# EXPERIMENT 6

**AIM** : Write a program to implement knapsack problem using greedy method.

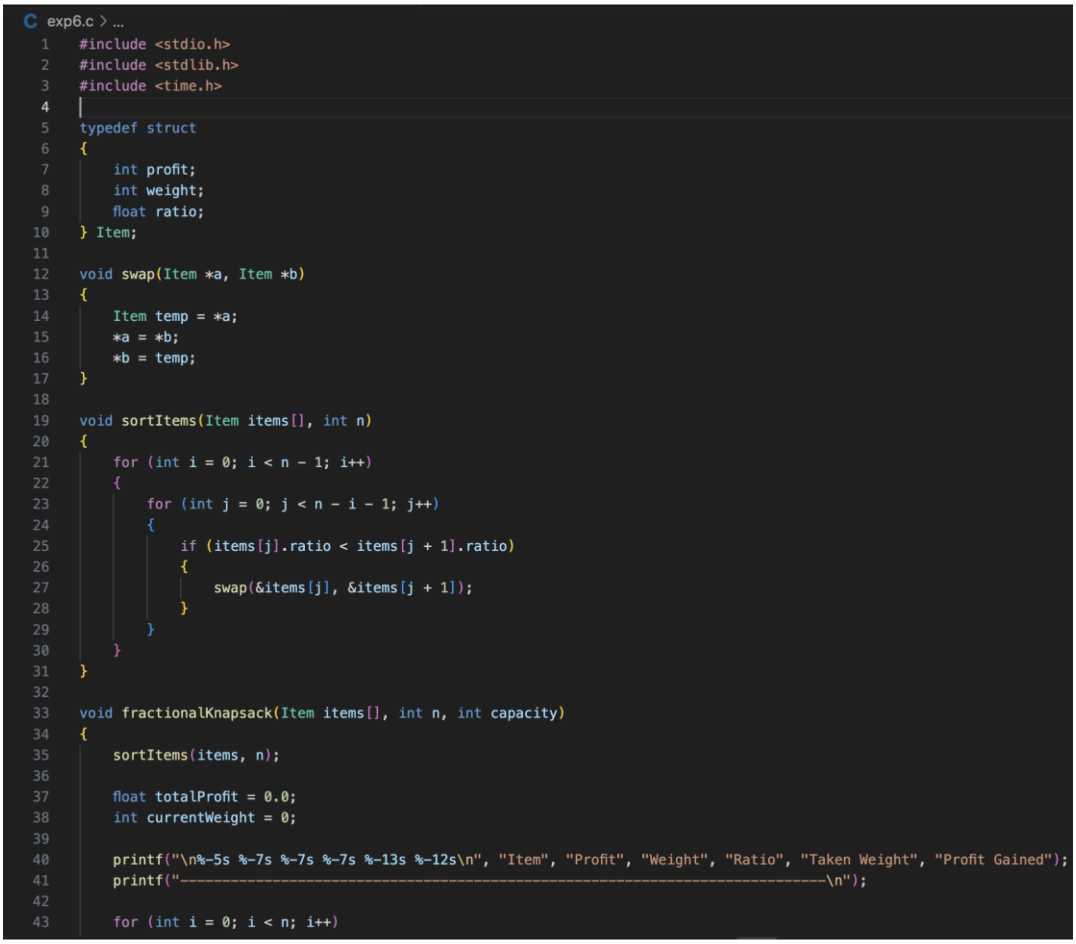
**THEORY :**

The Greedy method for the Knapsack problem is mainly applied to the Fractional Knapsack, where items can be split into smaller portions. The algorithm first calculates each item's profit-to-weight ratio and then sorts all items in descending order of this ratio. It keeps adding items with the highest ratio to the knapsack so that the profit increases as quickly as possible. If the knapsack cannot accommodate the full next item, only the proportional fraction required to fill the remaining capacity is taken. This strategy ensures a fast and optimal solution for the fractional case, but it does not work for the 0/1 Knapsack, where items must be taken as a whole.

**ALGORITHM :**

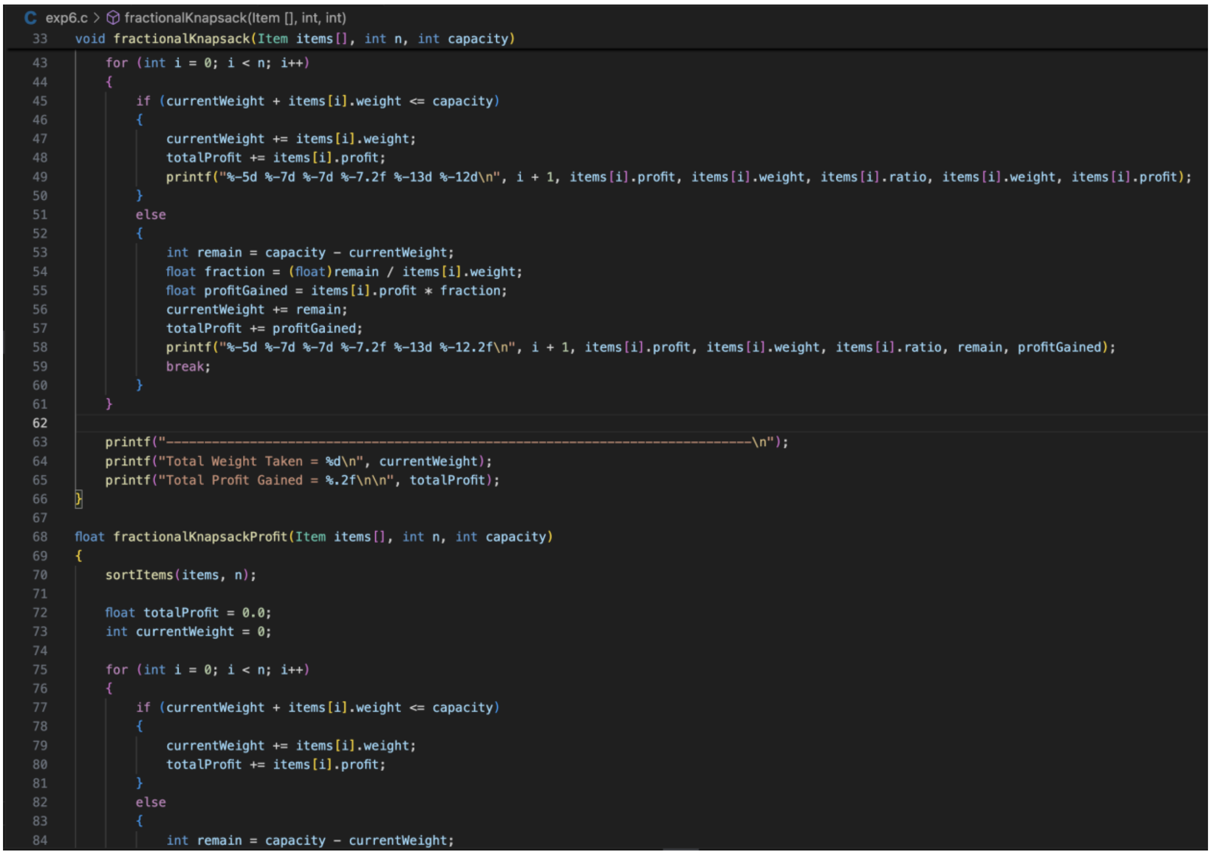
```
Algorithm FractionalKnapsack(W, items)
    // W = Capacity of knapsack
    // items = list of (profit, weight)

    For each item in items:
        compute ratio = profit / weight

    Sort items in descending order of ratio

    totalProfit = 0

    For each item in items:
        if item.weight ≤ W:
            W = W - item.weight
            totalProfit = totalProfit + item.profit
        else:
            fraction = W / item.weight
            totalProfit = totalProfit + (item.profit * fraction)
            W = 0
            break

    return totalProfit
End Algorithm
```

# SOURCE CODE:



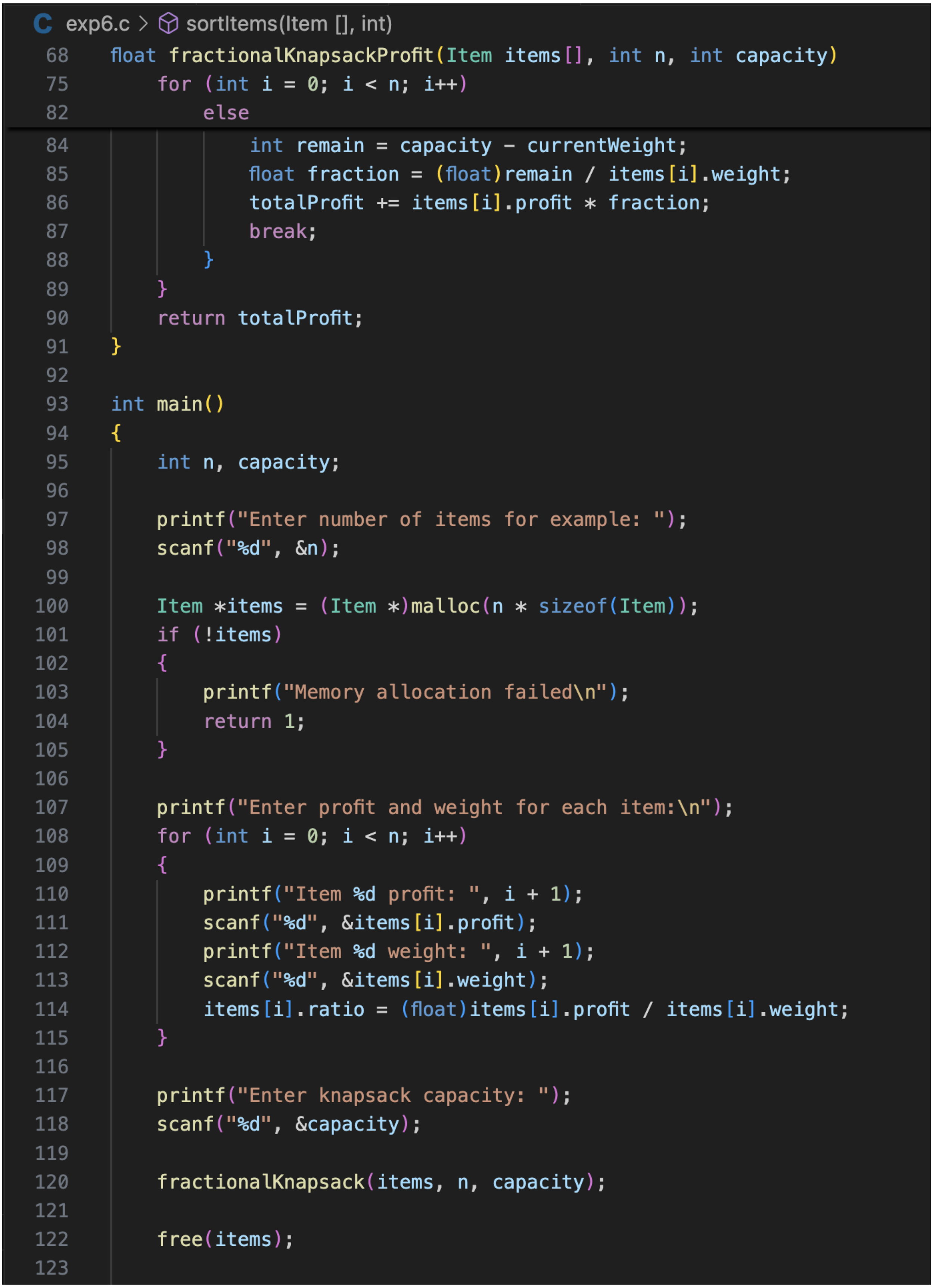
exp6.c > ...

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct
{
    int profit;
    int weight;
    float ratio;
} Item;

void swap(Item *a, Item *b)
{
    Item temp = *a;
    *a = *b;
    *b = temp;
}

void sortItems(Item items[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (items[j].ratio < items[j + 1].ratio)
            {
                swap(&items[j], &items[j + 1]);
            }
        }
    }
}

void fractionalKnapsack(Item items[], int n, int capacity)
{
    sortItems(items, n);

    float totalProfit = 0.0;
    int currentWeight = 0;

    printf("\n%-5s %-7s %-7s %-7s %-13s %-12s\n", "Item", "Profit", "Weight", "Ratio", "Taken Weight", "Profit Gained");
    printf("---------------------------------------------------------------------------\n");

    for (int i = 0; i < n; i++)
```

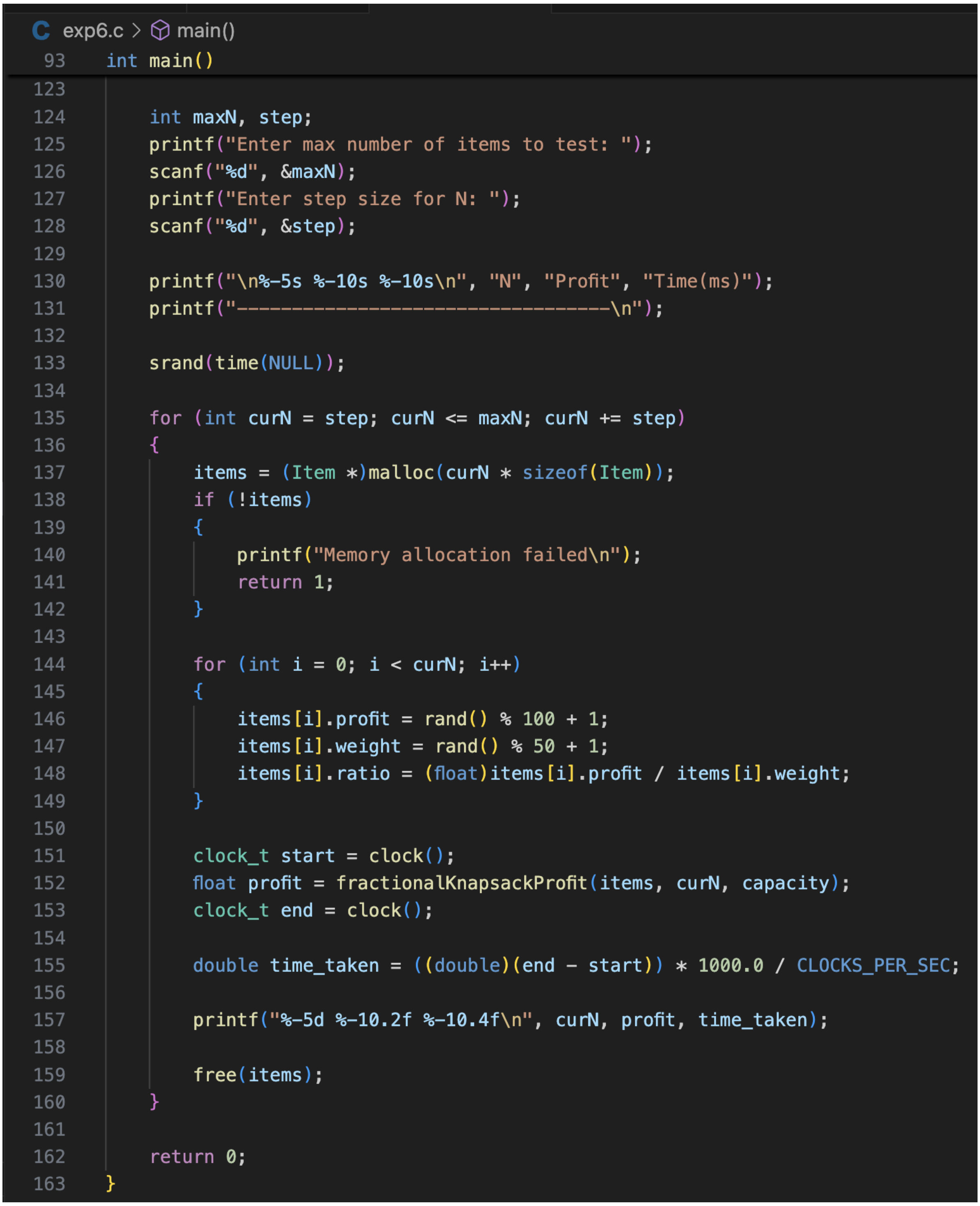


exp6.c > fractionalKnapsack(Item [], int, int)

```c
void fractionalKnapsack(Item items[], int n, int capacity)
    for (int i = 0; i < n; i++)
    {
        if (currentWeight + items[i].weight <= capacity)
        {
            currentWeight += items[i].weight;
            totalProfit += items[i].profit;
            printf("%-5d %-7d %-7d %-7.2f %-13d %-12d\n", i + 1, items[i].profit, items[i].weight, items[i].ratio, items[i].weight, items[i].profit);
        }
        else
        {
            int remain = capacity - currentWeight;
            float fraction = (float)remain / items[i].weight;
            float profitGained = items[i].profit * fraction;
            currentWeight += remain;
            totalProfit += profitGained;
            printf("%-5d %-7d %-7d %-7.2f %-13d %-12.2f\n", i + 1, items[i].profit, items[i].weight, items[i].ratio, remain, profitGained);
            break;
        }
    }

    printf("---------------------------------------------------------------------------\n");
    printf("Total Weight Taken = %d\n", currentWeight);
    printf("Total Profit Gained = %.2f\n\n", totalProfit);
}

float fractionalKnapsackProfit(Item items[], int n, int capacity)
{
    sortItems(items, n);

    float totalProfit = 0.0;
    int currentWeight = 0;

    for (int i = 0; i < n; i++)
    {
        if (currentWeight + items[i].weight <= capacity)
        {
            currentWeight += items[i].weight;
            totalProfit += items[i].profit;
        }
        else
        {
            int remain = capacity - currentWeight;
```

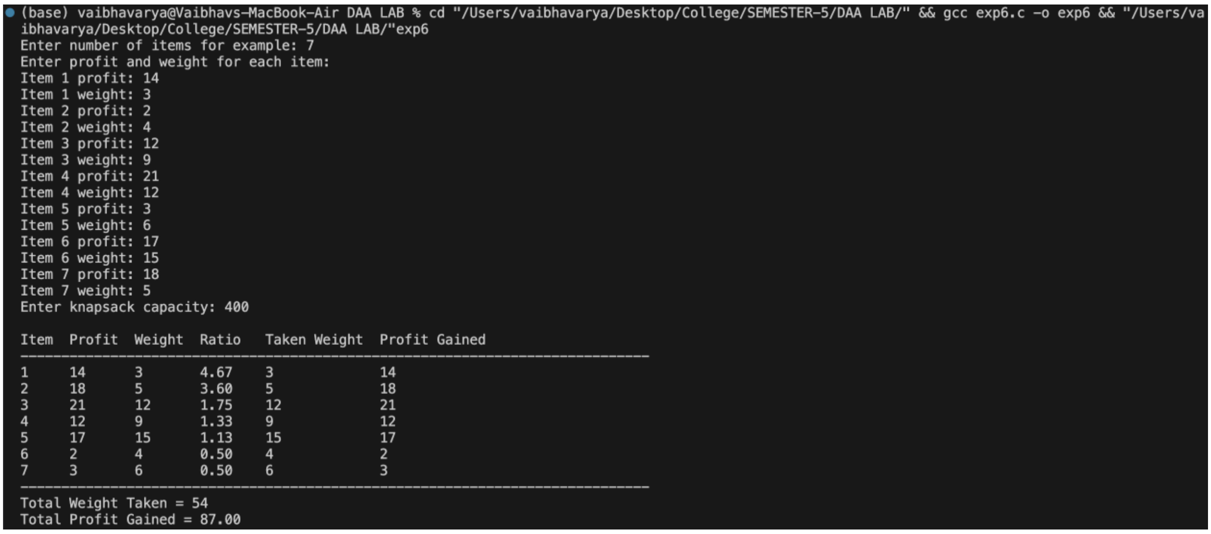


```c
float fractionalKnapsackProfit(Item items[], int n, int capacity)
    for (int i = 0; i < n; i++)
        else
            int remain = capacity - currentWeight;
            float fraction = (float)remain / items[i].weight;
            totalProfit += items[i].profit * fraction;
            break;
        }
    }
    return totalProfit;
}

int main()
{
    int n, capacity;

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

    Item *items = (Item *)malloc(n * sizeof(Item));
    if (!items)
    {
        printf("Memory allocation failed\n");
        return 1;
    }

    printf("Enter profit and weight for each item:\n");
    for (int i = 0; i < n; i++)
    {
        printf("Item %d profit: ", i + 1);
        scanf("%d", &items[i].profit);
        printf("Item %d weight: ", i + 1);
        scanf("%d", &items[i].weight);
        items[i].ratio = (float)items[i].profit / items[i].weight;
    }

    printf("Enter knapsack capacity: ");
    scanf("%d", &capacity);

    fractionalKnapsack(items, n, capacity);

    free(items);

```



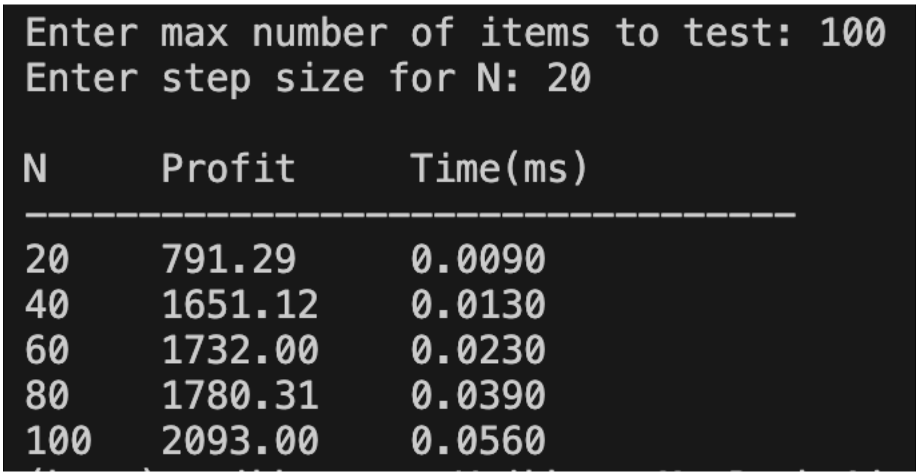
```c
int main()

    int maxN, step;
    printf("Enter max number of items to test: ");
    scanf("%d", &maxN);
    printf("Enter step size for N: ");
    scanf("%d", &step);

    printf("\n%-5s %-10s %-10s\n", "N", "Profit", "Time(ms)");
    printf("----------------------------------\n");

    srand(time(NULL));

    for (int curN = step; curN <= maxN; curN += step)
    {
        items = (Item *)malloc(curN * sizeof(Item));
        if (!items)
        {
            printf("Memory allocation failed\n");
            return 1;
        }

        for (int i = 0; i < curN; i++)
        {
            items[i].profit = rand() % 100 + 1;
            items[i].weight = rand() % 50 + 1;
            items[i].ratio = (float)items[i].profit / items[i].weight;
        }

        clock_t start = clock();
        float profit = fractionalKnapsackProfit(items, curN, capacity);
        clock_t end = clock();

        double time_taken = ((double)(end - start)) * 1000.0 / CLOCKS_PER_SEC;

        printf("%-5d %-10.2f %-10.4f\n", curN, profit, time_taken);

        free(items);
    }

    return 0;
}
```

## OUTPUT:



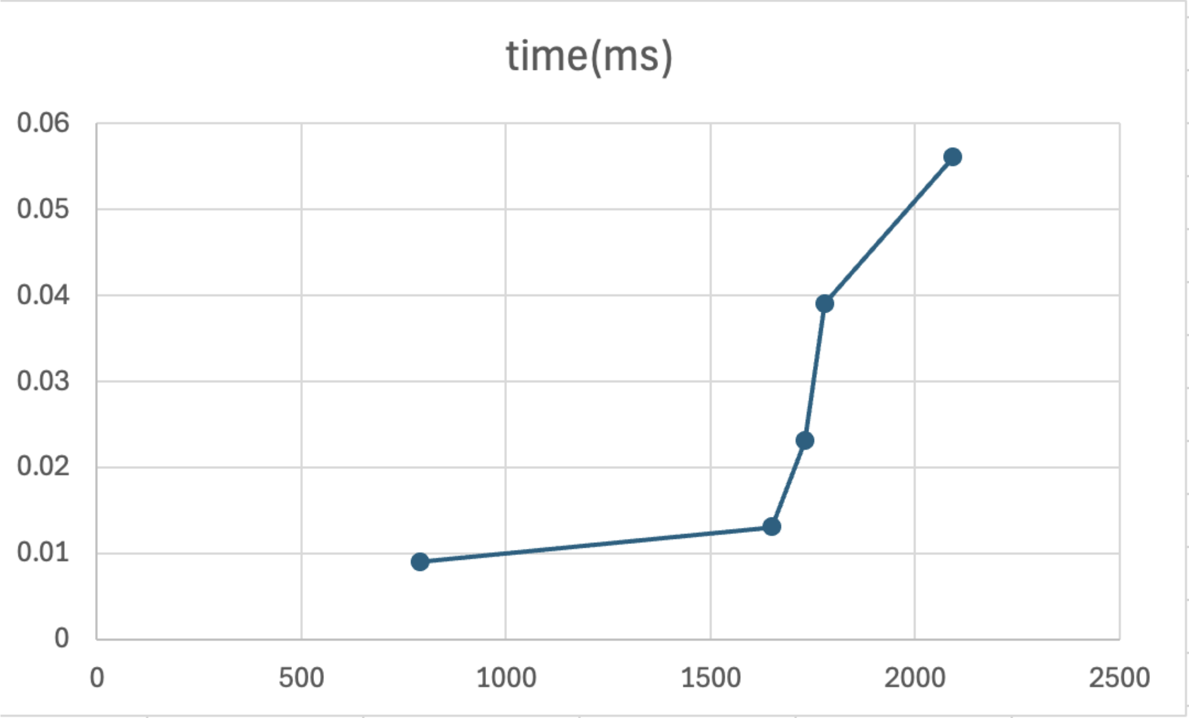
```
● (base) vaibhavarya@Vaibhavs-MacBook-Air DAA LAB % cd "/Users/vaibhavarya/Desktop/College/SEMESTER-5/DAA LAB/" && gcc exp6.c -o exp6 && "/Users/va
  ibhavarya/Desktop/College/SEMESTER-5/DAA LAB/"exp6
  Enter number of items for example: 7
  Enter profit and weight for each item:
  Item 1 profit: 14
  Item 1 weight: 3
  Item 2 profit: 2
  Item 2 weight: 4
  Item 3 profit: 12
  Item 3 weight: 9
  Item 4 profit: 21
  Item 4 weight: 12
  Item 5 profit: 3
  Item 5 weight: 6
  Item 6 profit: 17
  Item 6 weight: 15
  Item 7 profit: 18
  Item 7 weight: 5
  Enter knapsack capacity: 400

  Item  Profit  Weight  Ratio   Taken Weight  Profit Gained
  ----------------------------------------------------------------------------
  1     14      3       4.67    3             14
  2     18      5       3.60    5             18
  3     21      12      1.75    12            21
  4     12      9       1.33    9             12
  5     17      15      1.13    15            17
  6     2       4       0.50    4             2
  7     3       6       0.50    6             3
  ----------------------------------------------------------------------------
  Total Weight Taken = 54
  Total Profit Gained = 87.00
```

## Profit Vs Time Taken (ms) Table:



```
Enter max number of items to test: 100
Enter step size for N: 20

N     Profit     Time(ms)
----------------------------------
20    791.29     0.0090
40    1651.12    0.0130
60    1732.00    0.0230
80    1780.31    0.0390
100   2093.00    0.0560
```

## Graph:



**TIME COMPLEXITY :**

| | BEST CASE | WORST CASE | AVERAGE CASE |
|---|---|---|---|
| Knapsack Algorithm | O(nlogn) | O(nlogn) | O(nlogn) |

**LEARNING OUTCOMES :**