**CSA0683 -DESIGN AND ANALYSIS OF ALGORITHMS**

**(LAB PRACTICAL DAY-2)**

AJAY TANGUTURI

192125078

1. Write a program to perform MST using greedy techniques.

**Input:**

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]= {

0

}

,min,mincost=0,cost[10][10];

int main() {

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:\n");

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

for (j=1;j<=n;j++) {

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

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne<n) {

for (i=1,min=999;i<=n;i++)

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

if(cost[i][j]<min)

if(visited[i]!=0) {

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0) {

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

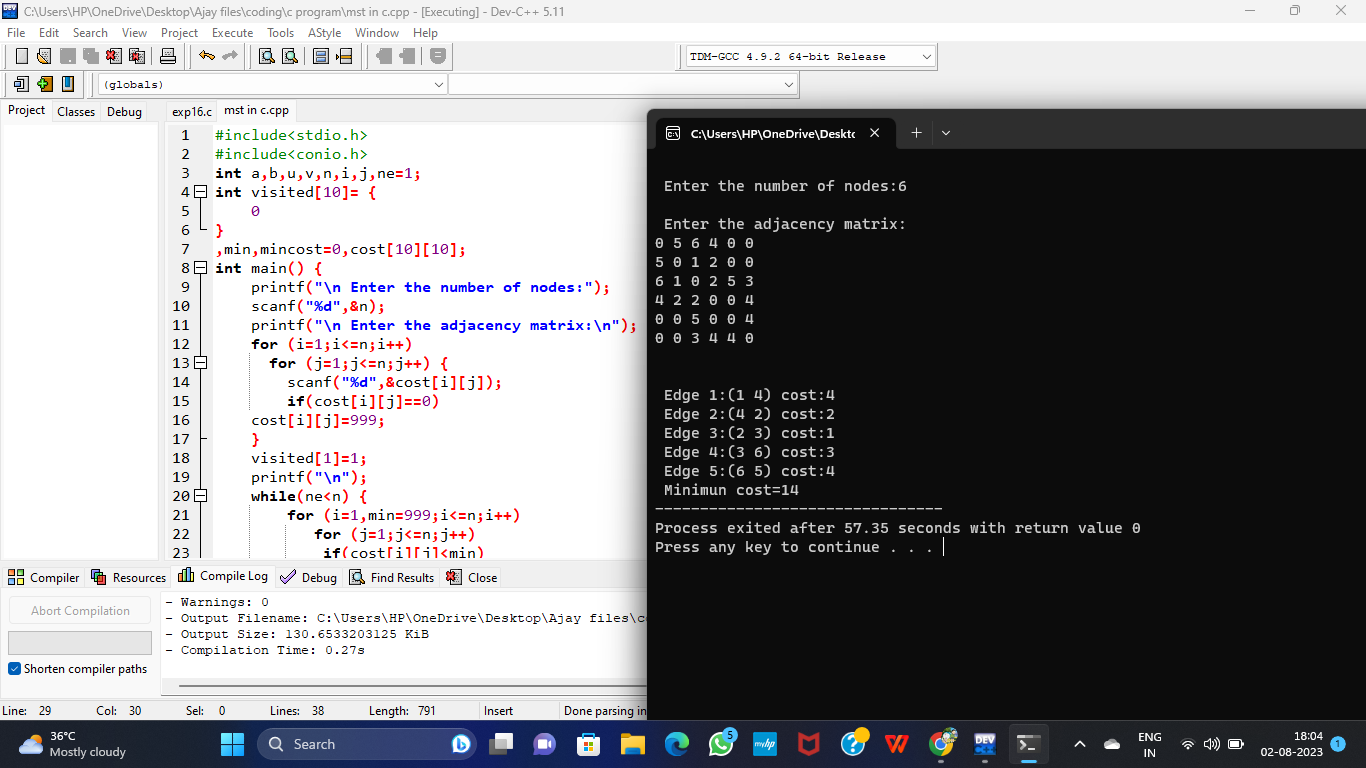
}

printf("\n Minimun cost=%d",mincost);

return 0;

}

**Output:**



1. Write a program to perform Knapsack problem using greedy techniques

**Input:**

# include<stdio.h>

void knapsack(int n, float weight[], float profit[], float capacity)

{

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if (i < n)

x[i] = u / weight[i];

tp = tp + (x[i] \* profit[i]);

printf("\nThe result vector is:");

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

printf("%f\t", x[i]);

printf("\nMaximum profit is: %f", tp);

}

int main() {

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

printf("\nEnter the no. of objects: ");

scanf("%d", &num);

printf("\nEnter the wts and profits of each object: ");

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

scanf("%f %f", &weight[i], &profit[i]);

}

printf("\nEnter the capacityacity of knapsack: ");

scanf("%f", &capacity);

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

{

ratio[i] = profit[i] / weight[i];

}

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

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

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

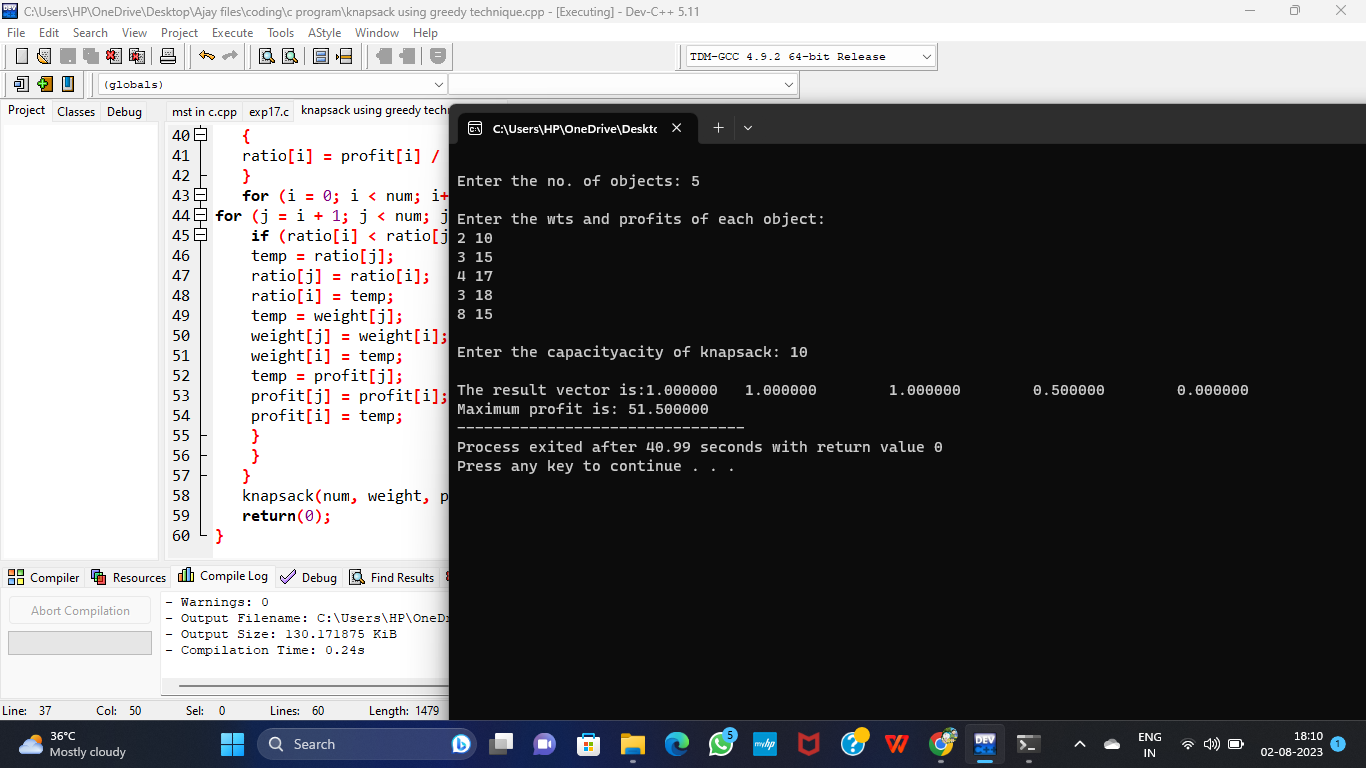
}

knapsack(num, weight, profit, capacity);

return(0);

}

**Output:**

****

1. Using Dynamic programming concept to find out Optimal binary search tree.

**Input:**

#include<stdio.h>

#include<limits.h>

double sumProbabilities(double probabilities[], int start, int end) {

double sum = 0.0;

for (int i=start;i<=end;i++)

sum += probabilities[i];

return sum;

}

double optimalBSTCost(double keys[], double probabilities[], int n) {

double dp[n][n];

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

dp[i][i] = probabilities[i];

for (int L = 2; L <= n; L++) {

for (int i = 0; i <= n - L + 1; i++) {

int j = i + L - 1;

dp[i][j] = INT\_MAX;

for (int r = i; r <= j; r++) {

double cost = ((r > i) ? dp[i][r - 1] : 0) + ((r < j) ? dp[r + 1][j] : 0) + sumProbabilities(probabilities, i, j);

if (cost < dp[i][j])

dp[i][j] = cost;

}

}

}

return dp[0][n - 1];

}

int main()

{

int n;

printf("Enter the number of keys: ");

scanf("%d", &n);

double keys[n], probabilities[n];

printf("Enter the keys and their probabilities:\n");

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

scanf("%lf %lf", &keys[i], &probabilities[i]);

}

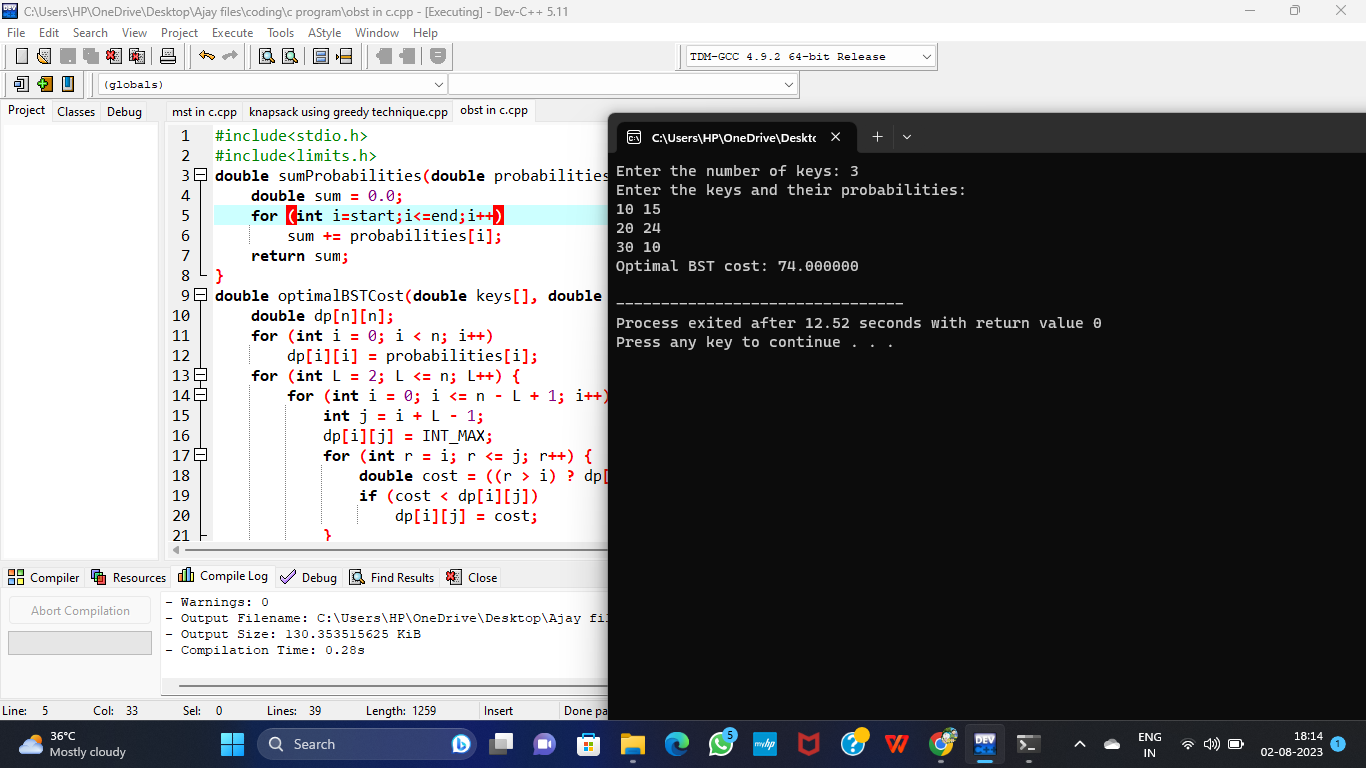
double cost = optimalBSTCost(keys, probabilities, n);

printf("Optimal BST cost: %lf\n", cost);

return 0;

}

**Output:**



1. Using Dynamic programming techniques to find binomial coefficient of a given number

**Input:**

#include<stdio.h>

int main()

{

int i,n=1,r=1,k=1,n1,n2,n3,binomial;

printf("\nEnter the value of n: ");

scanf("%d",&n1);

printf("\nEnter the value of r:");

scanf("%d",&n2);

n3=n1-n2;

for(i=1;i<=n1;i++)

{

n=n\*i;

}

for(i=1;i<=n2;i++)

{

r=r\*i;

}

for(i=1;i<=n3;i++)

{

k=k\*i;

}

binomial = n/(r\*k);

if(n1<0)

{

printf("\nInvalid.enter correct value.\n");

}

else if(n2<0)

{

printf("\nInvalid.enter correct value.\n");

}

else

{

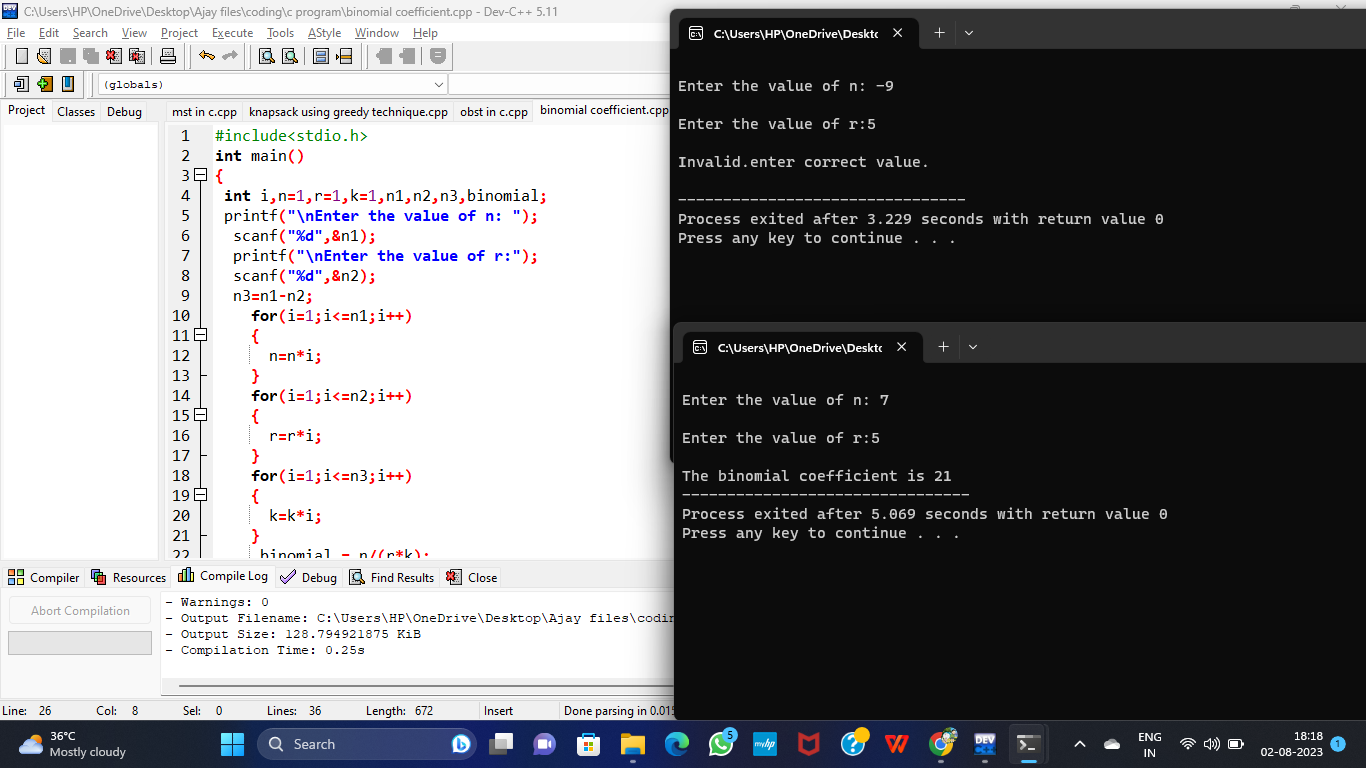
printf("\nThe binomial coefficient is %d",binomial);

}

return 0;

}

**Output:**



1. Write a program to perform Merge Sort

**Input:**

#include <stdio.h>

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[n1], M[n2];

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

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

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

M[j] = arr[q + 1 + j];

int i, j, k;

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

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

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = M[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[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 arr[], int size) {

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

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

printf("\n");

}

int main()

{

int n,c;

printf("Enter number of elements\n");

scanf("%d", &n);

int arr[n];

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

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

scanf("%d",&arr[c]);

int size = sizeof(arr) / sizeof(arr[0]);

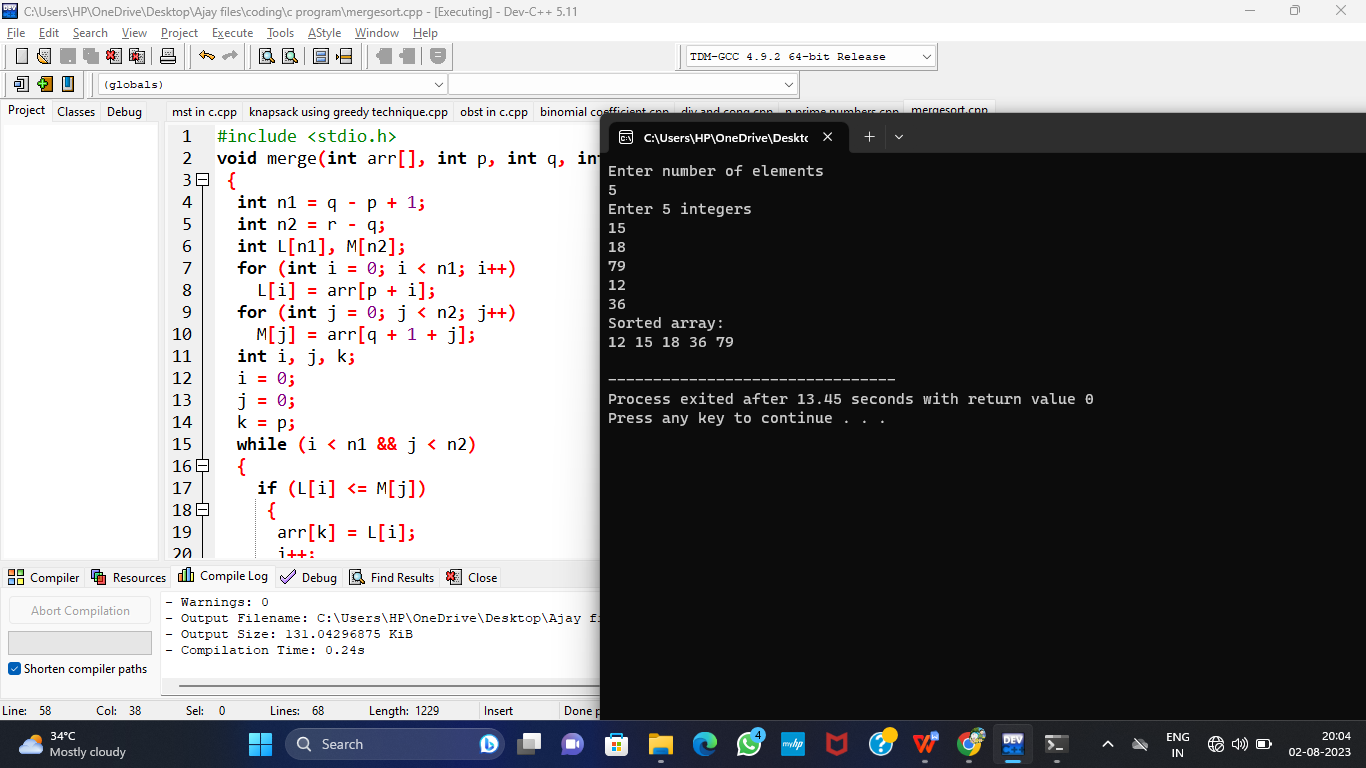
mergeSort(arr, 0, size - 1);

printf("Sorted array: \n");

printArray(arr, size);

}

**Output:**



21)Using Divide and Conquer strategy to find Max and Min value in the list.

**Input:**

#include<stdio.h>

int max, min;

int a[100];

void maxmin(int i, int j)

{

int max1, min1, mid;

if(i==j)

{

max = min = a[i];

}

else

{

if(i == j-1)

{

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

{

max = a[j];

min = a[i];

}

else

{

max = a[i];

min = a[j];

}

}

else

{

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);

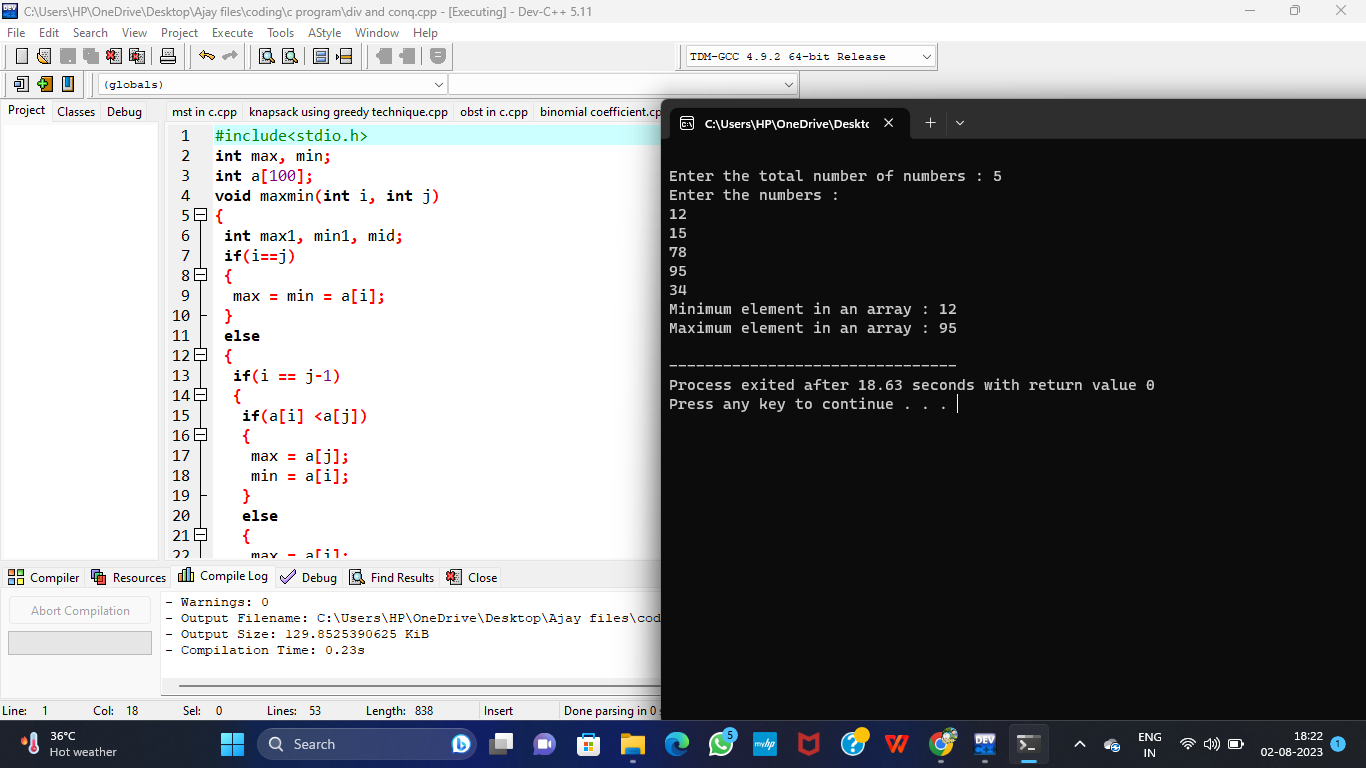
printf ("Minimum element in an array : %d\n", min);

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

return 0;

}

**Output:**

****

1. Write a program to generate all the prime numbers.

**Input:**

#include<stdio.h>

int main()

{

int num,i,count,n;

printf("Enter max range: ");

scanf("%d",&n);

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

{

count = 0;

for(i=2;i<=num/2;i++){

if(num%i==0){

count++;

break;

}

}

if(count==0 && num!= 1)

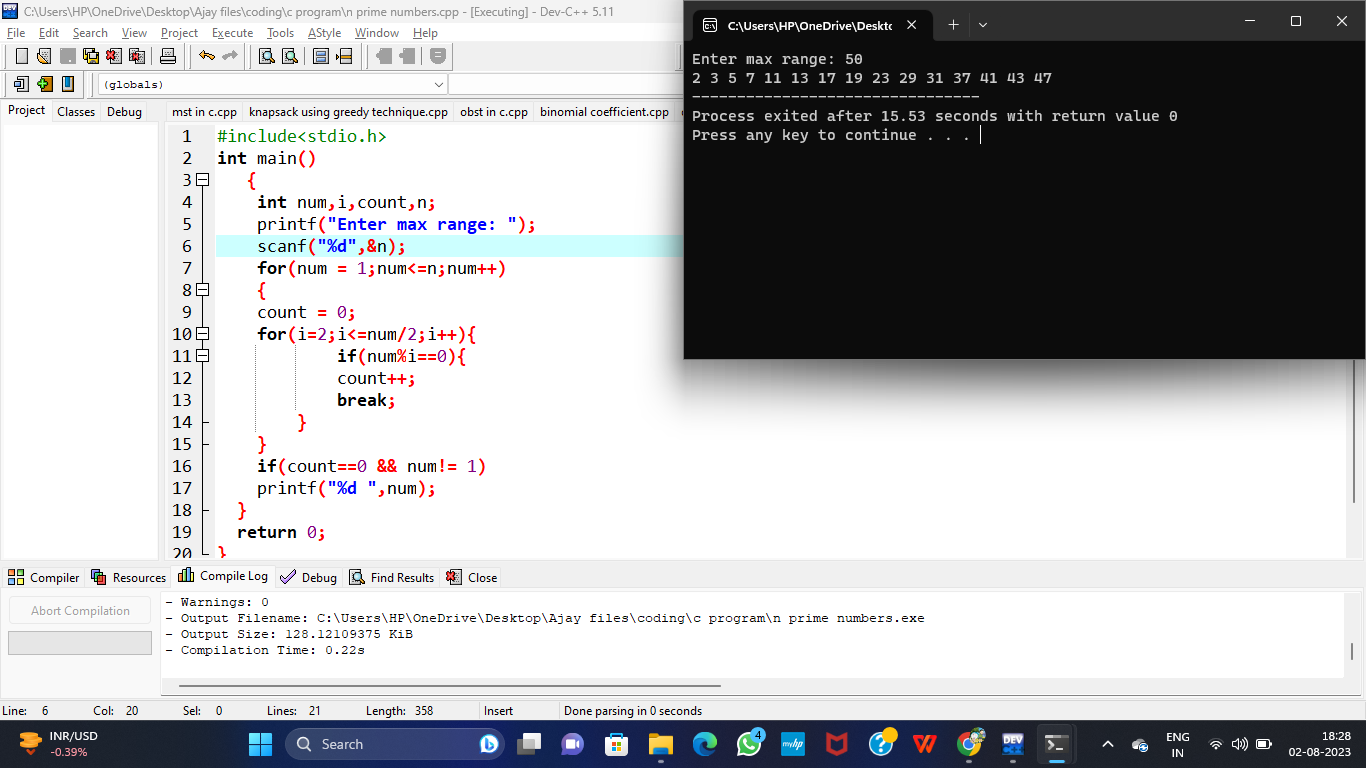
printf("%d ",num);

}

return 0;

}

**Output:**

****