Priyansh Salian
2003148
C31

Experiment 1B

Aim:- To implement insertion sort

Theory:-

In simple words we can say that, take an element from the unsorted array, place it in its corresponding position in the sorted part, & shift the elements accordingly.
~~Insertion Sort~~

Insertion sort is a simple sorting algorithm that works ~~simp~~ similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the sorted part are picked and placed at the correct position in the sorted part.

Algorithm

To sort ~~the~~ an array ~~of~~ size n in ascending order

1. Iterate from arr[1] to arr[n] over the array.
2. Compare the current element (key) to its ~~prece~~ predecessor
3. If the key element is smaller than ~~the~~ its predecessor, compare it to the elements ~~bef~~ before. Move the greater elements one position up to make space ~~for~~ the swapped element.

Priyansh Salian
2003148
C31

Page:
Date:

Example

Let consider an array
arr = 12, 11, 13, 5, 6

We will consider first element as the smallest

Now lets loop from i = 1 (second element in the array) To 4 (last element ~~of~~ the array)
Since 11 is smaller than 12, move 12 and insert 11 before 12.

11, 12, 13, 5, 6

Now i = 2, here 13 will remain at its position as all elements in A[0... i-1] are smaller than 13

11, 12, 13, 5, 6
i = 3, here 5 will move ~~at~~ to the beginning and all other elements from 11 to ~~st~~ 13 will move one position ~~as~~ ahead ~~of~~ their current position.

5, 11, 12, 13, 6
i = 4, here 6 will move ~~at~~ after 5, and elements from 11 to 13 will move one position ahead ~~of~~ their current position.

Finally we get our sorted array

5, 6, 11, 12, 13

Priyansh Salian
2003148
C31

Page :
Date :

Application

1. Insertion sort is used when number ~~of~~ elements ~~is~~ are small.

2. ~~Wh~~ Used when input array is almost sorted, only few elements are ~~misle~~ ~~pr~~ misplaced in complete big array.

~~3. This al type of sorting is an im~~
3. Insertion sort is an in-place algorithm, ~~meaning~~ meaning it requires no extra space.

~~3~~ 4. Maintains relative order ~~of~~ the input data in case ~~of~~ ~~the~~ two equal values.

Analysis

Let's ~~find~~ the time required to execute each line

	Cost	Times
~~for~~ (int i = 1; i < N; i ++)	C_1	n
{		
k = arr[i];	C_2	$n-1$
space = i;	0	$n-1$
~~for~~ (int j = space-1); j<0; j++)	C_4	$n-1$
{	C_5	$\sum_{j=2}^{n} t_j$
if (arr[j] > key)	C_6	$\sum_{j=2}^{n} (t_j - 1)$
{		
arr[j+1] = arr[j]	C_7	$\sum_{j=2}^{n} (t_j - 1)$
space = j;		
continue;		
}		
break;	C8	contin
} arr [space] = key;		

Priyansh Salian
2003148
C31

Page:
Date:

$$\text{Total Time} = C_1(n) + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n} t_j + C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 \sum_{j=2}^{n} (t_j - 1) + C_8(n-1)$$

Best Case :- Here array is already sorted so the if statement ~~of~~ the inner loop will never be executed

$$\text{Total time} = C_1 n + C_2(n-1) + C_4(n-1) + C_5(n-1) + C_8(n-1)$$
$$= (C_1 + C_2 + C_4 + C_5 + C_8)n + (C_2 + C_4 + C_5 + C_8)$$
$$= an + b = O(n)$$

Worst case :- Here the array will be in descending order & we want to arrange it ~~as~~ in ascending order. Each and every ~~sp~~ step will be executed

$$\text{Total time } (T(n)) :- C_1 n + C_2(n-1) + C_4(n-1) + C_5\left[\frac{(n+1)n}{2} - 1\right] + C_6 \frac{n(n-1)}{2} + C_7 \frac{n(n-1)}{2} + C_8(n-1)$$

$$= O(n^2)$$

Average Case :- Here some elements will not be sorted but the overall time complexity will come out to be $O(n^2)$

Space Complexity $= O(1)$:- It iterates over every element extract that out to a variable, and compare it against all ~~of~~ its left elements. So only space taken is for that variable. Space utilized does not depend on how big the array is so it will be executed in constant time

CODE:-

ex_1_b.cpp

```cpp
#include <iostream>

using namespace std;

int main()
{
    int a;
    cout << "Enter the no of elements you want to sort:-";
    cin >> a;
    const int n = a;
    int arr[n];
    cout << "Enter the elements you want to sort:-";

    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    for (int i = 1; i < n; i++)
    {
        int current = arr[i];
        int j = i - 1;
        while (arr[j] > current && j >= 0)
```

```cpp
        cin >> arr[i];
    }
    for (int i = 1; i < n; i++)
    {
        int current = arr[i];
        int j = i - 1;
        while (arr[j] > current && j >= 0)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = current;
    }
    for (int i = 0; i < n; i++)
    {
        cout << arr[i];
        cout << "\n"
    }
}
```

OUTPUT:-

```
Enter the elements you want to sort:56
32
24
44
12
Elements in sorted order are as follows:-12
24
32
44
56


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

Priyansh Salian
2003148
C 31

Page :
Date :

Conclusion

Insertion sort works best with small number ~~of~~ of elements. The ~~tu~~ worst case runtime complexity of insertion sort is $O(n^2)$ similar to that ~~of~~ of Bubble Sort. However, ~~Insr~~ Insertion Sort is considered better than Bubble Sort.