AIM:-

Assignment No. 04

1. Sort the data in ascending order using selection sort and descending order by using insertion sort (Display pass by pass output).
2. Search a perticular data using linear search.

OBJECTIVE:-

To understand the sorting techniques i.e. **Selection sort and Insertion sort**. To know the difference between these two sorting techniques and there inner loops i.e. sorting methods. Also to learn and to implement **Linear search** over the sorted array.

THEORY:-

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.

Insertion Sort

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order. The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

ALGORITHM:-

SELECTION SORT

1. Set the first element as minimum.
2. Compare minimum with the second element. If the second element is smaller than, minimum assign second element as minimum. Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.
3. After each iteration, minimum is placed in the front of the unsorted list.
4. For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.

INSERTION SORT

1. The first element in the array is assumed to be sorted. Take the second element and store it separately in key.  
   Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element.
2. Now, the first two elements are sorted.  
   Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.
3. In a similar way, place every unsorted element at its correct position.

LINEAR SEARCH

1. Here in this program we have taken the array as an input from the user along with the key to be searched. We have created a separate function for Linear Search.  
2. When the function is called we have to run a loop n times where n is the number of elements in an array.  
3. In each iteration we are comparing the value of key with elements of array in increasing order of array index.  
4. If key is equal to one of the array elements we print the value of index at which we found them to be equal.

SOURCE CODE:-

#include<iostream>

using namespace std;

void selection(int arr[20],int n)

{

int i,j,temp;

for(i=0;i<n-1;i++)

{

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

{

if (arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

cout<<"Using selection sort \n Sorted array is:\n"<<endl;

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

{

cout<<arr[i]<<endl;

}

}

void insertion(int arr[20],int n)

{

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;

}

cout<<"Using insertion sort \n Sorted array is:\n";

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

{

cout<<arr[i]<<endl;

}

}

int main()

{

int arr[20],i,n,choice;

cout<<"Enter number of elements in the array :";

cin>>n;

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

{

cin>>arr[i];

cout<<"\n";

}

selection(arr,n);

insertion(arr,n);

cout<<"Enter number you want to search\n";

cin>>choice;

int k=0;

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

{

if(choice==arr[i])

{

Cout<<”Element found : ”<<arr[i];

k++;

}

}

if(k==0)

{

cout<<"Element not found";

}

return 0;

}

OUTPUT:-

Enter number of elements in the array: 5

12

45

96

23

1

Using selection sort

Sorted array is:

1

12

23

45

96

Using insertion sort

Sorted array is:

96

45

23

12

1

Enter number you want to search

23

Element found : 23

CONCLUSION:-

SELECTION SORT

**Number of comparisons:**(n-1) + (n-2) + (n-3) +.....+ 1 = n(n-1)/2 nearly equals to n2  
  
**Complexity** = O(n2)

Also, we can analyze the complexity by simply observing the number of loops. There are 2 loops so the complexity is n\*n = n2.

**Time Complexities:**

* **Worst Case Complexity:** O(n2)  
    
  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.
* **Best Case Complexity:** O(n2)  
    
  It occurs when the the array is already sorted
* **Average Case Complexity:** O(n2)  
    
  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

The time complexity of selection sort is the same in all cases. At every step, you have to find the minimum element and put it in the right place. The minimum element is not known until the end of the array is not reached

INSERTION SORT

* **Worst Case Complexity:**O(n2)  
    
  Suppose, an array is in ascending order, and you want to sort it in descending order. In this case, worse case complexity occers.  
    
  Each element has to be compared with each of the other elements so, for every nth element, (n-1) number of comparisons are made.  
    
  Thus, the total number of comparisons = n\*(n-1) ~ n2
* **Best Case Complexity:**O(n)  
    
  When the array is already sorted, the outer loop runs for n number of times whereas the inner loop does not run at all. So, there is only n number of comparison. Thus, complexity is linear.
* **Average Case Complexity:**O(n2)  
    
  It occurs when the elements of a array are in jumbled order (neither ascending nor descending).

LINEAR SEARCH

**Case 1. Best Case:** When the element to be searched is the first element of the array.  
For example:

If the input array is {4, 6, 1, 2, 5, 3}

and the element to be searched is 4,

then the output will be Position: 1

**Best Case time complexity: o(1)**

**Case 2. Average Case:**When the element to be searched is any random element in an array.  
For example:

If the input array is {66, -3, 31}

and the element to be searched is 31,

the output will be Position: 3

**Average Case time Complexity: O(n)**

**Case 3. Worst Case:** When the element to be searched is absent.  
For example:

If the input array is {1, 3, 6, 1, 9}

and the element to be searched is 10,

then the output will be "Element not present".

**Worst Case time complexity: O(n)**