**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.

*Insertion Sort*

**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, 5 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.*

***Illustrations:***

**

**Bubble Sort** is the simplest [sorting algorithm](https://www.geeksforgeeks.org/sorting-algorithms/) that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

***Input:****arr[] = {6, 3, 0, 5}*

***First Pass:***

* *Bubble sort starts with very first two elements, comparing them to check which one is greater.*
  + *( 6 3 0 5 ) –> (****3 6****0 5 ), Here, algorithm compares the first two elements, and swaps since 6 > 3.*
  + *( 3****6******0****5 ) –>  ( 3****0******6****5 ), Swap since 6 > 0*
  + *( 3**0****6******5****) –>  ( 3**0****5 6****), Swap since 6 > 5*

***Second Pass:***

* *Now, during second iteration it should look like this:*
  + *(****3 0****5 6 ) –>  (****0 3****5 6 ), Swap since 3 > 0*
  + *( 0****3 5****6 ) –>  ( 0****3 5****6 ), No change as 5 > 3*

***Third Pass:***

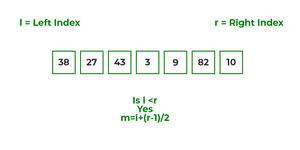
* *Now, the array is already sorted, but our algorithm does not know if it is completed.*
* *The algorithm needs one****whole****pass without****any****swap to know it is sorted.*
  + *(****0 3****5 6 ) –>  (****0 3****5 6 ), No change as 3 > 0*

*Array is now sorted and no more pass will happen.*

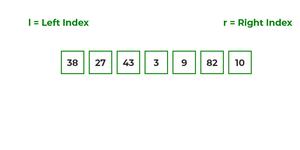
***Merge sort****is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.*

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

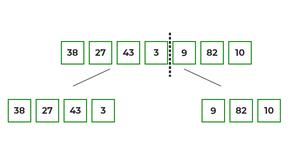
* *At first, check if the left index of array is less than the right index, if yes then calculate its mid point*

**

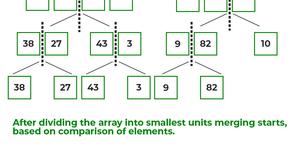
* *Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.*
* *Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.*

**

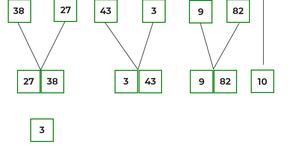
* *Now, again find that is left index is less than the right index for both arrays, if found yes, then again calculate mid points for both the arrays.*

**

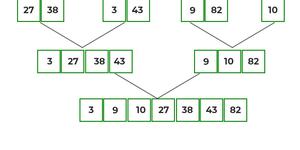
* *Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.*

**

* *After dividing the array into smallest units, start merging the elements again based on comparison of size of elements*
* *Firstly, compare the element for each list and then combine them into another list in a sorted manner.*

**

* *After the final merging, the list looks like this:*

**

The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}.

If we take a closer look at the diagram, we can see that the array is recursively divided into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.

