**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**JAI SHANKAR K S(1BM20CS062)**

***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 **JIA SHANKAR K S(1BM20CS062),** who is 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. Nagarathna N**               **Dr. Jyothi S Nayak**

Designation:- **Professor** Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru 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** | 1. Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} 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. |

1. **Write a recursive program to Solve**

**a)Towers-of-Hanoi problem**

**Program:-**

#include<iostream>

using namespace std;

void towerh(int n,char A,char B,char C){

    if(n==1){

        cout<<"move disc from "<<A<<" to "<<C<<endl;

    }

    else{

        towerh(n-1,A,C,B);

        cout<<"move disc from "<<A<<" to "<<C<<endl;

        towerh(n-1,B,A,C);

    }

}

int main(){

    int n;

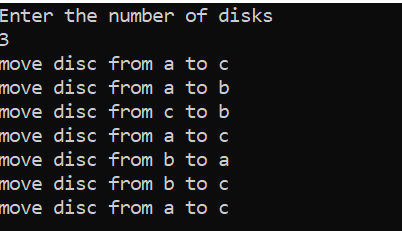
    cout<<"Enter the number of disks\n";

    cin>>n;

    towerh(n,'a','b','c');

}

**Output:-**



**1.b)GREATEST COMMON DIVISOR**

**Program:-**

#include<iostream>

using namespace std;

int gcd(int m,int n){

    if(n==0)

        return m;

    return gcd(n,m%n);

}

int main()

{

    int a,b,c;

    cout<<"Enter two numbers: ";

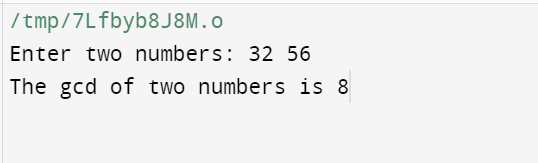
    cin>>a>>b;

    c=gcd(a,b);

    cout<<"The gcd of two numbers is "<<c;

}

**Output:-**



**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.**

**Program:-**

#include<iostream>

#include<vector>

#include<ctime>

using namespace std;

int linear\_search(int arr[],int low,int high,int key){

    if(low>high)

        return -1;

    if(arr[low]==key)

        return low;

    if(arr[high]==key)

        return high;

    else

        return linear\_search(arr,low+1,high-1,key);

}

void sort(int a[],int n){

    for(int i=0;i<n;i++){

        for(int j=0;j<n-i;j++)

        {

            if(a[j]>a[j+1])

                swap(a[j+1],a[j]);

        }

    }

}

int binary\_search(int arr[],int low,int high,int key){

    sort(arr,high+1);

    if(low>high)

        return -1;

    int mid=(low+high)/2;

    if(arr[mid]==key)

        return mid;

    else if(key>arr[mid])

        return binary\_search(arr,mid+1,high,key);

    else

        return binary\_search(arr,low,mid-1,key);

}

int main(){

     int n;

    clock\_t start,end;

    cout<<"Enter n:";

    cin>>n;

    int arr[n];

    cout<<"Elements:";

    for(int i=0;i<n;i++){

        arr[i]=i+1;

        cout<<arr[i]<<" ";

    }

    cout<<endl<<"Search Element?";

    int key=rand()%n+1;

    cout<<key;

    cout<<"\n----Linear Search----\n";

    cout<<"Element Index:";

    start=clock();

    cout<<linear\_search(arr,0,n-1,key);

    end=clock();

    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

    cout<<"\n----Binary Search----\n";

    cout<<"Element Index:";

    start=clock();

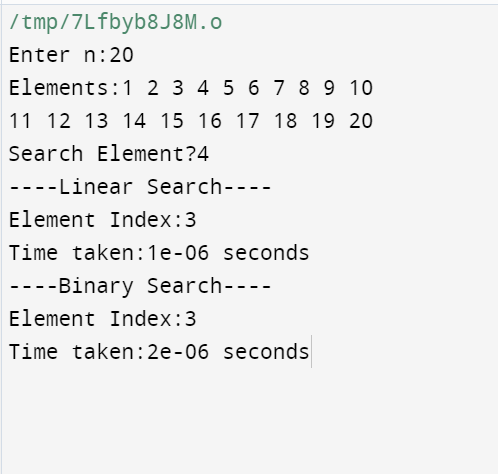
    cout<<binary\_search(arr,0,n-1,key);

    end=clock();

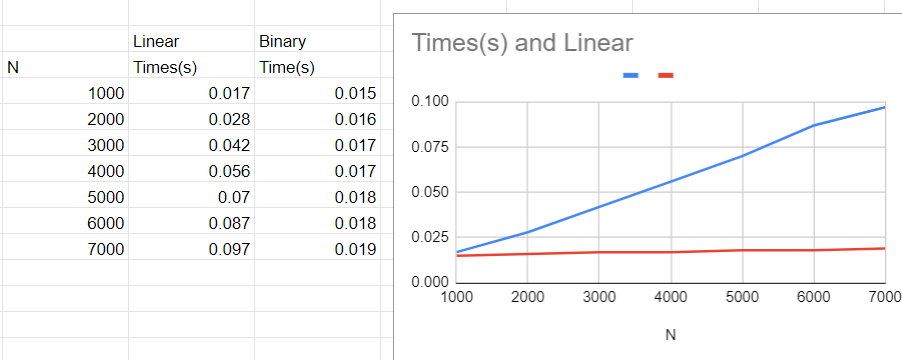
    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



**Graph:-**



**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.**

**Program:-**

#include<iostream>

#include<vector>

#include<ctime>

using namespace std;

vector<int> selection\_sort(vector<int> a){

    for(int i=0;i<a.size();i++){

        int min=i;

        for(int j=i+1;j<a.size();j++)

                if(a[j]<a[min])

                    min=j;

    swap(a[i],a[min]);

    }

    return a;

}

int  main(){

    int n;

    vector<int> arr;

    clock\_t start,end;

    cout<<"Enter no of elements:";

    cin>>n;

    cout<<"Elements Before Sorting:";

    for(int i=0;i<n;i++){

        arr.push\_back(rand()%100+1);

        cout<<arr[i]<<" ";

    }

    start=clock();

    arr=selection\_sort(arr);

    end=clock();

    cout<<endl<<"Elements After Sorting:";

    for(int i=0;i<n;i++){

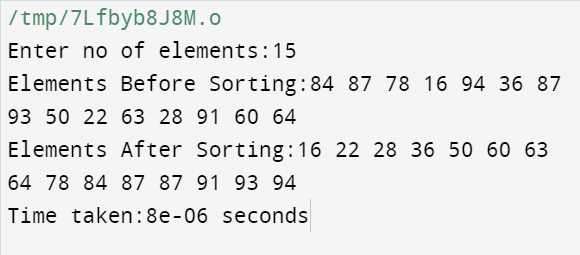
        cout<<arr[i]<<" ";

    }

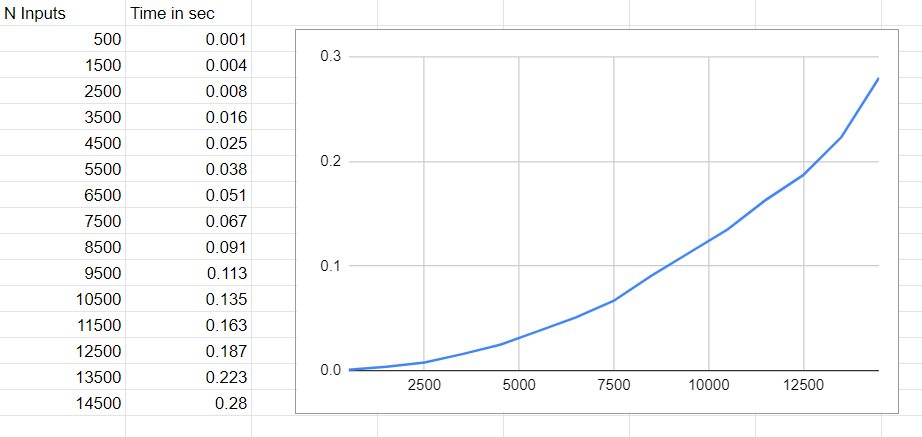
    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



**Graph:-**



**4.Write program to do the following:**

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

**Program:-**

#include<iostream>

using namespace std;

int order[20],visited[20],a[20][20];

int l=0,r=0;

void bfs(int v,int n){

    for(int i=0;i<n;i++){

        if(a[v][i] && !visited[i]){

            order[r++]=i;

            visited[i]=1;

        }

    }

    if(l<=r){

        bfs(l++,n);

    }

}

int main(){

    int n, s,flag=0;

    cout<<"Enter no of nodes:";

    cin>>n;

    cout<<"Enter Adjacency matrix:";

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

            cin>>a[i][j];

        }

    }

    cout<<"Enter starting vertex:";

    cin>>s;

    visited[s-1]=1;

    bfs(s-1,n);

    cout<<"Order of Traversing: v"<<s;

    for(int i=0;i<r;i++){

        cout<<" v"<<order[i]+1;

    }

    cout<<endl;

    for(int i=0;i<n;i++){

        if(visited[i]){

            flag=1;

        }

    }

    if(flag){

        cout<<"Graph connected"<<endl;

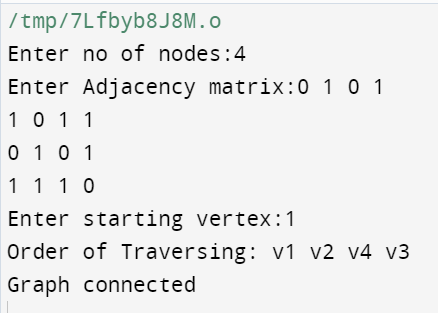
    }else{

        cout<<"Graph not connected"<<endl;

    }

}

**Output:-**



**4.b) Check whether a given graph is connected or not using DFS method.**

**Program:-**

#include<iostream>

using namespace std;

int order[20],visited[20],a[20][20];

int top=0;

void dfs(int v,int n){

    order[top++]=v;

    visited[v]=1;

    for(int i=0;i<n;i++){

        if(a[v][i] && !visited[i])

            dfs(i,n);

    }

}

int main(){

    int n, s,flag=0;

    cout<<"Enter no of nodes:";

    cin>>n;

    cout<<"Enter Adjacency matrix:";

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

            cin>>a[i][j];

        }

    }

    cout<<"Enter starting vertex:";

    cin>>s;

    dfs(s-1,n);

    cout<<"Order of Traversing: ";

    for(int i=0;i<top;i++){

        cout<<" v"<<order[i]+1;

    }

    cout<<endl;

    for(int i=0;i<n;i++){

        if(visited[i]){

            flag=1;

        }

    }

    if(flag){

        cout<<"Graph connected"<<endl;

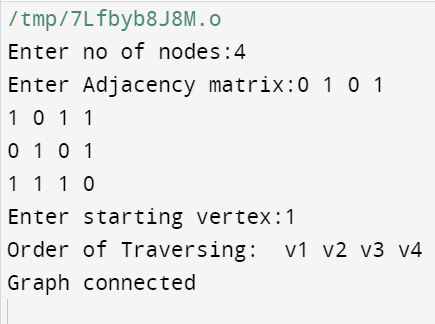
    }else{

        cout<<"Graph not connected"<<endl;

    }

}

**Output:-**



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

**Program:-**

#include<iostream>

#include<vector>

#include<ctime>

using namespace std;

vector<int> insertion\_sort(vector<int> a){

    int j,temp;

    for(int i=1;i<a.size();i++){

        j=i-1;

        temp=a[i];

        while(j>=0 && a[j]>temp){

            a[j+1]=a[j];

            j--;

        }

        a[j+1]=temp;

    }

    return a;

}

int  main(){

    int n;

    vector<int> a;

    clock\_t start,end;

    cout<<"Enter no of elements:";

    cin>>n;

    cout<<"Before Sorting:";

    for(int i=0;i<n;i++){

        a.push\_back(rand()%100+1);

        cout<<a[i]<<" ";

    }

    start=clock();

    a=insertion\_sort(a);

    end=clock();

    cout<<endl<<"After Sorting:";

    for(int i=0;i<n;i++){

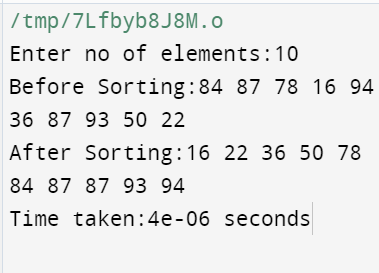
        cout<<a[i]<<" ";

    }

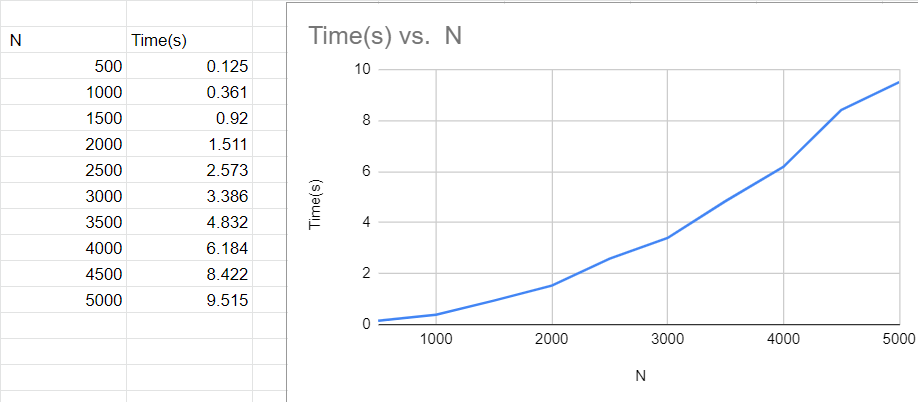
    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



**Graph:-**



1. **Write program to obtain the Topological ordering of vertices in a given digraph.**

**Program:-**

#include<iostream>

#include<vector>

#include<ctime>

using namespace std;

vector<int> selection\_sort(vector<int> a){

    for(int i=0;i<a.size();i++){

        int min=i;

        for(int j=i+1;j<a.size();j++)

                if(a[j]<a[min])

                    min=j;

    swap(a[i],a[min]);

    }

    return a;

}

int  main(){

    int n;

    vector<int> arr;

    clock\_t start,end;

    cout<<"Enter no of elements:";

    cin>>n;

    cout<<"Elements Before Sorting:";

    for(int i=0;i<n;i++){

        arr.push\_back(rand()%100+1);

        cout<<arr[i]<<" ";

    }

    start=clock();

    arr=selection\_sort(arr);

    end=clock();

    cout<<endl<<"Elements After Sorting:";

    for(int i=0;i<n;i++){

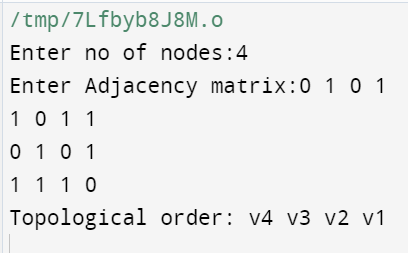
        cout<<arr[i]<<" ";

    }

    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



1. **Implement Johnson Trotter algorithm to generate permutations.**

**Program:-**

#include<iostream>

using namespace std;

int getpos(int a[],int n,int mob){

    for(int i=1;i<=n;i++){

        if(a[i]==mob)

            return i;

    }

    return 0;

}

int getmobile(int a[],bool dir[],int n){

    int mobile=0;

    for(int i=1;i<=n;i++){

        if(dir[a[i]]==false && i!=1){

            if(a[i]>a[i-1] && a[i]>mobile){

                mobile=a[i];

            }

        }

        if(dir[a[i]]==true && i!=n){

            if(a[i]>a[i+1] && a[i]>mobile){

                mobile=a[i];

            }

        }

    }

    return mobile;

}

void printpermutation(int a[],bool dir[],int n){

    int mobile=getmobile(a,dir,n);

    int pos=getpos(a,n,mobile);

    (dir[a[pos]]) ?  swap(a[pos+1],a[pos]):swap(a[pos],a[pos-1]);

    for(int i=1;i<=n;i++){

        if(a[i]>a[pos]){

            dir[a[i]]=(dir[a[i]]) ? false:true;

        }

        cout<<a[i];

    }

    cout<<" ";

}

int fact(int n){

    if(n==1)return 1;

    else return fact(n-1)\*n;

}

int main(){

    int n;

    cout<<"Enter no of elements:";

    cin>>n;

    int a[n];

    bool dir[n];

    for(int i=1;i<=n;i++){

        a[i]=i;

        dir[i]=false;

        cout<<a[i];

    }

    cout<<" ";

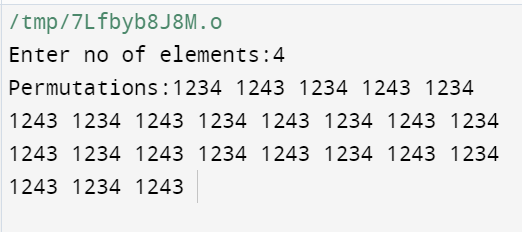
    for(int i=1;i<fact(n);i++){

        printpermutation(a,dir,n);

    }

}

**Output:-**



1. **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:-**

#include<iostream>

#include<ctime>

using namespace std;

void merge(int left[],int l,int right[],int r,int a[]){

    int i=0,j=0,k=0;

    while(i<l && j<r){

        if(left[i]<right[j]){

            a[k]=left[i];

            i++;

        }

        else{

            a[k]=right[j];

            j++;

        }

        k++;

    }

    while(i<l){

        a[k]=left[i];

        i++;

        k++;

    }

    while(j<r){

        a[k]=right[j];

        j++;

        k++;

    }

}

void mergesort(int a[],int len){

    if(len<=1)

    return;

    int mid=len/2;

    int left[mid],right[len-mid];

    for(int i=0;i<mid;i++)

        left[i]=a[i];

    for(int i=0;i<len-mid;i++)

        right[i]=a[mid+i];

    mergesort(left,mid);

    mergesort(right,len-mid);

    merge(left,mid,right,len-mid,a);

}

int main(){

    int n;

    clock\_t start,end;

    cout<<"Enter n:";

    cin>>n;

    int arr[n];

    cout<<"Before Sorting:";

    for(int i=0;i<n;i++){

        arr[i]=(rand()%100+1);

        cout<<arr[i]<<" ";

    }

    start=clock();

    mergesort(arr,n);

    end=clock();

    cout<<endl<<"Sorted Elements:";

    for(int i=0;i<n;i++){

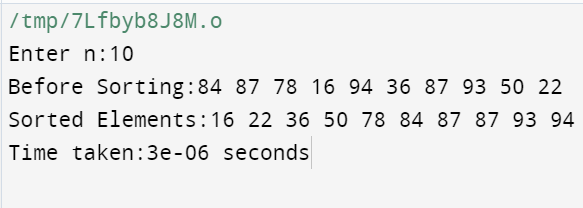
        cout<<arr[i]<<" ";

    }

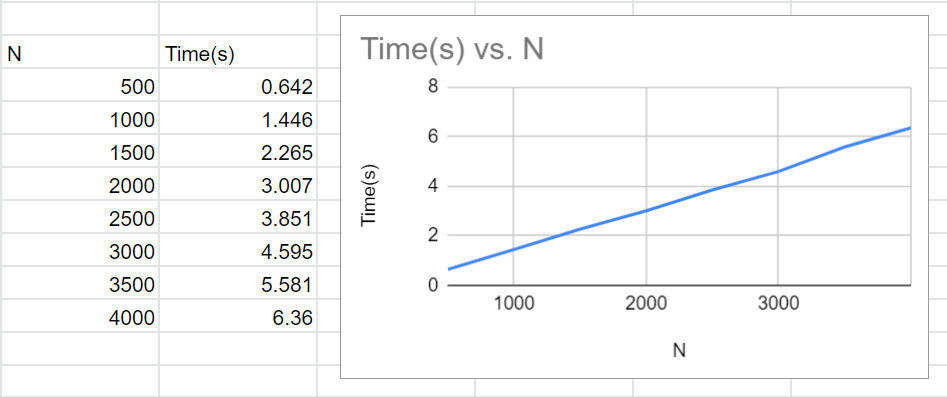
    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



**Graph:-**



1. **Sort a given set of N integer elements using Quick Sort technique and compute its time taken.**

**Program:-**

#include<iostream>

#include<ctime>

using namespace std;

void merge(int left[],int l,int right[],int r,int a[]){

    int i=0,j=0,k=0;

    while(i<l && j<r){

        if(left[i]<right[j]){

            a[k]=left[i];

            i++;

        }

        else{

            a[k]=right[j];

            j++;

        }

        k++;

    }

    while(i<l){

        a[k]=left[i];

        i++;

        k++;

    }

    while(j<r){

        a[k]=right[j];

        j++;

        k++;

    }

}

void mergesort(int a[],int len){

    if(len<=1)

    return;

    int mid=len/2;

    int left[mid],right[len-mid];

    for(int i=0;i<mid;i++)

        left[i]=a[i];

    for(int i=0;i<len-mid;i++)

        right[i]=a[mid+i];

    mergesort(left,mid);

    mergesort(right,len-mid);

    merge(left,mid,right,len-mid,a);

}

int main(){

    int n;

    clock\_t start,end;

    cout<<"Enter n:";

    cin>>n;

    int arr[n];

    cout<<"Before Sorting:";

    for(int i=0;i<n;i++){

        arr[i]=(rand()%100+1);

        cout<<arr[i]<<" ";

    }

    start=clock();

    mergesort(arr,n);

    end=clock();

    cout<<endl<<"Sorted Elements:";

    for(int i=0;i<n;i++){

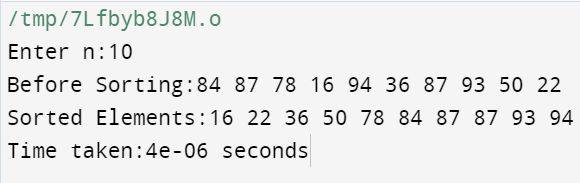
        cout<<arr[i]<<" ";

    }

    cout<<endl<<"Time taken:"<<difftime(end,start)/CLOCKS\_PER\_SEC<<" seconds";

}

**Output:-**



**Graph:-**

