Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

SE – COMP (SE-A/08) **Sub- DAA Lab**

Name	**Naman Badlani**
UID No.	**2021300008**
Subject	**Design And Analysis Of Algorithm**
Class	**Comps A**
Experiment No.	**7**
AIM	To implement fractional knapsack in C

Theory –

The fractional knapsack problem is a classic optimization problem in computer science and mathematics. Given a set of items, each with a weight and a value, and a knapsack with a capacity, the goal is to fill the knapsack with items in such a way that the total value of the items is maximized without exceeding the capacity of the knapsack. Unlike the 0-1 knapsack problem, where items must be either included or excluded entirely, in the fractional knapsack problem, we can include fractions of an item in the knapsack.

Algorithm –

1. Calculate the ratio (profit/weight) for each item.
2. Sort all the items in decreasing order of the ratio.
3. Initialize res = 0, curr_cap = given_cap.
4. Do the following for every item i in the sorted order:
5. If the weight of the current item is less than or equal to the remaining capacity, then add the value of that item into the result
6. Else add the current item as much as we can and break out of the loop.
7. Return res.

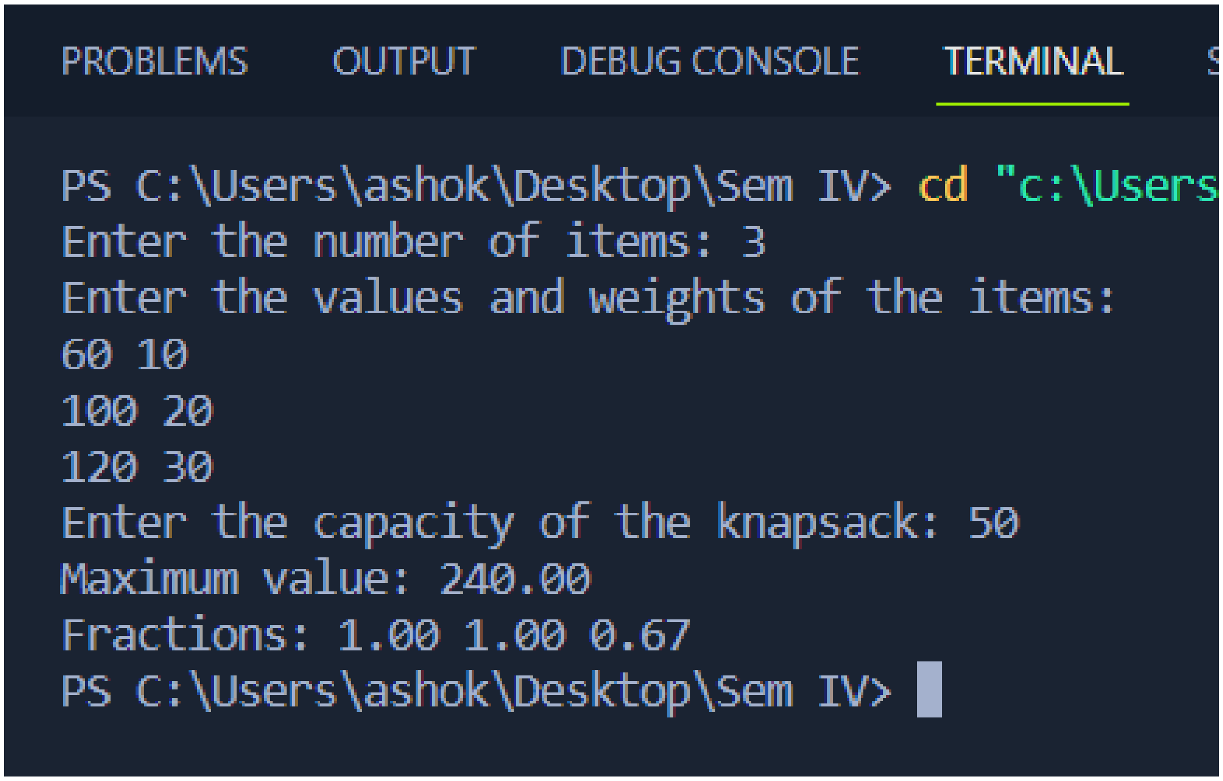
Program –

```
#include <stdio.h>

// Function to solve the fractional knapsack problem
```

```
void fractional_knapsack(int n, float value[], float weight[], float capacity) {
    // Calculate value-to-weight ratio for each item.
    float ratio[n];
    for (int i = 0; i < n; i++) {
        ratio[i] = value[i] / weight[i];
    }

    // Sort items by value-to-weight ratio in decreasing order using insertion
sort.
    for (int i = 1; i < n; i++) {
        float key_ratio = ratio[i];
        float key_value = value[i];
        float key_weight = weight[i];
        int j = i - 1;
        while (j >= 0 && ratio[j] < key_ratio) {
            ratio[j + 1] = ratio[j];
            value[j + 1] = value[j];
            weight[j + 1] = weight[j];
            j--;
        }
        ratio[j + 1] = key_ratio;
        value[j + 1] = key_value;
        weight[j + 1] = key_weight;
    }

    float max_value = 0.0;
    float fractions[n];
    for (int i = 0; i < n; i++) {
        if (weight[i] <= capacity) {
            fractions[i] = 1.0;
            max_value += value[i];
            capacity -= weight[i];
        } else {
            fractions[i] = capacity / weight[i];
            max_value += value[i] * capacity / weight[i];
            break;
        }
    }

    // Print the maximum value and the fraction of each item that was included in
the knapsack
```

```c
    printf("Maximum value: %.2f\n", max_value);
    printf("Fractions: ");
    for (int i = 0; i < n; i++) {
        printf("%.2f ", fractions[i]);
    }
}

// Driver program to test the above function
int main() {
    int n;
    float value[100], weight[100], capacity;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the values and weights of the items:\n");
    for (int i = 0; i < n; i++) {
        scanf("%f %f", &value[i], &weight[i]);
    }

    printf("Enter the capacity of the knapsack: ");
    scanf("%f", &capacity);

    fractional_knapsack(n, value, weight, capacity);

    return 0;
}
```

Result Analysis –



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\ashok\Desktop\Sem IV> cd "c:\Users
Enter the number of items: 3
Enter the values and weights of the items:
60 10
100 20
120 30
Enter the capacity of the knapsack: 50
Maximum value: 240.00
Fractions: 1.00 1.00 0.67
PS C:\Users\ashok\Desktop\Sem IV>
```

- One way to solve the fractional knapsack problem is by using a greedy approach. In the greedy approach, we sort the items in decreasing order of their value-to-weight ratio and iteratively add as much of each item as possible to the knapsack until either the knapsack is full, or we have no more items left. By always adding the items with the highest value-to-weight ratio first, we ensure that we are adding the most valuable items to the knapsack first. If an item cannot be added to the knapsack entirely, we add a fraction of it that maximizes the total value of the knapsack.

- The greedy approach works well for the fractional knapsack problem and has a time complexity of O (n log n), where n is the number of items.