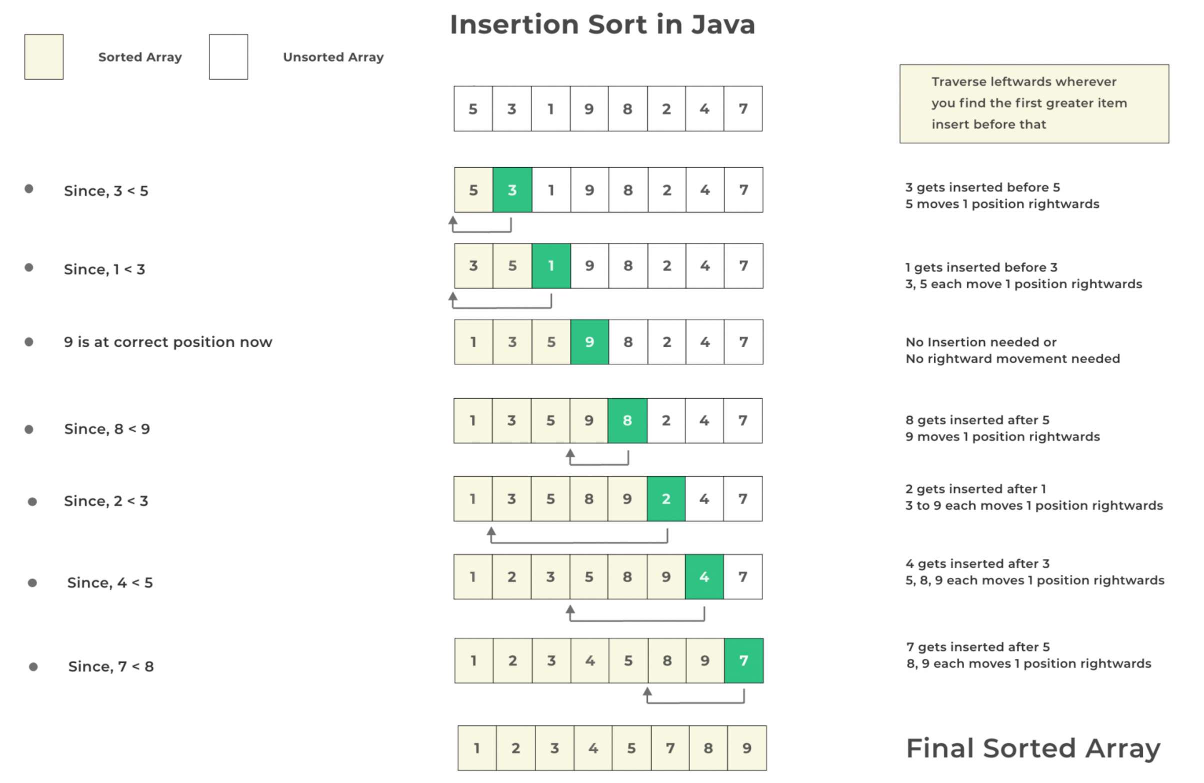
Insertion Sort in Java

This is an in-place comparison-based sorting algorithm, which is idle for small data-sets. Insertion sort is almost a resemblance of the way we sort playing cards in our hand. In this algorithm, we try to insert the elements of the array into an already sorted array.

Time Complexity	O(n2)
Best Case	Ω(n)
Worst Case	O(n2)
Aux. Space Complexity	O(1)
Best case when	Array is already sorted
Worst case when	Array is reverse sorted

Algorithm

```
Step 1: Repeat Steps 2 to 5 for K = 1 to N-1
Step 2: SET TEMP = ARR[K]
Step 3: SET J = K ? 1
Step 4: Repeat while TEMP <=ARR[J]
      SET ARR[J + 1] = ARR[J]
      SET J = J ? 1
      [END OF INNER LOOP]
 Step 5: SET ARR[J + 1] = TEMP
      [END OF LOOP]
 Step 6: EXIT
```



Insertion Sort in Java

Sorted Array
Unsorted Array
Traverse leftwards wherever you find the first greater item insert before that
5 3 1 9 8 2 4 7

- Since, 3 < 5 — 5 3 1 9 8 2 4 7 — 3 gets inserted before 5, 5 moves 1 position rightwards
- Since, 1 < 3 — 3 5 1 9 8 2 4 7 — 1 gets inserted before 3, 3, 5 each move 1 position rightwards
- 9 is at correct position now — 1 3 5 9 8 2 4 7 — No Insertion needed or No rightward movement needed
- Since, 8 < 9 — 1 3 5 9 8 2 4 7 — 8 gets inserted after 5, 9 moves 1 position rightwards
- Since, 2 < 3 — 1 3 5 8 9 2 4 7 — 2 gets inserted after 1, 3 to 9 each moves 1 position rightwards
- Since, 4 < 5 — 1 2 3 5 8 9 4 7 — 4 gets inserted after 3, 5, 8, 9 each moves 1 position rightwards
- Since, 7 < 8 — 1 2 3 4 5 8 9 7 — 7 gets inserted after 5, 8, 9 each moves 1 position rightwards

1 2 3 4 5 7 8 9 — Final Sorted Array

```
package Sort;

public class Insertion {
    /*Function to sort array using insertion sort*/
    static void insertionSort(int arr[])
    {
        int len = arr.length; //calculating the length of the array
        for (int i = 1; i < len; i++)
        {
            int key = arr[i];
            int j = i - 1;
            /* Shift elements of a[i-1 .... 0], that are greater
            than key, to one position right 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 a[])
    {
        int len = a.length;
        for (int i = 0; i < len; ++i)
            System.out.print(a[i] + " ");
        System.out.println();
    }
```

```
 // Main method
    public static void main(String args[])
    {
        int a[] = {11, 9, 7, 15, 6, 10, 5, 17};

        System.out.println("Array Before Insertion Sort: ");
        printArray(a);

        insertionSort(a);

        System.out.println("Array After Insertion Sort: ");
        printArray(a);
    }


}
```

- Works efficiently on smaller data sets in number
- Uses no additional memory for sorting as is in place algorithm with O(1) space complexity
- If data sets are already sorted or nearly sorted then has O(n) space complexity

- Very bad average time complexity of $O(n^2)$
- Shifting items because of insertion can be costly