**1.BACKTRACKING:**

**PROGRAM:**

#include <stdio.h>

#define TRUE 1

#define FALSE 0

int inc[50],w[50],sum,n,count=0;

void sumset(int i,int wt,int total);

int promising(int i,int wt,int total) {

return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}

void main() {

int i,j,n,temp,total=0;

printf("\n Enter how many numbers:\n");

scanf("%d",&n);

printf("\n Enter %d numbers to th set:\n",n);

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

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

total+=w[i];

}

count++;

printf("\n Input the sum value to create sub set:\n");

scanf("%d",&sum);

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

for (j=0;j<n-1;j++)

if(w[j]>w[j+1]) {

temp=w[j];

w[j]=w[j+1];

w[j+1]=temp;

}

count++;

printf("\n The given %d numbers in ascending order:\n",n);

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

printf("%d \t",w[i]);

count++;

if((total<sum))

printf("\n Subset construction is not possible"); else {

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

inc[i]=0;

printf("\n The solution using backtracking is:\n");

sumset(-1,0,total);

}

count++;

}

void sumset(int i,int wt,int total) {

int j;

if(promising(i,wt,total)) {

if(wt==sum) {

printf("\n{\t");

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

if(inc[j])

printf("%d\t",w[j]);

printf("}\n");

} else {

inc[i+1]=TRUE;

sumset(i+1,wt+w[i+1],total-w[i+1]);

inc[i+1]=FALSE;

sumset(i+1,wt,total-w[i+1]);

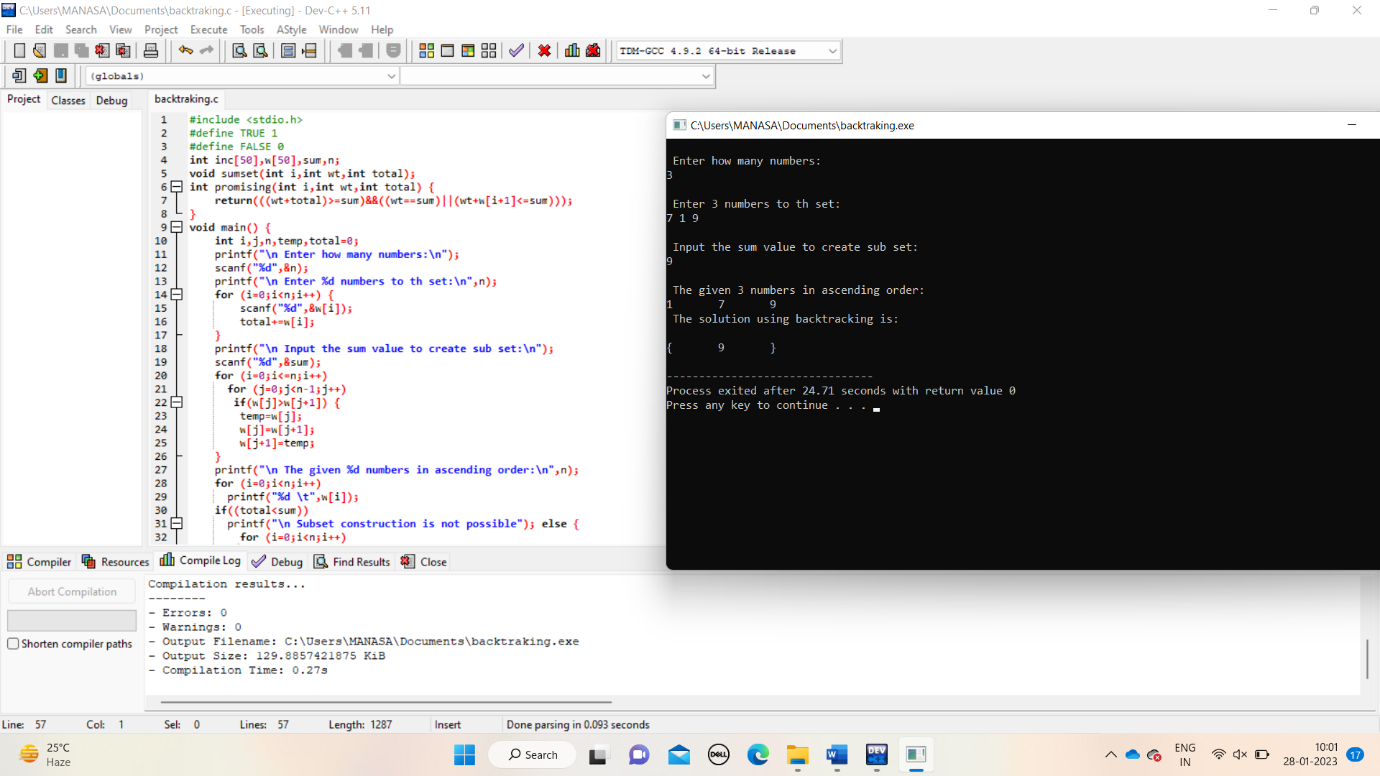
}

}

printf(" time complexity is%d",count);

}

**Output:**

****

2.sttrens matrix.

PROGRAM:

#include<stdio.h>

int main(){

int a[2][2], b[2][2], c[2][2], i, j,count=0;

int m1, m2, m3, m4 , m5, m6, m7;

printf("Enter the 4 elements of first matrix:");

count++;

for(i = 0;i < 2; i++)

{

count++;

for(j = 0;j < 2; j++)

{

count++;

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

}

}

count++;

count++;

printf("Enter the 4 elements of second matrix: ");

for(i = 0; i < 2; i++)

{

count++;

for(j = 0;j < 2; j++)

{

count++;

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

}

}

count++;

count++;

printf("\nThe first matrix is\n");

for(i = 0; i < 2; i++){

count++;

printf("\n");

for(j = 0; j < 2; j++){

count++;

printf("%d\t", a[i][j]);

}

}

count++;

count++;

printf("\nThe second matrix is\n");

for(i = 0;i < 2; i++){

count++;

printf("\n");

for(j = 0;j < 2; j++){

count++;

printf("%d\t", b[i][j]);

}

}

count++;

count++;

m1= (a[0][0] + a[1][1]) \* (b[0][0] + b[1][1]);

count++;

m2= (a[1][0] + a[1][1]) \* b[0][0];

count++;

m3= a[0][0] \* (b[0][1] - b[1][1]);

count++;

m4= a[1][1] \* (b[1][0] - b[0][0]);

count++;

m5= (a[0][0] + a[0][1]) \* b[1][1];

count++;

m6= (a[1][0] - a[0][0]) \* (b[0][0]+b[0][1]);

count++;

m7= (a[0][1] - a[1][1]) \* (b[1][0]+b[1][1]);

count++;

c[0][0] = m1 + m4- m5 + m7;

count++;

c[0][1] = m3 + m5;

count++;

c[1][0] = m2 + m4;

count++;

c[1][1] = m1 - m2 + m3 + m6;

count++;

printf("\nAfter multiplication using Strassen's algorithm \n");

for(i = 0; i < 2 ; i++){

count++;

printf("\n");

for(j = 0;j < 2; j++){

count++;

printf("%d\t", c[i][j]);

}

}

count++;

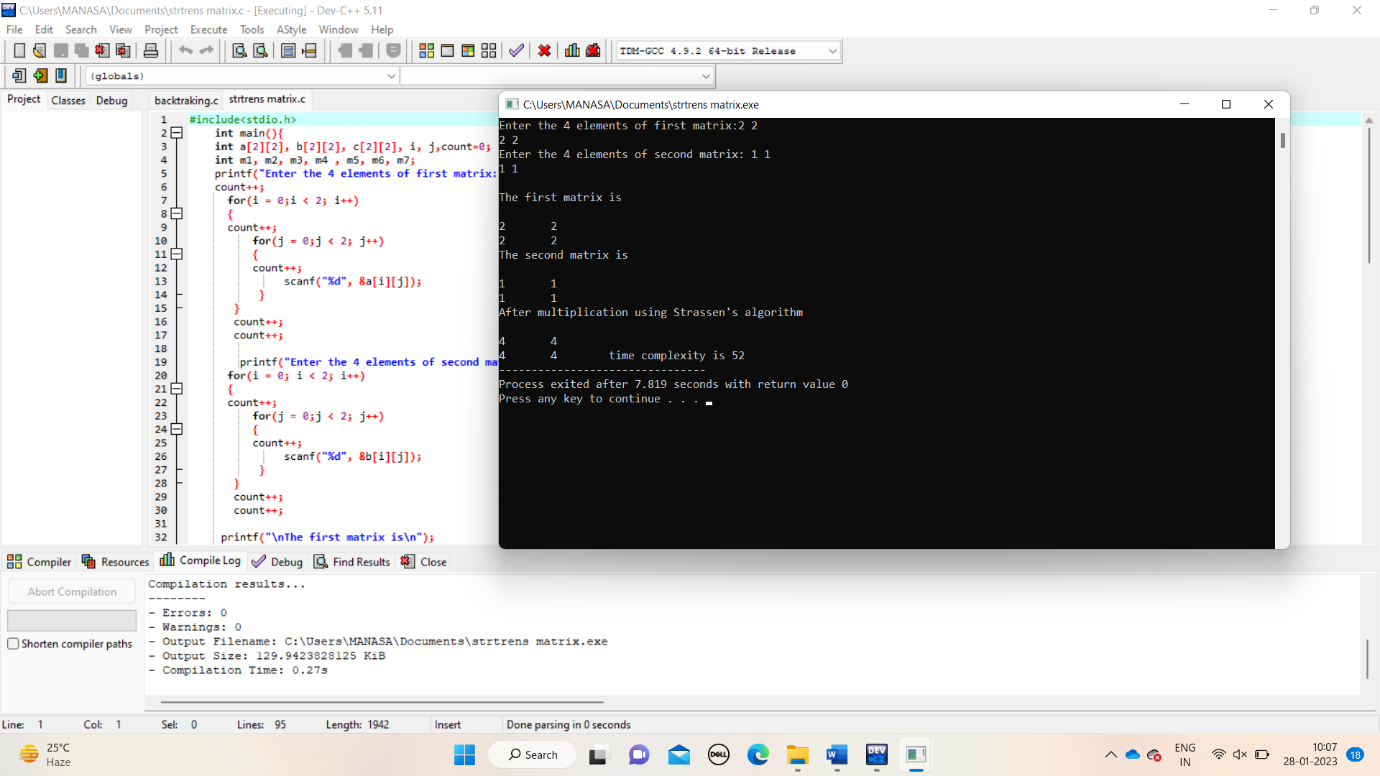
count++;

printf(" time complexity is %d",count);

return 0;

}

Output:



3.min\_max:

Program:

#include<stdio.h>

#include<stdio.h>

int max, min;

int a[100];

int count=0;

void maxmin(int i, int j)

{

int max1, min1, mid;

if(i==j)

{

count++;

max = min = a[i];

count++;

}

else

{

if(i == j-1)

{

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

{

max = a[j];

count++;

min = a[i];

count++;

}

else

{

max = a[i];

count++;

min = a[j];

count++;

}

}

else

{

mid = (i+j)/2;

count++;

maxmin(i, mid);

count++;

max1 = max; min1 = min;

count++;

maxmin(mid+1, j);

count++;

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);

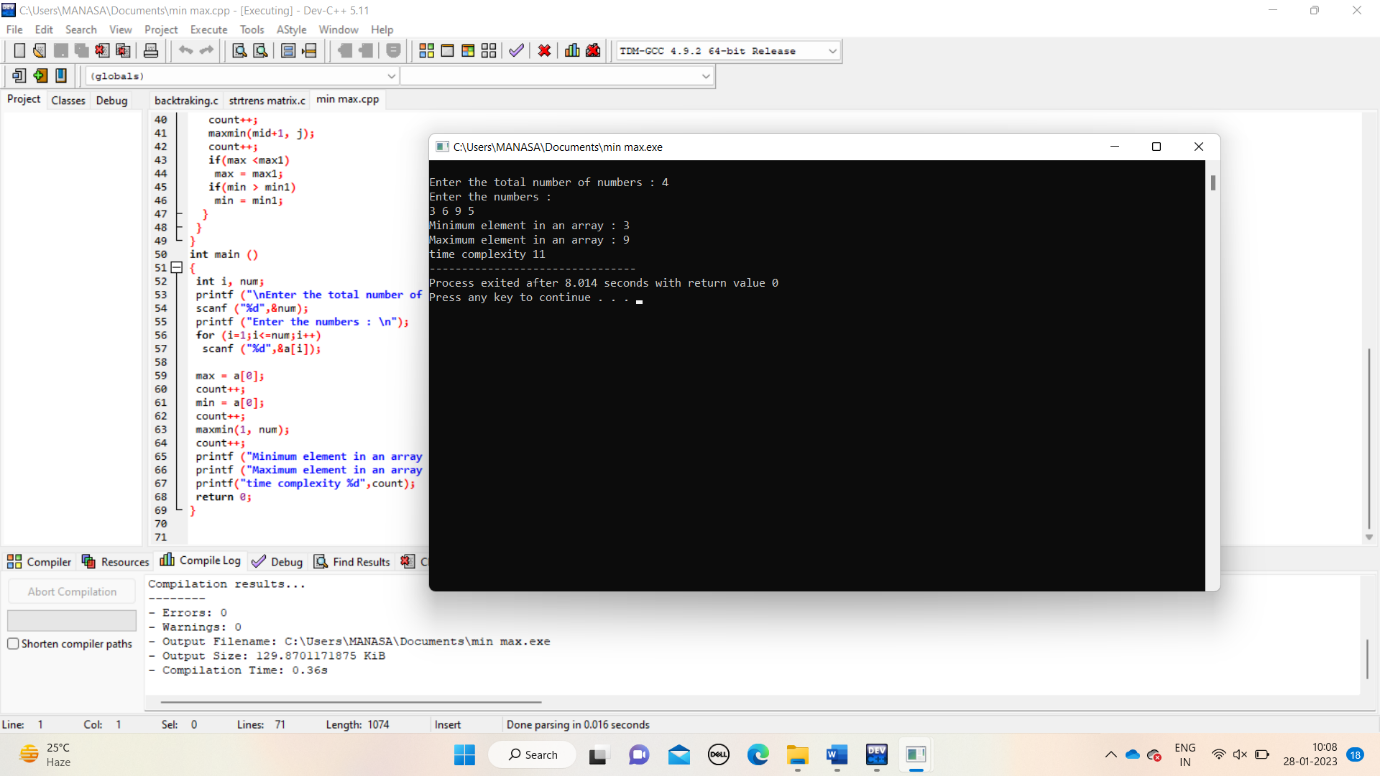
printf ("Maximum element in an array : %d\n", max);

printf("time complexity:%d",count);

return 0;

}

Output:



4.MERGE SORT:

PROGRAM:

#include<stdlib.h>

#include<stdio.h>

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

{

int count=0;

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];

count++;

i = 0;

count++;

j = 0;

count++;

k = l;

count++;

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++;

}

}

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);

}

}

void printArray(int A[], int size)

{

int i;

printf("\n");

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

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

}

int main()

{ int count=0;

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

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

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

printArray(arr, arr\_size);

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

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

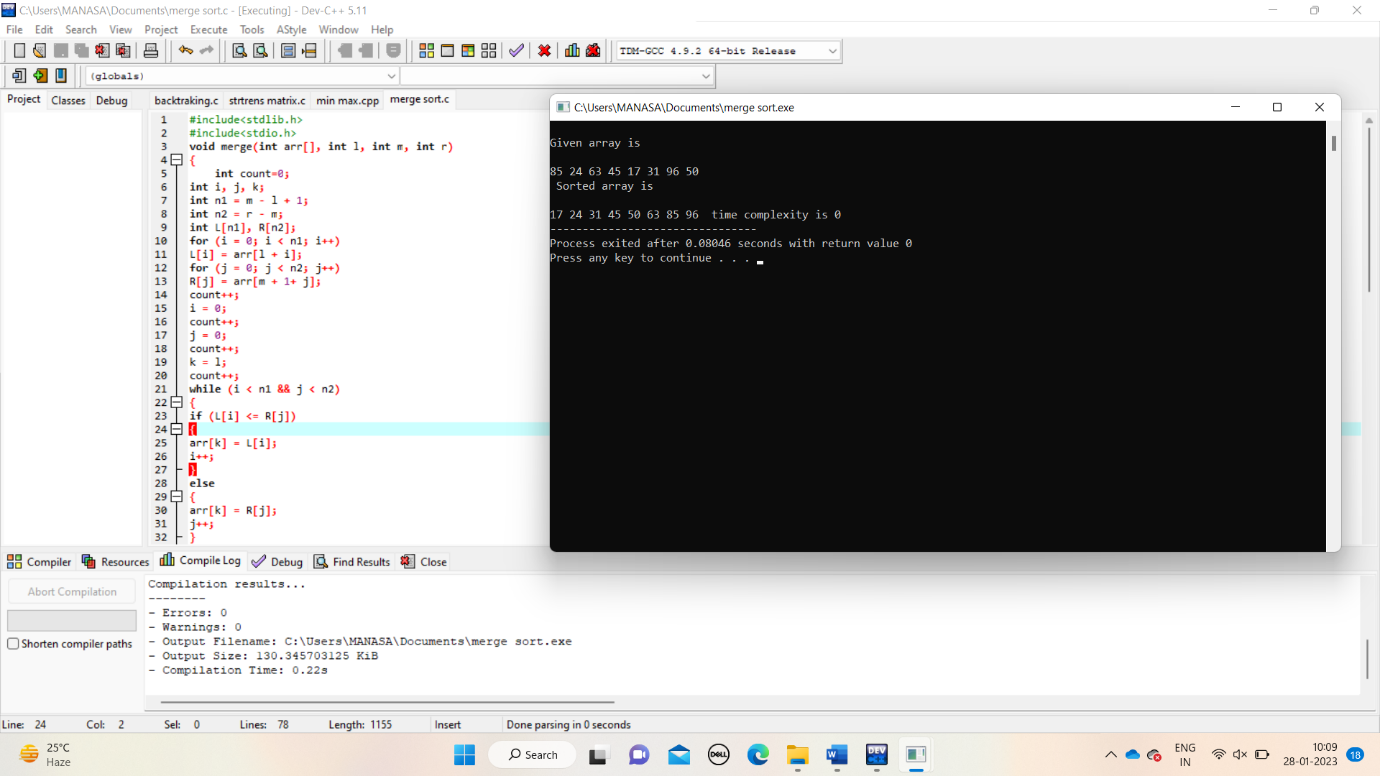
printArray(arr, arr\_size);

printf(" time complexity is %d",count);

return 0;

}

OUTPUT:



5.BINARY SEARCH:

PROGRAM:

#include <stdio.h>

int main()

{

int i, low, high, mid, n, key, array[100];

int count=0;

printf("Enter number of elements");

scanf("%d",&n);

printf("Enter %d integers", n);

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

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

printf("Enter value to find");

scanf("%d", &key);

low = 0;

count++;

high = n - 1;

count++;

mid = (low+high)/2;

count++;

while (low <= high) {

count++;

if(array[mid] < key)

low = mid + 1;

else if (array[mid] == key) {

count++;

printf("%d found at location %d ", key, mid+1);

break;

}

else

high = mid - 1;

mid = (low + high)/2;

count++;

}

if(low > high)

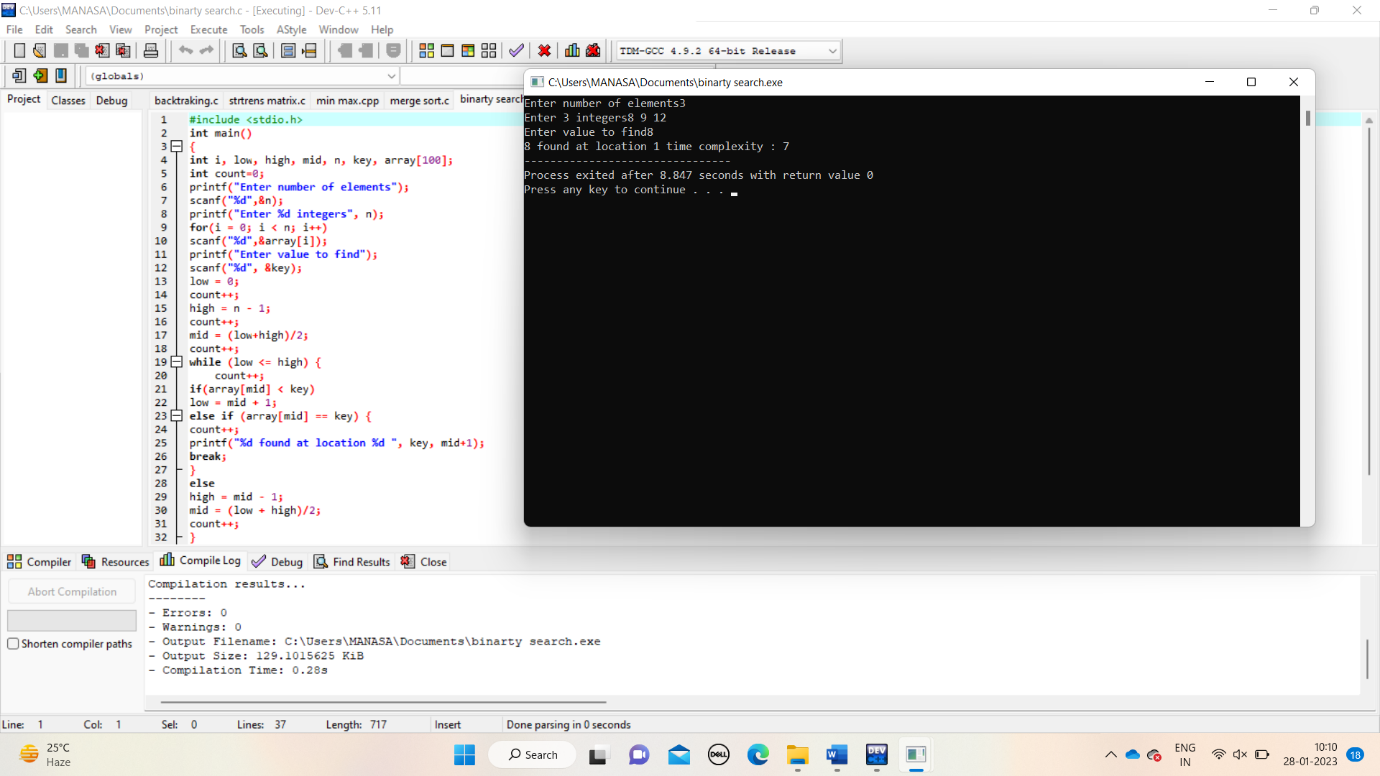
printf("Not found! %d isn't present in the list.n", key);

printf("time complexity : %d",count);

return 0;

}

OUTPUT:



6.PRIMES ALORITHM:

PROGRAM:

#include <stdio.h>

#include <limits.h>

#define vertices 5

int minimum\_key(int k[], int mst[])

{

int minimum = INT\_MAX, min,i,count=0;

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

if (mst[i] == 0 && k[i] < minimum )

minimum = k[i], min = i;

return min;

count++;

}

void prim(int g[vertices][vertices])

{

int parent[vertices];

int k[vertices];

int mst[vertices];

int i, count,edge,v;

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

{

k[i] = INT\_MAX;

count++;

mst[i] = 0;

count++;

}

count++;

k[0] = 0;

count++;

parent[0] = -1;

count++;

for (count = 0; count < vertices-1; count++)

{

edge = minimum\_key(k, mst);

mst[edge] = 1;

for (v = 0; v < vertices; v++)

{

if (g[edge][v] && mst[v] == 0 && g[edge][v] < k[v])

{

parent[v] = edge, k[v] = g[edge][v];

}

}

}

count++;

count++;

count++;

printf("\n Edge \t Weight\n");

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

printf(" %d <-> %d %d \n", parent[i], i, g[i][parent[i]]);

count++;

printf(" time complexity is :%d",count);

}

int main()

{

int g[vertices][vertices] = {{0, 0, 3, 0, 0},

{0, 0, 10, 4, 0},

{3, 10, 0, 2, 6},

{0, 4, 2, 0, 1},

{0, 0, 6, 1, 0},

};

prim(g);

return 0;

}

OUTPUT:

