**Experiment 11**

**Write a program to implement the dynamic algorithm to solve the Zero-one Knapsack problem.**

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <time.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapsack(int W, int wt[], int val[], int n)

{

int i, w;

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

// Build table K[n][w] in bottom up manner

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

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

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

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);

else

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

}

}

return K[n][W];

}

int main()

{

int i, n, val[1000], wt[1000], W;

int randNum1, randNum2 ;

double time;

clock\_t start, end;

printf("Enter number of items:");

scanf("%d", &n);

printf("Enter size of knapsack:");

scanf("%d", &W);

start = clock();

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

{

randNum1 = rand() % 1000;

wt[i] = randNum1;

randNum2 = rand() % 1000;

val[i] = randNum2;

printf("cost :%d \t value:%d \n", wt[i], val[i]);

}

printf("Maximum profit:%d", knapsack(W, wt, val, n));

// end clock

end = clock();

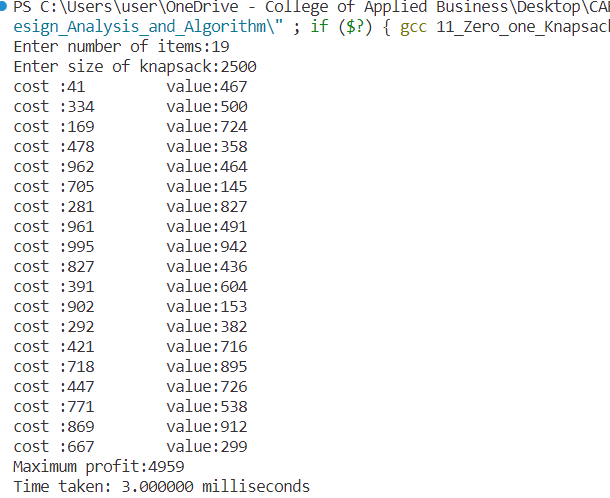
time = ((double)(end - start) \* 1000) / CLOCKS\_PER\_SEC;

printf("\nTime taken: %lf milliseconds\n", time);

return 0;

}

**Output:**



**Conclusion:**

This experiment had been conducted in a 64-bit system with 16 GB RAM and Processor 12th Gen Intel(R) Core (TM) i5-12500H 3.10 GHz. The algorithm was implemented in C programming language in Visual Studio Code 1.85.1 Code Editor. The time taken by this algorithm for 15 number of input size is 1 milliseconds.. The running time is analyzed as O(nW).