Name: Sai Gulve

PRN No: 202201040145

Batch: A4

Roll No: 63

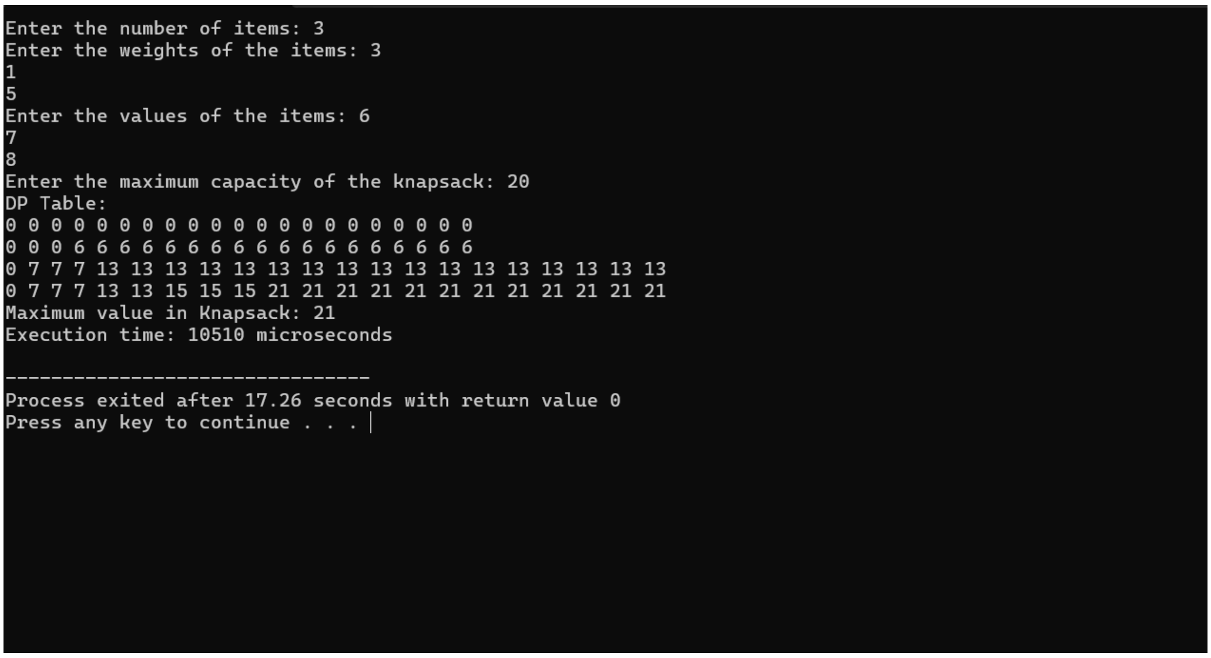
Problem Statement :- 3. Given the weights and profits of N items, in the form of {profit, weight} put these items in a knapsack of capacity W to get the maximum total profit using the Fractional Knapsack with the Greedy method.

Code :-
```
#include <iostream>

#include <chrono>


using namespace std;

using namespace std::chrono;


// Function to solve the 0/1 Knapsack problem

int knapsack(int weights[], int values[], int n, int W) {

   // Create a DP table

   int dp[n + 1][W + 1];


   // Initialize the DP table

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

      for (int w = 0; w <= W; ++w) {

         if (i == 0 || w == 0) {

            dp[i][w] = 0; // Base case: 0 items or 0 capacity

         } else if (weights[i - 1] <= w) {

            dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);

         } else {

            dp[i][w] = dp[i - 1][w];

         }

      }

   }
```

```
    // Display the DP table
    cout << "DP Table:" << endl;
    for (int i = 0; i <= n; ++i) {
        for (int w = 0; w <= W; ++w) {
            cout << dp[i][w] << " ";
        }
        cout << endl;
    }

    return dp[n][W]; // Return the maximum value obtained
}

int main() {
    int n; // Number of items
    cout << "Enter the number of items: ";
    cin >> n;

    int weights[n]; // Array to store weights
    int values[n];  // Array to store values

    // Input weights and values from the user
    cout << "Enter the weights of the items: ";
    for (int i = 0; i < n; ++i) {
        cin >> weights[i];
    }

    cout << "Enter the values of the items: ";
    for (int i = 0; i < n; ++i) {
        cin >> values[i];
    }
```

```
int W; // Maximum capacity of the knapsack
cout << "Enter the maximum capacity of the knapsack: ";
cin >> W;

// Start measuring time
auto start_time = high_resolution_clock::now();

int max_value = knapsack(weights, values, n, W);

// Stop measuring time
auto end_time = high_resolution_clock::now();

// Calculate duration
auto duration = duration_cast<microseconds>(end_time - start_time);

cout << "Maximum value in Knapsack: " << max_value << endl;
cout << "Execution time: " << duration.count() << " microseconds" << endl;

return 0;
```

}



```
Enter the number of items: 3
Enter the weights of the items: 3
1
5
Enter the values of the items: 6
7
8
Enter the maximum capacity of the knapsack: 20
DP Table:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
0 7 7 7 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
0 7 7 7 13 13 15 15 15 21 21 21 21 21 21 21 21 21 21 21 21
Maximum value in Knapsack: 21
Execution time: 10510 microseconds

--------------------------------
Process exited after 17.26 seconds with return value 0
Press any key to continue . . .
```