1)PROGRAM:

To find minimum and maximum in a array using divide and conquer approach with its time complexity.

CODE:

#include<stdio.h>

#include<stdio.h>

int max, min;

int a[100];

int c=0;

void maxmin(int i, int j)

{

int max1, min1, mid;

if(i==j)

{

c++;

max = min = a[i];

}

else

{

if(i == j-1)

{

if(a[i] <a[j])

{

c++;

max = a[j];

min = a[i];

}

else

{

c++;

max = a[i];

min = a[j];

}

}

else

{

c++;

mid = (i+j)/2;

maxmin(i, mid);

max1 = max; min1 = min;

maxmin(mid+1, j);

if(max <max1)

max = max1;

if(min > min1)

min = min1;

}

}

}

int main ()

{

int i, num;

printf ("\nEnter the total number of numbers : ");

scanf ("%d",&num);

printf ("Enter the numbers : \n");

for (i=1;i<=num;i++)

scanf ("%d",&a[i]);

max = a[0];

min = a[0];

maxmin(1, num);

printf ("Minimum element in an array : %d\n", min);

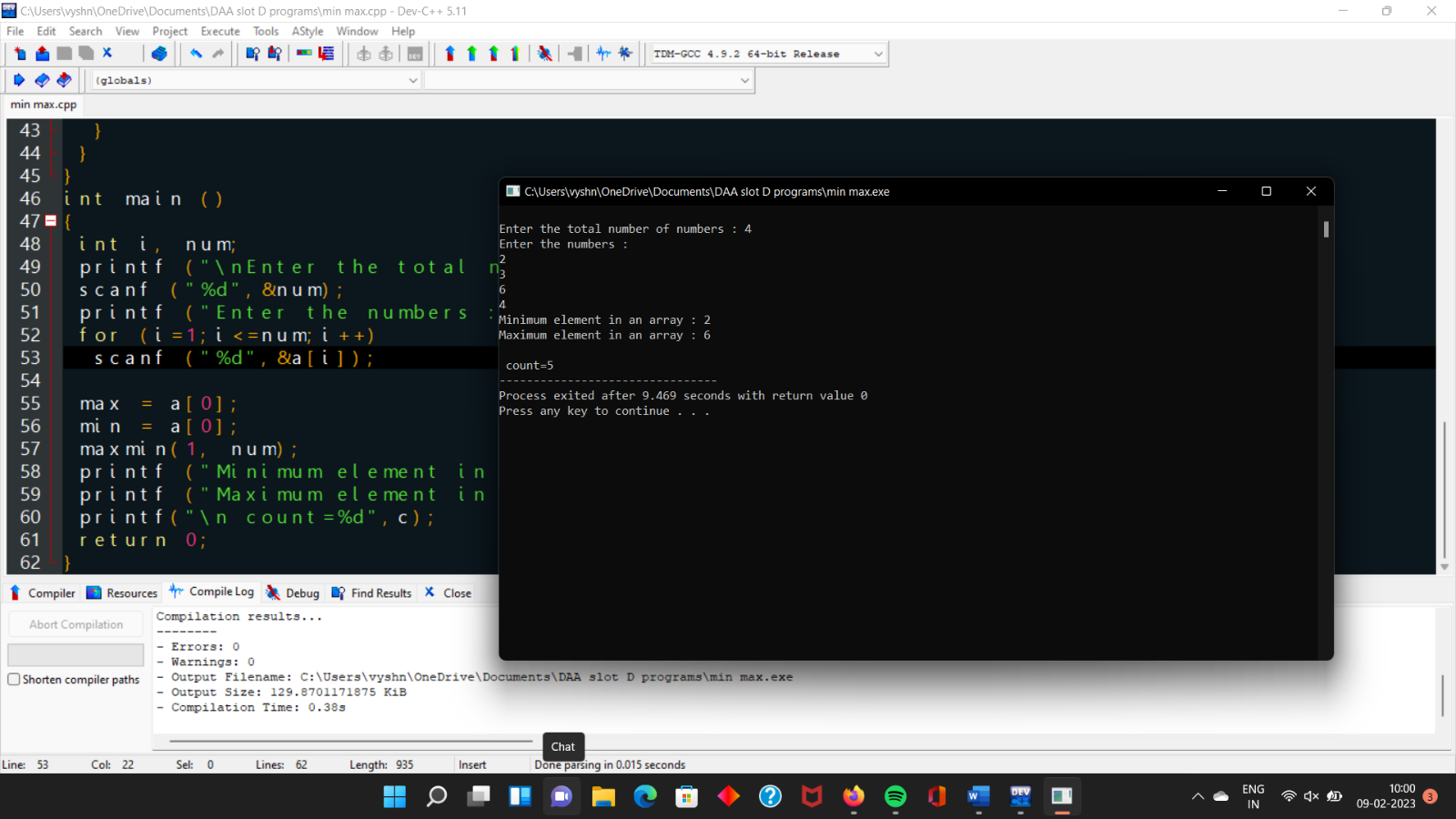
printf ("Maximum element in an array : %d\n", max);

printf("\n count=%d",c);

return 0;

}

OUTPUT:



2)PROGRAM:

Merge sort

CODE:

#include<stdlib.h>

#include<stdio.h>

// Merge Function

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort Function in C

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

// Functions to Print Elements of Array

void printArray(int A[], int size)

{

int i;

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

printf("%d ", A[i]);

printf("\n");

}

//main

int main()

{

int arr[] = {85, 24, 63, 45, 17, 31, 96, 50};

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

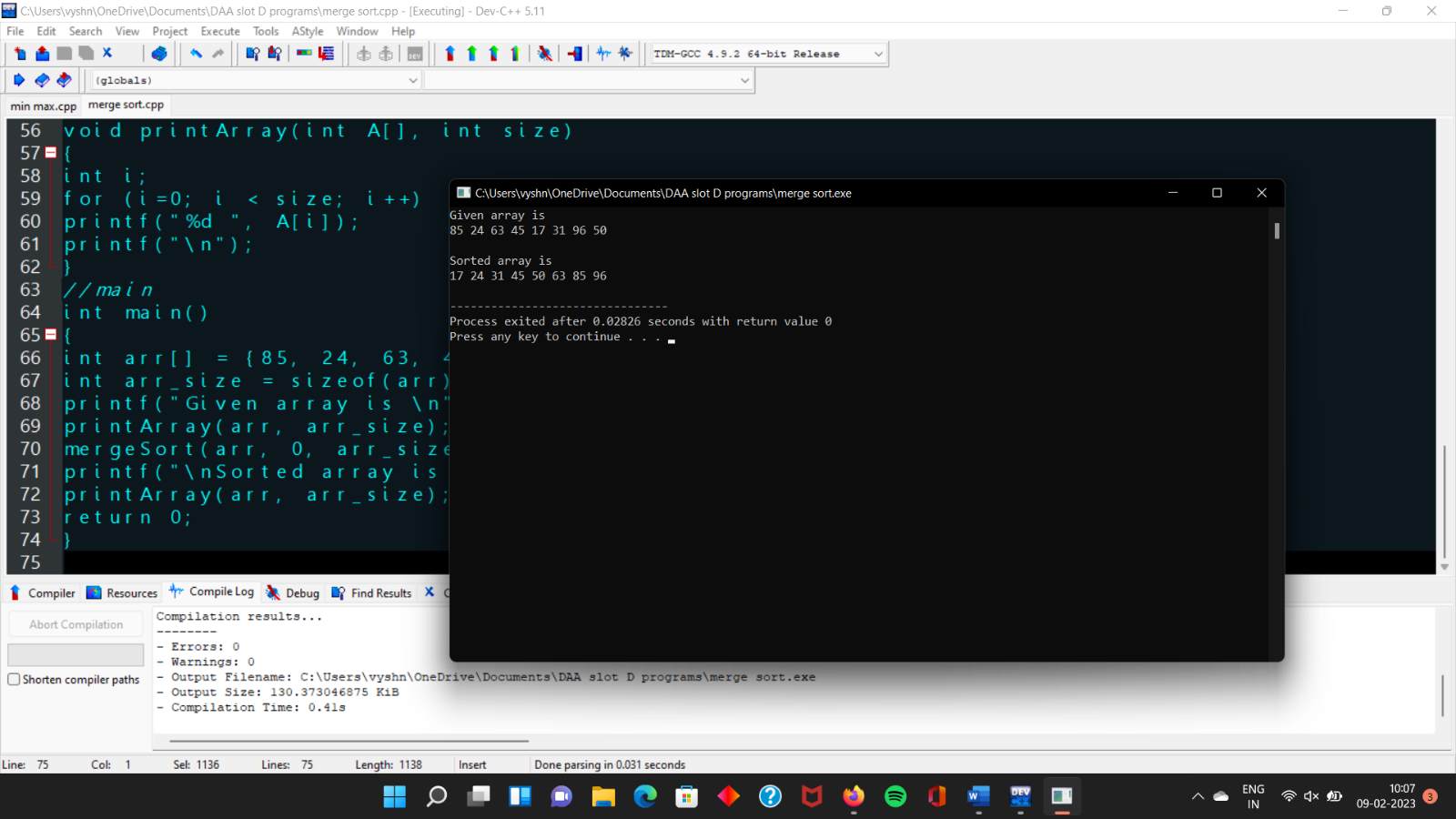
printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0;

}

OUTPUT:



3) PROGRAM:

Write a program to return all the possible subsets for a given integer array. Return the

solution in any order.

Input nums= [1,2,3]

Output : [ [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]]

CODE:

#include <stdio.h>

char string[50], n;

void subset(int, int, int);

int main()

{

int i, len;

printf("Enter the len of main set : ");

scanf("%d", &len);

printf("Enter the elements of main set : ");

scanf("%s", string);

n = len;

printf("The subsets are :\n");

for (i = 1;i <= n;i++)

subset(0, 0, i);

}

void subset(int start, int index, int num\_sub)

{

int i, j;

if (index - start + 1 == num\_sub)

{

if (num\_sub == 1)

{

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

printf("%c\n", string[i]);

}

else

{

for (j = index;j < n;j++)

{

for (i = start;i < index;i++)

printf("%c", string[i]);

printf("%c\n", string[j]);

}

if (start != n - num\_sub)

subset(start + 1, start + 1, num\_sub);

}

}

else

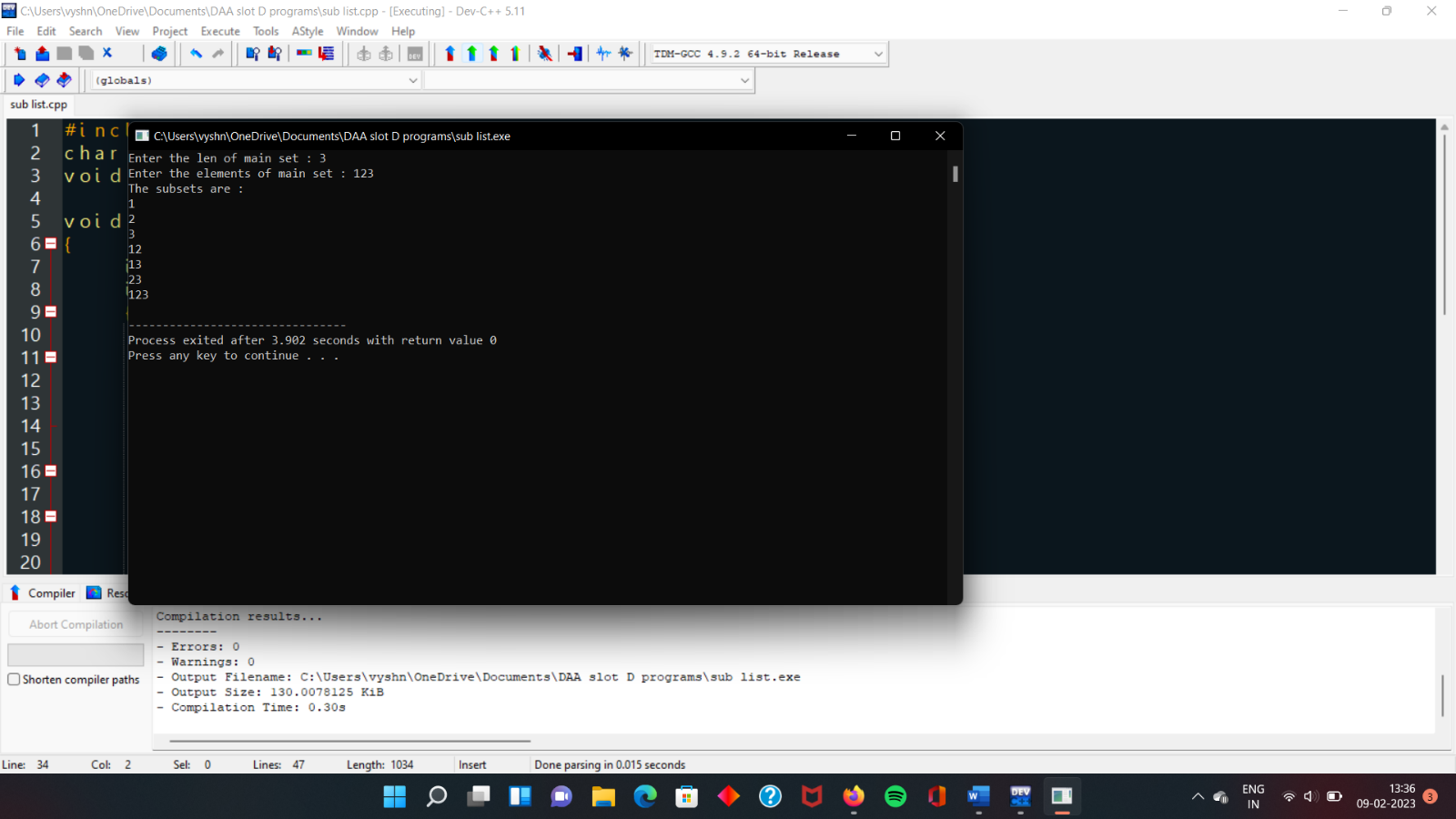
{

subset(start, index + 1, num\_sub);

}

}

OUTPUT:



4)PROGRAM:

Write a program to find a minimum spanning tree using prims technique for the

given graph

CODE:

#include <stdio.h>

#include <limits.h>

#define V 5

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, minIndex;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], minIndex = v;

return minIndex;

}

void printMST(int parent[], int graph[V][V]) {

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++)

printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool mstSet[V];

for (int i = 0; i < V; i++)

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, graph);

}

int main() {

int graph[V][V] = {{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

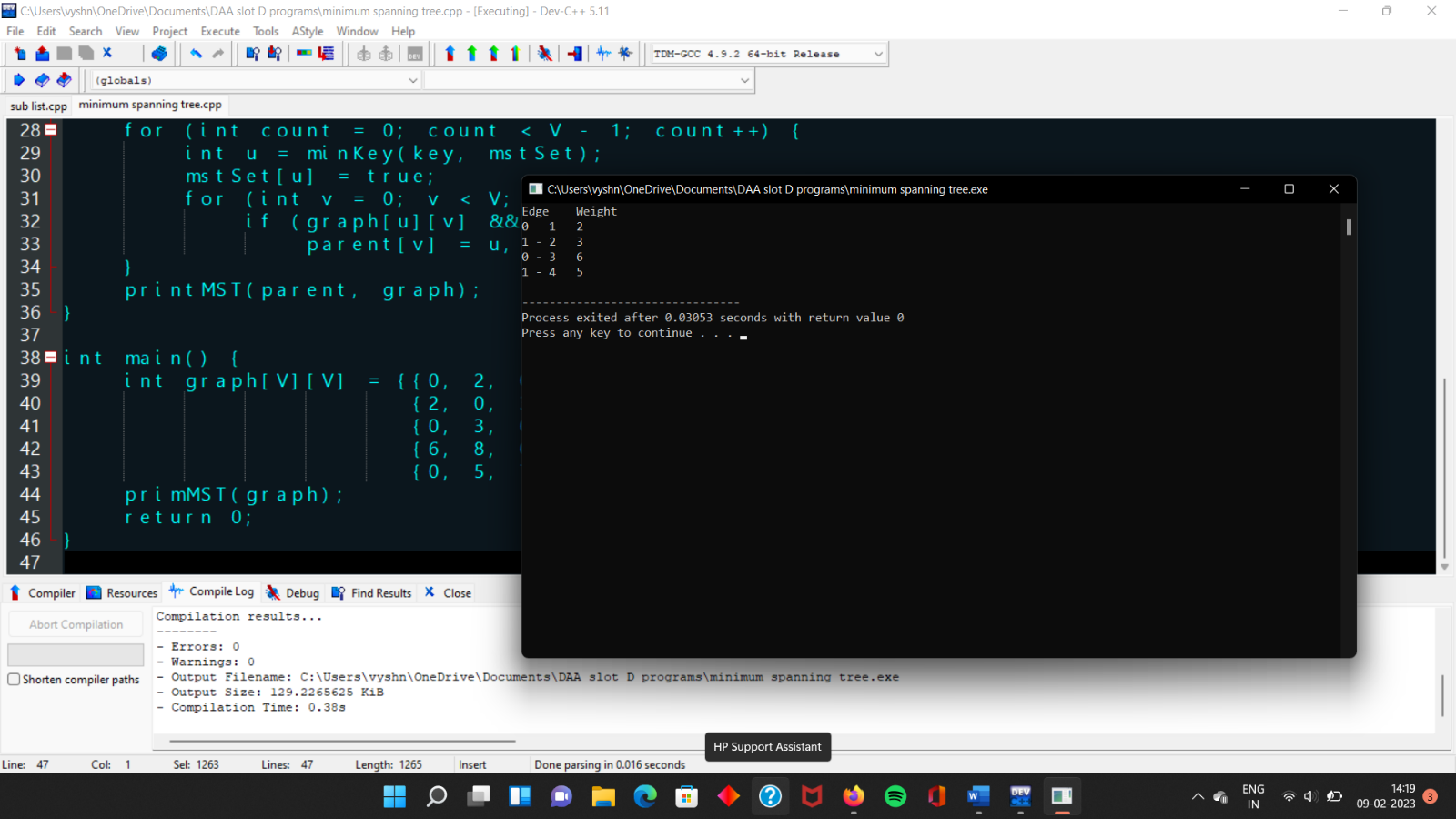
{0, 5, 7, 9, 0}};

primMST(graph);

return 0;

}

OUTPUT:



5) PROGRAM:

Write a program to compute container loader Problem for the given values and estimate time complexity.

CODE:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_ITEMS 100

#define MAX\_WEIGHT 100

int weight[MAX\_ITEMS];

int value[MAX\_ITEMS];

int dp[MAX\_ITEMS][MAX\_WEIGHT];

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapsack(int n, int w) {

int i, j;

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

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

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

dp[i][j] = 0;

} else if (weight[i-1] <= j) {

dp[i][j] = max(value[i-1] + dp[i-1][j-weight[i-1]], dp[i-1][j]);

} else {

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

}

}

}

return dp[n][w];

}

int main() {

int n = 4;

int w = 10;

weight[0] = 1;

weight[1] = 2;

weight[2] = 4;

weight[3] = 5;

value[0] = 5;

value[1] = 4;

value[2] = 6;

value[3] = 8;

int result = knapsack(n, w);

printf("Result: %d\n", result);

return 0;

}

OUTPUT:

