Experiment No. 2
To implement Insertion Sort
Date of Performance:15/02/2024
Date of Submission:22/02/2024

Experiment No. 2

Title: Insertion Sort

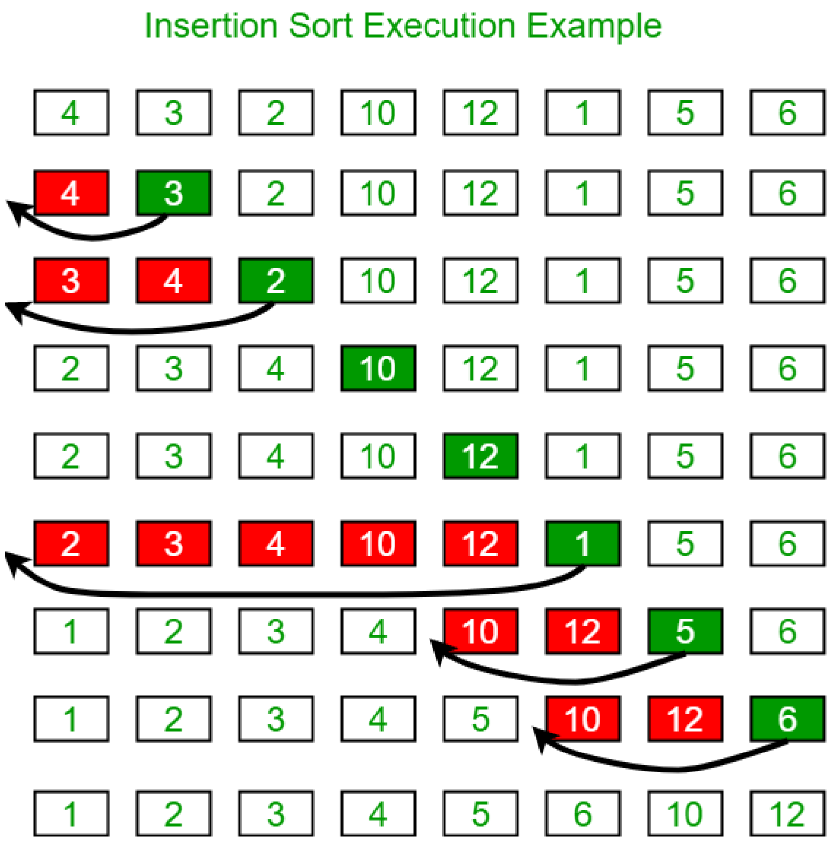
Aim: To study, implement and Analyze Insertion Sort Algorithm

Objective: To introduce the methods of designing and analyzing algorithms

Theory:

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

Example:



Algorithm and Complexity:

Algorithm Insertion Sort (A)	**Cost**	**Time**
// A is an array of size n		
for j ← 2 to n **do**	c_1	n
key ← A[j]	c_2	n - 1
i ← j - 1	c_3	n - 1
while (i > 0 && A[i] > key) **do**	c_4	$\sum_{j=2}^{n} t_j$
A [i + 1] ← A[i]	c_5	$\sum_{j=2}^{n}(t_j - 1)$
i ← i - 1	c_6	$\sum_{j=2}^{n}(t_j - 1)$
end	–	–
A[i + 1] ← key	c_7	n - 1
end		

Best case analysis:

- Let size of the input array is n. Total time taken by algorithm is the summation of time taken by each of its instruction.

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot \left(\sum_{j=2}^{n} t_j\right) + c_5 \cdot \sum_{j=2}^{n} (t_j - 1) + c_6 \cdot \sum_{j=2}^{n} (t_j - 1) + c_7 \cdot (n-1)$$

- The best case offers the lower bound of the algorithm's running time.
- When data is already sorted, the best scenario for insertion sort happens.
- In this case, the condition in the while loop will never be satisfied, resulting in tj = 1.

$$T(n) = c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \sum_{j=2}^{n} 1 + c_5 \sum_{j=2}^{n} 0 + c_6 \sum_{j=2}^{n} 0 + c_7 \cdot (n-1)$$

Where,

$$\sum_{j=2}^{n} 1 = 1 + 1 + \ldots + 1 \ (n-1 \text{ times}) = n - 1$$

$= c_1 \times n + c_2 \times n - c_2 + c_3 \times n - c_3 + c_4 \times n - c_4 + c_7 \times n - c_7$

$= (c_1 + c_2 + c_3 + c_7)\ n - (c_3 + c_4 + c_4 + c_7)$

$= a\ n + b$ Which is linear function of n

$= O(n)$

Worst case analysis:

- The worst-case running time gives an upper bound of running time for any input.
- The running time of algorithm cannot get worse than its worst-case running time.
- Worst case for insertion sort occurs when data is sorted in reverse order.
- So we must have to compare A[j] with each element of sorted array A[1 … j – 1]. So, $t_j = j$

$$\begin{aligned}\sum_{j=2}^{n} j &= 2 + 3 + 4 + \ldots + n \\ &= (1 + 2 + 3 + \ldots + n) - 1 = \sum n - 1 \\ &= \frac{n(n+1)}{2} - 1\end{aligned}$$

$$\sum_{j=2}^{n} (j-1) = 1 + 2 + 3 + \ldots + n - 1 = \Sigma\,(n-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned}T(n) &= c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot \left(\sum_{j=2}^{n} j\right) + c_5 \cdot \sum_{j=2}^{n} (j-1) + c_6 \cdot \sum_{j=2}^{n} (j-1) + c_7 \cdot (n-1) \\ &= c_1 \cdot n + c_2 \cdot (n-1) + c_3 \cdot (n-1) + c_4 \cdot \left(\frac{n(n+1)}{2} - 1\right) + c_5 \cdot \frac{n(n-1)}{2} + c_6 \cdot \frac{n(n-1)}{2} + c_7 \cdot (n-1) \\ &= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) \cdot n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) \cdot n - (c_2 + c_3 + c_4 + c_7)\end{aligned}$$

$= an^2 + bn + c$ which is quadratic function of n

$= O(n^2)$

Average Case Analysis

- Let's assume that $t_j = (j-1)/2$ to calculate the average case
  Therefore,
  $T(n) = C_1 * n + (C_2 + C_3) * (n - 1) + C_4/2 * (n - 1)(n) / 2 + (C_5 + C_6)/2 * ((n - 1)(n) / 2 - 1) + C_8 * (n - 1)$

further simplified has dominating factor of $\mathbf{n^2}$ and gives $T(n) = C * (n^2)$ or $O(n^2)$

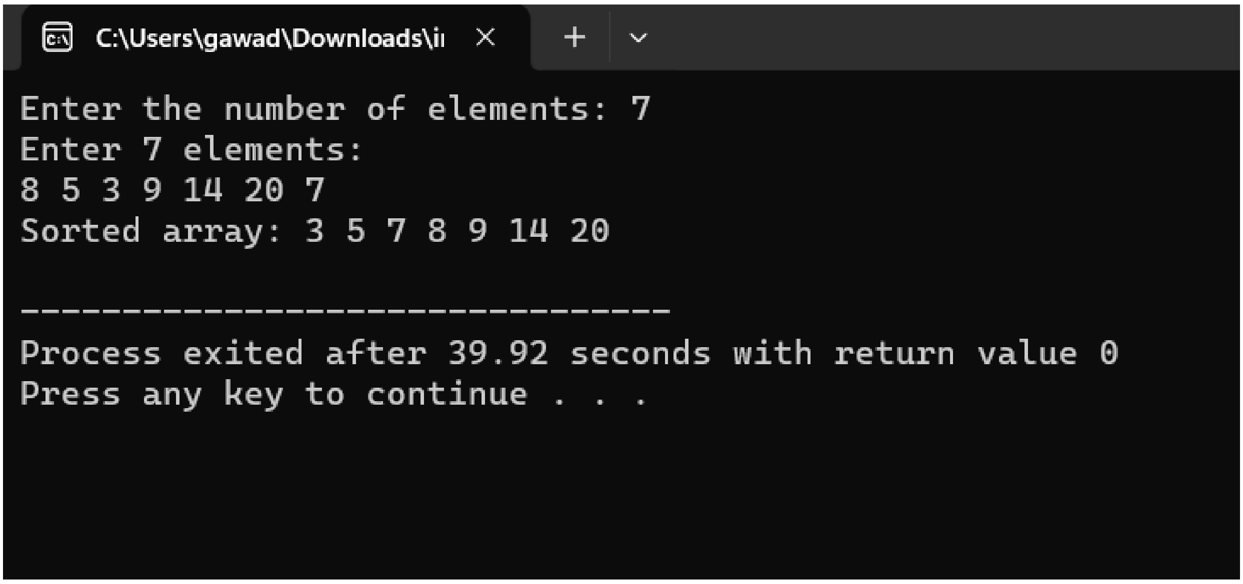
Code:

```
#include <stdio.h>


// Function to sort an array using insertion sort

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++)

    {

        key = arr[i];

        j = i - 1;



        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j = j - 1;
```

```
        }
        arr[j + 1] = key;
    }
}



void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}



int main()
{
    int n;



    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
```

```
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);



    insertionSort(arr, n);



    printf("Sorted array: ");
    printArray(arr, n);


    return 0;
}
```

Output:



Conclusion:

In conclusion, Insertion Sort is a straightforward and practical sorting technique, ideal for small datasets or partially sorted arrays. Although its time complexity is O(n^2), it remains a valuable option for scenarios where simplicity is prioritized over speed.