Lab 4:
Sorting and Searching

➔ **Aim:**

a. Sort the data in ascending order using Selection sort and descending order using Insertion sort.
b. Search a particular data using Linear search.

➔ **Theory**

● **Sorting**

The arrangement of data sequentially by ascending or descending order.
There are many techniques to sort the data by applying sorting criteria. We can apply any technique to sort the data. Following are some sorting techniques.

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Quick Sort
5. Merge Sort
6. Radix Sort
7. Shell sort

Each Sort has its own time and space complexity. **Time Complexity** is the amount of time taken by the set of code and **Space Complexity** is the amount of memory taken by the data in the code.It is represented by Big *O*. Following chart shows the time complexity for each sorting technique as it is the important factor in code.

Sorting Technique	Time Complexity
1. Bubble Sort	O(*n*2)
2. Selection Sort	O(*n*2)
3. Insertion Sort	O(*n*2)
4. Quick Sort	O(*nlogn*)
5. Merge Sort	O(*nlogn*)
6. Radix Sort	O((*n* + *base*) * *logbase*(*k*))
7. Shell Sort	O(*n*) to O(*n*2)

In this session we will see how Selection sort and Insertion sort works and their complexity.

● **Selection Sort**

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1. The subarray which is already sorted
2. Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

For example, arr[ ] = 64 25 12 22 11

Pass 1 :

Find the minimum element in arr[0...4] and place it at the beginning

64 | 25 12 22 **11** *min*

11 | 25 12 22 **64**

Pass 2 :

Find the minimum element in arr[1...4] and place it at the beginning of arr[1...4]

11 **| 25 12** *min* 22 64

11 **12 | 25** 22 64

Pass 3 :

Find the minimum element in arr[2...4] and place it at the beginning of arr[2...4]

11 12 **| 25 22** *min* 64

11 12 **22 | 25** 64

Pass 4:

Find the minimum element in arr[3...4] and place it at the beginning of arr[3...4]

11 12 22 **| 25** *min* 64

11 12 22 **25 |** 64

Sorted : 11 12 22 25 64

● **Pseudocode for Selection Sort in Ascending order.**

```
void selectionSort(int n)
{
int i,j,min,temp;
for(i=0;i<n-1;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(arr[j]<arr[min])
{
min=arr[j];
}
}
temp=arr[minI];
arr[minI]=arr[i];
arr[i]=temp;
}
```

}

Time Complexity:

- The outer loop executes N-1 times
- The inner loop executes N-i-1 times

Therefore, Complexity for Selection sort is $O(n2)$.

● **Insertion Sort**

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

This technique sorts the array by shifting elements one by one.

Though this is the simplest way to sort the data but it is inefficient for large amount of data sets.

For example, arr[ ] = 12, 11, 13, 5, 6, 3

loop for i = 1 (second element of the array) to 5 (the last element of the array) Consider

the 0 th element as a sorted list.

12 | 11 *key* 13 5 6 3

Pass 1:

i = 1 Since 11 is smaller than 12, move 12 and insert 11 before 12

11 12 | 13 *key* 5 6 3

Pass 2:

i = 2. 13 will remain at its position as all elements in A[0..I-1] are smaller than 13

11 12 13 | 5 *key* 6 3

Pass 3:

i = 3. 5 will move to the beginning and all other elements from 11 to 13 will move one

position ahead of their current position.

5 11 12 13 | 6 *key* 3

Pass 4:

i = 4. 6 will move to position after 5, and elements from 11 to 13 will move one position

ahead of their current position.

5 6 11 12 13 | 3 *key*

Pass 5:

i = 5. 3 will move to the beginning, and elements from 5 to 13 will move one position

ahead of their current position.

3 5 6 11 12 13 |

Sorted : 3 5 6 11 12 13 |

● **Pseudocode for Selection Sort in Descending order.**

void insertionSort(int n)

```
{
int i,j,key;
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 Complexity**

- The outer loop execute for N times
- The inner loop executes for N times

Therefore, the time complexity for Insertion Sort is $O(n2)$.

● **Searching**

Searching is a way to find any element from the set of elements. The searching can be done by three methods. They are as follows

1. Linear Search ($O(n)$)
2. Binary Search ($O(log2n)$)
3. Hashing ($O(1)$)

Each technique can be used according to the efficiency of complexity in the program. In this session we will see the Linear Search technique and its Complexity.

● **Linear Search**

Linear Search works by simply traversing the array or list sequentially to find the element in the array.

For example, If we want to find 5 in the given array arr[ ] = 1,2,3,4,5,6

Then,

Case 1: Element is Present in the List

Compare 5 with first element

1 2 3 4 5 6

Not matching, therefore move to the next position.

1 **2** 3 4 5 6

Not matching, therefore move to the next position.

1 2 **3** 4 5 6

Not matching, therefore move to the next position.

1 2 3 **4** 5 6
Not matching, therefore move to the next position.
1 2 3 4 **5** 6
Element Found at 4 th position.
As element is found there is no need to traverse the rest of the array.
Case 2: Element is not Present in the list.
Now, if we had to search for 7 instead of 5 we will move to the next position.
1 2 3 4 5 **6**
Here, 6 does not matches with 7 and also it is the last element in the list so we cannot move further. Therefore we can say that element is not present in the list

● **Pseudocode for Searching the element using Linear Search.**

```
void linerSearch(int n)
{
int item,i,flag=0;
cout<<"\nEnter the Element to be Searched\n";
cin>>item;
for(i=0;i<n;i++)
{
if(arr[i]==item)
{
cout<<"\nElement found at location "<<i+1;
flag=0;
break;
}
else
{
flag=1;
}
}
if(flag==1)
{
cout<<"\nElement not found\n";
}
}
```

● **Time Complexity**

- If the element to be searched is present at the last position of the array or list we have to traverse all N elements Therefore,
Worst Case Complexity: *O(n)*
- If the element is present at the first position of the list then only one comparison will be need Therefore,

}

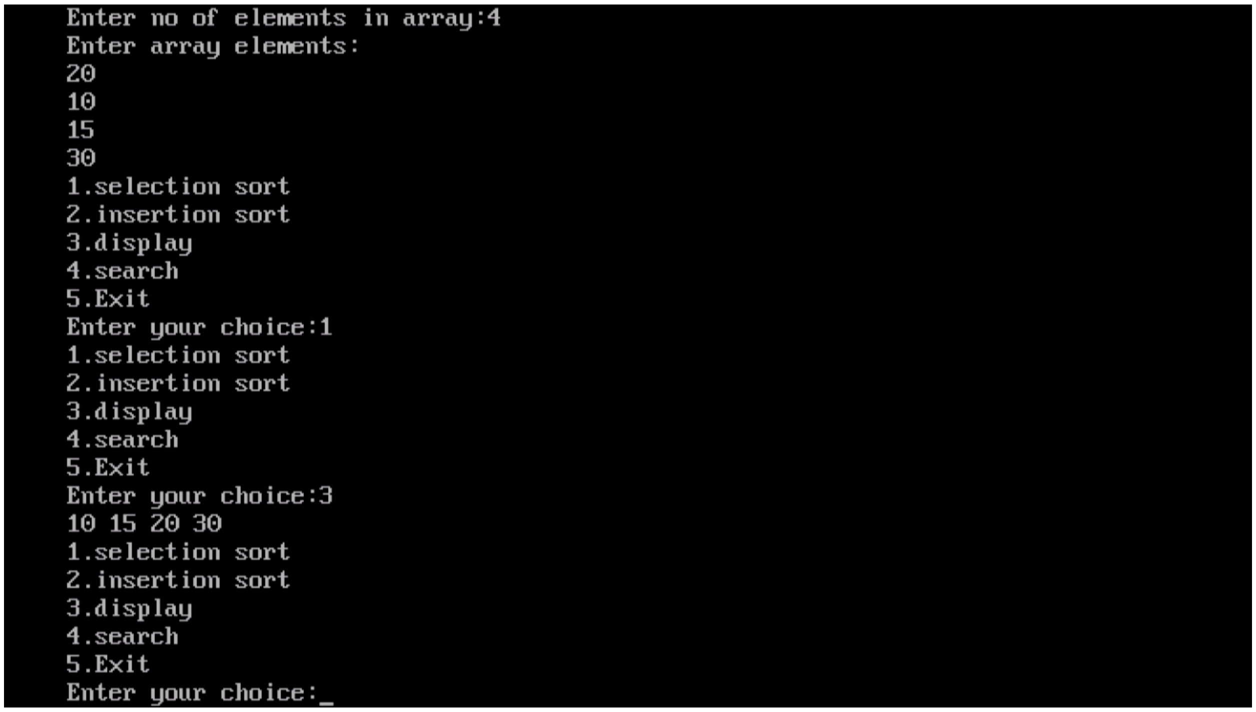
Best Case Complexity: *O(* 1 *)*

Program:

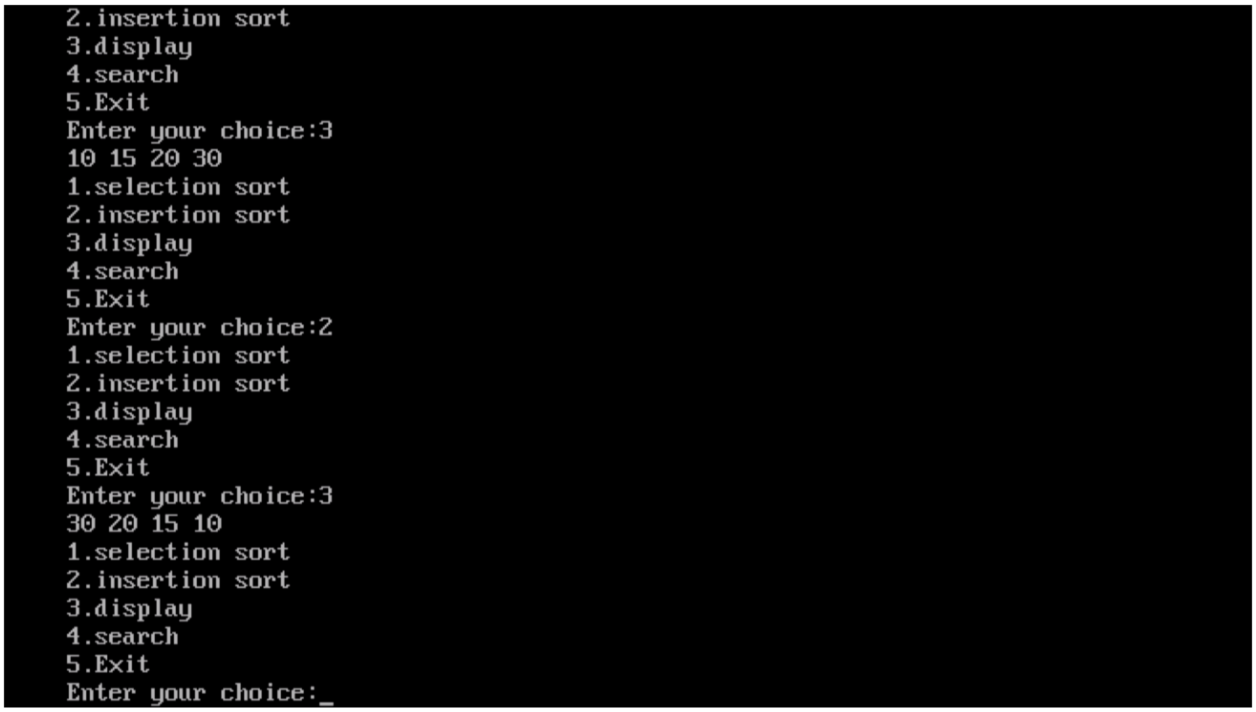
```
#include<iostream.h>

#include<conio.h>

#include<stdlib.h>

void selection(int b[],int n)

{

int i,j,pos,t;

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

{

pos=i;

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

{

if(b[j]<b[pos])

{

pos=j;

}

}

t=b[i];

b[i]=b[pos];

b[pos]=t;

}
```

```
}
void display(int b[],int n)
{
int i;
for(i=0;i<n;i++)
{
cout<<b[i]<<" ";
}
cout<<"\n";
}
void insertion(int b[],int n)
{
  int j,t;
for(int i=1;i<n;i++)
{
t=b[i];
j=i-1;
while((t>b[j])&&(j>=0))
{
b[j+1]=b[j];
j--;
}
b[j+1]=t;

}


}

void search(int b[],int n)

{

int el,flag=0;

cout<<"Enter element to be search:";

cin>>el;

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

{

if(b[i]==el)

{

flag=1;

cout<<"Element found "<<endl;

}

}

if(flag==0)

cout<<"Element not found"<<endl;

}


void main()

{
```

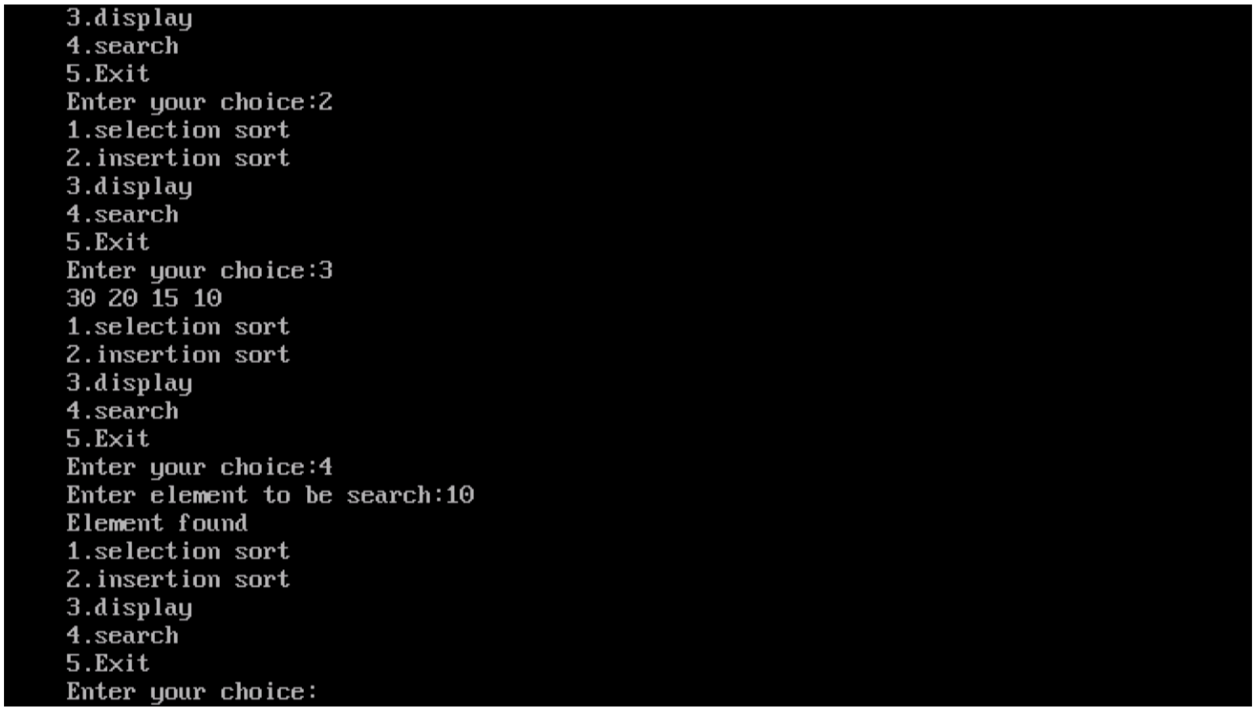
```
int t,n,a[100],i,j,pos,choice;

clrscr();

cout<<"Enter no of elements in array:";

cin>>n;

cout<<"Enter array elements:";

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

{

cin>>a[i];


}

do{

cout<<"1.selection sort"<<endl<<"2.insertion
sort"<<endl<<"3.display"<<endl<<"4.search"<<endl<<"5.Exit"<<endl;

cout<<"Enter your choice:";

cin>>choice;

switch(choice)

{

case 1:

selection(a,n);

break;

case 2:

insertion(a,n);

break;
case 3:
display(a,n);
break;
case 4:
search(a,n);
break;
}
}while(choice!=5 && choice<5);
system("pause");
getch();
}
```



```
Enter no of elements in array:4
Enter array elements:
20
10
15
30
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:1
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:3
10 15 20 30
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:_
```



```
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:3
10 15 20 30
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:2
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:3
30 20 15 10
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:_
```



```
3.display
4.search
5.Exit
Enter your choice:2
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:3
30 20 15 10
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:4
Enter element to be search:10
Element found
1.selection sort
2.insertion sort
3.display
4.search
5.Exit
Enter your choice:
```

➔ **Conclusion :**

In this way we have seen how to sort the elements in ascending order using selection sort and in descending order using insertion sort. Also we have seen how to search element in array or list using Linear Search technique