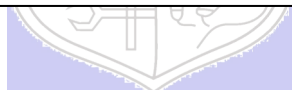




Experiment No. : 1

Title: Basic Sorting algorithm and its analysis



Batch: B-4

Roll No.: 16010422234

Name: Chandana Ramesh Galgali

Experiment No.: 1**Aim:** To implement and analyse time complexity of insertion sort.**Explanation and Working of insertion sort:**

To sort an array of size N in ascending order, iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

We take an unsorted array for our example.



Insertion sort compares the first two elements.



It finds that both 14 and 33 are already in ascending order. For now, 14 is in the sorted sub-list.



Insertion sort moves ahead and compares 33 with 27.



And finds that 33 is not in the correct position.



It swaps 33 with 27. It also checks with all the elements of the sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.



By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.



These values are not in a sorted order.



So we swap them.



However, swapping makes 27 and 10 unsorted.



Hence, we swap them too.



Again we find 14 and 10 in an unsorted order.



We swap them again. By the end of the third iteration, we have a sorted sub-list of 4 items.



This process goes on until all the unsorted values are covered in a sorted sub-list.

Algorithm of insertion sort:

The simple steps of achieving the insertion sort are listed as follows -

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step 2 - Pick the next element, and store it separately in a key.

Step 3 - Now, compare the key with all elements in the sorted array.

Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element. Else, shift greater elements in the array towards the right.

Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Derivation of Analysis insertion sort:**Worst Case Analysis**

The worst-case time complexity of insertion sort is $O(n^2)$, where n is the number of elements in the array. This occurs when the array is in reverse sorted order, and each element must be compared and shifted to the beginning of the array.

Best Case Analysis

The best-case time complexity of insertion sort is $O(n)$, which happens when the array is already sorted. In this case, the inner loop will not execute, and only the outer loop will iterate through the elements.

Average Case Analysis

The average-case time complexity of insertion sort is also $O(n^2)$. This is because, on average, each element will need to be compared and shifted approximately $\frac{n}{2}$ times, resulting in a quadratic time complexity.

Program(s) of insertion sort:


```
#include<stdio.h>
#include<stdlib.h>
int insertion_sort(int arr[], 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--;
        }
        arr[j+1]=key;
    }
    return arr[n];
}
```

```

}
int main(){
    int n, i;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("\nEnter the elements of the array:\n");
    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
    arr[n]=insertion_sort(arr, n);
    printf("\nThe sorted array is:\n");
    for(i=0; i<n; i++){
        printf("%d\t", arr[i]);
    }
    return 0;
}

```

Output(o) of insertion sort:

 "C:\Users\chand\Downloads\IV SEM\AA\EXP-1\exp-1\Untitled1.exe"

```

Enter the size of the array: 8

Enter the elements of the array:
14 33 27 10 35 19 42 44

The sorted array is:
10      14      19      27      33      35      42      44
Process returned 0 (0x0)   execution time : 66.807 s
Press any key to continue.

```

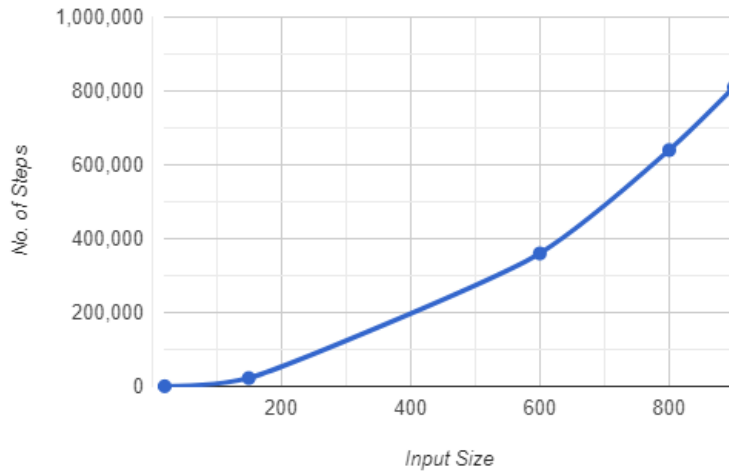
Results:

Time Complexity of Insertion sort:

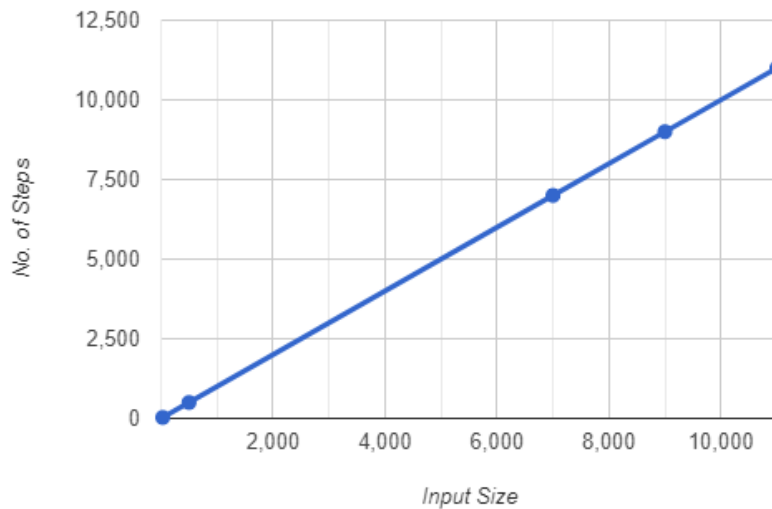
Worst Case Analysis:

| Sr. No. | Input size | No: of steps from Algorithm analysis | No: of steps from Theoretical analysis |
|---------|------------|---|---|
| 1) | 20 | 400 | 400 |
| 2) | 150 | 22500 | 22500 |
| 3) | 600 | 360000 | 360000 |

| | | | |
|----|-----|--------|--------|
| 4) | 800 | 640000 | 640000 |
| 5) | 900 | 810000 | 810000 |

GRAPH:**Best Case Analysis:**

| Sr. No. | Input size | No: of steps from Algorithm analysis | No: of steps from Theoretical analysis |
|---------|------------|--------------------------------------|--|
| 1) | 3 | 3 | 3 |
| 2) | 50 | 50 | 50 |
| 3) | 700 | 700 | 700 |
| 4) | 9000 | 90000 | 90000 |
| 5) | 11000 | 11000 | 11000 |

GRAPH:

Conclusion: (Based on the observations)

Insertion sort is a straightforward algorithm with a time complexity that makes it suitable for small input sizes or nearly sorted arrays. However, for larger input sizes, other sorting algorithms with better average time complexity, such as merge sort or quicksort, may be more appropriate.

Outcome: Analyze time and space complexity of basic algorithms

References:

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Cormen ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.

