**B. M. S. College of Engineering**

**Bull Temple Road, Bangalore-560019**

(Affiliated to Visvesvaraya Technological

University, Belgaum)



**DEPARTMENT:** COMPUTER SCIENCE AND ENGINEERING

**PROGRAM:** UG

**NAME:** RAMYA RAMESH

**USN:** 1BM19CS227

**SECTION:** 4-‘D’

**COURSE NAME:** Analysis and Design of Algorithms

**COURSE CODE:** 19CS4PCADA

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 1: TOWER OF HANOI AND GCD OF TWO NUMBERS**

TOWER OF HANOI:

#include <stdio.h>

#include <time.h>

void towers(int,char,char,char);

int main()

{

clock\_t start,end;

double time;

int num;

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

scanf("%d",&num);

printf("The sequence of moves involved in Tower of Hanoi are: \n");

start = clock();

towers(num,'A','C','B');

end = clock();

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

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

return 0;

}

void towers(int num,char frompeg,char topeg,char auxpeg)

{

if (num==1){

printf("\n Move disk 1 from peg %c to peg %c",frompeg,topeg);

return;

}

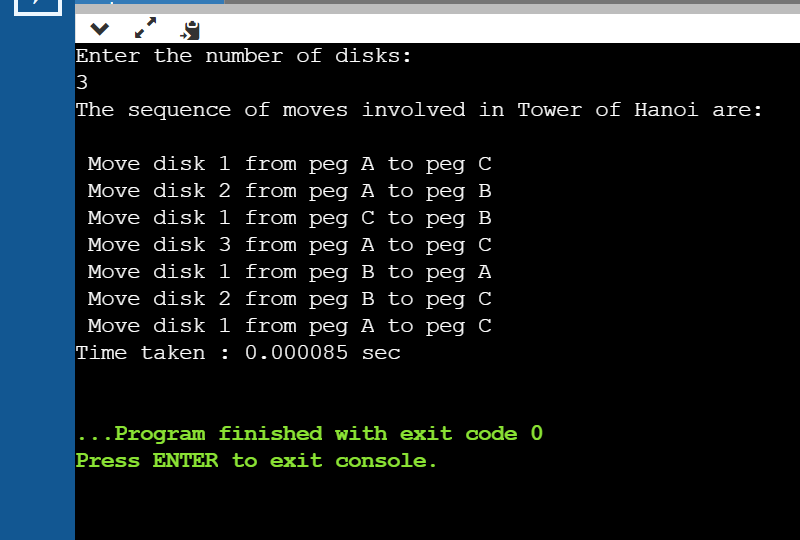
towers(num - 1,frompeg,auxpeg,topeg);

printf("\n Move disk %d from peg %c to peg %c",num,frompeg,topeg);

towers(num - 1,auxpeg,topeg,frompeg);

}

OUTPUT:



GCD OF TWO NUMBERS:

#include <stdio.h>

#include <time.h>

int gcd(int num1, int num2);

int main() {

clock\_t start,end;

double time;

int num1, num2;

printf("Enter two positive integers:\n");

scanf("%d %d", &num1, &num2);

start = clock();

printf("\n G.C.D of %d and %d is : %d", num1, num2, gcd(num1, num2));

end = clock();

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

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

return 0;

}

int gcd(int num1, int num2) {

if (num2 != 0)

{

return gcd(num2, num1 % num2);

}

else

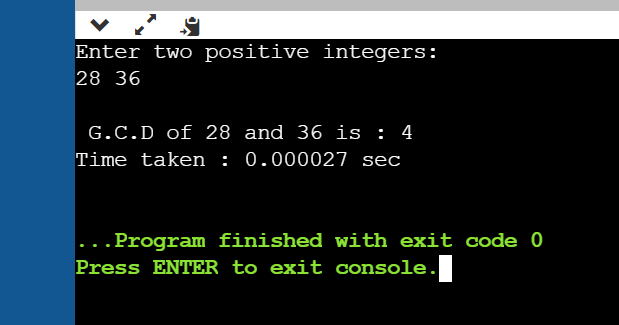
{

return num1;

}

}

OUTPUT:



NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 2: Implement Recursive Binary search and Linear search and determine the time required to search an element**

RECURSIVE LINEAR AND BINARY SEARCH:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int Linear\_Search(int array[10000], int search, int index, int n);

int Binary\_Search(int array[10000], int low,int high, int search);

int main()

{

clock\_t start,end;

double time;

int i,j,a,low, high, mid, n, search, array[10000],choice,loc,res,index;

printf("Enter number of elements in the array:\n");

scanf("%d", &n);

printf("The elements in the array are: \n");

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

{

array[i] = rand()%100;

printf("%d\n",array[i]);

}

printf("\n Enter the number to be searched: ");

scanf("%d",&search);

while(1){

printf("\n\*\*MENU\*\*\n");

printf("\n 1. Linear search");

printf("\n 2. Binary search");

printf("\n 3. Exit");

printf("\n Enter your choice: ");

scanf("%d",&choice);

switch(choice){

case 1:

start = clock();

loc = Linear\_Search(array, search, 0, n);

end = clock();

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

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

if (loc != 0)

{

printf("\nElement found at location: %d", loc);

}

else

{

printf("\nElement not found in the array!");

}

break;

case 2 : for (i = 0; i < n; ++i)

{

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

{

if (array[i] > array[j])

{

a = array[i];

array[i] = array[j];

array[j] = a;

}

}

}

printf("\n The sorted array is: \n");

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

printf("%d\n", array[i]);

start = clock();

res = Binary\_Search(array, 0, n-1, search);

end = clock();

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

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

if(res == -1){

printf("Element is not found in array!");

}

else{

printf("Element is present at location: %d", res);

}

break;

case 3: exit(0);

break;

default: printf("/n Wrong choice!");

break;

}

}

}

int Linear\_Search(int arr[], int search, int index, int n)

{

int pos = 0;

if(index >= n)

{

return 0;

}

else if (arr[index] == search)

{

pos = index + 1;

return pos;

}

else

{

return Linear\_Search(arr, search, index+1, n);

}

return pos;

}

int Binary\_Search(int arr[], int low, int high, int search)

{

if (high >= low)

{

int mid = low + (high - low)/2;

if (arr[mid] == search) return mid;

if (arr[mid] > search) return Binary\_Search(arr, low, mid-1, search);

return Binary\_Search(arr, mid+1, high, search);

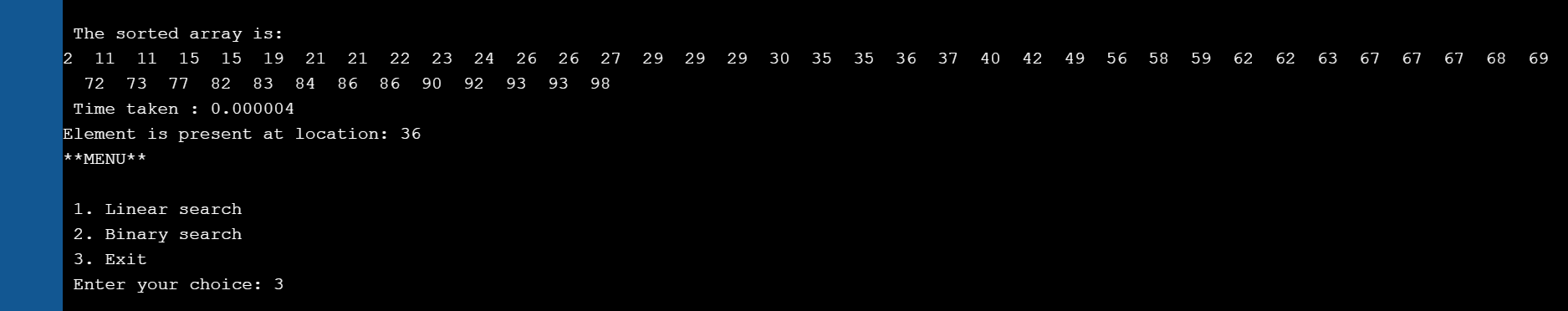
}

return -1;

}

OUTPUT:





|  |  |  |
| --- | --- | --- |
| N | Time (Linear) | Time (Binary) |
| 50 | 0.00001 | 0.000004 |
| 100 | 0.000012 | 0.000009 |
| 150 | 0.000016 | 0.000012 |
| 200 | 0.00002 | 0.000013 |

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 3: Sort a given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort**

NON-RECURSIVE SELECTION SORT:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

clock\_t start,end;

double time;

int a[100000], n, i, j, loc, temp;

printf("Enter number of elements in the array:\n");

scanf("%d", &n);

printf("The elements in the array are: \n");

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

{

a[i] = rand()%100;

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

}

start = clock();

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

{

loc=i;

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

{

if(a[loc] > a[j])

loc=j;

}

if(loc != i)

{

temp=a[i];

a[i]=a[loc];

a[loc]=temp;

}

}

end = clock();

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

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

printf("Sorted Array using Selection Sort is:\n");

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

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

return 0;

}

RECURSIVE SELECTION SORT:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void Sel\_Sort(int [], int, int);

int main()

{

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

arr[i] = rand()%100;

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

}

start = clock();

Sel\_Sort(arr, 0, n);

end = clock();

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

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

printf("\nThe sorted list in ascending order is\n");

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

{

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

}

return 0;

}

void Sel\_Sort(int arr[], int i, int n)

{

int j;

int min = i;

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

{

if (arr[j] < arr[min]) {

min = j;

}

}

int temp = arr[min];

arr[min] = arr[i];

arr[i] = temp;

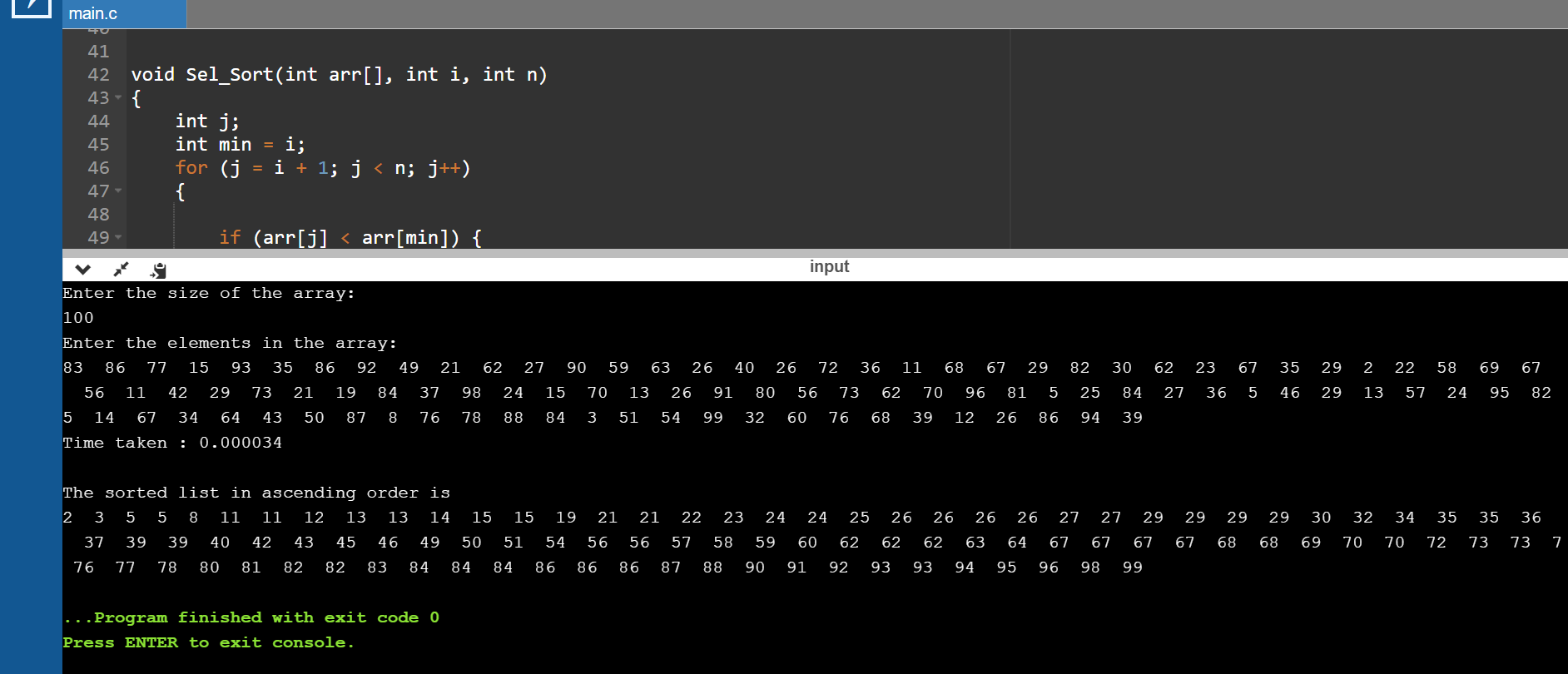
if (i + 1 < n) {

Sel\_Sort(arr, i + 1, n);

}

}

OUTPUT:



|  |  |
| --- | --- |
| **N** | **Time Taken** |
| 50 | 0.000015 |
| 100 | 0.000034 |
| 200 | 0.000103 |
| 500 | 0.000479 |
| 1000 | 0.001893 |
| 5000 | 0.041123 |
| 10000 | 0.156707 |
| 15000 | 0.405502 |
| 20000 | 0.607911 |
| 25000 | 1.080052 |

RECURSIVE BUBBLE SORT:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void bubbleSort(int [], int);

int main()

{

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

arr[i] = rand()%100;

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

}

start = clock();

bubbleSort(arr, n);

end = clock();

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

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

printf("\nThe sorted list in ascending order is\n");

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

{

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

}

return 0;

}

void bubbleSort(int arr[], int n)

{

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

{

if (arr[i] > arr[i + 1]) {

int temp = arr[i];

arr[i] = arr[i+1];

arr[i+1] = temp;

}

}

if (n - 1 > 1) {

bubbleSort(arr, n - 1);

}

}

OUTPUT:

|  |  |
| --- | --- |
| **N** | **Time Taken** |
| 50 | 0.000016 |
| 100 | 0.000044 |
| 200 | 0.000166 |
| 500 | 0.000721 |
| 1000 | 0.002964 |
| 5000 | 0.068924 |
| 10000 | 0.328939 |
| 15000 | 0.992609 |
| 20000 | 1.382367 |
| 25000 | 2.260302 |

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 4: a.) Print all the nodes reachable from a given starting node in a digraph using BFS method. b.) Check whether a given graph is connected or not using DFS method**

PROGRAM 4 = DFS & BFS

1. Print all the nodes reachable from a given starting node in a digraph using BFS method.

#include<stdio.h>

#include<time.h>

int arr[50][50],queue[50],visit[50],n,i,j,f=0,r=-1;

void bfs(int v)

{

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

if(arr[v][i] && !visit[i])

queue[++r]=i;

if(f<=r)

{

visit[queue[f]]=1;

bfs(queue[f++]);

}

}

int main()

{

clock\_t start,end;

double time;

int v;

printf("\nEnter the number of nodes:");

scanf("%d",&n);

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

{

queue[i]=0;

visit[i]=0;

}

printf("\nEnter graph data in matrix form:\n");

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

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

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

}

}

printf("\n Enter the starting vertex:");

scanf("%d",&v);

start = clock();

bfs(v);

end = clock();

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

printf("\n The nodes which are reachable are:\n");

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

if(visit[i])

printf("%d\t",i);

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

}

OUTPUT :



1. Check whether a given graph is connected or not using DFS method.

#include<stdio.h>

#include<time.h>

int arr[50][50],reachable[50],n;

void dfs(int v)

{

int i;

reachable[v]=1;

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

if(arr[v][i] && !reachable[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main()

{

clock\_t start,end;

double time;

int i,j,count=0;

printf("\nEnter number of nodes: ");

scanf("%d",&n);

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

{

reachable[i]=0;

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

arr[i][j]=0;

}

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

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

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

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

start = clock();

dfs(1);

end = clock();

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

printf("\n");

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

{

if(reachable[i])

count++;

}

if(count==n)

printf("\nGraph is connected!");

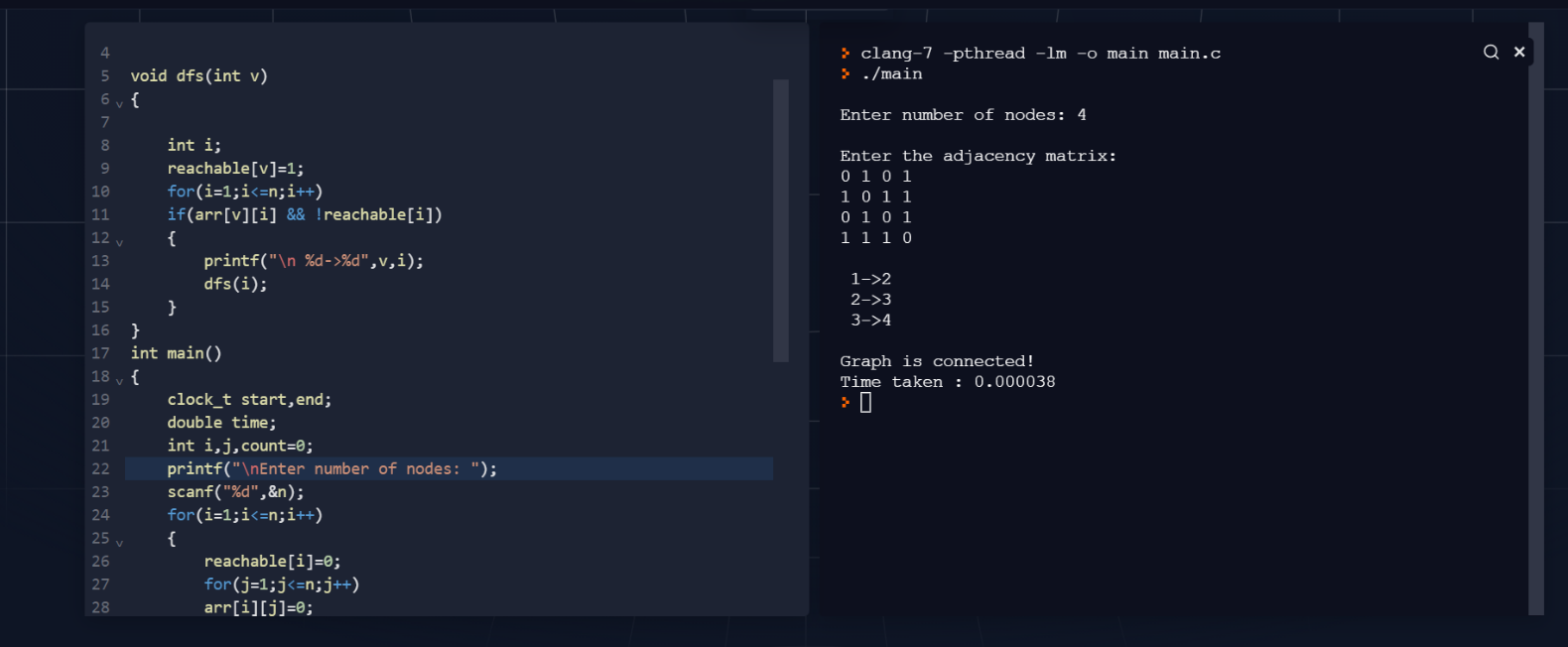
else

printf("\nGraph is not connected!");

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

}

OUTPUT:



NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 5: Sort a given set of N integer elements using Insertion Sort technique and compute its time taken**

NON-RECURSIVE INSERTION SORT

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

void Insert\_Sort(int [], int);

int main()

{

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

arr[i] = rand()%100;

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

}

start = clock();

Insert\_Sort(arr, n);

end = clock();

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

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

printf("\nThe sorted list in ascending order is\n");

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

{

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

}

return 0;

}

void Insert\_Sort(int arr[], int n)

{

int i, search, j;

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

search = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > search) {

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

j = j - 1;

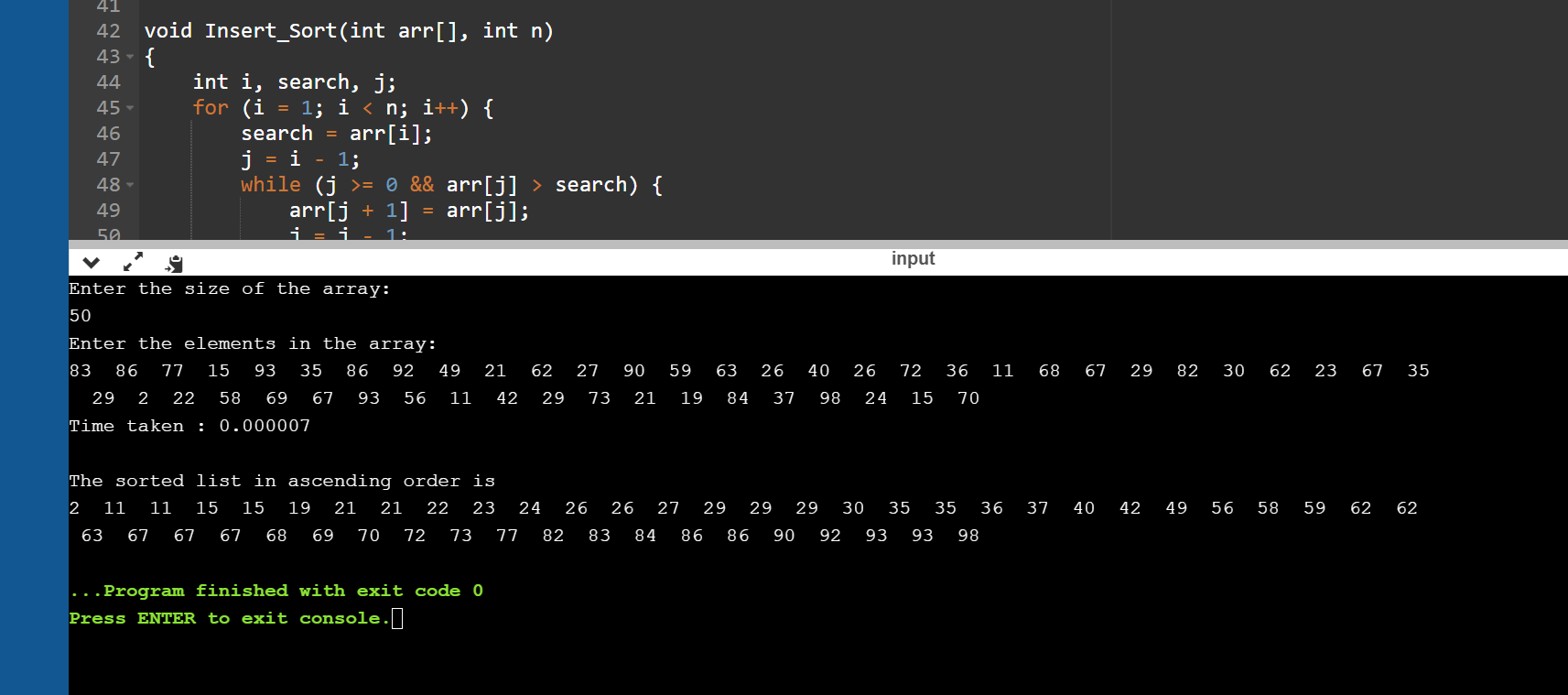
}

arr[j + 1] = search;

}

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000007 |
| 100 | 0.000015 |
| 200 | 0.000065 |
| 500 | 0.000312 |
| 1000 | 0.001089 |
| 5000 | 0.026088 |
| 10000 | 0.087888 |
| 15000 | 0.183433 |
| 20000 | 0.40368 |
| 25000 | 0.537591 |

RECURSIVE INSERTION SORT

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void Insertion\_Sort(int [], int);

int main()

{

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

arr[i] = rand()%100;

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

}

start = clock();

Insertion\_Sort(arr, n);

end = clock();

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

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

printf("\nThe sorted list in ascending order is\n");

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

{

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

}

return 0;

}

void Insertion\_Sort(int arr[], int n){

if (n <= 1)

return;

Insertion\_Sort( arr, n-1 );

int last = arr[n-1];

int j = n-2;

while (j >= 0 && arr[j] > last){

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

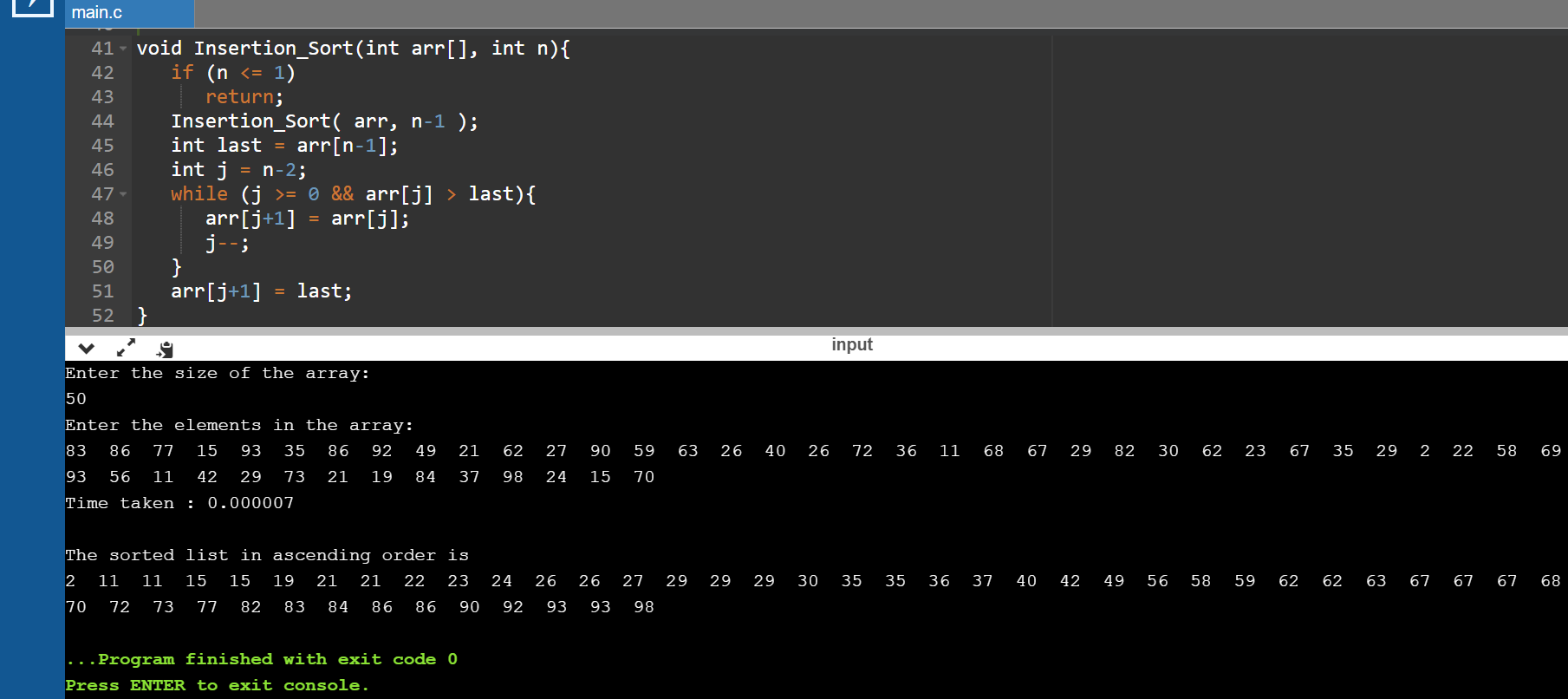
j--;

}

arr[j+1] = last;

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000007 |
| 100 | 0.000024 |
| 200 | 0.000063 |
| 500 | 0.000341 |
| 1000 | 0.001115 |
| 5000 | 0.024857 |
| 10000 | 0.088347 |
| 15000 | 0.189293 |
| 20000 | 0.403715 |
| 25000 | 0.580936 |

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 6: Write program to obtain the Topological ordering of vertices in a given digraph**

PROGRAM 6: TOPOLOGICAL SORTING

#include<stdio.h>

#include<time.h>

int a[50][50],n,indeg[50];

void indegree()

{

int j,i,sum;

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

{

sum=0;

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

sum+=a[i][j];

indeg[j]=sum;

}

}

void topology()

{

int i,u,v,t[50],s[50],tos=-1,k=0;

indegree();

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

{

if(indeg[i]==0)

s[++tos]=i;

}

while(tos!=-1)

{

u=s[tos--];

t[k++]=u;

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

{

if(a[u][v]==1)

{

indeg[v]--;

if(indeg[v]==0)

s[++tos]=v;

}

}

}

printf("The Topological Sequence is:\n");

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

printf("%d ",(t[i]+1));

}

int main()

{

clock\_t start,end;

double time;

int i,j;

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

scanf("%d",&n);

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

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

{

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

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

}

start = clock();

topology();

end = clock();

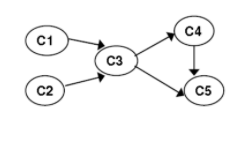
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

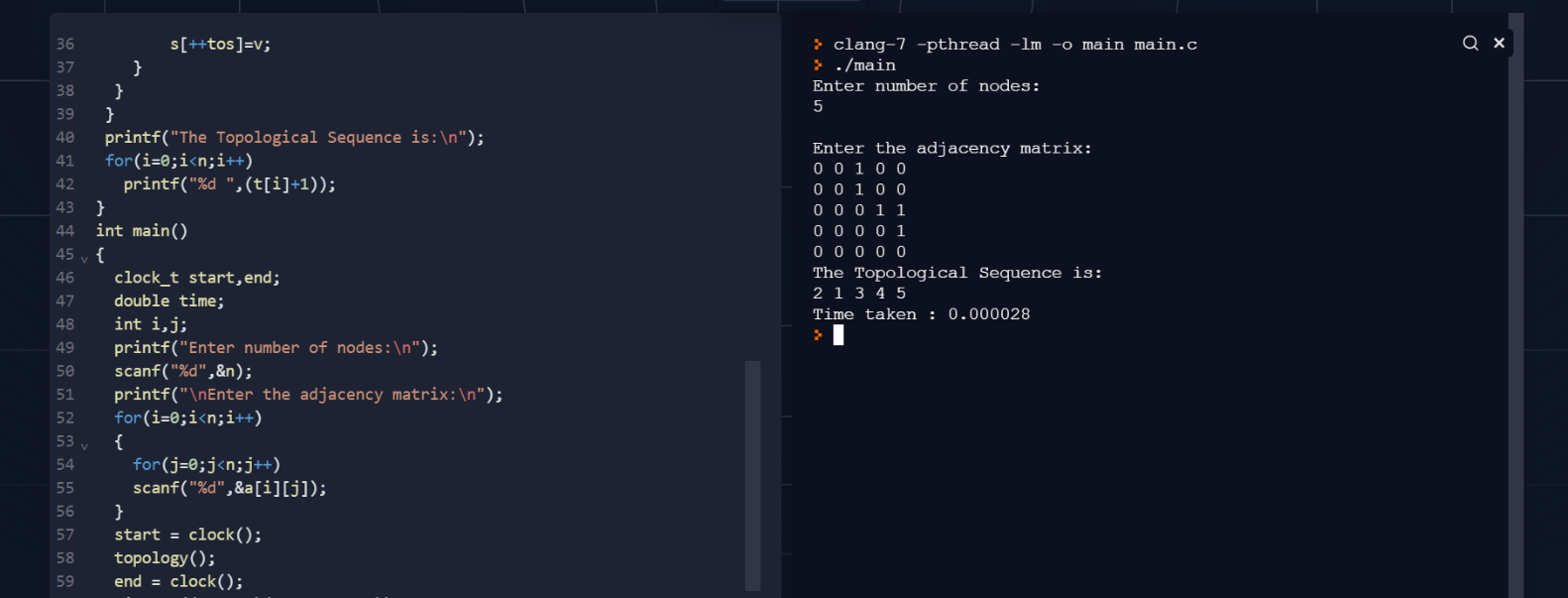
printf("\nTime taken : %lf\n",time);

}

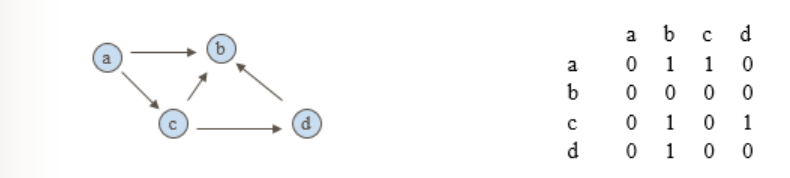
OUTPUT:

Graph 1:





Graph 2:





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 7: Implement Johnson Trotter algorithm to generate permutations**

PROGRAM 7: JOHNSON TROTTER ALGORITHM

CODE:

#include <stdio.h>

#include<time.h>

int N, i;

int p[10000], q[10000];

int direct[10000];

void Move(int x, int d)

{

int z;

printf("\n");

z = p[q[x]+d];

p[q[x]] = z;

p[q[x]+d] = x;

q[z] = q[x];

q[x] = q[x]+d;

}

void Permutation(int n)

{

int i;

if (n > N){

int i;

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

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

}

else

{

Permutation( n+1 );

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

{

Move(n,direct[n]);

Permutation(n+1);

}

direct[n] = -direct[n];

}

}

int main ()

{

clock\_t start,end;

double time;

printf("Enter the value of N:");

scanf("%d", &N);

printf("\n");

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

{

direct[i] = -1; p[i] = i;

q[i] = i;

}

printf("The permutations generated are:\n");

start = clock();

Permutation(1);

end = clock();

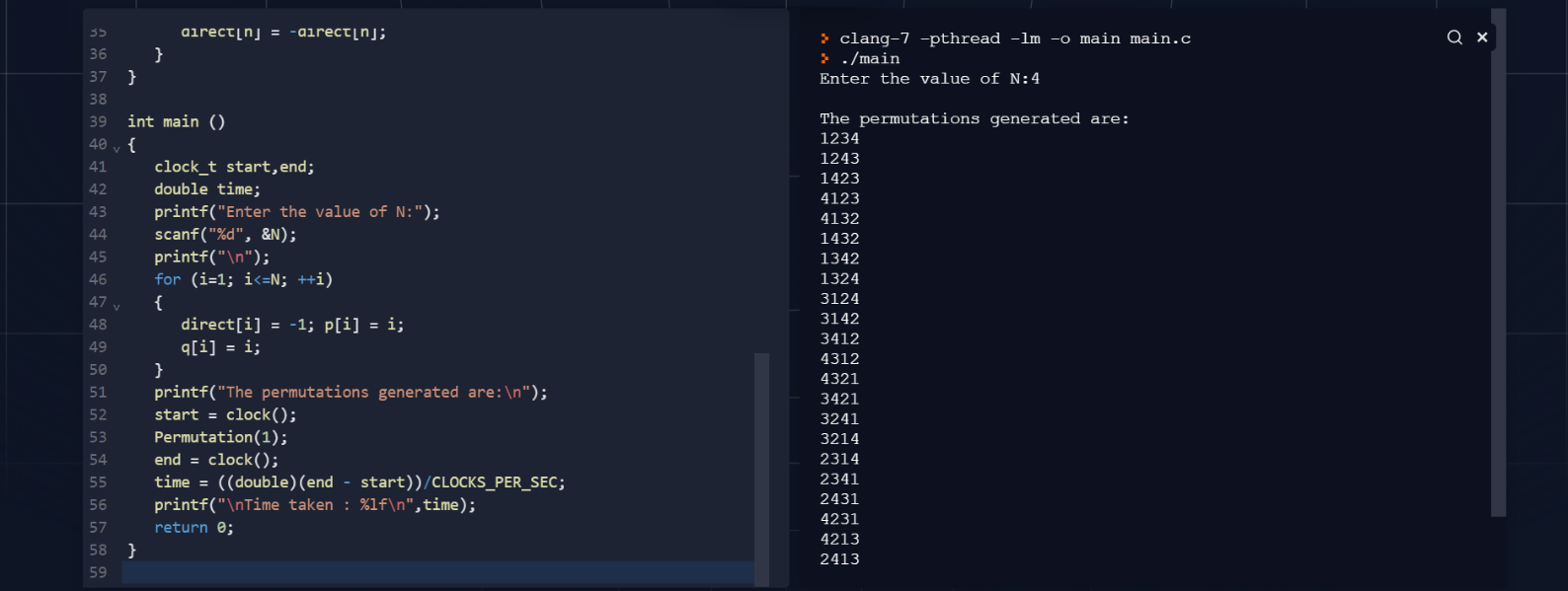
time = ((double)(end - start))/CLOCKS\_PER\_SEC;

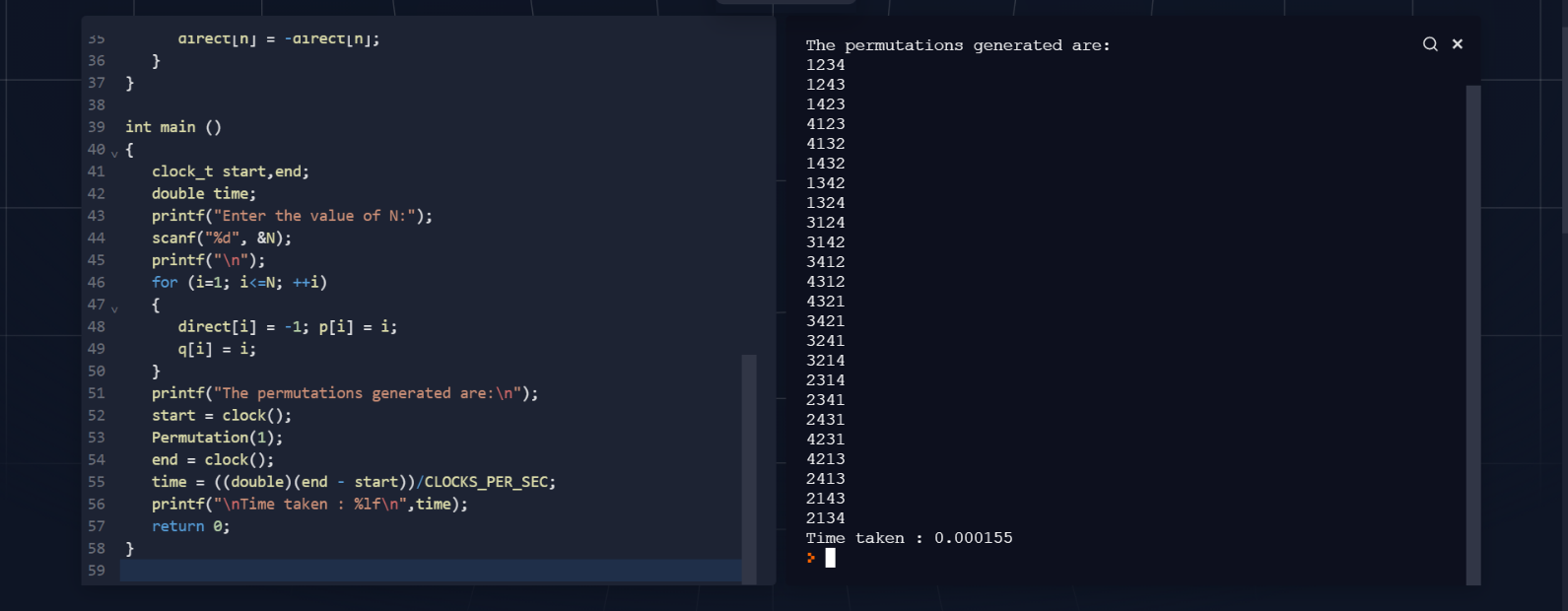
printf("\nTime taken : %lf\n",time);

return 0;

}

OUTPUT:





NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 8: Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort**

PROGRAM 8: RECURSIVE MERGE SORT

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void combine(int [], int, int, int);

void split(int [],int, int);

int main()

{

clock\_t start,end;

double time;

int arr[100000], n, i;

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

arr[i] = rand()%100;

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

}

start = clock();

split(arr, 0, n - 1);

end = clock();

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

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

printf("\nThe sorted list in ascending order after applying MergeSort is:\n");

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

{

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

}

return 0;

}

void split(int arr[],int low,int high)

{

int mid;

if(low < high)

{

mid = (low + high) / 2;

split(arr, low, mid);

split(arr, mid + 1, high);

combine(arr, low, mid, high);

}

}

void combine(int arr[],int low,int mid,int high)

{

int i, m, k, l, temp[100000];

l = low;

i = low;

m = mid + 1;

while ((l <= mid) && (m <= high))

{

if (arr[l] <= arr[m])

{

temp[i] = arr[l];

l++;

}

else

{

temp[i] = arr[m];

m++;

}

i++;

}

if (l > mid)

{

for (k = m; k <= high; k++)

{

temp[i] = arr[k];

i++;

}

}

else

{

for (k = l; k <= mid; k++)

{

temp[i] = arr[k];

i++;

}

}

for (k = low; k <= high; k++)

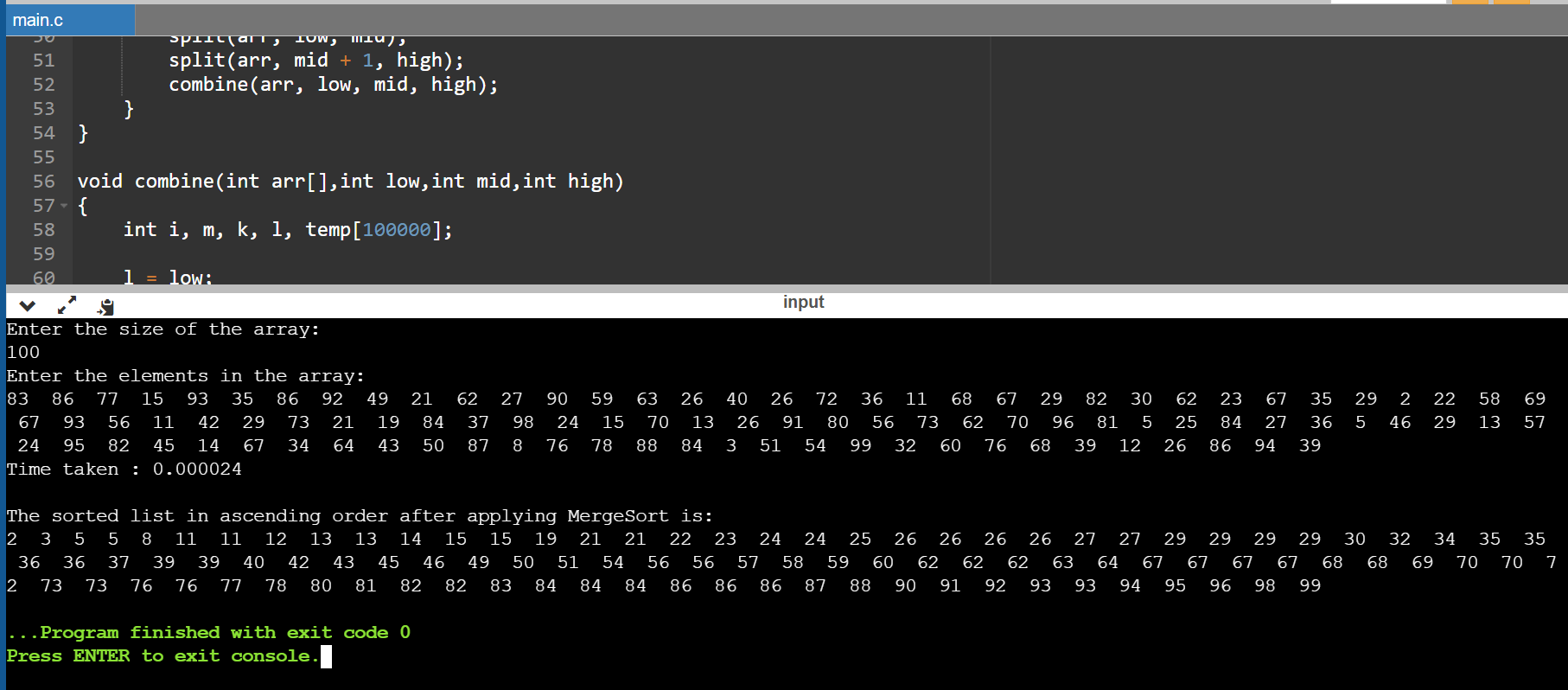
{

arr[k] = temp[k];

}

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000017 |
| 100 | 0.000024 |
| 200 | 0.000046 |
| 500 | 0.000111 |
| 1000 | 0.00023 |
| 5000 | 0.000956 |
| 10000 | 0.002056 |
| 15000 | 0.002789 |
| 20000 | 0.004008 |
| 25000 | 0.005103 |

NAME: RAMYA RAMESH

USN: 1BM19CS227

**PROGRAM 9: Sort a given set of N integer elements using Quick Sort technique and compute its time taken**

PROGRAM 9 : QUICKSORT

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void quicksort(int [], int, int);

int main()

{

clock\_t start,end;

double time;

int i, n, a[100000];

printf("Enter the size of the array:\n");

scanf("%d", &n);

printf("Enter the elements in the array:\n");

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

{

a[i] = rand()%100;

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

}

start = clock();

quicksort(a,0,n-1);

end = clock();

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

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

printf("\nThe sorted list in ascending order is\n");

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

{

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

}

return 0;

}

void quicksort(int a[100000],int first,int last){

int i, j, pivot, temp;

if(first<last){

pivot=first;

i=first;

j=last;

while(i<j){

while(a[i]<=a[pivot]&&i<last)

i++;

while(a[j]>a[pivot])

j--;

if(i<j){

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

temp=a[pivot];

a[pivot]=a[j];

a[j]=temp;

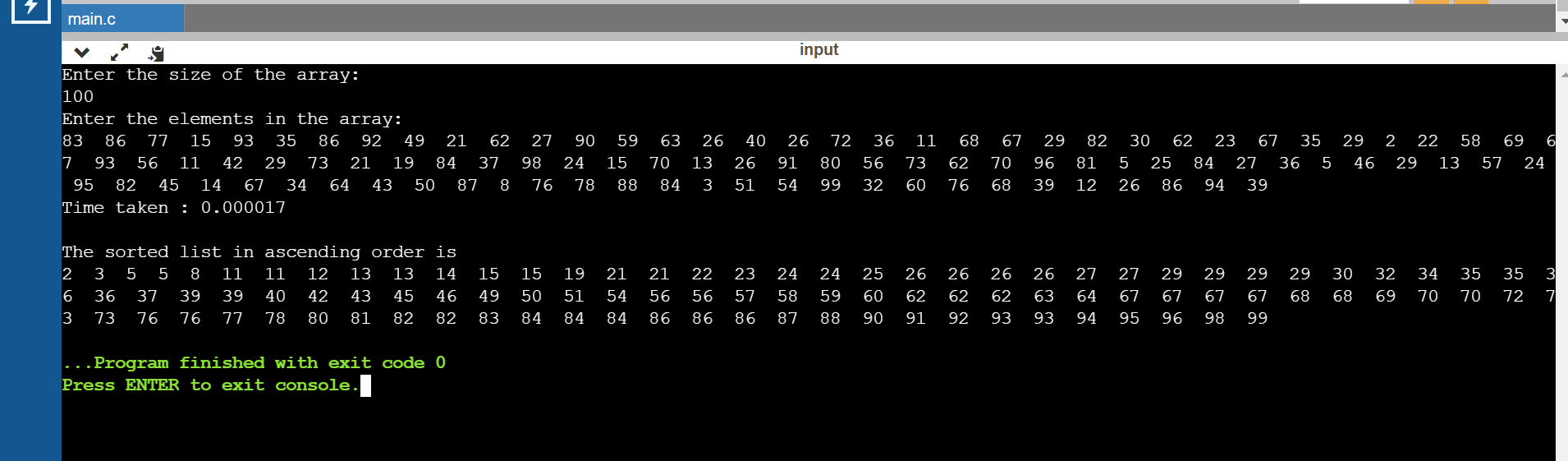
quicksort(a,first,j-1);

quicksort(a,j+1,last);

}

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000008 |
| 100 | 0.000017 |
| 200 | 0.000026 |
| 500 | 0.000083 |
| 1000 | 0.000122 |
| 5000 | 0.000858 |
| 10000 | 0.002929 |
| 15000 | 0.005951 |
| 20000 | 0.009067 |
| 25000 | 0.012314 |