Group 51 -Write an efficient algorithm to sort a given list of numbers using insertion sort. Trace the content of every location of the array in this process. Also, track the movement of every element in the list while sorting.

IHM2016501[1]IHM2016005[2] BIM2016004[3] IIM2016006[4] RIT2015050[5]

I. INTRODUCTION AND LITERATURE SURVEY

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.It works the way many people sort a hand of playing cards. We start with an empty left hand and the cards face down on the table. We then remove one card at a time from the table and insert it into the correct position in the left hand. To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left like below image.So,here in the question we are given a array of numbers of size n, we need to sort the given array using the above mentioned insertion sort.We also need to track and movement of every element in the list while sorting i.e trace the path of each and every element and trace the content of every location of the array in this process i.e we have to show how the array is been changed during the sorting.

II. ALGORITHM DESIGN

A. Input Format

In first few lines of the code we use $rand()$ function to generate a random number and assign it to the variable n which defines the size of the array.And then we use $rand()$ function to generate n random numbers and store them in the array.So.we now have a array of size n filled with n random numbers.

- A structure $element$ is been used which had a filed of type integer to stores the key and value of type integer which stores the value present in the array.
- We have a array $a[]$ of the above mentioned structure type and also a variable temp(which is used for swapping).
- Each element of the array has a unique key value.
- This Structure is declared globally.
- Also the variables n and iteration are declared globally

B. Algorithm

Algorithm 1 Insertion_sort

```
procedure INSERTION_SORT($pos[][102]$)          ▷ using insertion sort
    for i=0 to n-1 do
        for k=0 to n-1 do
            $pos[k][iteration + 1] \leftarrow pos[k][iteration]$
        $iteration + +$
        $temp.value \leftarrow a[i].value$
        $temp.key \leftarrow a[i].key$
        $j \leftarrow i - 1$
        while 0<=j & temp.value <arr[j].value do
            $a[j + 1].value \leftarrow a[j].value$
            $pos[a[i].key][iteration] \leftarrow j + 1$
            $j - -$
        $arr[j + 1].value \leftarrow temp.value$
        $arr[j + 1].key \leftarrow temp.key$
        print(i)
        for j=0 to n-1 do print(a[j].value)
```

```
procedure MAIN          ▷ main function
    for j=0 to n-1 do
        $a[j].key \leftarrow j$
        $pos[a[j].key][iteration] \leftarrow j$
    for j=0 to n-1 do print(a[j].value)
    $insertion\_sort(pos)$
    for i=0 to n-1 do
        for j=0 to n-1 do
            if a[j].key==i then
                break
        for j=0 to iteration do print(pos[i][j])
    for i=0 to n-1 do print(a[i].value)
```

C. Tracing the content of every location and movement of every element

- To trace the content of every location we are simply printing the modified array after each and every block of sorting.
- But tracking the movement of every element is not that easy as the array may have repetition of same digit.So,we provide unique key value to each and every

element of the array and thus identify them uniquely.We then track the movement of the every element based on the value.

D. Output Format

As mentioned in the question we are printing the array to trace the elements after each step of the insertion sort and then we are printing the original array

III. ANALYSIS

A. Analysis

1) Best Case Analysis Insertion Sort: When the array is sorted in increasing order we get the best case the example is Suppose we have the array [2, 3, 5, 7, 11], where the sorted subarray is the first four elements, and we're inserting the value 11. Upon the first test, we find that 11 is greater than 7, and so no elements in the subarray need to slide over to the right. Then this call of insert takes just constant time. Suppose that every call of insert takes constant time. Because there are n-1 calls to insert, if each call takes time that is some constant c, then the total time for insertion sort is c (n-1), which is $O(n)$

$$=> time_{best} \propto n$$

$$=> O(n) \text{ - A linear Time Computation}$$

2) Best Case Analysis: the best case is in which the array is in increasing order but here we have to trace the the path of every element so we have to insert and keep the track of every element as from the above analysis of insertion sort the best case for insert is constant c and we running a loop upto n to keep track of every element so it is cXn and there n elements in array so the time is c(n-1)(n-1).

$$time_{worst} \propto 3n^2 + k \text{ instructions.(k-constant)}$$
$$=> time_{worst} \propto n^2$$

$$=> O(n^2) \text{ - A Quadratic Time Computation}$$

3) Worst Case Analysis : As discussed above in the worst case senario also we have A call to insert causes every element to slide over if the key being inserted is less than every element to its left. So, if every element is less than every element to its left, the running time of insertion sort is $O(n^2)$

$$time_{worst} \propto 4n^2 + k \text{ instructions.(k-constant)}$$
$$=> time_{worst} \propto n^2$$

$$=> O(n^2) \text{ - A Quadratic Time Computation}$$

$$=> t_{best} = t_{worst}$$

4) Average Case Analysis : We know that

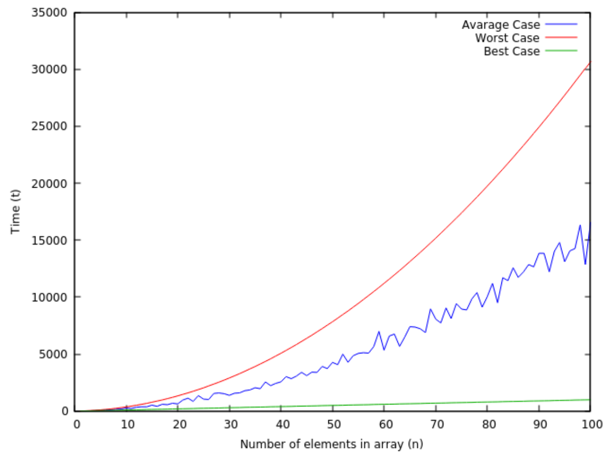
$$time_{best} <= t_{average} <= t_{worst}$$

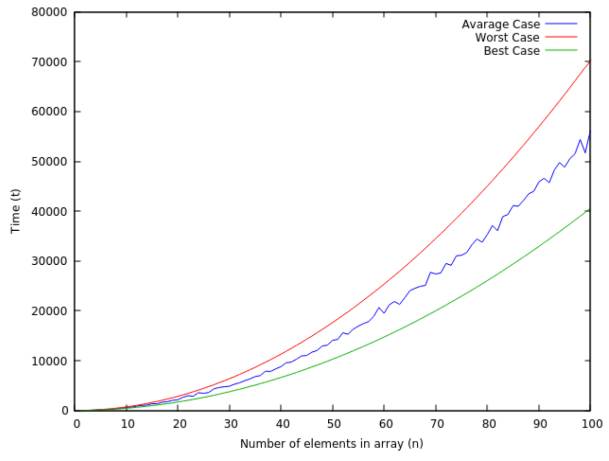
But from above
$$t_{best} = t_{worst}$$
So
$$=> t_{best} = t_{avg} = t_{worst}$$

IV. EXPERIMENTALSTUDY

The best way to study an algorithm is by graphs and profiling.

1) Graphs-Time-Complexity: We have seen that $t_{best} = t_{avg} = t_{worst}$ So,the graphs for best case,average,worst case time complexity remain the same.We can clearly see that as though Brute force and efficient algorithm are of order n^2 efficient algorithm is a lot better than the other one.





2) Profiling:

$NumberOfComputations$

n	t_{avg}	t_{best}	t_{worst}
3	44	44	62
5	156	120	180
10	630	450	720
20	2142	1710	2850
50	14070	10290	17640
70	27390	20010	34500
80	35322	26070	45030
100	56136	40590	70290

- We can seen that $t_{best} = t_{avg} = t_{worst}$
- We can also see that time increases as the value of n increases and $time_{worst} \propto n^2$,$time_{avg} \propto n^2$,$time_{best} \propto n^2$.

V. DISCUSSIONS

A. Understanding the Use Of File Handling and generation of random numbers

In this context the file handling is used to generate the random test cases through which analysis and experimentation of algorithm becomes easier for handling different test cases. We have used $srand()$ and $rand()$ to generate the random values to fill the matrix with the help of $< time.h >$ header file.

B. Tracing the content of every location and movement of every element

- To trace the content of every location we are simply printing the modified array after each and every block of sorting.
- But tracking the movement of every element is not that easy as the array may have repetition of same digit.So,we provide unique key value to each and every element of the array and thus identify them uniquely.We then track the movement of the every element based on the value.

VI. CONCLUSIONS

- We can see that the Complexity of Insertion Sort is $O(n)$ in best case,$O(n^2)$ in average and worst cases.But as we need to track the movement of every element we need to run another loop havind n iterations so now the complexity is $O(n^2)$ in all the cases
- In this solution we have bounded the size of the matrix between 1 and 100 as these values are sufficient enough for us to understand and analyze the result and draw conclusions.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)