

Design and Analysis of Algorithm

Lab [KCS -553]

FOR

ACADEMIC YEAR 2022-2023



**BUNDELKHAND INSTITUTE OF ENGINEERING
AND TECHNOLOGY, JHANSI**

**(BIET JHANSI)-
284128**

Submitted to: -

Er.Shashank Gupta

Submitted by: -

Saransh Gupta

Roll No. 2004310051

INDEX

Name: Saransh Gupta

Roll No.:2004310051

S.No.	Date	Aim/Objective	T. Sign
1		Sorting: Bubble, Selection and Insertion Sort Heap Sort	
2		Searching: Liner Search and Binary Search.	
3		Divide and conquer method: Merge Sort Quick Sort	
4		Sorting in linear time: Counting Sort, Radix Sort.	
5		Order Statistics: Maximum/Minimum element K^{th} Smallest element	
6		Greedy Method: Minimum Spanning Tree using Prim's Algorithm Minimum Spanning Tree using Kruskal's Algorithm	
7		Dynamic Programming: Matrix Chain Multiplication Longest Common Subsequence	
8		Case Study of P, NP, NP Complete and NP Hard Problems.	

OBJECTIVE - To implement quick sort

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

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

int partition(int array[], int low, int
high) {

int pivot = array[high];

int i = (low - 1);

for (int j = low; j < high; j++) {
if (array[j] <= pivot) {
    i++;

j
    swap(&array[i], &array[j]);
}
}

// swap pivot with the greater element
at i

swap(&array[i + 1], &array[high]);

// return the partition point
return (i + 1);
}

void quickSort(int array[], int low, int
high) {

if (low < high) {

// find the pivot element such that
// elements smaller than pivot are on
left of pivot

// elements greater than pivot are on
right of pivot

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

}
```

```

int main()

{

freopen("File2.txt","r",stdin);
vector<int> v1(400000);
for(int i=0;i<400000;i++)
cin>>v1[i];
int
Range[10]={20000,30000,40000,50000
,60000,70000,80000,90000,100000,110
000};

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

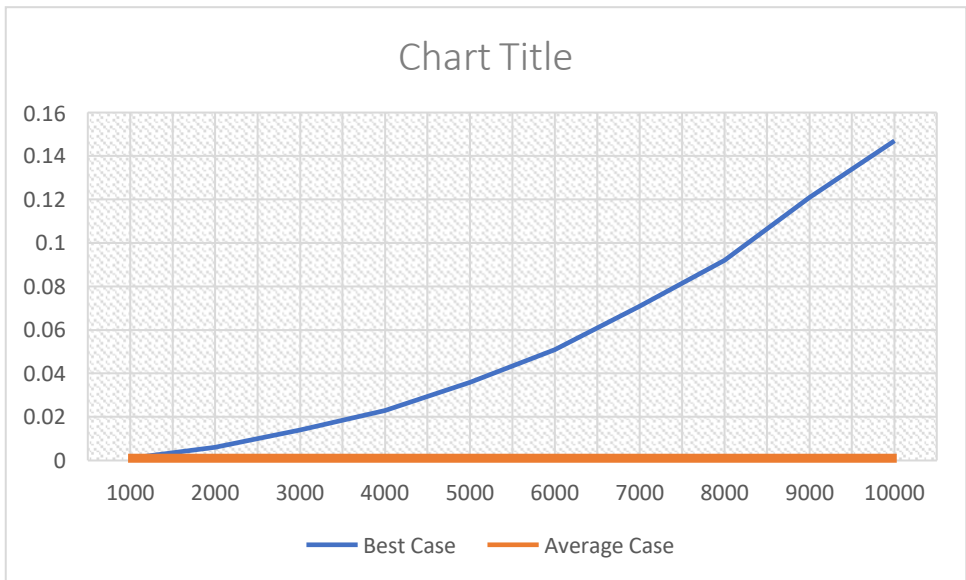
{ clock_t start,end;
int tests=Range[i];
vector<int>v=v1;
start=clock();
quickSort(data, 0, n - 1);
end=clock();
double total_time=double(end-
start)/CLOCKS_PER_SEC;

cout<<total_time<<" ";

}

```

Range	Best	Average
1000	0.001	0
2000	0.006	0
3000	0.014	0.001
4000	0.023	0
5000	0.036	0.001
6000	0.051	0.001
7000	0.071	0.002
8000	0.092	0.001
9000	0.121	0.002
10000	0.147	0.002



OBJECTIVE- To implement 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++)

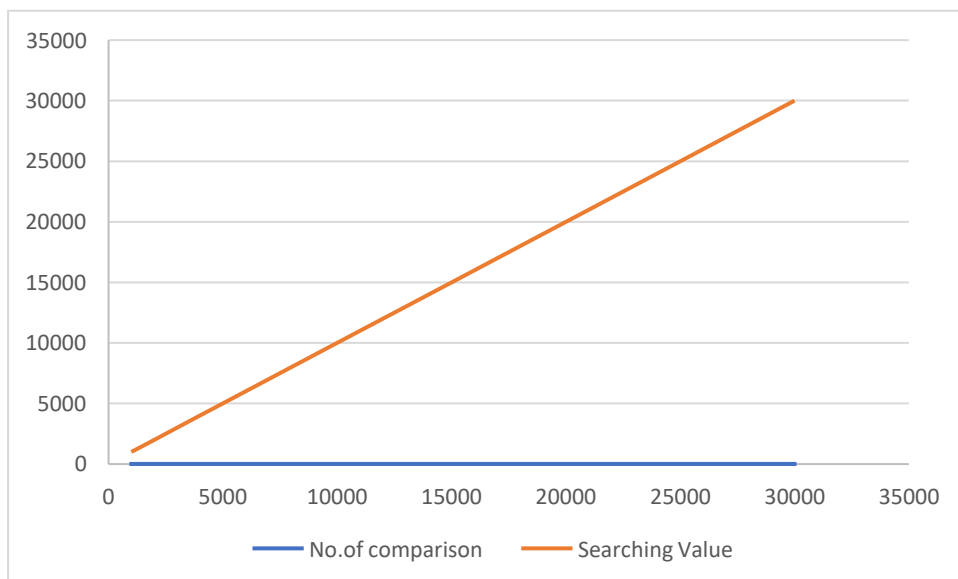
    { int k=size[i];int a[k];
  for(int
j=0;j<k;j++){ cin>>a[
j];
    }int x=a[k-1];
  int count=0;for( int p=0;p<k;p++){
count++;if(a[p]==x){ break; }}

  cout<<"no of comparison"<<" "<<count<<endl;}

  return 0;}
```

Input-

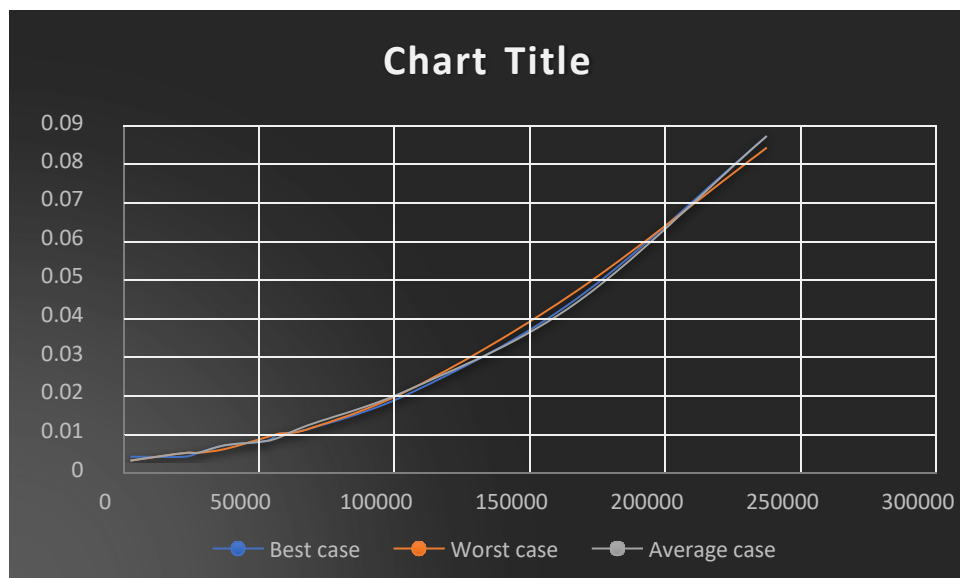
Range	No.of comparison	Searching Value
30000	30000	30000
25000	25000	25000
20000	20000	20000
15000	15000	15000
10000	10000	10000
5000	5000	5000
4000	4000	4000
3000	3000	3000
2000	2000	2000
1000	1000	1000



Objective - To implement 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(l 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_SE
C
//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



OBJECTIVE – To implement 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++){

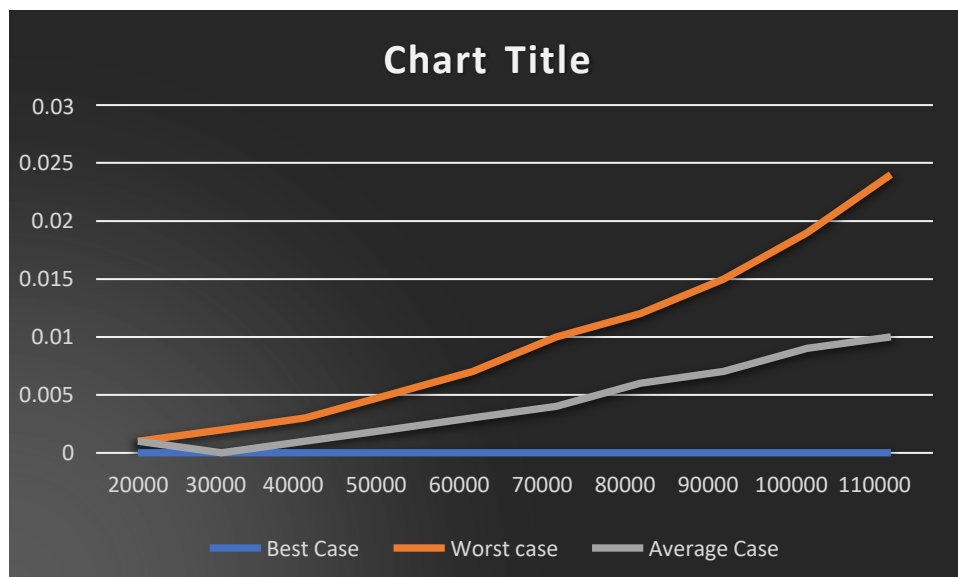
        long long int n=ran[k];long long int arr[n]; for(int i=0;i<n;i++) cin>>arr[i];
        time_t t1=time(NULL);
        int i, key, j;for (i = 1; i <
        n; i++){
            key = arr[i];j = 1;
            while (j >= 0 && arr[j] > key){
                arr[j + 1] = arr[j];j = j - 1;
            }

            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

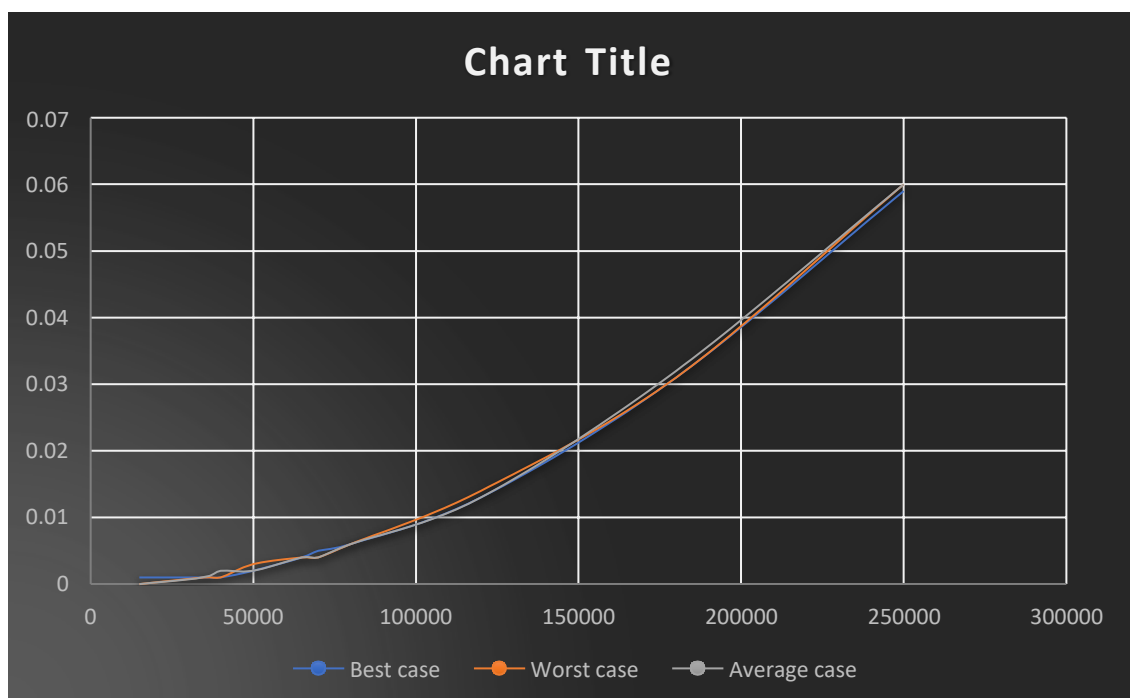


Objective – To Implement Selection Sort

```
#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



OBJECTIVE – To implement Binary Search

Code:

```
#include<bits/stdc++.h>
using namespace std; int
main()
{ freopen("file1.txt", "r", stdin);

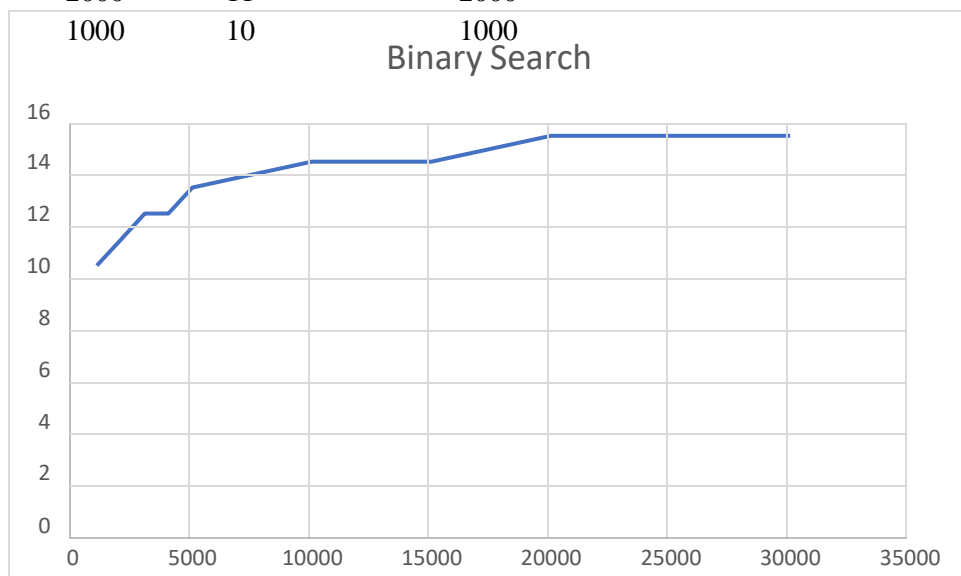
int size[]={ 30000,25000,20000,15000,10000,5000,4000,3000,2000,1000};

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){ in
t 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



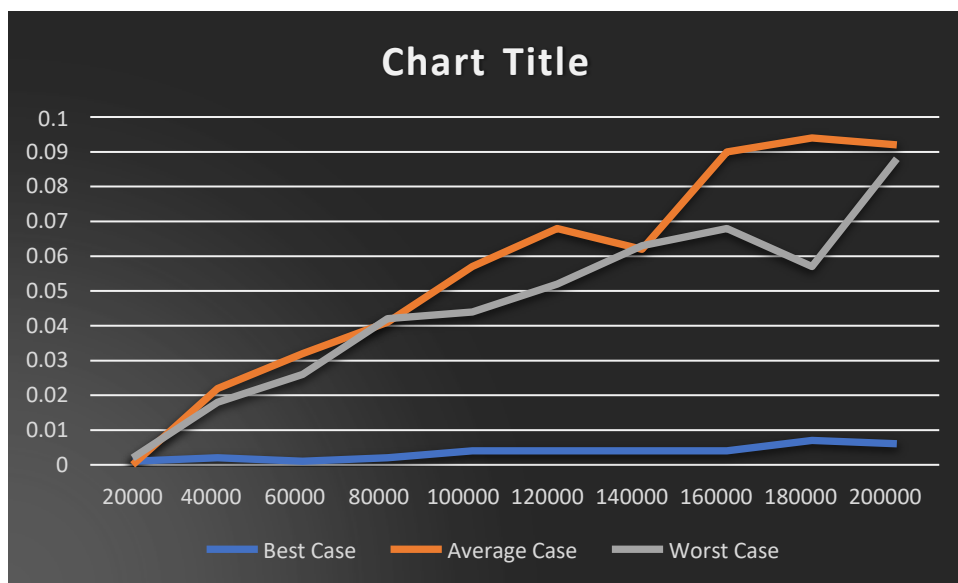
OBJECTIVE – To implemmt Heap Sort

```
#include<bits/stdc++.h>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] >
        arr[largest])largest = left;
    if (right < n && arr[right] >
        arr[largest])largest = right;
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
{
    for (int i = n / 2 - 1; i >= 0;
        i--)heapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
int main()
{
    freopen("file1.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);
        heapSort(arr, n);

        time_t t2=time(NULL);
        double ts=(double)(t2-
        t1)/CLOCKS_PER_SEC;cout<<ts<<" ";
    }
}
```

Range	Best case	Worst case	Average case
20000	0.001	0.000	0.002
40000	0.002	0.022	0.018
60000	0.001	0.032	0.026
80000	0.002	0.041	0.042
100000	0.004	0.057	0.044
120000	0.004	0.068	0.052
140000	0.004	0.062	0.063
160000	0.007	0.094	0.0057
180000	0.006	0.092	0.088
200000			



OBJECTIVE – To implement Merge Sort

Code

```
#include<bits/stdc++.h>
using namespace std;

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);
    }
}

int main()
{
    long long int
    ran[]={100000,30000,40000,50000,60000,70000,80000,90000,100000,110000};
    for(int k=0;k<10;k++)
    { FILE *fp;
    fp=fopen("File2.txt","r",stdi
n); int n=ran[k];
    int arr[n];
    for(in
```

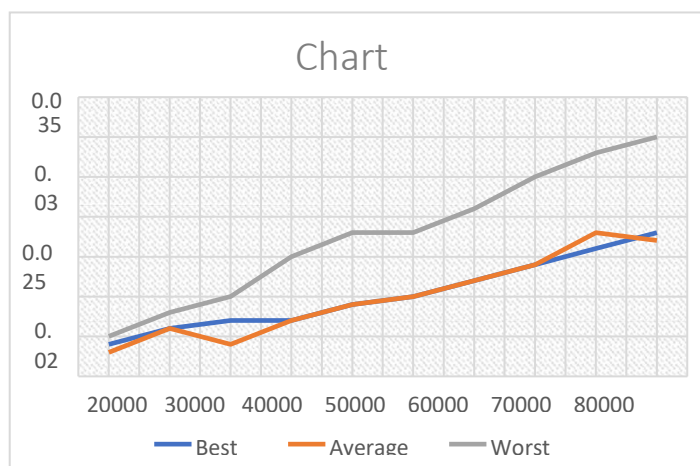
```

i=0;i<n;i++)
cin>>arr[i];
int size = sizeof(arr) /
sizeof(arr[0]);time_t
t1=time(NULL); mergeSort(arr,
0, size - 1);
time_t t2=time(NULL);

double ts=(double)(t2-
t1)/CLOCKS_PER_SEC;cout<<ts<<" ";
fclose(fp);
}
}

```

Range	Best	Worst	Average
20000	0.004	0.003	0.005
30000	0.006	0.006	0.008
40000	0.007	0.007	0.015
50000	0.007	0.007	0.01
60000	0.009	0.009	0.018
70000	0.01	0.01	0.018
80000	0.012	0.012	0.021
90000	0.016	0.018	0.028
100000	0.018	0.017	0.03
110000	0.018	0.020	0.05



Count Sort

```
#include<bits/stdc++.h>
using namespace std;
// Count sort

void countSort(int array[], int size) {
    int output[10];
    int count[10];

    int max = array[0];

    for (int i = 1; i < size; i++) {
        if (array[i] > max)
            max = array[i]; }

    for (int i = 0; i <= max; ++i) {
        count[i] = 0; }

    for (int i = 0; i < size; i++) {
        count[array[i]]++;}

    for (int i = 1; i <= max; i++) {
        count[i] += count[i - 1];}

    for (int i = size - 1; i >= 0; i--) {
        output[count[array[i]] - 1] = array[i];
        count[array[i]]--;}

    for (int i = 0; i < size; i++) {
        array[i] = output[i];}

}

int main()

{ freopen("count.txt","r",stdin);
vector<int> v1(100000);
for(int i=0;i<100000;i++)
cin>>v1[i];
int range[10]={ 10000,20000,30000,40000,50000,60000,70000,80000,90000,99000};

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

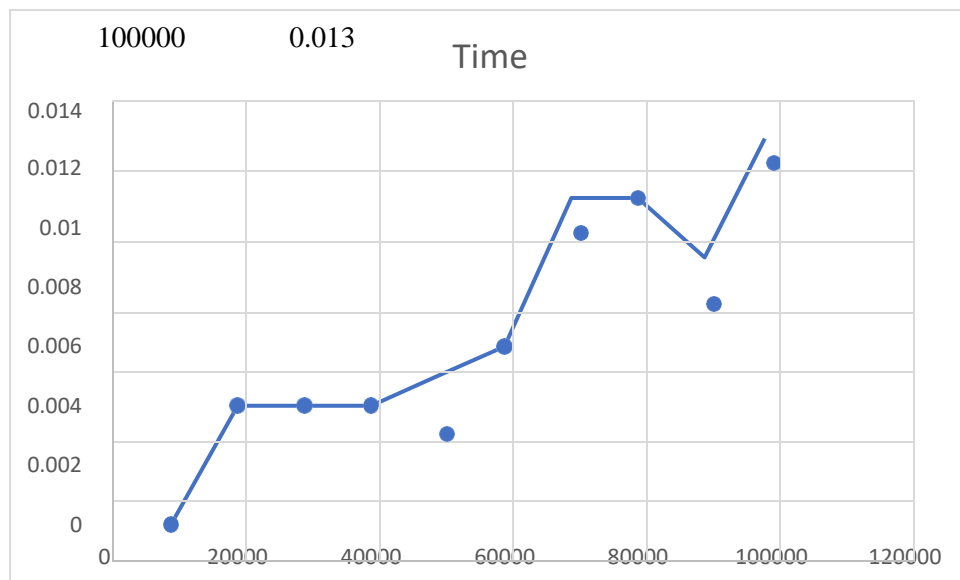
{ clock_t ti,tf;

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;

//printArray(v);

cout<<v.size()<<" "<<tt<<" "<<endl;}}
```

Range	Time
10000	0.00
20000	0.004
30000	0.004
40000	0.004
50000	0.005
60000	0.006
70000	0.011
80000	0.011
90000	0.009



Radix Sort

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;
// radix sort

// Function to get the largest element from an array
int getMax(int array[], int n) {
    int max = array[0];

    for (int i = 1; i < n; i++)
        if (array[i] > max)
            max = array[i];
    return max;
}

void countingSort(int array[], int size, int place) {
    const int max = 10;
    int output[size];
    int count[max];

    for (int i = 0; i < max; ++i)
        count[i] = 0;
    for (int i = 0; i < size; i++)
        count[(array[i] / place) % 10]++;
    for (int i = 1; i < max; i++)
        count[i] += count[i - 1];

    for (int i = size - 1; i >= 0; i--) {
        output[count[(array[i] / place) % 10] - 1] = array[i];
        count[(array[i] / place) % 10]--;
    }

    for (int i = 0; i < size; i++)
        array[i] = output[i];
}

void radixsort(int array[], int size) {

    int max = getMax(array, size);

    // Apply counting sort to sort elements based on place value.
    for (int place = 1; max / place > 0; place *= 10)
        countingSort(array, size, place);
}

int main()

{
    freopen("radix.txt", "r", stdin);
    vector<string> v1(100000);
    for(int i=0;i<100000;i++)
        cin>>v1[i];
    int range[10]={ 10000,20000,30000,40000,50000,60000,70000,80000,90000,100000};
    for(int i=0;i<10;i++)
```



```

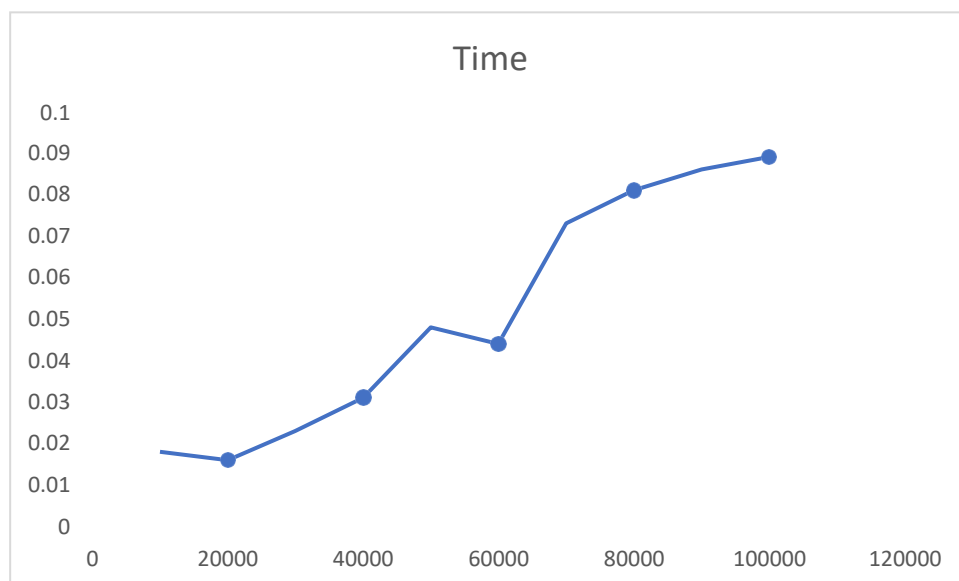
{ clock_t ti,tf;
int inputs=range[i];
vector<string>v=v1;
v.resize(inputs);
ti=clock();
radixsort(v,0,inputs-1,0);
tf=clock();
double tt=double(tf-ti)/CLOCKS_PER_SEC;

//print(v,inputs);
cout<<" "<<tt<<endl;
}

}

```

Range	Time
10000	0.018
20000	0.016
30000	0.023
40000	0.031
50000	0.048
60000	0.044
70000	0.073
80000	0.081
90000	0.086
100000	0.089



OBJECTIVE – To find minimum and maximum element in given array.

Theory - If n is odd then initialize min and max as the first element.

If n is even then initialize min and max as minimum and maximum of the first two elements respectively.

For the rest of the elements, pick them in pairs and compare their maximum and minimum with max and min respectively

```
#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;

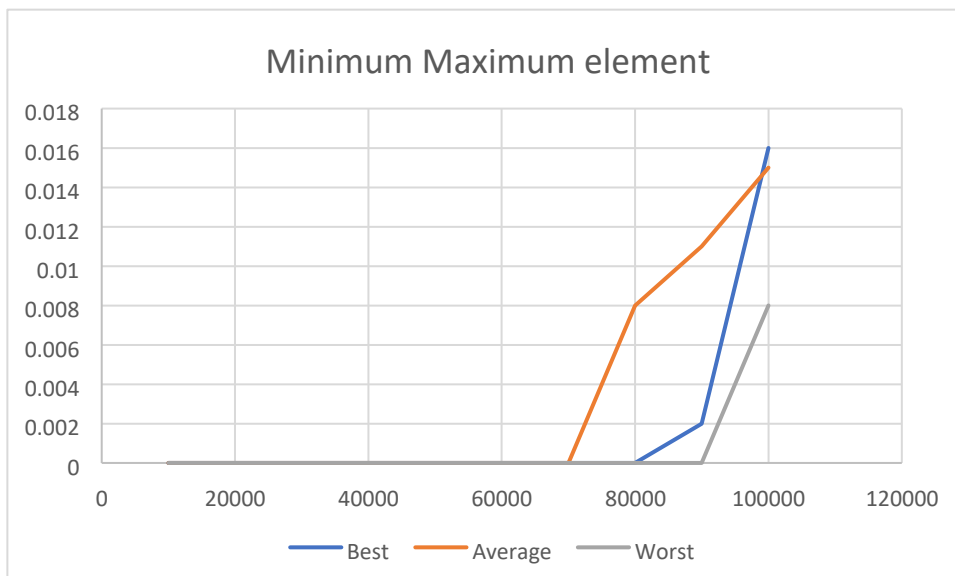
        int tests=Range[i];
        vector<int>v=v1;

        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; } }

```



OBJECTIVE- To find the Kth smallest element in a given array

Theory- To find the Kth minimum element in an array, insert the elements into the priority queue until the size of it is less than K, and then compare remaining elements with the root of the priority queue and if the element is less than the root then remove the root and insert this element into the priority queue and finally return root of the priority queue

```
#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+1-1);

}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);

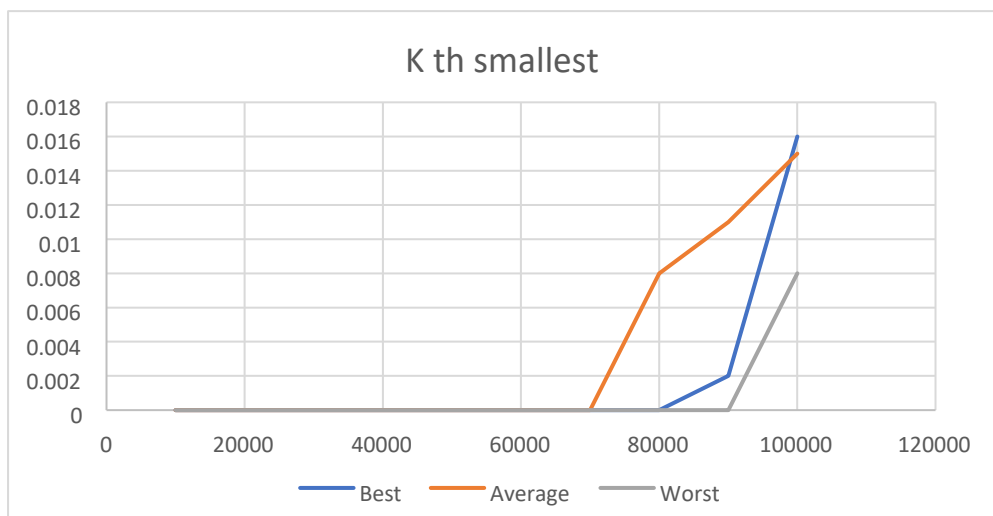
end=clock();

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



OBJECTIVE - To implement Prims algorithm to find minimum cost spanning tree

Theory - Use a boolean array mstSet[] to represent the set of vertices included in MST. If a value mstSet[v] is true, then vertex v is included in MST, otherwise not. Array key[] is used to store key values of all vertices. Another array parent[] to store indexes of parent nodes in MST. The parent array is the output array, which is used to show the constructed MST.

```
#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);}

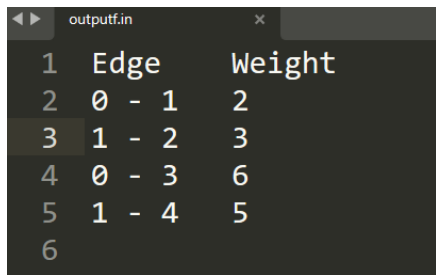
int main()
{
    int graph[V][V] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };
```

```

primMST(graph);

return 0;}

```



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

Output

OBJECTIVE- To implement Kruskals algorithm to find minimum cost spanning tree

```

#include <bits/stdc++.h>

using namespace std;

class DSU {
    int* parent; int* rank;
public:DSU(int n)
    { parent = new int[n]; rank = new int[n];
      for (int i = 0; i < n; i++) { parent[i] = -1; rank[i] = 1; } }
    int find(int i)
    { if (parent[i] == -1) return i; return parent[i] = find(parent[i]); }
    void unite(int x, int y)
    { int s1 = find(x); int s2 = find(y);
      if (s1 != s2) {
          if (rank[s1] < rank[s2]) { parent[s1] = s2; rank[s2] += rank[s1]; }
          else { parent[s2] = s1; rank[s1] += rank[s2]; } } }
};

class Graph {
    vector<vector<int>> > edgelist;
    int V;

public:
    Graph(int V) { this->V = V; }

    void addEdge(int x, int y, int w)
    {

```

```

        edgelist.push_back({ w, x, y });
    }

void kruskals_mst()
{
    sort(edgelist.begin(), edgelist.end());
    DSU s(V); int ans = 0;
    cout << "Following are the edges in the " "constructed MST"<< endl;
    for (auto edge : edgelist) {
        int w = edge[0];
        int x = edge[1];
        int y = edge[2];
        if (s.find(x) != s.find(y)) {
            s.unite(x, y);
            ans += w;
            cout << x << " -- " << y << " == " << w
                << endl;
        }
    }
    cout << "Minimum Cost Spanning Tree: " << ans;}};

// Driver's code
int main()
{
    Graph g(4);
    g.addEdge(0, 1, 10); g.addEdge(1, 3, 15);    g.addEdge(2, 3, 4); g.addEdge(2, 0, 6);
    g.addEdge(0, 3, 5);
    return 0;
}

```

```

outputf.in
1 Following are the edges in the constructed MST
2 2 -- 3 == 4
3 0 -- 3 == 5
4 0 -- 1 == 10
5 Minimum Cost Spanning Tree: 19

```


Objective : To implement matrix chain multiplication using dynamic programming

Theory: we can use dynamic programming to solve the problem in pseudo-polynomial time. Here's how:

1. First, it will divide the matrix sequence into two subsequences.
2. You will find the minimum cost of multiplying out each subsequence.
3. You will add these costs together and in the price of multiplying the two result matrices.
4. These procedures will be repeated for every possible matrix split and calculate the minimum.

And this is how dynamic programming solves the problem. Now implement this solution in C++.

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; }

```

The minimum cost is 4500

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

Objective : To implement longest common subsequence using dynamic programming

Theory: The dynamic programming paradigm consists of two methods known as the top-down approach and the bottom-up approach. The top-down approach memorizes results in a recursive solution, whereas the bottom-up approach builds a solution from the base case. Hence, for this particular problem, we will formulate a bottom-up solution.

Code:

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

int LCS( char *A, char *B, int x, int y ){

    int L[x+1][y+1]; int i, j; for(i =0; i<=x; i++){

        for(j=0; j<=y; 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] = maximum(L[i-1][j], L[i][j-1]);}}return L[x][y];}

int maximum(int a, int b)

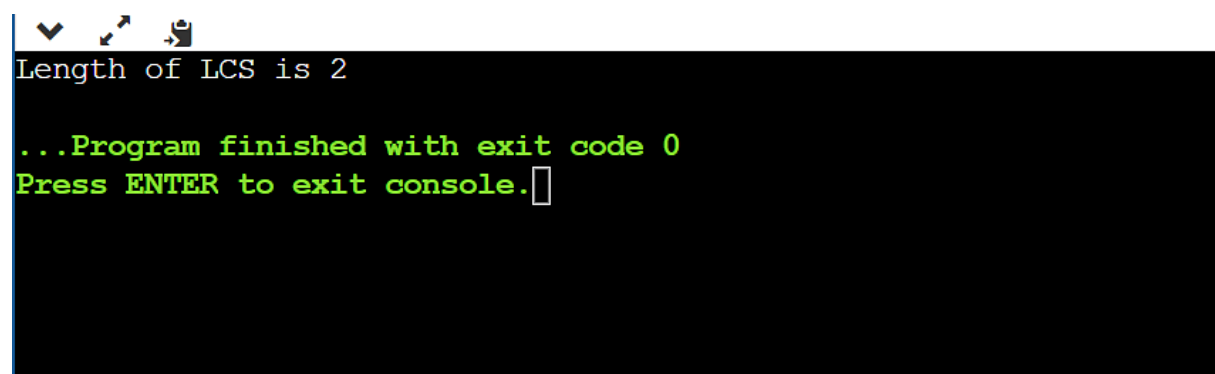
{return (a > b)? a : b;}

int main()

{char A[] = "XY";char B[] = "XPYQ";int x = strlen(A);int y = strlen(B);

printf("The Length of Longest Common Subsequence is %d", LCS( A, B, x, y ) );

return 0;}
```



```
Length of LCS is 2

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

Objective : To do the study of P ,NP ,NP Complete ,Np Hard problem .

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.