PBLE Report

Semester	S.E. Semester III – Computer Engineering
Subject	Analysis of Algorithm
Subject Professor In-charge	Prof. Swapnil S. Sonawane

Roll Number	Name of Students
24102C0086	Jay Keluskar
24102C0075	Rishi Shah
25102C2008	Samarth Pagaria

Name of the Project:

Warehouse Packing Optimization – Maximize value of stored goods within weight/space limit. Use Cases: (a) Knapsack packing. (b) Compare 0/1 vs Fractional. (c) Add Branch & Bound optimization.

Project Description:

The *Warehouse Packing Optimization* project aims to determine the most efficient way to store items in a warehouse so that the total value of goods is maximized without exceeding the available weight or space capacity. Each item has two parameters — weight and value — and the system must decide how much of each item to include to achieve the highest possible total value.

This project applies the Fractional Knapsack Algorithm, a Greedy strategy that selects items based on their value-to-weight ratio. The algorithm prioritizes items that offer the highest value per unit of weight and allows taking fractions of items when the remaining capacity cannot accommodate the entire item. This approach ensures that the solution is optimal and computationally efficient, making it ideal for real-time warehouse or logistics operations.

Project Code:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct {
    int weight;
    int value;
} Item;
int knapsack01(Item items[], int n, int capacity) {
    int i, w;
    int dp[50][50];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0) {
                dp[i][w] = 0;
            }
            else if (items[i - 1].weight <= w) {
                int include = items[i - 1].value + dp[i - 1][w - items[i -
1].weight];
                int exclude = dp[i - 1][w];
                dp[i][w] = (include > exclude) ? include : exclude;
            }
            else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }
    return dp[n][capacity];
}
int main() {
    int n, capacity;
    int i;
    printf("Enter number of items: ");
    if (scanf("%d", &n) != 1 || n < 0 || n > 50) {
        printf("Invalid number of items. Exiting.\n");
        return 1;
    }
    Item items[50];
    for (i = 0; i < n; i++) {
        printf("Enter weight and value of item %d: ", i + 1);
        if (scanf("%d %d", &items[i].weight, &items[i].value) != 2) {
            printf("Invalid input for item. Exiting.\n");
            return 1;
```

```
        }
    }
    printf("Enter capacity of warehouse: ");
    if (scanf("%d", &capacity) != 1 || capacity < 0 || capacity > 50) {
        printf("Invalid capacity. Exiting.\n");
        return 1;
    }
    int maxValue = knapsack01(items, n, capacity);
    printf("\n--- Warehouse Packing Optimization ---\n");
    printf("Maximum value that can be stored: %d\n", maxValue);
    printf("---------------------------------------\n");
    return 0;
}
```

Summary Table — When Program Exits:

Input Stage	Condition Causing Exit	Message
Number of items (n)	Non-integer, n < 0, n > 50	Invalid number of items. Exiting.
Item weight/value	Non-integer or missing values	Invalid input for item. Exiting.
Warehouse capacity	Non-integer, capacity < 0, capacity > 50	Invalid capacity. Exiting.

Summary Table — When Program Succeeds:

Input Stage	Condition for Success	Description / Outcome
Number of items (n)	Integer, 0 < n ≤ 50	Program accepts and processes the number of items.
Item weight/value	Integer values provided for all items	Program reads weights and values correctly.
Warehouse capacity	Integer, 0 < capacity ≤ 50	Program accepts capacity and runs DP algorithm.
Overall constraints	Individual item weights ≤ capacity (optional, DP handles exceeding weights)	Program calculates maximum value that can be stored without exceeding capacity.
Computation	DP table successfully built	Output displays maximum value achievable.

Output Screenshots:

1. **Testcase A :**

```
Enter number of items: 4
Enter weight and value of item 1: 10 60
Enter weight and value of item 2: 20 100
Enter weight and value of item 3: 30 120
Enter weight and value of item 4: 25 75
Enter capacity of warehouse: 50

--- Warehouse Packing Optimization ---
Maximum value that can be stored: 175
--------------------------------------


--------------------------------
Process exited after 34.51 seconds with return value 0
Press any key to continue . . .
```

2. **Testcase B :**

```
Enter number of items: -3
Invalid number of items. Exiting.

--------------------------------
Process exited after 5.657 seconds with return value 1
Press any key to continue . . .
```

GitHub Repository Link of Project: https://github.com/Jayz-yuors/AOA-Project.git