A COMPARISON BETWEEN THREE SORTING ALGORITHMS BASED UPON THE TIME COMPLEXITY

Shweta Popli[1], Arshi Talwar[2], Sneha Gupta[3]
[1, 2, 3] *Dronacharya College Of Engineering*

ABSTRACT

This research paper presents the different types of sorting algorithms of data structures like bubble sort, insertion sort and selection sort and also give their performance analysis with respect to time complexity. These four sorting algorithms have been an area of focus for a long time but still the question remain same of which to use when ? which is the main reason to perform this research. This research paper provides the detailed study of the algorithms and compares them on the basis of time complexity to reach the conclusion.

Keywords- *bubble sort, selection sort, insertion sort, time complexity.*

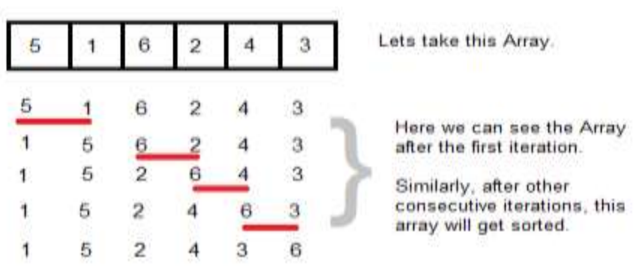
I. INTRODUCTION

In computer science, a sorting algorithm is an efficient algorithm which performs an important task that puts elements of a list in a certain order or arranges a collection of items into a particular order. Sorting data has been developed to arrange the array values in various ways for a database. Sorting algorithms are an important part of managing data. Sorting algorithms are open problems and many researchers in past have attempted to optimize them with optimal space and time scale.Most sorting algorithms work by comparing the data being sorted. Sorting algorithms are usually judged by their efficiency. In this case, efficiency refers to the algorithmic efficiency as the size of the input grows large and is generally based on the number of elements to sort. Most of the algorithms in use have an algorithmic efficiency of either O (n^2) or O(n*log(n)).

II. BUBBLE SORT

Bubble sort is an algorithm which is used to sort the n elements in an array. It compares all the elements one by one and sort them. It is called bubble sort because in each iteration the smaller element bubbles up towards the first place. It takes place by comparing the adjacent data items and swapping each pair.

2.1 Working Procedure Of Algorithms


5 1 6 2 4 3
Lets take this Array.
5 1 6 2 4 3
1 5 6 2 4 3
1 5 2 6 4 3
1 5 2 4 6 3
1 5 2 4 3 6
Here we can see the Array after the first iteration.
Similarly, after other consecutive iterations, this array will get sorted.

2.2 Algorithm

```
Read the array
int i,j,temp;
for(i=0;i<n-1;i++)
  {
    for(j=0;j<n-i-1;j++)
       {
          if(a[j]>?a[j+1])
            {
               temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
         }
  }
```

2.3 Complexity

In bubble sort n-1 comparisons will be done in 1st pass , n-2 comparisons will be done in 2nd pass and so on, The total number of comparisons will be

(n-1)+(n-2)+(n-3)+………..+3+2+1

Sum=n(n-1)/2

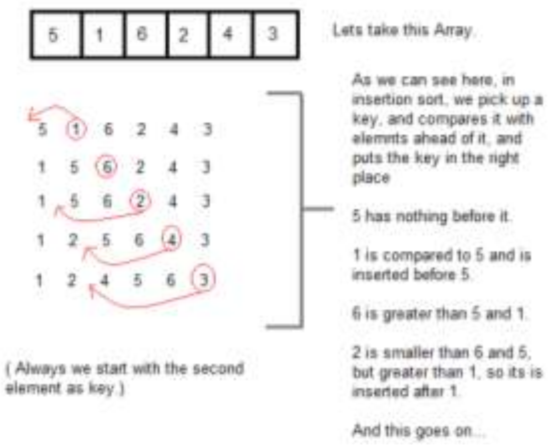
i.e $O(n^2)$

III. INSERTION SORT

It is the simple sorting algorithm which sorts the array by shifting the elements one by one. The characterstics of insertion sort are:

1. It is one of the simplest implemented form.
2. It is efficient for smaller data sets.
3. It is stable as it does not change the order.
4. It is better than bubble and selection sort algorithms.

3.1 Working Procedure Of Algorithms



Lets take this Array.

As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

And this goes on...

(Always we start with the second element as key.)

3.2 Algorithm

```
Read the array
int i,j,temp;
for(i=1;i<n-1;i++)
  {
      temp=a[i];
       j=i-1;
         while(j>0 && temp<a[j])
           {
               a[j+1]=a[j];
                j--;
            }
              a[j+1]=temp;
      }
```

3.3 Complexity

To insert last element we require n-1 comparisons and n-1 movements

To insert n-1st element we require n-2 comparisons and n-2 movements

To sum up

$2^*(1+2+3+\ldots\ldots n-1) = 2^*(n-1)^*n/2 = (n-1)^*n = O(n^2)$

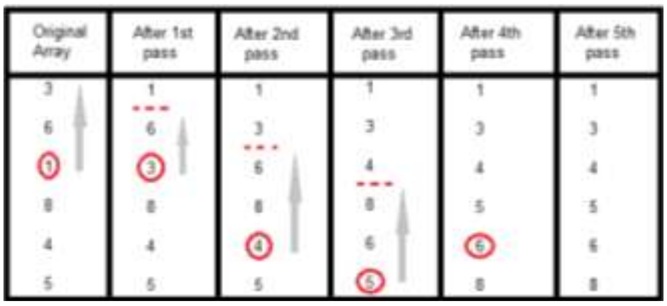
If the greater part of array is sorted the complexity is O(n).

So the average complexity is O(n)

IV. SELECTION SORT

It is conceptually the most simplest algorithm. This algorithm finds the most smallest elements and exchanges it with the element in the first position, then finds the second smallest element and exchanges it with the element in the second position and the procedure goes on.

4.1 WORKING PROCEDURE OF ALGORITHM



In the first pass , the smallest element found is 1 , so it is placed at the first position , then leaving the first element , smallest element is searched form rest of the elements , 3 is the smallest , so it is placed at the second position. Then leaving 1 and 3, from rest of the elements we search for the smallest element and put it in the third position , keep doing this until the array is sorted.

4.2 Algorithm

```
Read the array
for(i=0;i<n-2;i++)
   {
       for(j=i+1;j<n-1;j++)
         {
            if(a[i]>a[j])
               {
                   temp=a[i];
                   a[i]=a[j];
                  a[j]=temp;
               }
          }
     }
```

4.3 Complexity

T(n)=n+(n-1)+(n-2)+............+2+1 make pair the first and last terms, the second and next to last terms, and so forth. it ends up looking like this:-

T(n)=[n+1]+[(n-1)+2]+[(n-2)+3]+......... =(n+1)+(n+1)+(n+1)+...........

In an algebra , the sum of first N integers is given by the formula,

n(n+1)/2 therefore, T(n)=n(n+1)/2 =((n^2)+n)/2 =((n^2)/2)+(n/2)

So,Complexity of selection sort is O(n2)

V. CONCLUSION

- Bubble Sort

Bubble sort	Comparisons
Best case	O(n^2)
Average case	O(n^2)
Worst case	O(n^2)

- ✓ It takes several passes to sort the elements in an array.
- ✓ Every pass need to do comparisons between the elements and exchange the data if the elements are not in right order.

- ✓ Hence the complexity of bubble sort is same in all the cases.

- ➢ Selection Sort

Selection Sort	Comparisons
Best case	O(n^2)
Average case	O(n^2)
Worst case	O(n^2)

- ✓ The time complexity of selection sort is same in all the cases.
- ✓ The efficiency of selection sort does not depend upon the initial arrangement of the data.

- ➢ Insertion sort

Insertion Sort	Comparisons
Best case	O(n)
Average case	O(n^2)
Worst case	O(n^2)

- ✓ Adaptive (i.e., efficient) for data sets that are already substantially sorted: the time complexity is O(n + d), where d is the number of inversions.
- ✓ More efficient in practice than most other simple quadratic (i.e., O(n2)) algorithms such as selection sort or bubble sort; the best case (nearly sorted input) is O(n).

REFERENCES

[1] *C* and *C++* written by Sumita arora and D.S salaria.

[2] *goggle search*