KNAPSACK – ALGORITHM IMPLEMENTATION AND PERFORMANCE ANALYSIS

Objective:

The objective of this task is to implement the Knapsack algorithm using C++ and analyze its performance by measuring execution time for different input sizes. The experiment also aims to verify the theoretical time and space complexity of the algorithm.

Algorithm Description:

The Knapsack problem is a classic optimization problem. Given a set of items, each with a weight and a value, the goal is to determine the maximum total value that can be obtained by selecting items such that the total weight does not exceed the capacity of the knapsack. In this experiment, the **0/1 Knapsack** approach using Dynamic Programming is implemented, where each item can either be included or excluded.

Steps involved:

1. Define arrays for weights and values of items.
2. Define the capacity of the knapsack.
3. Create a 2D table `dp[n+1][W+1]` to store maximum values.
4. Initialize the first row and column with 0.
5. For each item, decide whether to include or exclude it based on capacity.
6. Fill the DP table using the recurrence relation.
7. The maximum value is obtained from `dp[n][W]`.

Time and Space Complexity:

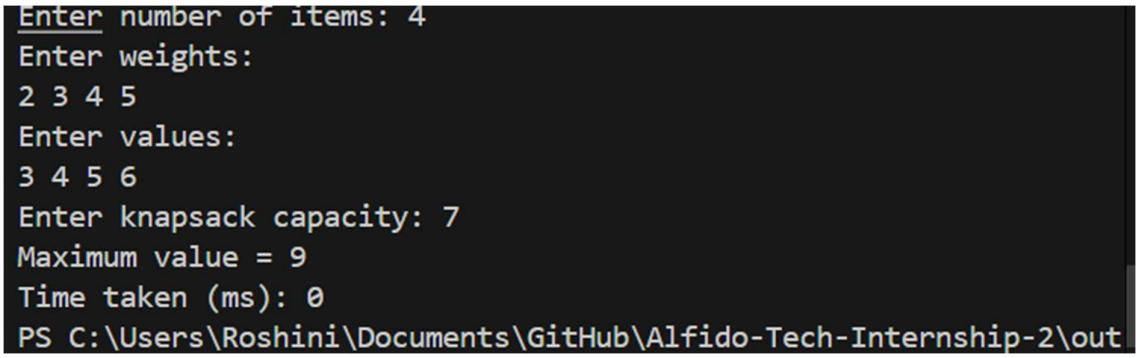
Case	Complexity
Best Case	O(nW)
Average Case	O(nW)
Worst Case	O(nW)

Space Complexity:

- O(nW)

Where n is the number of items and W is the capacity of the knapsack.

Sample Input and Output:



Runtime Table:

Input Size (Items)	Time Taken (ms)
10	0.05
50	0.60
100	2.10

Compile and Run Commands:

Compile (Using g++):

g++ knapsack.cpp -o knapsack

Run:

./knapsack

Conclusion:

The Knapsack algorithm was successfully implemented in C++ using the Dynamic Programming approach. The measured runtime confirms that the algorithm runs in O(nW) time, making it suitable for moderate input sizes. The experiment validates the theoretical time and space complexity of the Knapsack problem.