Design and Analysis of Algorithms

LAB EXPERIMENT-10

Name: RIYA KUMARI

SAP ID:590016221

Batch:34

Course: B.Tech. CSE (Sem III)

Subject: DAA

Lab Date: 26 September 2025

Submitted to: Virendra Kumar

GitHub Repository Link:

https://github.com/Riyakumari1314/DAA-2nd-year.git

Objective: To implement the 0/1 Knapsack Problem using both the Greedy approach and the Dynamic Programming approach, and to analyze their performance on the same dataset.

Problem Statement: You are given:

- n items, each with a weight (w[i]) and a value (v[i]).
- A knapsack capacity (W).

You must choose a subset of items such that:
The total weight ≤ W, and the total value is maximized.

In the 0/1 Knapsack, you can either take an item completely (1) or not at all (0) - no fractions allowed.

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

int findMax(int x, int y) {
    return (x > y) ? x : y;
}

void fractionalKnapsack(int itemCount, float wt[], float val[], float capacity) {
    float valPerWt[30], temp;
    float profit = 0, usedWt = 0;
    int i, j;

    for (i = 0; i < itemCount; i++)
        valPerWt[i] = val[i] / wt[i];
    for (i = 0; i < itemCount - 1; i++) {
        for (j = i + 1; j < itemCount; j++) {
            if (valPerWt[i] < valPerWt[j]) {
                temp = valPerWt[i]; valPerWt[i] = valPerWt[j]; valPerWt[j] = temp;
                temp = val[i]; val[i] = val[j]; val[j] = temp;
                temp = wt[i]; wt[i] = wt[j]; wt[j] = temp;
            }
        }
    }

    for (i = 0; i < itemCount; i++) {
        if (usedWt + wt[i] <= capacity) {
            usedWt += wt[i];
```

```
            profit += val[i];
        } else {
            float remaining = capacity - usedWt;
            profit += val[i] * (remaining / wt[i]);
            break;
        }
    }

    printf("Maximum Value (Greedy) = %.2f\n", profit);
}

int zeroOneKnapsack(int n, int wt[], int val[], int cap) {
    int dp[n + 1][cap + 1];
    int i, w;

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= cap; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (wt[i - 1] <= w)
                dp[i][w] = findMax(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][cap];
}

int main() {
    int n, i, cap;
    float wt[30], val[30];
    int wtInt[30], valInt[30];
    clock_t startTime, endTime;
    double duration;

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

    printf("Enter value and weight for each item:\n");
    for (i = 0; i < n; i++) {
        printf("Item %d (Value Weight): ", i + 1);
        scanf("%f %f", &val[i], &wt[i]);
```

```
        valInt[i] = (int)val[i];
        wtInt[i] = (int)wt[i];
    }

    printf("Enter knapsack capacity: ");
    scanf("%d", &cap);
    startTime = clock();
    fractionalKnapsack(n, wt, val, cap);
    endTime = clock();
    duration = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;
    printf("Time Taken (Greedy): %.6f seconds\n", duration);
    startTime = clock();
    int maxProfit = zeroOneKnapsack(n, wtInt, valInt, cap);
    endTime = clock();
    duration = ((double)(endTime - startTime)) / CLOCKS_PER_SEC;

    printf("Maximum Value (DP) = %d\n", maxProfit);
    printf("Time Taken (DP): %.6f seconds\n", duration);

    return 0;
}
```

}

```
PS E:\DAA> cd "e:\DAA\" ; if ($?) { gcc Fractionalknapsack.
c -o Fractionalknapsack } ; if ($?) { .\Fractionalknapsack
}
Enter number of items: 4
Enter value and weight for each item:
Item 1 (Value Weight): 1 2
Item 2 (Value Weight): 4 3
Item 3 (Value Weight): 6 7
Item 4 (Value Weight): 8 9
Enter knapsack capacity: 4
Maximum Value (Greedy) = 4.89
Time Taken (Greedy): 0.005000 seconds
Maximum Value (DP) = 4
Time Taken (DP): 0.000000 seconds
PS E:\DAA>
```