Design & Analysis of Algorithm Lab [KCS-503]

ACADEMIC YEAR 2022-2023

Department of Computer Science and Engineering

Bundelkhand Institute of Engineering and Technology

Jhansi (U.P.)

Submitted to:

Er. Shashank Gupta

Submitted by:

Abhishek Mittal

Roll No.2004310006

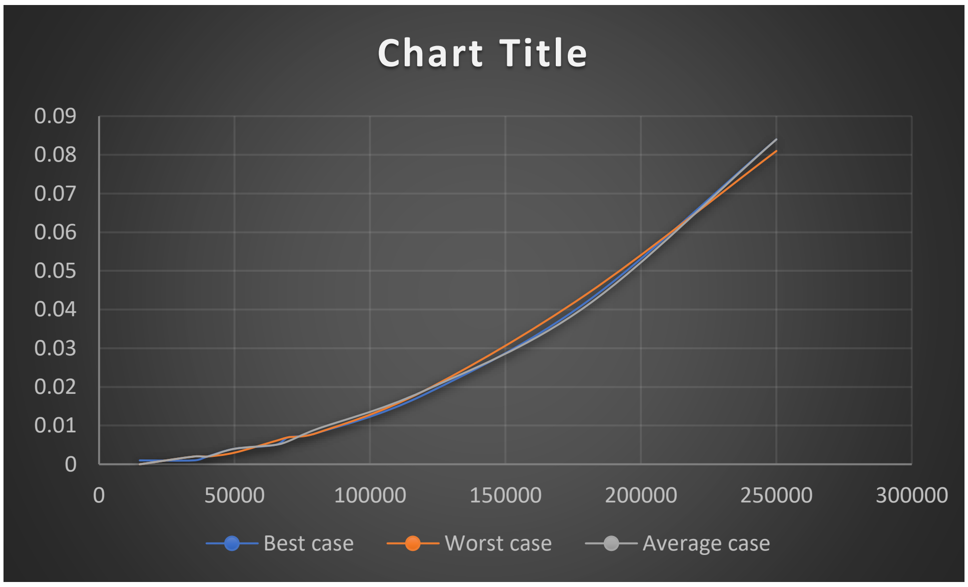
INDEX

Name: Abhishek Mittal **Roll No. 2004310006**

Bubble Sort

```
#include<bits/stdc++.h>
using namespace std;
int main(){
freopen("random.txt","r",stdin);
long long int range[]={250000,180000,120000,80000,70000,65000,50000,40000,35000,15000};
 for(int k=0;k<10;k++){
 long long int n=range[k];
 long long int a[n];
 for(int i=0;i<n;i++) {
 cin>>a[i];
 }
 time_t first=time(NULL);
 double total_t;
for(long long int i=0;i<n-1;i++){
for(long long int j=0;j<n-i-1;j++){
if(a[j]>a[j+1]){
long long int temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
time_t second =time(NULL);
total_t=(double)(second-first)/CLOCKS_PER_SEC;
//t[k]=total_t;
cout<<total_t<<" ";
}
}
```

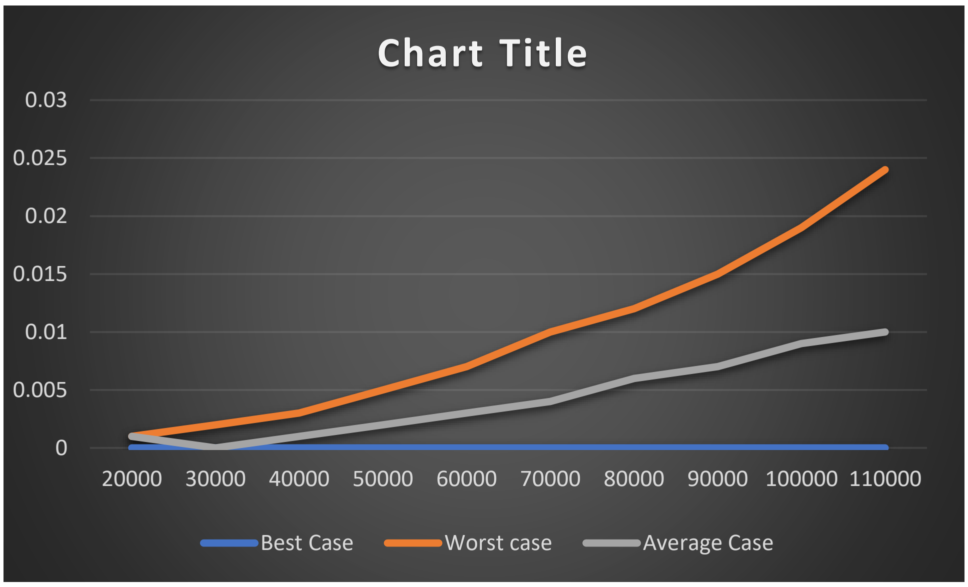
Range	Best case	Worst case	Average case
250000	0.084	0.081	0.084
180000	0.042	0.044	0.041
120000	0.018	0.019	0.019
80000	0.008	0.008	0.009
70000	0.007	0.007	0.006
65000	0.005	0.006	0.005
50000	0.004	0.003	0.004
40000	0.002	0.002	0.002
35000	0.001	0.002	0.002
15000	0.001	0	0



Insertion Sort

```
#include<bits/stdc++.h>

using namespace std;

int main()

{

freopen("random.txt","r",stdin);

long long int ran[]={20000,30000,40000,50000,60000,70000,80000,90000,100000,110000};

for(int k=0;k<10;k++)

{

long long int n=ran[k];

long long int arr[n];

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

cin>>arr[i];

time_t t1=time(NULL);

int i, key, j;

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

    {

        key = arr[i];

        j = i - 1;

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

        {

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

            j = j - 1;

        }

        arr[j + 1] = key;

    }


time_t t2=time(NULL);

double ts=(double)(t2-t1)/CLOCKS_PER_SEC;

cout<<ts<<" ";

}

}
```

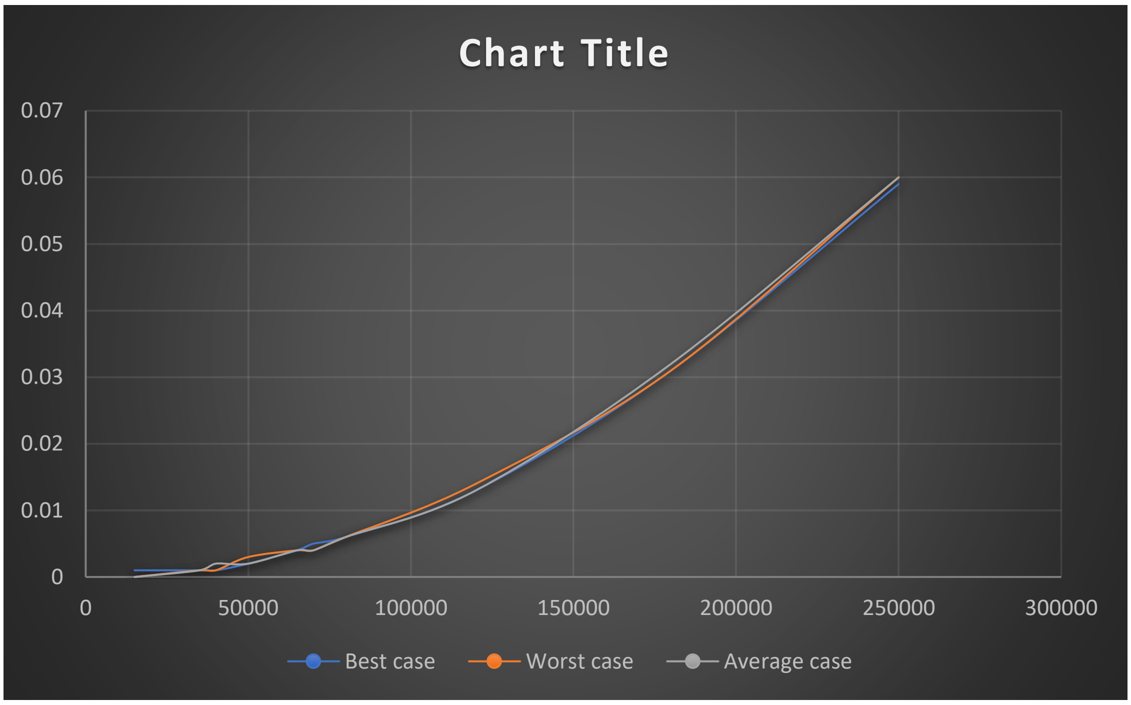
Range	Best case	Worst case	Average case
20000	0	0.001	0.001
30000	0	0.002	0
40000	0	0.003	0.001
50000	0	0.005	0.002
60000	0	0.007	0.003
70000	0	0.01	0.004
80000	0	0.012	0.006
90000	0	0.015	0.007
100000	0	0.019	0.009
120000	0	0.024	0.01



Selection Sort

```
#include<bits/stdc++.h>
using namespace std;
int main(){
 freopen("random.txt","r",stdin);
long long int range[]={250000,180000,120000,80000,70000,65000,50000,40000,35000,15000};
 for(int k=0;k<10;k++){
 long long int n=range[k];
 long long int a[n];
 for(int i=0;i<n;i++) {
 cin>>a[i];
 }
 int min_idx;
 time_t first=time(NULL);
 double total_t;
 for(int i=0;i<n;i++){
 min_idx=i;
 for(int j=i+1;j<n;j++){
 if(a[j]<a[min_idx]){
 min_idx=j;
 }
 }
 if(min_idx!=i){
 swap(a[min_idx],a[i]);
 }
}
 time_t second =time(NULL);
 total_t=(double)(second-first)/CLOCKS_PER_SEC;
cout<<total_t<<" ";
}
}
```

Range	Best case	Worst case	Average case
250000	0.059	0.06	0.06
180000	0.031	0.031	0.032
120000	0.013	0.014	0.013
80000	0.006	0.006	0.006
70000	0.005	0.004	0.004
65000	0.004	0.004	0.004
50000	0.002	0.003	0.002
40000	0.001	0.001	0.002
35000	0.001	0.001	0.001
15000	0.001	0	0

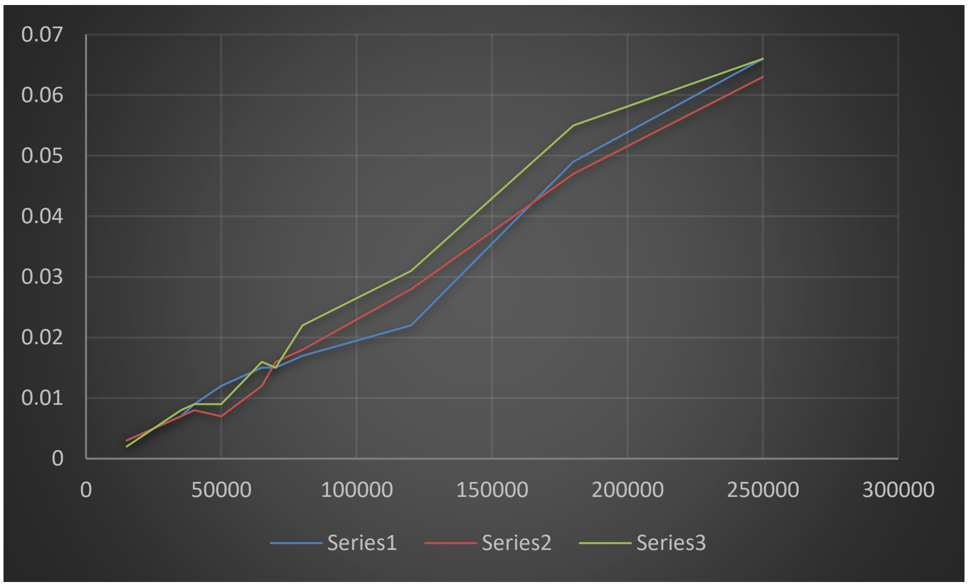


Heap Sort

```
#include<bits/stdc++.h>          //Abhishek Mittal

#include <chrono>
using namespace std;
void heapify(long long int arr[], long long int N, long long int i)
{
        //this is the heapify function
         long long int largest = i;
        long long int l = 2 * i + 1;
        long long int r = 2 * i + 2;
        if (l < N && arr[l] > arr[largest])
            largest = l;
        if (r < N && arr[r] > arr[largest])
            largest = r;
        if (largest != i) {
         swap(arr[i], arr[largest]);
                heapify(arr, N, largest);
   }
}
void heapSort(long long int arr[], long long int N)
{
        for (long long int i = N / 2 - 1; i >= 0; i--)
          heapify(arr, N, i);
        for (long long int i = N - 1; i > 0; i--) {
          swap(arr[0], arr[i]);
                heapify(arr, i, 0);
   }
}
void printarr(long long int a[], long long int n)
{
        for(long long int j=0; j<n; j++)
        cout<<a[j]<<" ";
}
```

```
int main(){ freopen("file2.txt","r",stdin);
long long int range[]={250000,180000,120000,80000,70000,65000,50000,40000,35000,15000};
 for(long long int k=0;k<10;k++){long long int
 n=range[k];
 long long int a[n];
 for(long long int i=0;i<n;i++)cin>>a[i];

auto start = chrono::steady_clock::now();
//heap sort starts hereheapSort(a,n);
auto end = chrono::steady_clock::now(); cout<<chrono::duration_cast<chrono::seconds>(end-start).count()<<" ";
}
}
```

Range	Best case (ms)	Worst case (ms)	Average case (ms)
250000	0.066	0.063	0.066
180000	0.049	0.047	0.055
120000	0.022	0.028	0.031
80000	0.017	0.018	0.022
70000	0.015	0.016	0.015
65000	0.015	0.012	0.016
50000	0.012	0.007	0.009
40000	0.009	0.008	0.009
35000	0.007	0.007	0.008
15000	0.003	0.003	0.002

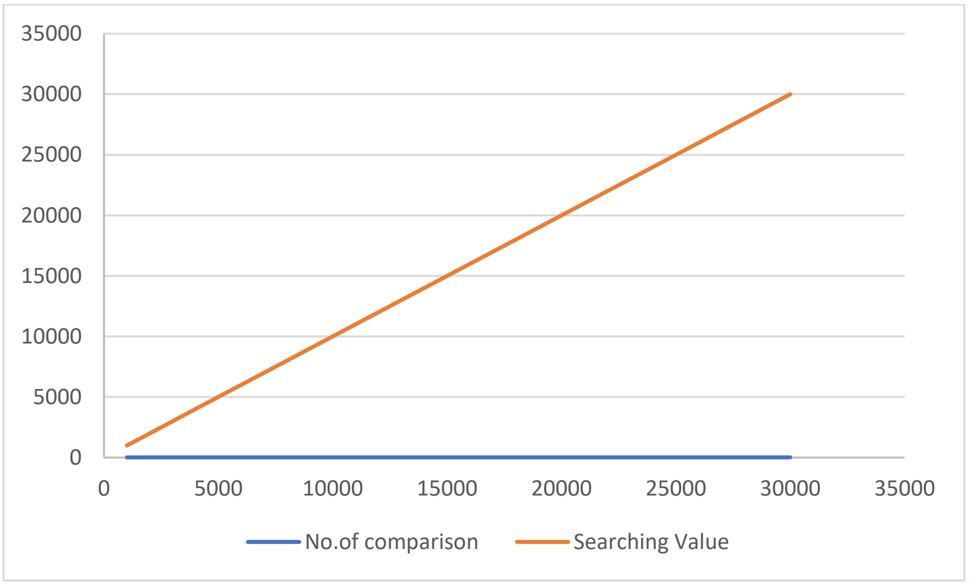


Linear Search

Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
 freopen("file1.txt", "r", stdin);
 int size[]={30000,25000,20000,15000,10000,5000,4000,3000,2000,1000};
 for(int i=0;i<10;i++)
 {
 int k=size[i];
 int a[k];
 for(int j=0;j<k;j++){
 cin>>a[j];
 }
 int x=a[k-1];

 int count=0;
 for( int p=0;p<k;p++)
 {
 count++;
 if(a[p]==x){
 break;
 }
 }
 cout<<"no of comparison"<<" "<<count<<endl;
}
 return 0;
}
```

Range	No.of comparison	Searching Value
30000	30000	30000
25000	25000	25000
20000	20000	20000
15000	15000	15000
10000	10000	10000
5000	5000	5000
4000	4000	4000
3000	3000	3000
2000	2000	2000
1000	1000	1000

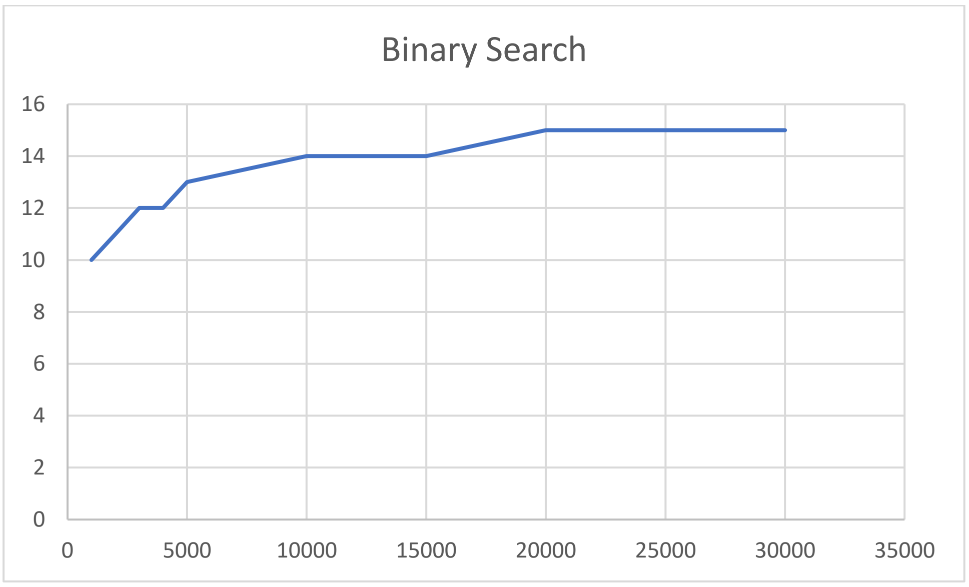


Binary Search

Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
 freopen("file1.txt", "r", stdin);
 int size[]={30000,25000,20000,15000,10000,5000,4000,3000,2000,1000};
 for(int i=0;i<10;i++)
 {
 int k=size[i];
 int a[k];
 for(int j=0;j<k;j++){
 cin>>a[j];
 }
 int x=a[k-1];

 int count=0;
 int l=0;
 int r=k-1;
while(l<=r)
 {
 int mid=(l+r)/2;
 count++;
 if(a[mid]==x)
 break;
 if(a[mid]>x)
 r=mid-1;
 else l=mid+1;
 }
 cout<<" "<<"no of comparison "<<count<<endl;
}
 return 0;
}
```

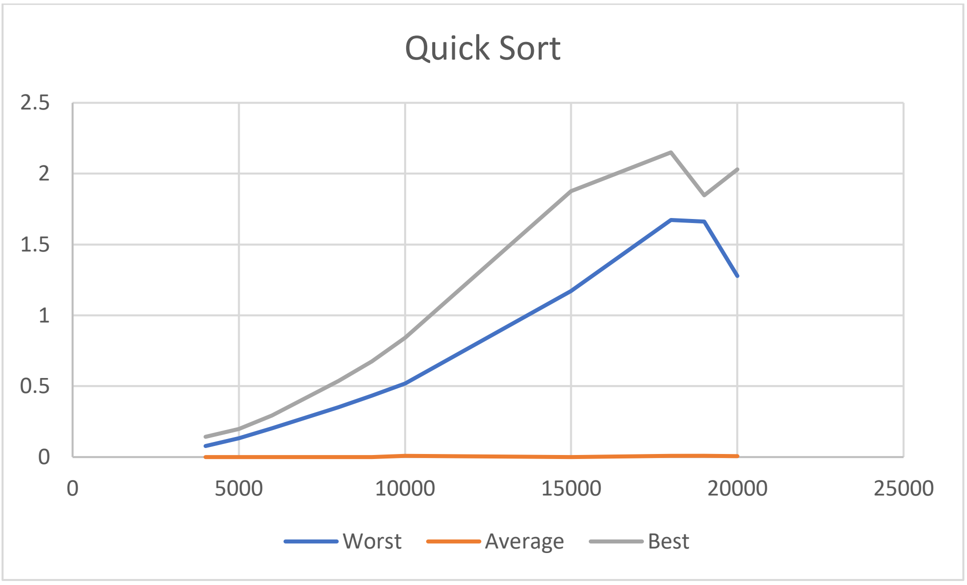
Range	No.of comparison	Searching Value
30000	15	30000
25000	15	25000
20000	15	20000
15000	14	15000
10000	14	10000
5000	13	5000
4000	12	4000
3000	12	3000
2000	11	2000
1000	10	1000



Quick Sort

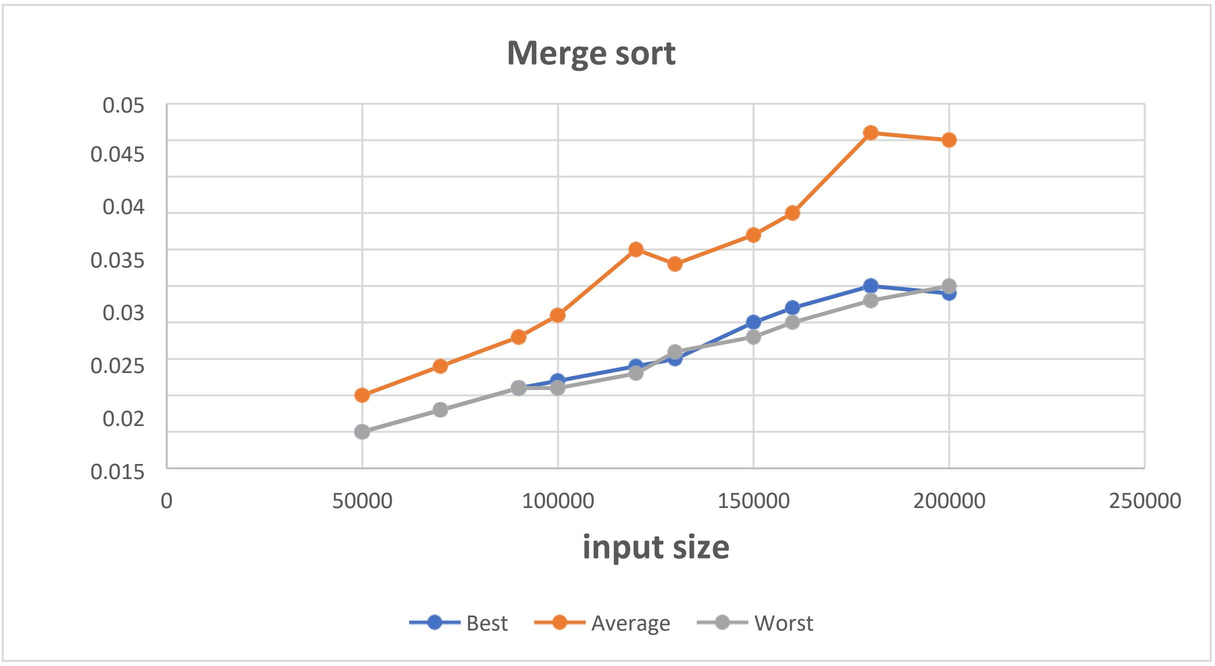
```
#include<bits/stdc++.h>    //Abhishek Mittal

using namespace std;

// quick sort

int partition( vector<int>&v, int low, int high)

{

int pivot =v[high]; int i= low-1;


for (int j = low; j <= high - 1; j++) { if (v[j] < pivot) {

i++;

swap(v[i],v[j]);

}

}

swap(v[i + 1],v[high]); return i + 1;

}


void quickSort(vector<int>&v, int low, int high)

{

if (low < high) {


int pi = partition(v, low, high); quickSort(v, low, pi - 1); quickSort(v, pi + 1, high);

}

}


int main()

{

freopen("file2.txt","r",stdin); vector<int> v2(100000);

for(int i=0;i<100000;i++) cin>>v2[i];



int range[10]={4000,5000,6000,8000,9000,10000,15000,18000,19000,20000};
for(int i=0;i<10;i++)

{           clock_t ti,tf;

int inputs=range[i]; vector<int>v=v2; ti=clock();

quickSort(v, 0,inputs-1); tf=clock();

double tt=double(tf-ti)/CLOCKS_PER_SEC; cout<<tt<<endl;

}

}
```

Range	Worst	Average	Best
4000	0.078	0	0.143
5000	0.132	0	0.199
6000	0.202	0	0.294
8000	0.352	0	0.537
9000	0.434	0	0.674
10000	0.52	0.008	0.842
15000	1.171	0	1.877
18000	1.673	0.008	2.15
19000	1.661	0.009	1.848
20000	1.277	0.007	2.031



Merge Sort

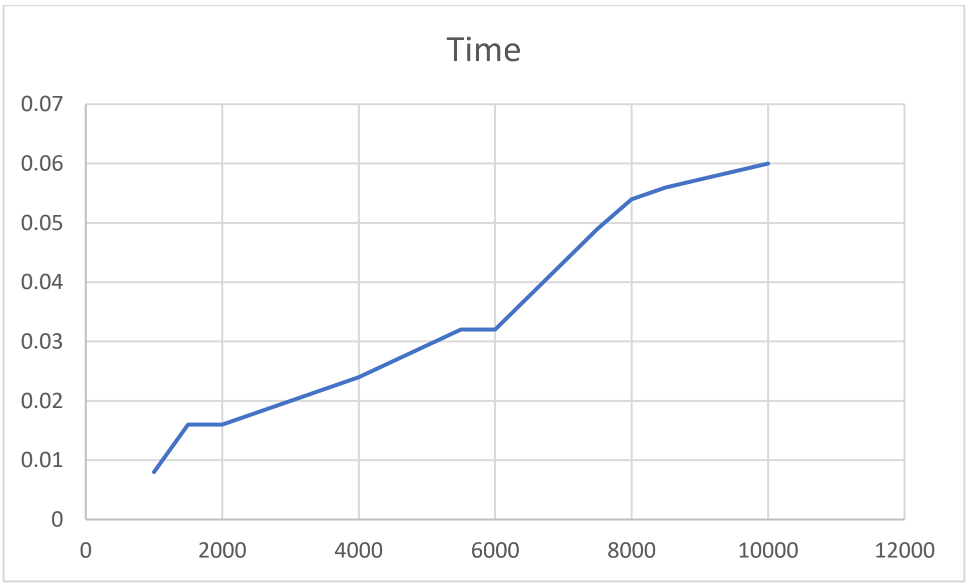
```
#include<bits/stdc++.h>   // Abhishel Mittal
using namespace std;
 void merge(vector<int>&v,int low,int mid,int high){
            int b[high-low+1];
            int i=low;
            int j=mid+1;
            int k=0;
            while(i<=mid && j<=high){
                        if(v[i]<=v[j]){
                                    b[k++]=v[i++];
                         }
                         else{
                                    b[k++]=v[j++];
                         }
             }
             while(i<=mid){
                        b[k++]=v[i++];
             }
             while(j<=high){
                        b[k++]=v[j++];
             }
             for(int i=0;i<=high-low;i++){
                        v[i+low]=b[i];
             }
 }
 void mergesort(vector<int>&v,int low,int high){
if(low<high){
            int mid=(low+high)/2;
            mergesort(v,low,mid);
            mergesort(v,mid+1,high);
            merge(v,low,mid,high);
 }
 }
int main()
{
            freopen("random.txt","r",stdin);
            vector<int> v1(210000);
            for(int i=0;i<210000;i++)
            cin>>v1[i];



            int
range[10]={50000,70000,90000,100000,120000,130000,150000,160000,180000,200000};
            for(int i=0;i<10;i++)
            {   clock_t ti,tf;
                        int inputs=range[i];
                        vector<int>v=v1;
                        ti=clock();
                        mergesort(v, 0,inputs-1);
                        tf=clock();
                        double tt=double(tf-ti)/CLOCKS_PER_SEC;
                        cout<<tt<<endl;
            }
}
```


Merge sort
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0
50000
100000
150000
200000
250000
input size
Best
Average
Worst

Counting Sort

```
# include<bits/stdc++.h>
using namespace std;
void countSort(vector<int>& arr)
{ int s=arr.size(),k=9;
   int max = *max_element(arr.begin(), arr.end());
int min = *min_element(arr.begin(), arr.end());
int range = max - min + 1;
 vector<int> count(range), output(arr.size());
for (int i = 0; i < arr.size(); i++)
count[arr[i] - min]++;
for (int i = 1; i < count.size(); i++)
count[i] += count[i - 1];
for (int i = arr.size() - 1; i >= 0; i--) {
output[count[arr[i] - min] - 1] = arr[i];
count[arr[i] - min]--;
}
 for (int i = 0; i < arr.size(); i++)
arr[i] = output[i];
}
void printArray(vector<int>& arr)
{
for (int i = 0; i < arr.size(); i++)
cout << arr[i] << " ";
cout << "\n";
}
int main()
{ freopen("count.txt","r",stdin);
vector<int> v1(100000);
for(int i=0;i<100000;i++)
cin>>v1[i];
int range[10]={15000,25000,35000,40000,50000,60000,75000,80000,95000,99000};

for(int i=0;i<10;i++)

{ clock_t ti,tf;

int inputs=range[i];

vector<int>v=v1;

v.resize(inputs);

ti=clock();

countSort(v);

tf=clock();

double tt=double(tf-ti)/CLOCKS_PER_SEC;

cout<<tt<<endl;

}

}
```

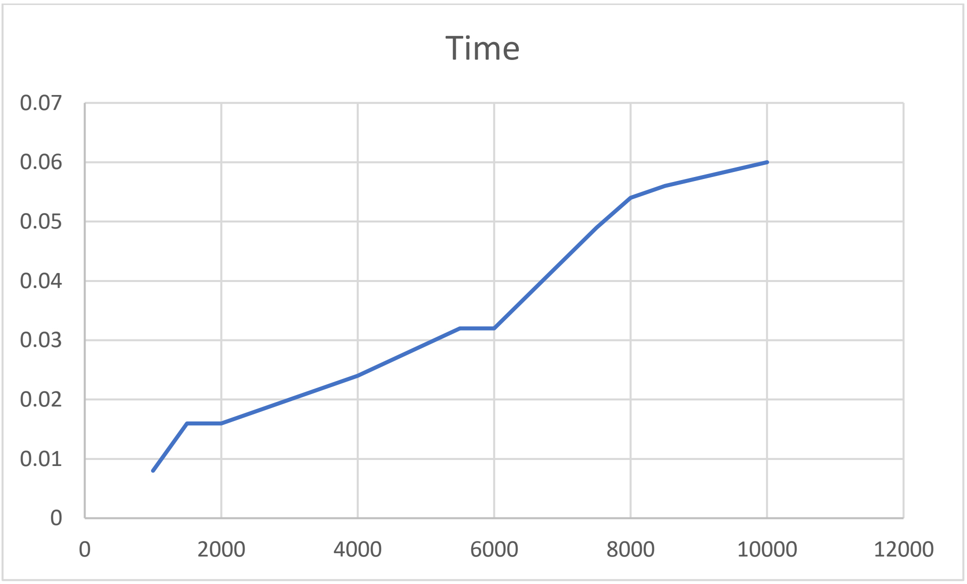
Range	Time
15000	0
25000	0.01
35000	0.002
40000	0.003
50000	0.005
60000	0.006
75000	0.008
80000	0.008
95000	0.008
99000	0.008



Radix Sort

```
#include<bits/stdc++.h>

#include<iostream>

using namespace std;

// radix sort

void print(vector<string>& str, int n)

{

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

  cout << str[k] << " ";

}

  cout <<endl;

}

int char_at(string str, int s)

{

  if (str.size() <= s)

  return -1;

  else

  return str.at(s);

}

void radixsort(vector<string>& str, int low, int high, int s)

{


if (high <= low) {

return;

}

int count[256 + 2] = { 0 };

unordered_map<int, string> temp;

for (int i = low; i <= high; i++) {

int c = char_at(str[i], s);

count[c+2]++;

}
for (int r = 0; r < 256 + 1; r++)
count[r + 1] += count[r];
for (int i = low; i <= high; i++) {
int c = char_at(str[i], s);
temp[count[c+1]++] = str[i];
}
for (int i = low; i <= high; i++)
str[i] = temp[i - low];
for (int r = 0; r < 256; r++)
radixsort(str, low + count[r], low + count[r + 1] - 1,s + 1);
}
int main()
{
freopen("radix.txt","r",stdin);
vector<string> v1(100000);
for(int i=0;i<100000;i++)
cin>>v1[i];
int range[10]={1000,1500,2000,4000,5500,6000,7500,8000,8500,10000};
for(int i=0;i<10;i++)
{ clock_t ti,tf;
int inputs=range[i];
vector<string>v=v1;
v.resize(inputs);
ti=clock();
radixsort(v,0,inputs-1,0);
tf=clock();
double tt=double(tf-ti)/CLOCKS_PER_SEC;

cout<<" "<<tt<<endl;
}
}
```

Range	Time
1000	0.008
1500	0.016
2000	0.016
4000	0.024
5500	0.032
6000	0.032
7500	0.049
8000	0.054
8500	0.056
10000	0.06

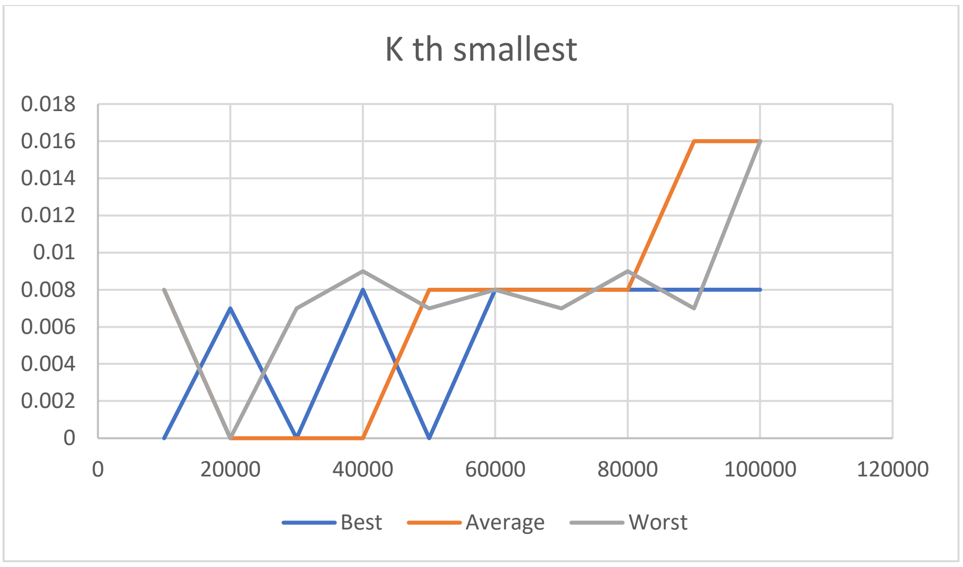


Kth smallest Element

```
#include<bits/stdc++.h>
using namespace std;
int partition(int arr[], int l, int r, int k);
int findMedian(int arr[], int n)
{
        sort(arr, arr+n);
        return arr[n/2];
}
int kthSmallest(int arr[], int l, int r, int k)
{
        if (k > 0 && k <= r - l + 1)
        {
                int n = r-l+1;
                int i, median[(n+4)/5];
                for (i=0; i<n/5; i++)
                        median[i] = findMedian(arr+l+i*5, 5);
                if (i*5 < n)
                {
                        median[i] = findMedian(arr+l+i*5, n%5);
                        i++;
                }
                int medOfMed = (i == 1)? median[i-1]:kthSmallest(median, 0, i-1, i/2);
                int pos = partition(arr, l, r, medOfMed);
                if (pos-l == k-1)
                        return arr[pos];
                if (pos-l > k-1)
                        return kthSmallest(arr, l, pos-1, k);
         return kthSmallest(arr, pos+1, r, k-pos+l-1);
        }
        return INT_MAX;
}
void swap(int *a, int *b)
{
        int temp = *a;
        *a = *b;
        *b = temp;
}int partition(int arr[], int l, int r, int x)
{
        int i;
        for (i=l; i<r; i++)
                if (arr[i] == x)
                break;   swap(&arr[i], &arr[r]);
        i = l;
        for (int j = l; j <= r - 1; j++)
        {if (arr[j] <= x)
                {swap(&arr[i], &arr[j]);
```

```
                        i++;
                    }
            }
            swap(&arr[i], &arr[r]);
            return i;
}
int main(){
            freopen("random.txt","r",stdin);
            vector<int> v1(100000);
            for(int i=0;i<100000;i++){
            cin>>v1[i];
}
            int Range[10]={100000,90000,80000,70000,60000,50000,40000,30000,20000,10000};
            for(int i=0;i<10;i++)
            {   clock_t start,end;
                        int tests=Range[i];
                        int arr[tests];
                        for(int i=0;i<tests;i++){
                                    arr[i]=v1[i];
                        }
                        start=clock();
                        int k=17;
                        int ans=kthSmallest(arr,0,tests-1,k);
                        end=clock();
                        double total_time=double(end-start)/CLOCKS_PER_SEC;
                        cout<<"time taken for "<<tests<<" inputs is : "<<total_time<<endl;
}
}
```

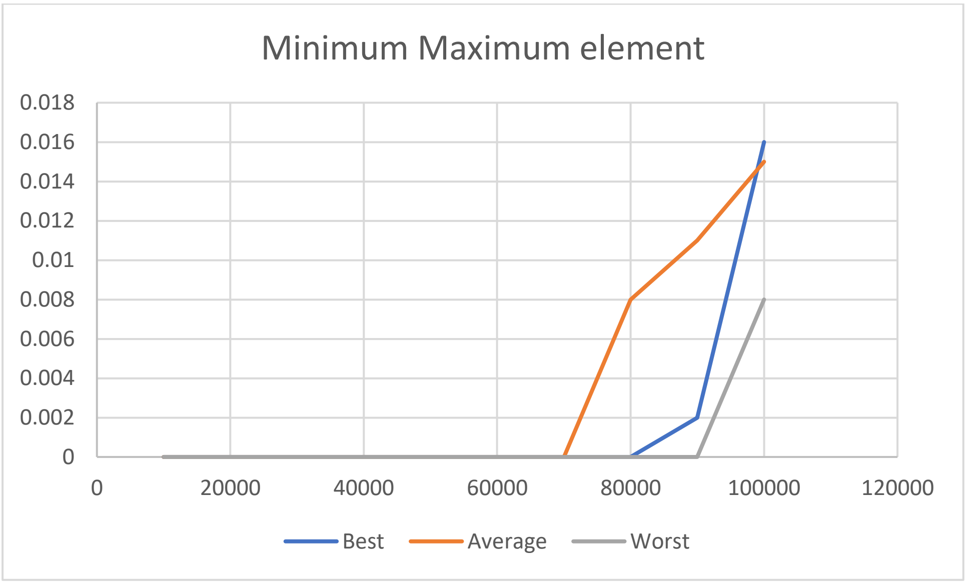
Range	Best	Average	Worst
100000	0.008	0.016	0.016
90000	0.008	0.016	0.007
80000	0.008	0.008	0.009
70000	0.008	0.008	0.007
60000	0.008	0.008	0.008
50000	0	0.008	0.007
40000	0.008	0	0.009
30000	0	0	0.007
20000	0.007	0	0
10000	0	0.008	0.008


K th smallest
0.018
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
20000
40000
60000
80000
100000
120000
Best
Average
Worst

Find the minimum and maximum element

```
#include<bits/stdc++.h>
using namespace std;
struct Pair
{
	int min;
	int max;
};
struct Pair getMinMax(vector<int>&arr, int n)
{
	struct Pair minmax;
	int i;
	if (n % 2 == 0)
	{
		if (arr[0] > arr[1])
		{
			minmax.max = arr[0];
			minmax.min = arr[1];
		}
		else
		{
			minmax.min = arr[0];
			minmax.max = arr[1];
		}
		i = 2;
	}
	else
	{
		minmax.min = arr[0];
		minmax.max = arr[0];
		i = 1;
	}
	while (i < n - 1)
	{
		if (arr[i] > arr[i + 1])
		{
			if(arr[i] > minmax.max)
				minmax.max = arr[i];
			if(arr[i + 1] < minmax.min)
				minmax.min = arr[i + 1];
		}
		else
		{
			if (arr[i + 1] > minmax.max)
				minmax.max = arr[i + 1];
			if (arr[i] < minmax.min)
				minmax.min = arr[i];
		}
```

```
                    i += 2;
          }
return minmax
}
int main(){
          freopen("File2.txt","r",stdin);
          vector<int> v1(100000);
          for(int i=0;i<100000;i++){
          cin>>v1[i];
}
int Range[10]={100000,90000,80000,70000,60000,50000,40000,30000,20000,10000};
          for(int i=0;i<10;i++)
          {   clock_t start,end;
                    int tests=Range[i];
                    vector<int>v=v1;
                    start=clock();
                    Pair minmax=getMinMax(v,tests-1);
                    end=clock();
                    double total_time=double(end-start)/CLOCKS_PER_SEC;
                    cout<<"time taken for "<<tests<<" inputs is : "<<total_time<<endl;
}
}
```



Experiment-06

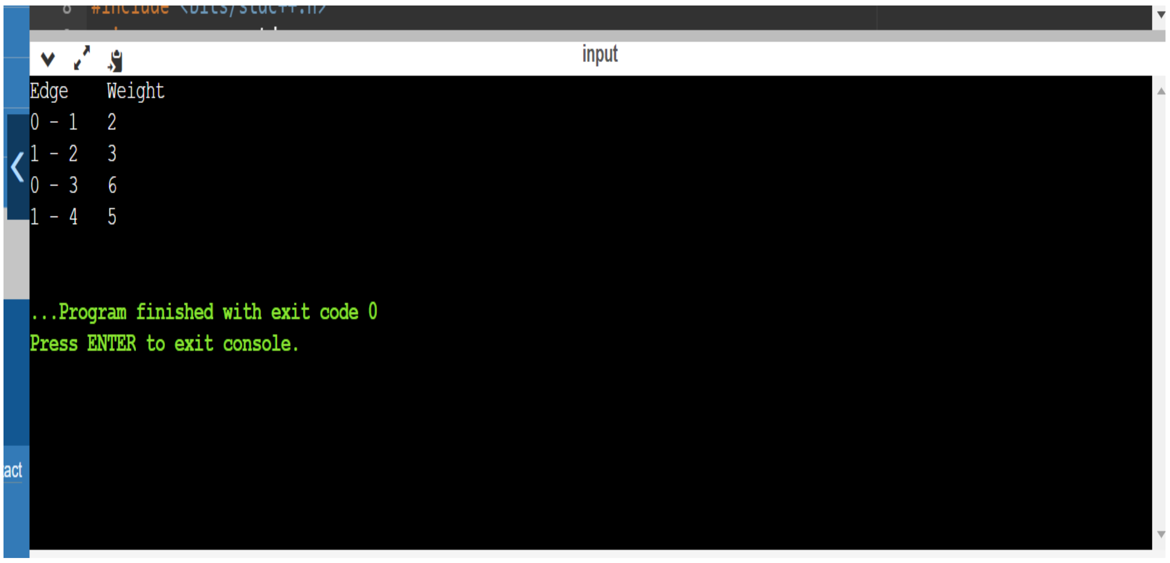
Objective: To Implement Minimum Spanning Tree using prims and Kruskal Algorithm.

Prims:

Code

```
#include <bits/stdc++.h>
using namespace std;
#define V 5
int minKey(int key[], bool mstSet[])
{
        int min = INT_MAX, min_index;
        for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min) {
         min = key[v];
         min_index = v;
        }
        return min_index;
         }
void printMST(int parent[], int graph[V][V]){
 cout << "Edge \tWeight\n";
for (int i = 1; i < V; i++)
 cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << " \n";
}
void primMST(int graph[V][V]){
 int parent[V];
int key[V];
bool mstSet[V];
        for (int i = 0; i < V; i++)
                key[i] = INT_MAX, mstSet[i] = false;
        key[0] = 0;
        parent[0] = -1; // First node is always root of MST
        for (int count = 0; count < V - 1; count++) {
                int u = minKey(key, mstSet);
                mstSet[u] = true;
                for (int v = 0; v < V; v++)
        if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                                parent[v] = u,
                   key[v] = graph[u][v];
}
        printMST(parent, graph);
}
int main()
{
        int graph[V][V] = { { 0, 2, 0, 6, 0 },
                                        { 2, 0, 3, 8, 5 },
                                        { 0, 3, 0, 0, 7 },
                                        { 6, 8, 0, 0, 9 },
                                        { 0, 5, 7, 9, 0 } };
        primMST(graph);
        return 0;
}
```

OUTPUT



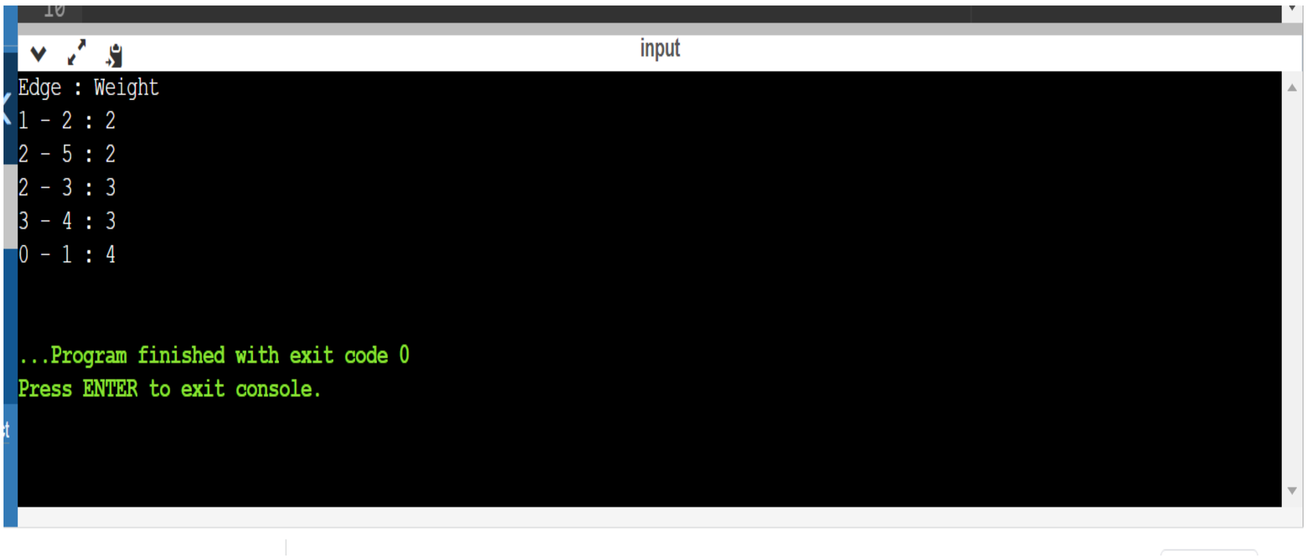
```
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5


...Program finished with exit code 0
Press ENTER to exit console.
```

Kruskal

Code:

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

#define edge pair<int, int>

class Graph {
   private:
  vector<pair<int, edge> > G;  // graph
  vector<pair<int, edge> > T;  // mst
  int *parent;
  int V;  // number of vertices/nodes in graph
   public:
  Graph(int V);
  void AddWeightedEdge(int u, int v, int w);
  int find_set(int i);
  void union_set(int u, int v);
  void kruskal();
  void print();
};
Graph::Graph(int V) {
  parent = new int[V];

  //i 0 1 2 3 4 5
  //parent[i] 0 1 2 3 4 5
  for (int i = 0; i < V; i++)
    parent[i] = i;
  G.clear();
  T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w) {
  G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i) {
  // If i is the parent of itself
  if (i == parent[i])
    return i;
  else
    // Else if i is not the parent of itself
    // Then i is not the representative of his set,
    // so we recursively call Find on its parent
    return find_set(parent[i]);
}

void Graph::union_set(int u, int v) {
  parent[u] = parent[v];
}
void Graph::kruskal() {
  int i, uRep, vRep;
  sort(G.begin(), G.end());  // increasing weight
  for (i = 0; i < G.size(); i++) {
    uRep = find_set(G[i].second.first);
    vRep = find_set(G[i].second.second);
    if (uRep != vRep) {
      T.push_back(G[i]);  // add to tree
      union_set(uRep, vRep);
    }
  }
}
void Graph::print() {
  cout << "Edge :"
     << " Weight" << endl;
  for (int i = 0; i < T.size(); i++) {
    cout << T[i].second.first << " - " << T[i].second.second << " : "
       << T[i].first;
    cout << endl;
  }
}
int main() {
  Graph g(6);
  g.AddWeightedEdge(0, 1, 4);
  g.AddWeightedEdge(0, 2, 4);
  g.AddWeightedEdge(1, 2, 2);
  g.AddWeightedEdge(1, 0, 4);
  g.AddWeightedEdge(2, 0, 4);
  g.AddWeightedEdge(2, 1, 2);
  g.AddWeightedEdge(2, 3, 3);
  g.AddWeightedEdge(2, 5, 2);
  g.AddWeightedEdge(2, 4, 4);
  g.AddWeightedEdge(3, 2, 3);
  g.AddWeightedEdge(3, 4, 3);
  g.AddWeightedEdge(4, 2, 4);
  g.AddWeightedEdge(4, 3, 3);
  g.AddWeightedEdge(5, 2, 2);
  g.AddWeightedEdge(5, 4, 3);
  g.kruskal();
  g.print();
  return 0;
}
```

OUTPUT:



```
Edge : Weight
1 - 2 : 2
2 - 5 : 2
2 - 3 : 3
3 - 4 : 3
0 - 1 : 4


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment-07

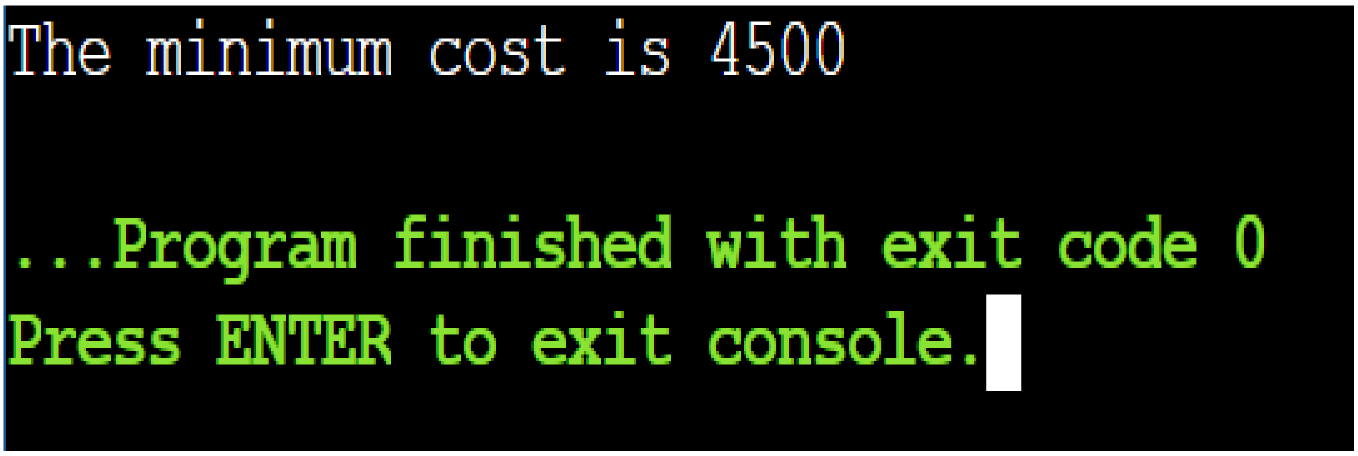
Objective: To implement Matrix Chain Multiplication and Longest Common Subsequence.

Matrix chain Multiplication:

Code:

```
#include <bits/stdc++.h>

using namespace std;

#define MAX 10

int look_up[MAX][MAX];

int mcm(int dims[], int i, int j)

{

    // A base case for one matrix;

    if (j <= i + 1) {

      return 0;

    }

    int min = INT_MAX;

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

    {

        for (int k = i + 1; k <= j - 1; k++)

        {

            int cost = mcm(dims, i, k);

            cost += mcm(dims, k, j);

            cost += dims[i] * dims[k] * dims[j];
            if (cost < min) {
                min = cost;
            }
        }
        look_up[i][j] = min;
    }     return look_up[i][j];
}
int main()
{
    int dims[] = { 10, 30, 5, 60 };
    int n = sizeof(dims) / sizeof(dims[0]);
    cout << "The minimum cost is " << mcm(dims, 0, n - 1);
     return 0;   }
```

OUTPUT

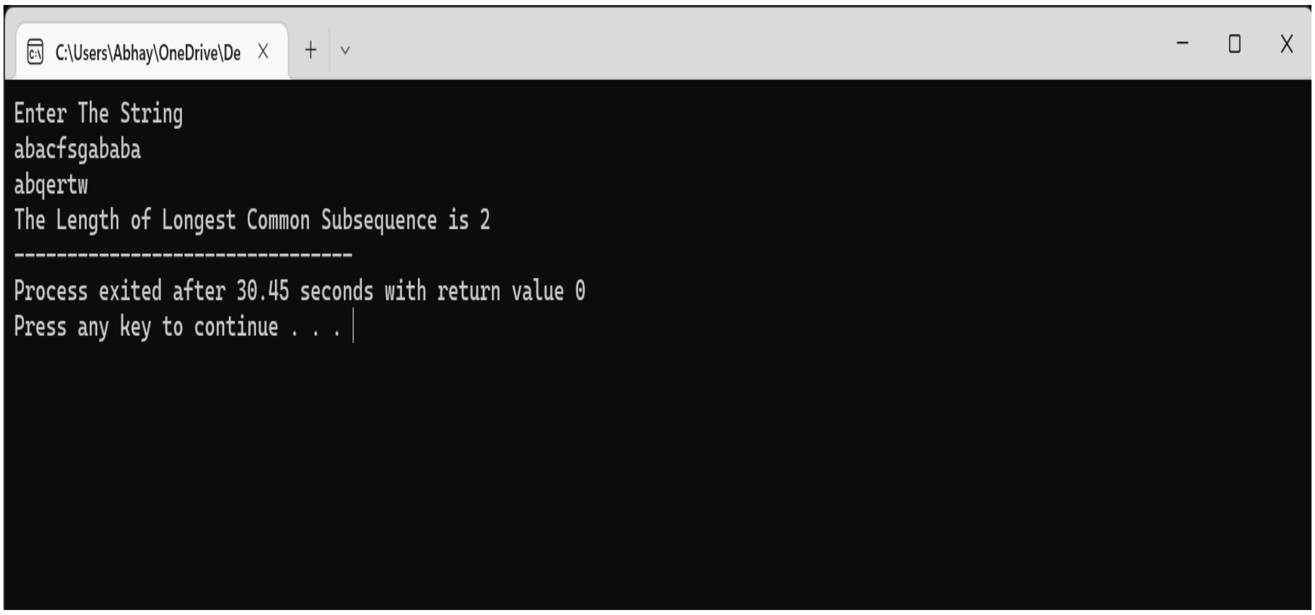


```
The minimum cost is 4500

...Program finished with exit code 0
Press ENTER to exit console.
```

Longest Common Subsequence

Code:

```
#include<bits/stdc++.h>
using namespace std;
int LCS( string A, string B, int m, int n )
{
   int L[m+1][n+1];
   int i, j;
   for(i =0; i<=m; i++){
      for(j=0; j<=n; j++){
         if( i==0|| j==0)
             L[i][j] = 0;
         else if(A[i-1] == B[j-1]){
            L[i][j] = L[i-1][j-1] + 1;
         }
         else{
            L[i][j] = max(L[i-1][j], L[i][j-1]);
         }
      }
   }
   return L[m][n];
}
int main()
{
string A;
string B;
cout<<"Enter The String"<<endl;
cin>>A>>B;
int x = A.size();
int y = B.size();
printf("The Length of Longest Common Subsequence is %d", LCS( A, B, x, y ) );
return 0;
}
```



```
Enter The String
abacfsgababa
abqertw
The Length of Longest Common Subsequence is 2
--------------------------------
Process exited after 30.45 seconds with return value 0
Press any key to continue . . .
```

Experiment-08

Objective: Case Study of P, NP, NP complete and NP Hard Problems.

Theory:

P-Class

The P in the P class stands for **Polynomial Time.** It is the collection of decision problems(problems with a “yes” or “no” answer) that can be solved by a deterministic machine in polynomial time.

Features:

1. The solution to P problems is easy to find.
2. P is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

NP Class

The class NP consists of those problems that are verifiable in polynomial time. NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the aid of a little extra information. Hence, we aren't asking for a way to find a solution, but only to verify that an alleged solution really is correct.

Every problem in this class can be solved in exponential time using exhaustive search.

NP Complete

A problem X is NP-Complete if there is an NP problem Y, such that Y is reducible to X in polynomial time. NP-Complete problems are as hard as NP problems.

A problem is NP-Complete if it is a part of both NP and NP-Hard Problem. A non-deterministic Turing machine can solve NP-Complete problem in polynomial time.

NP Hard

Intuitively, these are the problems that are *at least as hard as the NP-complete problems*. Note that NP-hard problems *do not have to be in NP, and they do not have to be decision problems.*

The precise definition here is that *a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time*.

But since any NP-complete problem can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-hard problem in polynomial time. Then, if there is a solution to one NP-hard problem in polynomial time, there is a solution to all NP problems in polynomial time.