**Experiment 16**

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

**Program:-**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int max(int a, int b) { return (a > b)? a : b; }

void knapSack(int W, int wt[], int val[], int n, int curWt, int curVal, int\* maxVal) {

if (curWt > W || n == 0) {

\*maxVal = max(\*maxVal, curVal);

return;

}

knapSack(W, wt, val, n-1, curWt, curVal, maxVal);

knapSack(W, wt, val, n-1, curWt+wt[n-1], curVal+val[n-1], maxVal);

}

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]);

}

int maxVal = 0;

knapSack(W, wt, val, n, 0, 0, &maxVal);

printf("Maximum profit:%d", maxVal);

// end clock

end = clock();

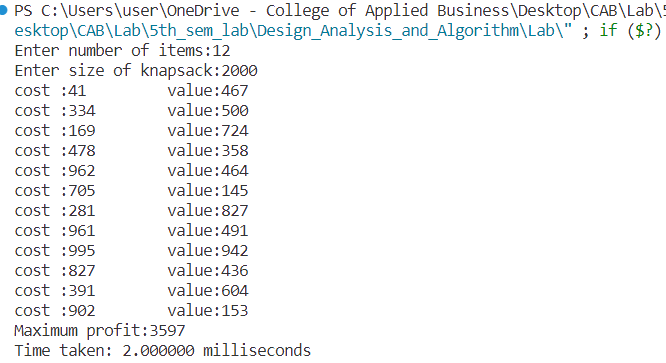
time = ((double)(end - start) \* 1000) / CLOCKS\_PER\_SEC;

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 12 number of input size is 2 milliseconds.. The running time is analyzed as O(2n).