

LABORATORY REPORT

Algorithm Laboratory (CS-39001)

B.Tech Program in ECSc

Submitted By

Name:- SANNIDHI DEB

Roll No: 2330044



Kalinga Institute of Industrial Technology

(Deemed to be University) Bhubaneswar, India

Autumn, 2025

Table of Contents

[illegible]

Experiment Number	5.1
Experiment Title	<p>Write an algorithm and a C program to find the maximum profit nearest to but not exceeding the given knapsack capacity using the Fractional Knapsack algorithm.</p> <p>Input: Number of items: 3 Weights of items, $\{w_1, w_2, w_3\} = \{18, 15, 10\}$ Profits of items, $\{P_1, P_2, P_3\} = \{25, 24, 15\}$ Maximum capacity of knapsack, $m = 20$</p>
Date of Experiment	18/09/2025
Date of Submission	24/09/2025

1.Algorithm:-

Exp-05: Knapsack
Frac_Knapsack(P, w, n)
1. for $i \leftarrow 1$ to n
2. Determine $\frac{P}{w}$
3. Arrange the ratio in descending order
4. for $i \leftarrow 1$ to n
5. if $m > 0$
6. if $(w_i \leq m)$
7. $P = P + P_i$
8. $m = m - w_i$
9. else $P = P + \frac{m}{w_i} \times P_i$
10. break

Sannidhi Deb, 2330044

2. Code:-

```
#include <stdio.h>
```

```
typedef struct {
```

```
    int weight;
```

```
    int profit;
```

```
    float ratio;
```

```
} Item;
```

```
void sortItems(Item items[], int n) {
```

```
    for (int i = 0; i < n - 1; i++) {
```

```
        for (int j = i + 1; j < n; j++) {
```

```
            if (items[i].ratio < items[j].ratio) {
```

```
                Item temp = items[i];
```

```
                items[i] = items[j];
```

```
                items[j] = temp;
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
int main() {
```

```
    int n = 3;
```

```
    int m = 20;
```

```
    int weights[] = {18, 15, 10};
```

```
int profits[] = {25, 24, 15};
```

```
Item items[n];
```

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

```
    items[i].weight = weights[i];
```

```
    items[i].profit = profits[i];
```

```
    items[i].ratio = (float)profits[i] / weights[i];
```

```
}
```

```
sortItems(items, n);
```

```
int remaining_capacity = m;
```

```
float total_profit = 0.0;
```

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

```
    if (items[i].weight <= remaining_capacity) {
```

```
        total_profit += items[i].profit;
```

```
        remaining_capacity -= items[i].weight;
```

```
    } else {
```

```
        total_profit += items[i].profit * ((float)remaining_capacity / items[i].weight);
```

```
        remaining_capacity = 0;
```

```
        break;
```

```

    }

}

printf("\nSannidhi Deb\n 2330044\n\n");

printf("Maximum Profit = %.2f\n", total_profit);

return 0;

}

```

3.Results/Output:- Entire Screen Shot including Date & Time:-

```

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>gcc exp5_1.c -o exp5_1.exe

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>exp5_1

Sannidhi Deb
2330044

Maximum Profit = 31.50

C:\Users\debsa\OneDrive\Desktop\AL_Lab_044>

```

4. Remarks:-

1. What type of algorithm is used?

① Fractional Knapsack uses 'greedy algorithm' where items are selected based on highest profit to weight ratio until capacity is filled.

Sannidhi Deb, 2330044

2. Analyze the complexity of your algorithm.

② Sorting the items by P/w ratio $\rightarrow O(n \log n)$
Selection after sorting $\rightarrow O(n)$
Therefore, overall time complexity is $O(n \log n)$
Sannidhi Deb, 2330044

3. Any other observations?

③ The greedy method always guarantees optimal solution for fractional case, and also allows breaking items unlike 0/1 Knapsack, making it easier to reach optimal solution.
Sannidhi Deb, 2330044
ALINGA

5. Conclusion:-

The fractional knapsack experiment successfully demonstrated how greedy optimization can be applied to maximize profit without exceeding the capacity constraint. By prioritizing items based on profit-to-weight ratio and allowing fractional selection, the algorithm achieved an optimal solution efficiently, highlighting its effectiveness in resource allocation problems

Sannidhi Deb
(2330044)

Sannidhi Deb

Signature of the FIC

(Name of the FIC)

