Symbiosis Institute of Technology

Department of Computer Science and Engineering

Academic Year 2025-26

Design Analysis of Algorithm– Lab

Batch 2023-27 - Sem V

Lab Assignment No:- 5	
Name of Student	Deepti Pal
PRN No.	23070122081
Batch	2023-27
Class	CSE- A3
Academic Year & Semester	2025-26 TY, 5th sem
Date of Submission	15 Sept 2025,
Title of Assignment:	You're packing a suitcase for a flight, and the airline allows a maximum weight of 15 kg. You have a set of items to pack, each with a weight and a utility value (how useful it is to you on the trip). But you can't take all items — so you need to choose the most valuable combination without exceeding 15 kg. Item Weight (kg) Value (usefulness) Laptop 3 9 Headphones 1 5 Jacket 5 10 Camera 4 7 Book 2 4 Shoes 6 6

	Problem: Choose the combination of items that gives maximum value but keeps the total weight ≤ 15 kg. This is the 0/1 Knapsack Problem, because you can either take an item (1) or leave it (0) — no partial items. You should be able to change the maximum allowed weight, weights of different items. You can also include new items, delete items given in the example.
Theory: (Handwritten)	**Fractional Knapsack (Greedy)** - Idea: Sort by value/weight ratio. - Time: O(n log n). - Space: O(1). **0/1 Knapsack (Dynamic Programming)** - Idea: Use DP table dp[i][w] = max value with first i items and capacity w. - Time: O(nW). - Space: O(nW). **Optimization:** - Fractional: early break when bag is full. - 0/1: Space optimization to O(W) possible.

Source code	#include <iostream>
	#include <vector>
	#include <string>
	using namespace std;
	
	struct Item {
	  string name;
	  int weight, value;
	};
	
	int knapsack(int maxW, vector<Item>& items, vector<string>& chosenItems) {
	  int n = items.size();
	
	  vector<vector<int>> dp(n + 1, vector<int>(maxWW + 1, 0));
	
	  for (int i = 1; i <= n; i++) {
	  for (int w = 0; w <= maxW; w++) {

```
            if (items[i-1].weight <= w) {
                dp[i][w] = max(dp[i-1][w],
                               dp[i-1][w - items[i-1].weight] + items[i-1].value);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }


    int w = maxW;
    for (int i = n; i > 0; i--) {
        if (dp[i][w] != dp[i-1][w]) {
            chosenItems.push_back(items[i-1].name);
            w -= items[i-1].weight;
        }
    }

    return dp[n][maxW];
}

int main() {
    int maxW = 15;

    vector<Item> items = {
        {"Laptop", 3, 9},
        {"Headphones", 1, 5},
        {"Jacket", 5, 10},
        {"Camera", 4, 7},
        {"Book", 2, 4},
        {"Shoes", 6, 6}
    };

    vector<string> chosenItems;
    int maxValue = knapsack(maxW, items, chosenItems);

    cout << "Maximum Value = " << maxValue << endl;
    cout << "Items chosen: ";
    for (auto &it : chosenItems) cout << it << " ";
    cout << endl;

    return 0;
}
```

Output Screenshots

Assignment 5 - C++

You're packing a suitcase for a flight, and the airline allows a **maximum weight of 15 kg**. You have a set of items to pack, each with a **weight** and a **utility value** (how useful it is to you on the trip). But you can't take all items — so you need to choose the most valuable combination **without exceeding 15 kg**.

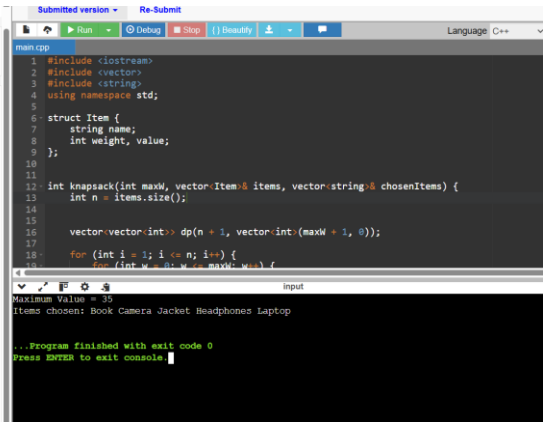
Item	Weight (kg)	Value (usefulness)
Laptop	3	9
Headphones	1	5
Jacket	5	10
Camera	4	7
Book	2	4
Shoes	6	6

Problem:

Choose the combination of items that gives **maximum value** but keeps the **total weight ≤ 15 kg**.

This is the **0/1 Knapsack Problem**, because you can either take an item (1) or leave it (0) — no partial items.

You should be able to change the maximum allowed weight, weights of different items. You can also include new items, delete items given in the example

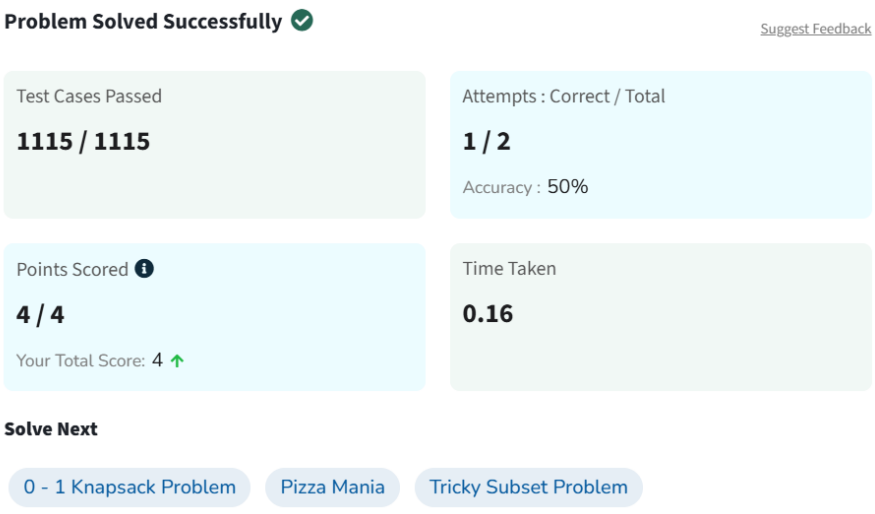


Problems Solved from GeeksforGeeks

1. Fractional Knapsack (Greedy) –

Fractional Knapsack | Practice | GeeksforGeeks

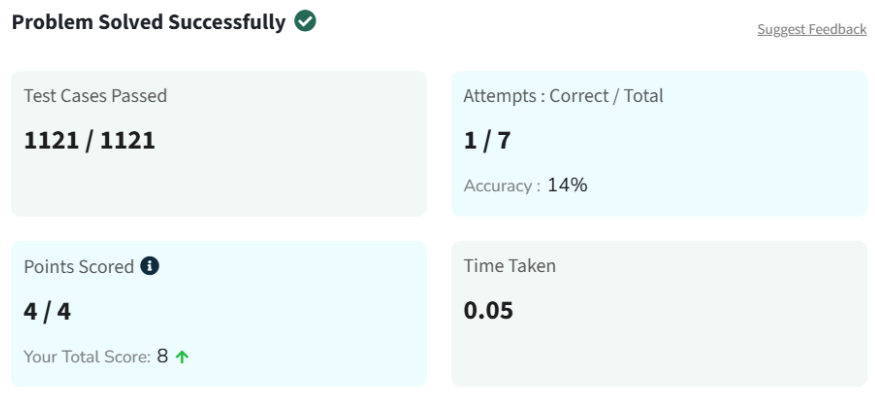
```cpp
class Solution {
public:
    double fractionalKnapsack(vector<int>& val, vector<int>& wt, int capacity) {
        int n = val.size();

        // Pair each item as {ratio, index}
        vector<pair<double, int>> ratio;
        for (int i = 0; i < n; i++) {
            double r = (double)val[i] / wt[i];
            ratio.push_back({r, i});
        }

        // Sort by ratio (highest first)
        sort(ratio.rbegin(), ratio.rend());

        double totalValue = 0.0;

        for (int i = 0; i < n; i++) {
            int idx = ratio[i].second;

            if (wt[idx] <= capacity) {
                // Take the whole item
                totalValue += val[idx];
                capacity -= wt[idx];
            } else {
                // Take fraction of item
                totalValue += (double)val[idx] * ((double)capacity / wt[idx]);
                break; // bag is full
            }
        }

        return totalValue;
    }
};
```



2. 0/1 Knapsack (DP) –

[0 - 1 Knapsack Problem | Practice | GeeksforGeeks]

```
class Solution {
public:
    int knapsack(int W, vector<int>& val, vector<int>& wt) {
        int n = val.size();
        vector<vector<int>> dp(n+1, vector<int>(W+1, 0));

        for (int i = 1; i <= n; i++) {
            for (int w = 1; w <= W; w++) {
                if (wt[i-1] <= w)
                    dp[i][w] = max(val[i-1] + dp[i-1][w - wt[i-1]], dp[i-1][w]);
                else
                    dp[i][w] = dp[i-1][w];
            }
        }
        return dp[n][W];
    }
};
```



3. Bounded Knapsack

[Knapsack with Duplicate Items | Practice | GeeksforGeeks]

```cpp
class Solution {
public:
    int knapSack(vector<int>& val, vector<int>& wt, int capacity) {
        int n = val.size();
        vector<int> dp(capacity + 1, 0);

        for (int w = 0; w <= capacity; w++) {
            for (int i = 0; i < n; i++) {
                if (wt[i] <= w) {
                    dp[w] = max(dp[w], val[i] + dp[w - wt[i]]);
                }
            }
        }
        return dp[capacity];
    }
};
```

Problem Solved Successfully

Suggest Feedback

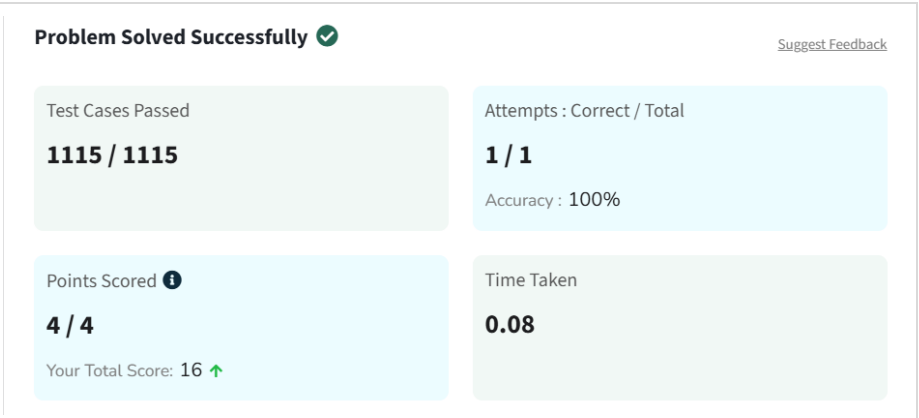
Test Cases Passed	Attempts : Correct / Total
1111 / 1111	**1 / 1**
	Accuracy : 100%

Points Scored	Time Taken
4 / 4	**0.02**
Your Total Score: 12 ↑	

4. Subset Sum Problem

[Subset Sum Problem | Practice | GeeksforGeeks]

```cpp
class Solution {
public:
    bool isSubsetSum(vector<int>& arr, int sum) {
        int n = arr.size();
        vector<vector<bool>> dp(n+1, vector<bool>(sum+1, false));

        for (int i = 0; i <= n; i++)
            dp[i][0] = true;

        for (int i = 1; i <= n; i++) {
            for (int s = 1; s <= sum; s++) {
                if (arr[i-1] <= s)
                    dp[i][s] = dp[i-1][s] || dp[i-1][s - arr[i-1]];
                else
                    dp[i][s] = dp[i-1][s];
            }
        }

        return dp[n][sum];
    }
};
```

	
Conclusion	Thus, we implemented and compared Fractional Knapsack using Greedy and 0/1 Knapsack using DP, discussed time/space complexities, and provided real-world use cases. The assignment matches exactly with online practice problems that reinforce the concepts.