

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Analysis and Design of Algorithms

Submitted by

Atharva Acharya (1BM20CS004)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by Atharva Acharya (1BM20CS004), who is a bonafide student of B. M. S. College of Engineering. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: DR GR PRASAD
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1	Write a recursive program to Solve a) Towers-of-Hanoi problem b) To find GCD	
2	Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.	
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.	
4	Write program to do the following: 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.	
5	Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.	
6	Write program to obtain the Topological ordering of vertices in a given digraph.	
7	Implement Johnson Trotter algorithm to generate permutations.	
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.	
9	Sort a given set of N integer elements using Quick Sort technique and compute its time taken.	
10	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	
11	Implement Warshall's algorithm using dynamic programming	
12	Implement 0/1 Knapsack problem using dynamic programming.	
13	Implement All Pair Shortest paths problem using Floyd's algorithm.	
14	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	
15	Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm.	
16	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	
17	Implement "Sum of Subsets" using Backtracking. "Sum of Subsets" problem: Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d. For example, if $S = \{1, 2, 5, 6, 8\}$ and $d = 9$ there are two solutions	

	{1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.	
18	Implement "N-Queens Problem" using Backtracking.	

Course Outcome

CO1	Ability to analyze time complexity of Recursive and Non-Recursive algorithms using asymptotic notations.
CO2	Ability to design efficient algorithms using various design techniques.
CO3	Ability to apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Ability to conduct practical experiments to solve problems using an appropriate designing method and find time efficiency.

1a)Recursive GCD

```
#include <stdio.h>

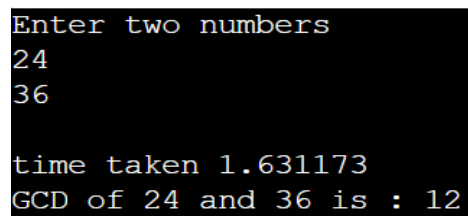
#include<time.h>

int gcd(int m,int n){
    if(n==0){
        return m;
    }

    return gcd(n,m%n);
}

int main()
{
    clock_t start,end;
    int m,n;
    printf("Enter two numbers\n");
    scanf("%d %d",&m,&n);
    start=clock();
    int result=gcd(m,n);
    end=clock();
    printf("\ntime taken %f ", difftime(end,start)/CLOCKS_PER_SEC);
    printf("\nGCD of %d and %d is : %d\n",m,n,result);
}
```

Output:

A screenshot of a terminal window with a black background and white text. The text shows the program's execution: it prompts for two numbers, receives 24 and 36, calculates the GCD, and displays the time taken and the result.

```
Enter two numbers
24
36

time taken 1.631173
GCD of 24 and 36 is : 12
```

1b) Tower Of Hanoi

```
#include <stdio.h>
```

```
#include <time.h>
```

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

```
{  
    if (num == 1)  
    {  
        printf("\n Move disk 1 from  %c to %c", frompeg, topeg);  
        return;  
    }
```

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

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

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

```
}
```

```
int main()
```

```
{
```

```
    int n;
```

```
    clock_t start,end;
```

```
    printf("Enter a number : ");
```

```
    scanf("%d", &n);
```

```
    start=clock();
```

```
    towers(n, 'A', 'C', 'B');
```

```
    end=clock();
```

```
    printf("\ntime taken %f ", difftime(end,start)/CLOCKS_PER_SEC);
```

```
    return 0;
```

```
}
```

Output:

```
Enter a number : 3  
  
Move disk 1 from A to C  
Move disk 2 from A to B  
Move disk 1 from C to B  
Move disk 3 from A to C  
Move disk 1 from B to A  
Move disk 2 from B to C  
Move disk 1 from A to C  
time taken 1.643774
```

2)Recursive Linear and Binary Search

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
void linear_search(int arr[], int index, int last, int key){
```

```
    if(index>last){
```

```
        printf("Element not found\n");
```

```
        return;
```

```
    }
```

```
    if(arr[index]==key){
```

```
        printf("Element found at position %d\n",index);
```

```
        return;
```

```
    }
```

```
    linear_search(arr,index+1,last,key);
```

```
}
```

```
void binary_search(int arr[], int low, int high, int key){
```

```
    if(low>high){
```

```
        printf("Element not found\n");
```

```
        return;
```

```
    }
```

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

```
    if(arr[mid]==key){
```

```
        printf("Element found at position %d\n",mid);
```

```
    }
```

```
    else if(arr[mid]>key){
```



```

        binary_search(arr,low,mid-1,key);
    }
    else{
        binary_search(arr,mid+1,high,key);
    }
}

```

```

int pivot_position(int arr[],int low, int high){

```

```

    int pivot = arr[low];

```

```

    int i=low;

```

```

    int j=high;

```

```

    while(i<j){

```

```

        // Move ahead till value is less than pivot

```

```

        while(arr[i]<pivot && i<=high){

```

```

            i++;

```

```

        }

```

```

        // Move ahead till value is greater than pivot

```

```

        while(arr[j]>pivot && j>=low){

```

```

            j--;

```

```

        }

```

```

        // point of violation: swap the values that cause the violation

```

```

        if(i<j){

```

```

            int temp=arr[i];

```

```

            arr[i]=arr[j];

```

```

            arr[j]=temp;

```

```

        }

```

```
}
```

```
// While traversing if i>j, it means we have found pivot position(j), so swap pivot with a[j];
```

```
arr[low] = arr[j];
```

```
arr[j] = pivot;
```

```
// returning j as the pivot position
```

```
return j;
```

```
}
```

```
void quickSort(int arr[],int low,int high){
```

```
    if(low<high){
```

```
        int pivot = pivot_position(arr,low,high);
```

```
        quickSort(arr,low,pivot-1);
```

```
        quickSort(arr,pivot+1,high);
```

```
    }
```

```
}
```

```
void main(){
```

```
    clock_t start1,end1,start2,end2;
```

```
    int n,key,choice;
```

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

```
    scanf("%d",&n);
```

```
    int arr[n];
```

```
    // printf("Enter the elements of the array\n");
```

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

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

```
// }
```

```
// For random input
```

```
for(int i=0;i<n;i++){  
    arr[i]=rand();  
}
```

```
printf("Enter the element you want to search\n");  
scanf("%d",&key);
```

```
do{  
    printf("Enter 1 to perform linear_search\n");  
    printf("Enter 2 to perform binary_search\n");  
    printf("Enter 3 to exit\n");  
    scanf("%d",&choice);
```

```
switch(choice){  
    case 1: start1 = clock();  
        linear_search(arr,0,n-1,key);  
        end1 = clock();  
        printf("time taken %f\n",difftime(end1,start1)/CLOCKS_PER_SEC);  
        break;  
    case 2: quickSort(arr,0,n-1);  
        start2 = clock();  
        binary_search(arr,0,n-1,key);  
        end2 = clock();  
        printf("time taken %f\n ",difftime(end2,start2)/CLOCKS_PER_SEC);  
        break;  
    case 3: printf("Exiting..");
```

```

        break;
    }

}while(choice!=3);
}

```

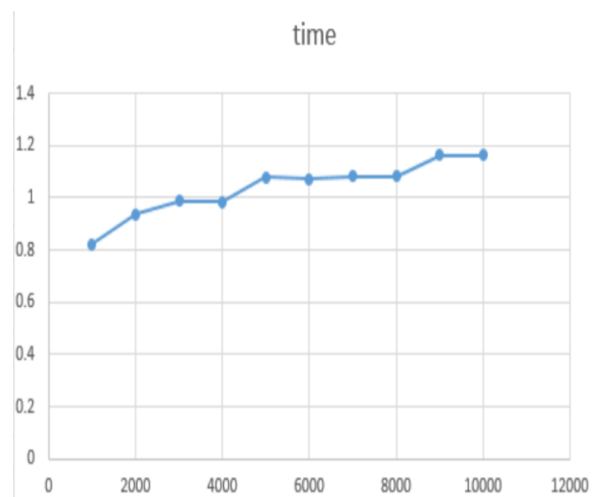
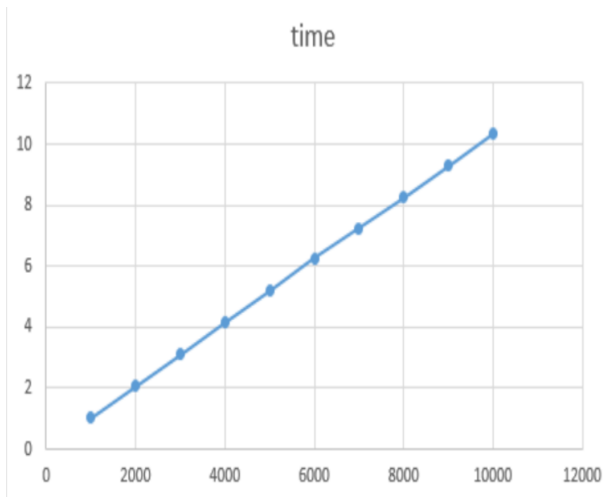
Output:

```

Enter number of elements
1000
Enter the element you want to search
2345
Enter 1 to perform linear_search
Enter 2 to perform binary_search
Enter 3 to exit
1
Element not found
time taken 0.000053
Enter 1 to perform linear_search
Enter 2 to perform binary_search
Enter 3 to exit
2
Element not found
time taken 0.000011
Enter 1 to perform linear_search
Enter 2 to perform binary_search
Enter 3 to exit
3
Exiting..

```

Graphs:



3)Selection Sort

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


void selection_sort(int arr[],int n){

    int i,j, temp,min;

    // Selection sort
    // n-1 since the last element will already be sorted
    for(i=0;i<n-1;i++){
        min=i;

        // Selecting the smallest element
        for(j=i+1;j<n;j++){
            if(arr[j]<arr[min]){
                min=j;
            }
        }

        // Insertion of the smallest element in the right place by swapping
        temp=arr[i];
        arr[i]=arr[min];
        arr[min]=temp;

    }
}
```

```
int main()
{
    clock_t start,end;

    int n;

    printf("Enter number of elements\n");
    scanf("%d",&n);

    int arr[n];

    printf("Enter the elements of the array\n");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }

    // // For random input
    // for(int i=0;i<n;i++){
    //     arr[i]=rand();
    // }

    start = clock();

    selection_sort(arr,n);

    end = clock();

    printf("Time taken: %f\n", difftime(end,start)/CLOCKS_PER_SEC);

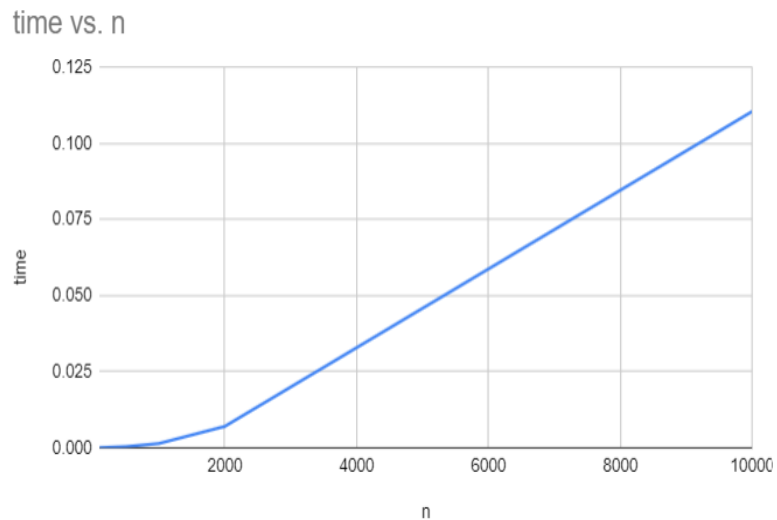
    // displaying the elements
    printf("Sorted array: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}
```

```
}  
printf("\n");  
  
}
```

Output:

```
Enter number of elements  
5  
Enter the elements of the array  
23 54 12 65 10  
Time taken: 0.000001  
Sorted array: 10 12 23 54 65
```

Graph



4a) BFS Reachable

```
#include <stdio.h>

int n, visited[10], adjacency[10][10], queue[10], f=0, r=-1;

void bfs(int v){
    for(int i=0; i<n; i++){
        if(visited[i]==0 && adjacency[v][i]==1){
            queue[++r]=i;
        }
    }
    if(f<=r){
        visited[queue[f]]=1;
        bfs(queue[f++]);
    }
}

int main()
{
    int start;

    printf("Enter number of vertices\n");
    scanf("%d", &n);

    for (int i=0; i<n; i++) {
        visited[i]=0;
        for (int j=0; j<n; j++)
            adjacency[i][j]=0;
    }

    printf("Enter the adjacency matrix\n");
    for(int i=0; i<n; i++){
        printf("Enter row %d\n", i);
        for(int j=0; j<n; j++){
```



```

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

printf("Enter the element you want to start with\n");
scanf("%d",&start);

bfs(start);

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

    for (int i=0;i<n;i++){
        if(visited[i]){
            printf("%d\t",i);
        }
        else{
            printf("\nBFS is not possible, all nodes are not reachable\n");
            break;
        }
    }
}

```

Output:

```

Enter number of vertices
5
Enter the adjacency matrix
Enter row 0
0 1 1 1 1
Enter row 1
1 0 0 0 0
Enter row 2
1 0 0 0 0
Enter row 3
1 0 0 0 0
Enter row 4
1 0 0 0 0
Enter the element you want to start with
0

The node which are reachable are:
0      1      2      3      4

```

4b) DFS Connected or Not

```
#include <stdio.h>
```

```
int n, visited[10], adjacency[10][10];
```

```
void dfs(int v){
```

```
    visited[v]=1;
```

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

```
        if(visited[i]==0 && adjacency[v][i]==1){
```

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

```
            dfs(i);
```

```
        }
```

```
    }
```

```
}
```

```
int main()
```

```
{
```

```
    int count = 0;
```

```
    printf("Enter number of vertices\n");
```

```
    scanf("%d",&n);
```

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

```
        visited[i]=0;
```

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

```
            adjacency[i][j]=0;
```

```
    }
```

```

printf("Enter the adjacency matrix\n");
for(int i=0;i<n;i++){
    printf("Enter row %d\n",i);
    for(int j=0;j<n;j++){
        scanf("%d",&adjacency[i][j]);
    }
}

printf("DFS traversal: ");
dfs(0);
printf("Graph is connected");

}

```

Output:

```

Enter number of vertices
5
Enter the adjacency matrix
Enter row 0
0 1 1 0 0
Enter row 1
1 0 0 1 1
Enter row 2
1 0 0 0 0
Enter row 3
0 1 0 0 0
Enter row 4
0 1 0 0 0
DFS traversal: 0->1 1->3 1->4 0->2 Graph is connected

```

5) Insertion Sort

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void insertion_sort(int arr[],int n){

    int j;

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

        int key = arr[i];

        j=i-1;

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

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

            j--;

        }

        arr[j+1]=key;

    }

}

int main()

{

    clock_t start,end;

    int n;

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

    scanf("%d",&n);

    int arr[n];

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

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

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

    }

    // // For random input

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

```

// arr[i]=rand();
// }

start = clock();

insertion_sort(arr,n);

end = clock();

printf("Time taken: %f\n", difftime(end,start)/CLOCKS_PER_SEC);

// displaying the elements

printf("Sorted array: ");

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

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

}

printf("\n");
}

```

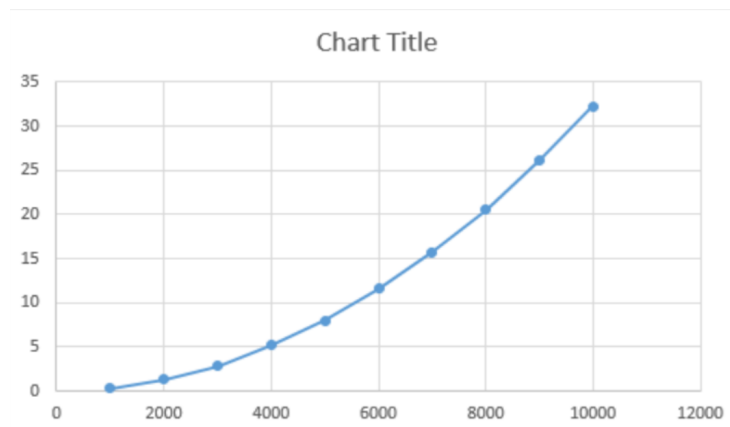
Output:

```

Enter number of elements
5
Enter the elements of the array
23 54 34 45 67
Time taken: 0.000001
Sorted array: 23 34 45 54 67

```

Graph:



6) Topological Ordering

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int n;
```

```
    int indeg[10],flag[10],adjacency[10][10];
```

```
    printf("Enter number of vertices\n");
```

```
    scanf("%d",&n);
```

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

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

```
        for(int j=0;j<n;j++){
```

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

```
        }
```

```
    }
```

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

```
        indeg[i]=0;
```

```
        flag[i]=0;
```

```
    }
```

```
    printf("the topological order is: \n");
```

```
    int count=0;
```

```
    while(count<n){
```

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

```
            if(indeg[i]==0 && flag[i]==0){
```

```
                printf("%d ",i);
```

```

        flag[i]=1;
    }

    for(int j=0;j<n;j++){
        if(adjacency[j][i]==1){
            indeg[i]--;
        }
    }
}
count++;

}

}

```

Output:

```

Enter number of vertices
4
Enter the adjacency matrix
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 1
the topological order is:
0 1 2 3

```

7)Johnson Trotter

```
#include <stdio.h>

#include <stdlib.h>

int flag = 0;

int swap(int *a,int *b)
{
    int t = *a;

    *a = *b;

    *b = t;
}

int search(int arr[],int num,int mobile)
{
    int g;

    for(g=0;g<num;g++)
    {
        if(arr[g] == mobile)
        {
            return g+1;
        }

        else
        {
            flag++;
        }
    }

    return -1;
}

int find_Moblie(int arr[],int d[],int num)
{
    int mobile = 0;
```



```

int mobile_p = 0;

int i;

for(i=0;i<num;i++)
{
    if((d[arr[i]-1] == 0) && i != 0)
    {
        if(arr[i]>arr[i-1] && arr[i]>mobile_p)
        {
            mobile = arr[i];
            mobile_p = mobile;
        }
        else
        {
            flag++; }
    }
    else if((d[arr[i]-1] == 1) & i != num-1)
    {
        if(arr[i]>arr[i+1] && arr[i]>mobile_p)
        {
            mobile = arr[i];
            mobile_p = mobile;
        }
        else
        {

            flag++;
        }
    }
    else
    {
        flag++;
    }
}

```

```

        }
    }
    if((mobile_p == 0) && (mobile == 0))
        return 0;
    else
        return mobile;
}

void permutations(int arr[],int d[],int num)
{
    int i;
    int mobile = find_Moblie(arr,d,num);
    int pos = search(arr,num,mobile);
    if(d[arr[pos-1]-1]==0)
        swap(&arr[pos-1],&arr[pos-2]);
    else
        swap(&arr[pos-1],&arr[pos]);
    for(int i=0;i<num;i++)
    {
        if(arr[i] > mobile)
        {
            if(d[arr[i]-1]==0)
                d[arr[i]-1] = 1;
            else
                d[arr[i]-1] = 0;
        }
    }
    for(i=0;i<num;i++)
    {
        printf(" %d ",arr[i]);
    }
}

```

```

int factorial(int k)
{
    int f = 1;
    int i = 0;
    for(i=1;i<k+1;i++)
    {
        f = f*i;
    }
    return f;
}

int main()
{
    int num = 0;
    int i;
    int j;
    int z = 0;

    printf("Johnson trotter algorithm to find all permutations of given numbers \n");
    printf("Enter the number\n");
    scanf("%d",&num);
    int arr[num],d[num];
    z = factorial(num);
    printf("The total permutations are %d",z);
    printf("\nAll possible permutations are: \n");
    for(i=0;i<num;i++)
    {
        d[i] = 0;
        arr[i] = i+1;
        printf(" %d ",arr[i]);
    }
    printf("\n");
}

```

```
for(j=1;j<z;j++)  
{  
    permutations(arr,d,num);  
    printf("\n");  
}  
return 0;  
}
```

Output:

```
Johnson trotter algorithm to find all permutations of given numbers  
Enter the number  
3  
The total permutations are 6  
All possible permutations are:  
1 2 3  
1 3 2  
3 1 2  
3 2 1  
2 3 1  
2 1 3
```

8) Merge Sort

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
void merge(int arr[],int low, int high,int mid){
```

```
    int i=low, j=mid+1;
```

```
    int c[high+1];
```

```
    int k=low;
```

```
    while(i<=mid && j<=high){
```

```
        if(arr[i]<arr[j]){
```

```
            c[k]=arr[i];
```

```
            i++;
```

```
        }
```

```
        else{
```

```
            c[k]=arr[j];
```

```
            j++;
```

```
        }
```

```
        k++;
```

```
    }
```

```
    while(i<=mid){
```

```
        c[k]=arr[i];
```

```
        i++;
```

```
        k++;
```

```
    }
```

```
    while(j<=high){
```

```
        c[k]=arr[j];
```

```

        j++;

        k++;
    }

    for(i=low;i<=high;i++){
        arr[i]=c[i];
    }
}

void mergeSort(int arr[],int low,int high){
    if(low<high){
        int mid = (low + high)/2;
        mergeSort(arr,low,mid);
        mergeSort(arr,mid+1,high);
        merge(arr,low,high,mid);
    }
}

int main()
{
    clock_t start,end;

    int n;

    printf("Enter number of elements\n");
    scanf("%d",&n);

    int arr[n];

    printf("Enter the elements of the array\n");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}

```

```
}

// // For random input
// for(int i=0;i<n;i++){
//   arr[i]=rand();
// }

start = clock();

mergeSort(arr,0,n-1);

end = clock();

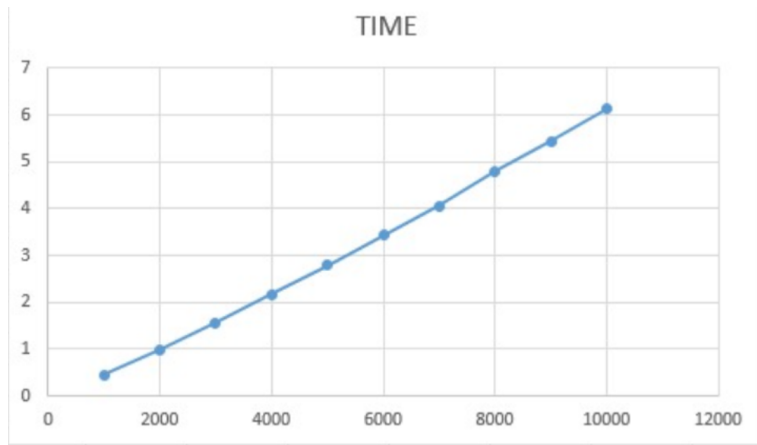
printf("Time taken: %f\n", difftime(end,start)/CLOCKS_PER_SEC);

// displaying the elements
printf("Sorted array: ");
for(int i=0;i<n;i++){
    printf("%d ",arr[i]);
}
printf("\n");
}
```

Output:

```
Enter number of elements
5
Enter the elements of the array
56 34 45 12 67
Time taken: 0.000003
Sorted array: 12 34 45 56 67
```

Graph:



9) Quick Sort

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
int pivot_position(int arr[],int low, int high){
```

```
    int pivot = arr[low];
```

```
    int i=low;
```

```
    int j=high;
```

```
    while(i<j){
```

```
        // Move ahead till value is less than pivot
```

```
        while(arr[i]<pivot && i<=high){
```

```
            i++;
```

```
        }
```

```
        // Move ahead till value is greater than pivot
```

```
        while(arr[j]>pivot && j>=low){
```

```
            j--;
```

```
        }
```

```
        // point of violation: swap the values that cause the violation
```

```
        if(i<j){
```

```
            int temp=arr[i];
```

```
            arr[i]=arr[j];
```

```
            arr[j]=temp;
```

```
        }
```

```
    }
```

```
// While traversing if i>j, it means we have found pivot position(j), so swap pivot with a[j];  
arr[low] = arr[j];  
arr[j] = pivot;  
// returning j as the pivot position  
return j;  
  
}
```

```
void quickSort(int arr[],int low,int high){  
    if(low<high){  
        int pivot = pivot_position(arr,low,high);  
        quickSort(arr,low,pivot-1);  
        quickSort(arr,pivot+1,high);  
    }  
}
```

```
int main()  
{  
    clock_t start,end;  
    int n;  
    printf("Enter number of elements\n");  
    scanf("%d",&n);  
  
    int arr[n];  
  
    printf("Enter the elements of the array\n");  
    for(int i=0;i<n;i++){  
        scanf("%d",&arr[i]);  
    }  
}
```

```
// // For random input
// for(int i=0;i<n;i++){
//   arr[i]=rand();
// }

start = clock();

quickSort(arr,0,n-1);

end = clock();

printf("Time taken: %f\n", difftime(end,start)/CLOCKS_PER_SEC);

// displaying the elements
printf("Sorted array: ");
for(int i=0;i<n;i++){
    printf("%d ",arr[i]);
}
printf("\n");
}
```

Output:

```
Enter number of elements
5
Enter the elements of the array
56 67 98 34 12
Time taken: 0.000002
Sorted array: 56 56 56 98 98
```

Graph:

