Experiment No.1
Insertion Sort
Date of Performance:
Date of Submission:

Title: Insertion Sort

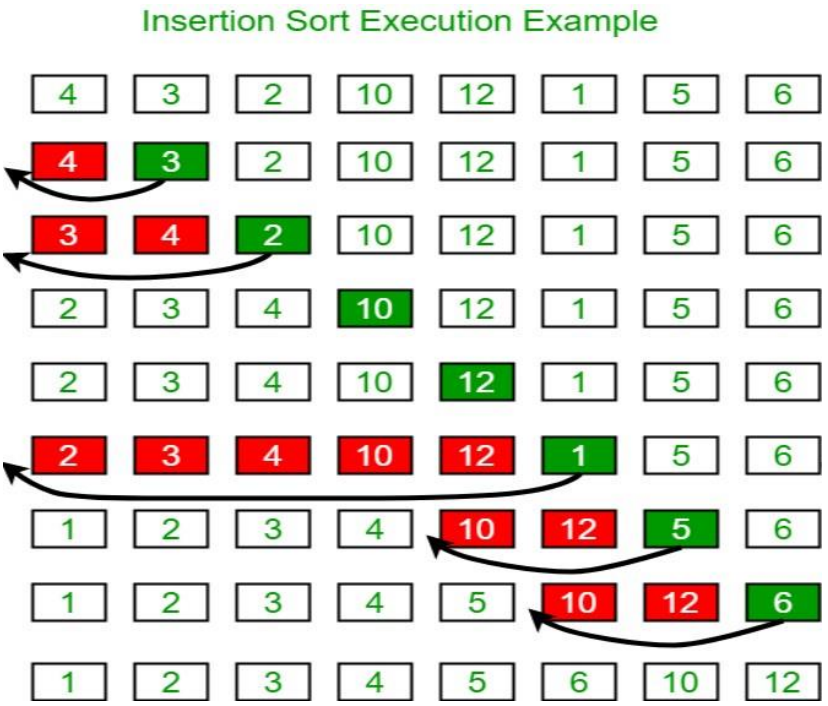
Aim: To implement Selection Comparative analysis for large values of ‘n’

Objective: To introduce the methods of designing and analysing algorithms

Theory:

Insertion sort is a simple sorting algorithm that works 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 unsorted part are picked and placed at the correct position in the sorted part.

Example:



Algorithm and Complexity:

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

Code :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to perform insertion sort
void insertionSort(int arr[], int n) {
   int i, key, j;
   for (i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;
      // Move elements of arr[0..i-1], that are greater than key, to one position ahead
of their current position
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}

// Function to generate random array of given size
void generateRandomArray(int arr[], int n) {
   srand(time(NULL));
```

```
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000; // Generating random numbers between 0 and 999
    }
}

int main() {
    int n, i;

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

    int arr[n];

    // Generating random array
    generateRandomArray(arr, n);

    // Sorting the array using Insertion Sort
    clock_t start = clock();
    insertionSort(arr, n);
    clock_t end = clock();
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Displaying sorted array
    printf("Sorted array: ");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Displaying time taken for sorting
    printf("Time taken for sorting: %f seconds\n", time_taken);

    return 0;
}
```

Output :

```
PS E:\Testing_Lang> cd "e:\Testing_Lang\" ; if ($?) { gcc test.c -o test } ; if ($?) { .\test }
Enter the number of elements: 5
Sorted array: 255 539 591 680 843
Time taken for sorting: 0.000000 seconds
PS E:\Testing_Lang>
```

Conclusion:

This program prompts the user to enter the number of elements 'n'. It then generates a random array of 'n' elements, sorts the array using the Insertion Sort algorithm, and displays the sorted array along with the time taken for sorting.

To perform a comparative analysis for large values of 'n', you can modify the program to execute multiple times with increasing values of 'n' and measure the time taken for each execution. Additionally, you can compare the performance of Insertion Sort with other sorting algorithms to analyze their efficiency for different input sizes.