**LINEAR SEARCH**

Linear search(int a[],int n,int k)   
for(i=0;i<n;i++){  
 if(a[i]==k)  
 return 1;  
}

**SORTING THE ELEMENTS BY USING LIBRARY FUNCTIONS**SYNTAX**:**  SORT(<STARTING VALUE>,<ENDING VALUE>)EXAMPLE:SORT(A,A+N);  
where n=size of the array.

For vectors:  
sort(a.begin(),a.end())  
Time complexity:O(nlogn)

Decreasing order  
bool cmp(int x,int y){  
return x<y;  
}  
Sort(a.begin(),a.end(),cmp)

Q)Arrange the dates in increasing order  
sol:

Int vector<vector<int>>dates;  
 for(i=0:i<n;i++)  
{  
 int d,m,y:cin>>d>>m>>y;dates.push\_back({d,m,y});  
}  
sort(dates.begin(),dates.end(),cmp)  
cout<<dates[i][0]<<” “<<dates[i][1]<<” “<<dates[i][2]<<”\n”;

Bool cmp(vector<int>x,vector<int>y)  
{  
if(x[2]<y[2])  
return 1;  
else if(x[2]>y[2])return 0;  
else if(x[1]<y[1])return 1;  
else if(x[1]>y[1])return 0;  
else return x[0]<y[0];  
}

**BINARY SEARCH**step 1:- sort the array using sort(a,a+n)int bs(int a[],int n,int k)  
{  
l=0,h=n-1;  
while(l<=h)  
{  
 int m=(l+h)+2;  
if(a[m]==k)return 1;  
else if(a[m]<k)l=m+1;  
else h=m-1;  
}  
return 0;  
}

Int k;  
cin>>k;  
cout<<bs(a,n,k);

**Library function of binary search:**syntax:binary\_search(a,a+n,k)

Leet code problem 35  
class Solution {  
public:  
int searchInsert(vector<int>& nums, int target) {  
int l=-1,h=nums.size()-1;  
if(nums[h]<target)return h+1;  
 while(h-l>1)

        {

            int m=(l+h)/2;

            if(nums[m]>=target)h=m;

            else l=m;

        }

        return h;

    }

};

Lower bound(a.begin(),a.end(),i)-it gives the next maximum value of the given number.

**SORTING:  
BUBBLE SORT**: We compare the adjacent elements and make it sorted  
It requires n-1 passes to complete the sorting

Bubble sort code  
void bubble(arr[],n){  
for(int i=0;i<n-1;i++){  
for(int j=1;j<n-i-1;j++){  
if(a[j]>a[j+1])swap(a[j],a[j+1]))  
}  
}  
}

Time complexity:O(n^2) for all the best,avg and worst cases.

**SELECTION SORT**: We find the smallest element in each pass from i to n and we swap it.  
greedy method is used for this sorting because we select min element from the remaining inputs  
  
SELECTION SORT CODE  
void selection(int a[],int n){  
for(int i=0;i<n-1;i++){  
int mi=i;  
for(int j=i+1;j<n;j++){  
if(a[mi]>a[j]) mi=j;  
}  
swap(a[i],a[mi]);  
}  
}

**INSERTION SORT**: Insertion sort is the simple method of sorting an array. In this technique, the array is virtually split into the sorted and unsorted part. An element from unsorted part is picked and is placed at correct position in the sorted part.

INSERTION SORT CODE  
void insertion(int a[],int n){  
for(int i=1;i<n;i++){  
int k=a[i],j=i-1;  
while(j>=0&&a[j]>k){  
a[j+1]=a[j];  
j--;  
}  
a[j+1]=k;  
}  
}

TIME COMPLEXITY: BEST-O(N)  
 AVG AND WORST-O(N^2)

**Merge Sort:** We divide the existing array into sub arrays until we get only one element in the sub arrays.  
MERGE SORT CODE  
void merge(int a[],int l,int m,int h){  
int b[h+1],i=l,j=m+1,k=l;  
while(i<=m && j<=h){  
if(a[i]<=a[j])  
b[k++]=a[j++];  
else b[k++]=a[j++];  
}  
while(i<=m)b[k++]=a[i++];  
while(j<=h)b[k++]=a[j++];  
for(int i=l;i<=h;i++)  
a[i]=b[i];  
}

Void mergesort(int a[],int l,int h)  
{

If(l<h){  
int m=(l+h)/2;  
mergesort(a,l,m);  
mergesort(a,m+1,h);  
merge(a,l,m,h);  
}  
}  
  
 **Difference b/w merge and quick sort**  
In merge sort will divide and order the list and in quick sort will order and divide the list.  
In merge sort will divide exactly at middle in quick sort will divide based on pivot.  
In merge sort it takes additional array and in quick sort it is in place algorithm

Merge sort-nlogn and stable  
quick sort- average and best n logn and it is unstable  
 worst n2   
Example: 3 4 18 25 20 7 6 10  
 step 1 : 3 4 18 25 20 7 6 10 i=0;j=0;  
 step 2 : 3 4 18 25 20 7 6 10 i=1;j=1;  
 step 3 : 3 4 18 25 20 7 6 10 i=2;j=2;  
 step 4 : 3 4 18 25 20 7 6 10 i=2;j=3;  
 step 5 : 3 4 18 25 20 7 6 10 i=2;j=4;  
 step 6 : 3 4 18 25 20 7 6 10 i=2;j=5;  
 step 7 : 3 4 7 25 20 18 6 10 i=2;j=5; //swap and increment i  
 step 8 : 3 4 7 6 20 18 25 10 i=3;j=6;//swap and increment i  
 step 9 : 3 4 7 6 10 18 25 20 i=4;j=7;//swap and increment i

**QUICK SORT**  
void quicksort (int a[],int l.int r)   
{  
 if(l<r)  
{  
 int pi=partition(a,l,r);  
 quicksort(a,l,pi-1)  
 quicksort(a,pi+1,r)  
}  
}  
int partition(int a[],int l, int r)  
{  
 int i=l;  
 for(j=1;j<r;j++)  
{  
 if(a[j]<=a[r])  
{  
 swap(a[i],a[j])  
 i++;  
}  
}  
swap(a[i],a[r]) return i ;  
}