



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

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



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

Title: Insertion Sort

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

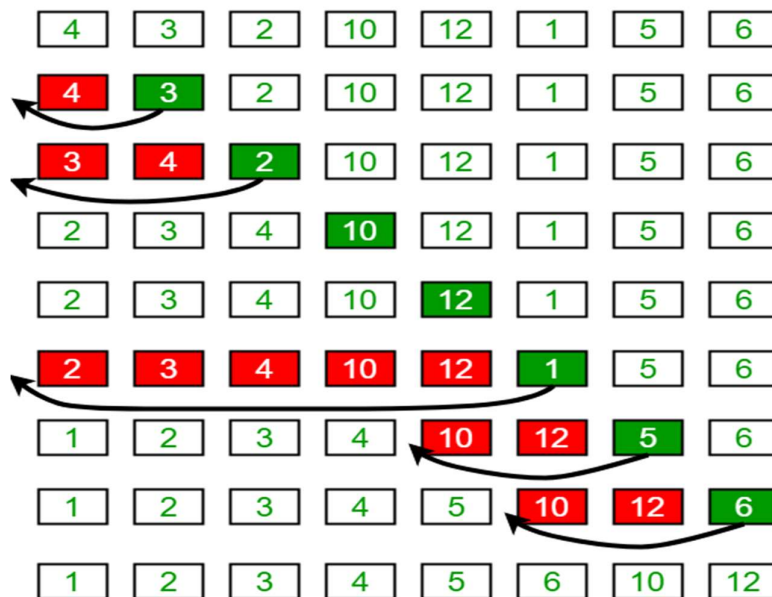
Objective: To introduce the methods of designing and analysing algorithms

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:

Insertion Sort Execution Example



Algorithm and Complexity:



INSERTION-SORT (<i>A</i>)	<i>cost</i>	<i>times</i>
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	$n - 1$

Implementation:

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

```
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;
```

```
    }
```

```
}
```

```
void printArray(int arr[], int n) {
```

```
    int i;
```



Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

```
for (i = 0; i < n; i++)  
    printf("%d ", arr[i]);  
printf("\n");  
}  
  
int main() {  
    int arr[] = { 12, 11, 13, 5, 6 };  
    int n = sizeof(arr) / sizeof(arr[0]);  
    printf("Given array is \n");  
    printArray(arr, n);  
    insertionSort(arr, n);  
    printf("Sorted array is \n");  
    printArray(arr, n);  
    return 0;  
}
```

Output:

A screenshot of a Turbo C++ console window. The title bar shows the path 'C:\TURBOC3\BIN' and the application name 'TC'. The output text is as follows:
Given array is
12 11 13 5 6
Sorted array is
5 6 11 12 13
The cursor is positioned at the end of the last line of output.

Conclusion: The implementation of the insertion sort algorithm demonstrated its effectiveness in sorting small to moderate-sized datasets. While its simplicity and efficiency are notable, scalability limitations highlight the need for alternative algorithms for larger datasets. Nonetheless, insertion sort remains a valuable foundational concept in computer science education.