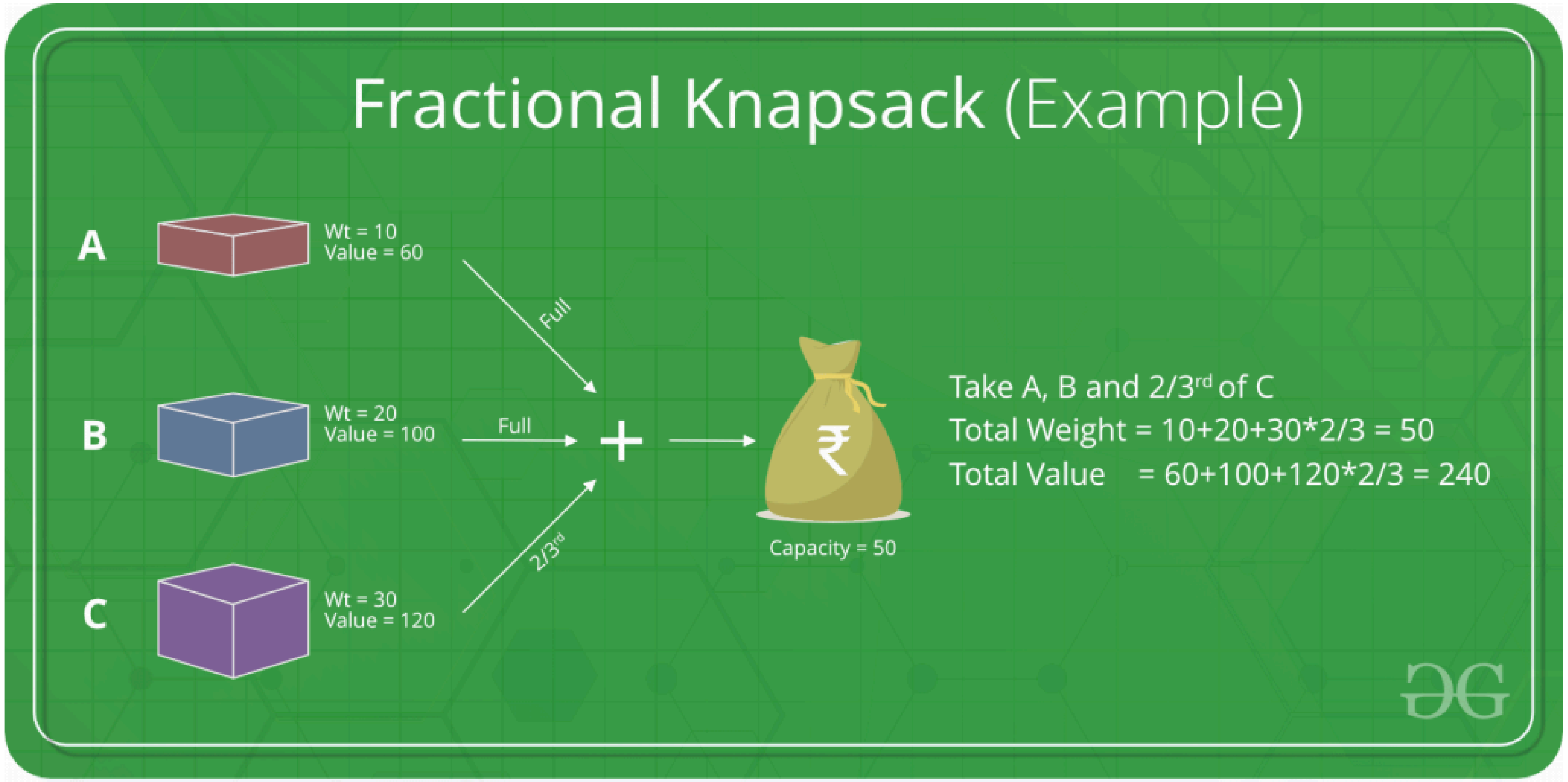
Introduction to Greedy Algorithms

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 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.

```
     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.

Dash
All
Articles
Videos
Problems
Quiz
Contest

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 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.

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.