time.  
 start[] = {1, 3, 0, 5, 8, 5};  
 finish[] = {2, 4, 6, 7, 9, 9};  
A person can perform at most four activities. The   
maximum set of activities that can be executed   
is {0, 1, 3, 4} [ These are indexes in start[] and   
finish[] ]

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.  
  
To do this:

1. Sort the activities according to their finishing time.
2. Select the first activity from the sorted array and print it.
3. Do the following for remaining activities in the sorted array.  
   * If the start time of this activity is greater than or equal to the finish time of previously selected activity then select this activity and print it.

**Topic 2 : Job Sequencing Problem | Greedy Approach**

**Problem**: Given an array of jobs where every job has a deadline and associated profit if the job is finished before the deadline. It is also given that every job takes a single unit of time, so the minimum possible deadline for any job is 1. How to maximize total profit if only one job can be scheduled at a time.  
  
**Examples:**

**Input**: Four Jobs with following deadlines and profits  
 JobID Deadline Profit  
 a 4 20   
 b 1 10  
 c 1 40   
 d 1 30  
**Output**: Following is maximum profit sequence of jobs  
 c, a   
  
**Input**: Five Jobs with following deadlines and profits  
 JobID Deadline Profit  
 a 2 100  
 b 1 19  
 c 2 27  
 d 1 25  
 e 3 15  
**Output**: Following is maximum profit sequence of jobs  
 c, a, e

This is a standard Greedy Algorithm problem. Following is the greedy algorithm to solve the above problem:

1) Sort all jobs in decreasing order of profit.  
2) Initialize the result sequence as the first job in sorted jobs.  
3) Do following for remaining n-1 jobs  
.......a) If the current job can fit in the current result sequence   
 without missing the deadline, add the current job to the result.  
 Else ignore the current job.

**Implementation**:

C++



1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

// C++ program to find the maximum profit

// job sequence from a given array

// of jobs with deadlines and profits

#include<iostream>

#include<algorithm>

using namespace std;

// A structure to represent a job

struct Job

{

char id; // Job Id

int dead; // Deadline of job

// Profit if job is over

// before or on deadline

int profit;

};

// This function is used for sorting all

// jobs according to profit

bool comparison(Job a, Job b)

{

return (a.profit > b.profit);

}

// Returns minimum number of platforms required

void printJobScheduling(Job arr[], int n)

{

// Sort all jobs according to

// decreasing order of profit

Run

Java



**Output:**

Following is maximum profit sequence of job : c a e

**Time Complexity** of the above solution is O(n2). It can be optimized using Disjoint Set Data Structure. Please refer below post for details.