Aim:

To write a program to analyze insertion and selection sort.

Theory:

Insertion Sort is a simple sorting algorithm that works by sorting an array one element at a time. It works by dividing the array into two parts: the sorted part and the unsorted part. The algorithm starts by assuming that the first element of the array is already sorted and then iterates through the remaining elements, inserting each element into its proper position in the sorted part of the array.

Insertion Sort has a time complexity of O(n^2), making it less efficient than other sorting algorithms such as Quick Sort or Merge Sort. However, it is often faster than these algorithms for small arrays or partially sorted arrays.

The algorithm is simple to understand and implement, making it a popular choice for simple sorting tasks. It is also an in-place sorting algorithm, meaning it does not require additional memory to sort the array, making it useful for situations where memory is limited.

Overall, Insertion Sort is a useful algorithm for sorting small or partially sorted arrays efficiently. Its simplicity and in-place nature make it a popular choice for simple sorting tasks.

Selection Sort is a simple sorting algorithm that works by repeatedly finding the minimum element from the unsorted part of an array and putting it at the beginning of the sorted part of the array. The algorithm works by dividing the array into two parts: the sorted part and the unsorted part. The sorted part is initially empty, while the unsorted part contains all the elements of the array.

Selection Sort has a time complexity of O(n^2), making it less efficient than other sorting algorithms such as Quick Sort or Merge Sort. However, it is useful for small arrays or partially sorted arrays.

The algorithm is simple to understand and implement, making it a popular choice for simple sorting tasks. It is also an in-place sorting algorithm, meaning it does not require additional memory to sort the array, making it useful for situations where memory is limited.

Overall, Selection Sort is a useful algorithm for sorting small or partially sorted arrays efficiently. Its simplicity and in-place nature make it a popular choice for simple sorting tasks.

Algorithm:

Insertion Sort:

```
InsertionSort(A)
  for i = 1 to length(A) - 1
    key = A[i]
    j = i - 1
    while j >= 0 and A[j] > key
      A[j+1] = A[j]
      j = j - 1
    end while
    A[j+1] = key
  end for
end InsertionSort
```

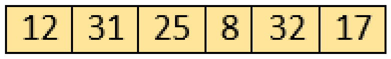
Selection Sort:

```
SelectionSort(A)
  for i = 0 to length(A) - 1
    min_idx = i
    for j = i + 1 to length(A) - 1
      if A[j] < A[min_idx]
        min_idx = j
      end if
    end for
    temp = A[i]
    A[i] = A[min_idx]
    A[min_idx] = temp
  end for
end SelectionSort
```

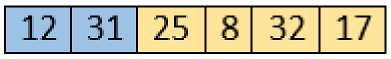
Example:

Insertion Sort:

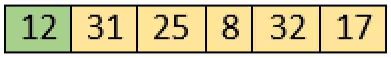
Let the elements of array are –



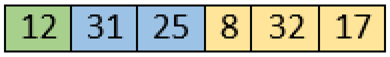
Initially, the first two elements are compared in insertion sort.

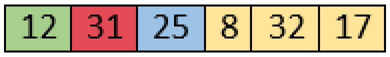


Here, 31 is greater than 12. That means both elements are already in ascending order. So, for now, 12 is stored in a sorted sub-array.



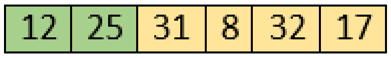
Now, move to the next two elements and compare them.



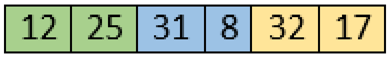


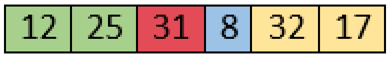
Here, 25 is smaller than 31. So, 31 is not at correct position. Now, swap 31 with 25. Along with swapping, insertion sort will also check it with all elements in the sorted array.

For now, the sorted array has only one element, i.e. 12. So, 25 is greater than 12. Hence, the sorted array remains sorted after swapping.

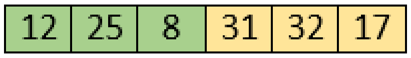


Now, two elements in the sorted array are 12 and 25. Move forward to the next elements that are 31 and 8.

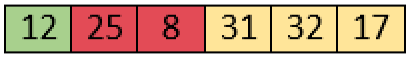




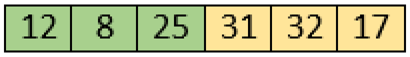
Both 31 and 8 are not sorted. So, swap them.



After swapping, elements 25 and 8 are unsorted.



So, swap them.



Now, elements 12 and 8 are unsorted.

12	8	25	31	32	17

So, swap them too.

8	12	25	31	32	17

Now, the sorted array has three items that are 8, 12 and 25. Move to the next items that are 31 and 32.

8	12	25	31	32	17

Hence, they are already sorted. Now, the sorted array includes 8, 12, 25 and 31.

8	12	25	31	32	17

Move to the next elements that are 32 and 17.

8	12	25	31	32	17

17 is smaller than 32. So, swap them.

8	12	25	31	17	32

8	12	25	31	17	32

Swapping makes 31 and 17 unsorted. So, swap them too.

8	12	25	17	31	32

8	12	25	17	31	32

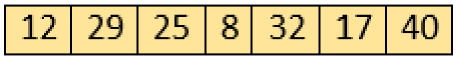
Now, swapping makes 25 and 17 unsorted. So, perform swapping again.

8	12	17	25	31	32

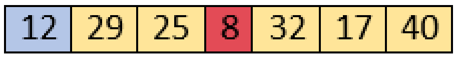
Now, the array is completely sorted.

Selection Sort:

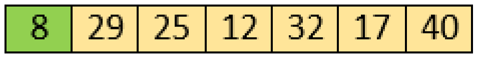
Let the elements of array are –



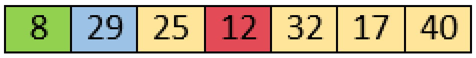
Now, for the first position in the sorted array, the entire array is to be scanned sequentially.
At present, 12 is stored at the first position, after searching the entire array, it is found that 8 is the smallest value.



So, swap 12 with 8. After the first iteration, 8 will appear at the first position in the sorted array.

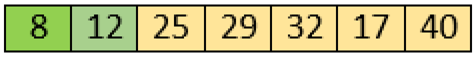


For the second position, where 29 is stored presently, we again sequentially scan the rest of the items of unsorted array. After scanning, we find that 12 is the second lowest element in the array that should be appeared at second position.

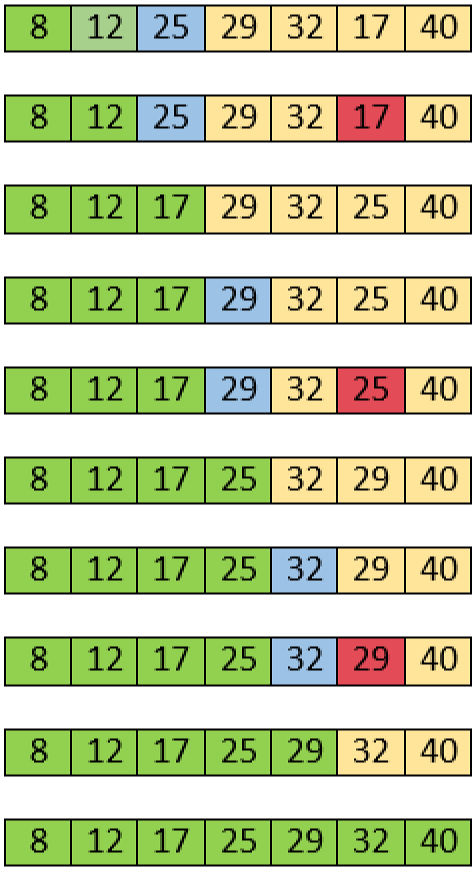


Now, swap 29 with 12. After the second iteration, 12 will appear at the second position in the sorted array.

So, after two iterations, the two smallest values are placed at the beginning in a sorted way.



The same process is applied to the rest of the array elements. Now, we are showing a pictorial representation of the entire sorting process.



Now, the array is completely sorted.

Algorithm:

Insertion_sort(A,n)

```
For j = 2 to n
    Key = A[j]
    i = j-1
    while(i>=1 && a[i]>key)
        a[i+1] = A[i]
        i—
    A[i+1]=key
```

Selection_sort(A,n)

```
    For i=1 to n-1
        Imin = i
        For j=i+1 to n
            If(A[min]>A[j])
                Imin=j
    Temp = A[imin]
    A[imin] = A[i]
    A[i] = Temp
```

Code:

Insertion Sort:

```
#include<stdio.h>
#include<time.h>
//insertion sort
void main ()
{
   int a[100000],i,j,temp,n;
   clock_t start, end;
    double cpu_time_used;
    start = clock();
   printf("Enter number of elements");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      a[i]=n-i;                //worst case:- n-1,best case:- i, average case :- rand()
   }
   for(i=0;i<n;i++)
   {
      temp=a[i];
      j=i-1;
      while(j>=0 && a[j]>=temp)
      {
         a[j+1]=a[j];
         j--;
      }
      a[j+1]=temp;
   }
   // for(i=0;i<n;i++)
```

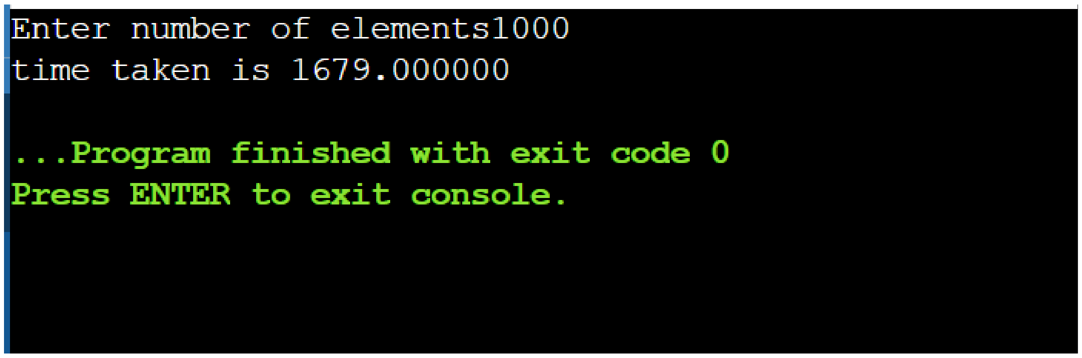
```
    // {
    //     printf("%d\t",a[i]);
    // }
    end = clock();
    cpu_time_used = ((double) (end - start));
    printf("time taken is %f",cpu_time_used);
}
```

Selection Sort:

```
//selection sort

#include<stdio.h>

#include<time.h>


void main ()

{

    int a[50000],i,j,imin,temp,n;

     clock_t start, end;

     double cpu_time_used;

     start = clock();

    printf("Enter number of elements");

    scanf("%d",&n);

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

    {

        a[i]=rand();     //worst case:- n-1,best case:- i, average case :- rand()


    }

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

    {

        imin=i;

        for(j=i+1;j<n;j++)
        {
            if(a[j]<a[imin])
            {
                a[imin]=a[j];
            }
        }
        temp=a[i];
        a[i]=a[imin];
        a[imin]=temp;
    }
    // for(i=0;i<n;i++)
    // {
    //     printf("%d\t",a[i]);
    // }
    end = clock();
    cpu_time_used = ((double) (end - start));
    printf("time taken is %f",cpu_time_used);
}
```

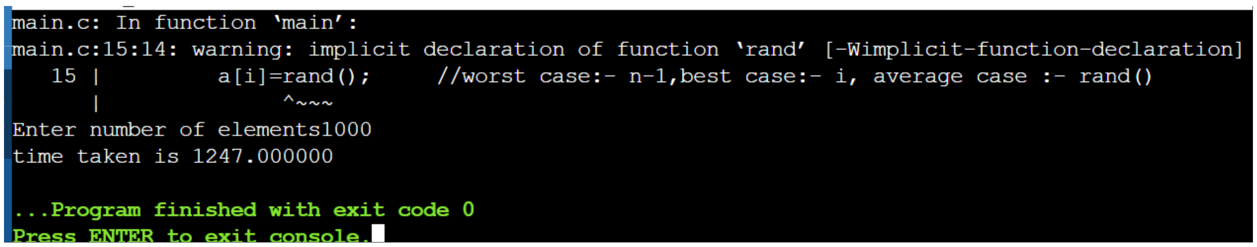
Output:

Insertion Sort:



```
Enter number of elements1000
time taken is 1679.000000

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

Selection Sort:



```
main.c: In function 'main':
main.c:15:14: warning: implicit declaration of function 'rand' [-Wimplicit-function-declaration]
   15 |         a[i]=rand();     //worst case:- n-1,best case:- i, average case :- rand()
      |              ^~~~
Enter number of elements1000
time taken is 1247.000000

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

Conclusion:

Insertion Sort is a simple and efficient sorting algorithm that iterates over an array and places each element in its correct position in the sorted part of the array. It has a time complexity of O(n^2), making it suitable for small or partially sorted arrays.

Selection Sort is a simple and easy-to-implement sorting algorithm that iterates over an array, finds the smallest element in the unsorted part of the array, and places it at the beginning of the sorted part. It has a time complexity of O(n^2), making it suitable for small or partially sorted arrays.