Name: Vivek Vikram Pundkar

Roll no: 77

GR no: 11910860

Division: C

Batch: 3

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Fractional Knapsack Problem**

**Problem Statement**

Given a set of N items each having value V with weight W and the total capacity of a knapsack. The task is to find the maximal value of fractions of items that can fit into the knapsack.

Examples:

Input: A[] = {{60, 20} , {100, 50}, {120, 30}}, Total\_capacity = 50

Output: 180.00

Explanation: Take the first item and the third item. Total value = 60 + 120 = 180 with a total capacity of 20 + 30 = 50

Input: A[] = {{500, 30}}, Total\_capacity = 10

Output: 166.67

Explanation: Since the total capacity of the knapsack is 10, consider one-third of the item.

**Brute Force Approach**

The most basic approach is to try all possible subsets and possible fractions of the given set and find the maximum value among all such fractions.

The time complexity will be exponential, as you need to find all possible combinations of the given set.

**Greedy Approach**

The Fractional Knapsack problem can be solved efficiently using the greedy algorithm, where you need to sort the items according to their value/weight ratio.

**Algorithm**

* Sort the given array of items according to weight / value(W /V) ratio in descending order.
* Start adding the item with the maximum W / V ratio.
* Add the whole item, if the current weight is less than the capacity, else, add a portion of the item to the knapsack.
* Stop, when all the items have been considered and the total weight becomes equal to the weight of the given knapsack.

struct item

{

    int value,weight;

};

bool cmp(item a,item b)

{

    double r1=(double)a.value/a.weight;

    double r2=(double)b.value/b.weight;

    return(r1>r2);

}

double fractionalknapsack(item A[],int Total\_Capacity,int n)

{

    sort(A,A + n,cmp);

    int cur\_weight = 0;

    double final\_val = 0.0;

    for(int i=0;i<n;i++)

    {

        if(cur\_weight + arr[i].weight <= Total\_Capacity)

        {

            Cur\_weight += A[i].weight;

            Final\_val += A[i].value;

        }

        else

        {

            int remain = Total\_capacity - cur\_weight;

            Final\_val += A[i].value \* ((double)remain / A[i].weight);

        }

    }

    return final\_val;

}

Time Complexity: O(N \*log N) where N is the size of the array.

Space Complexity: O(1) because no extra space is needed.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Huffman Coding using Greedy Algorithm**

Problem Statement