Insertion Sort:

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.

Characteristics of Insertion Sort:
This algorithm is one of the simplest algorithm with simple implementation
Basically, Insertion sort is efficient for small data values
Insertion sort is adaptive in nature, i.e. it is appropriate for data sets which are already partially sorted.
Working of Insertion Sort algorithm:

Consider an example: arr[]: {12, 11, 13, 5, 6}

12 11 13 5 6
First Pass:

Initially, the first two elements of the array are compared in insertion sort.
12 11 13 5 6
Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.
So, for now 11 is stored in a sorted sub-array.
11 12 13 5 6
Second Pass:

Now, move to the next two elements and compare them

11 12 13 5 6

Here, 13 is greater than 12, thus both elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11
Third Pass:

Now, two elements are present in the sorted sub-array which are 11 and 12

Moving forward to the next two elements which are 13 and 5

11 12 13 5 6

Both 5 and 13 are not present at their correct place so swap them

11 12 5 13 6

After swapping, elements 12 and 5 are not sorted, thus swap again

11 5 12 13 6

Here, again 11 and 5 are not sorted, hence swap again

5 11 12 13 6

here, it is at its correct position
Fourth Pass:

Now, the elements which are present in the sorted sub-array are 5, 11 and 12
Moving to the next two elements 13 and 6

5 11 12 13 6

Clearly, they are not sorted, thus perform swap between both

5 11 12 6 13

Now, 6 is smaller than 12, hence, swap again

5 11 6 12 13

Here, also swapping makes 11 and 6 unsorted hence, swap again

5 6 11 12 13
Finally, the array is completely sorted.

Insertion Sort Algorithm
To sort an array of size N in ascending order:

Iterate from arr[1] to arr[N] over the array.
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.

Below is the implementation of C#:

```
using System;

class InsertionSort {

    // Function to sort array
    // using insertion sort
    void sort(int[] arr)
    {
        int n = arr.Length;
        for (int i = 1; i < n; ++i) {
            int key = arr[i];
            int 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;
    }
}

// A utility function to print
// array of size n
static void printArray(int[] arr)
{
    int n = arr.Length;
    for (int i = 0; i < n; ++i)
        Console.Write(arr[i] + " ");

    Console.Write("\n");
}

// Driver Code
public static void Main()
{
    int[] arr = { 12, 11, 13, 5, 6 };
    InsertionSort ob = new InsertionSort();
    ob.sort(arr);
    printArray(arr);
}
}
```

Output
5 6 11 12 13