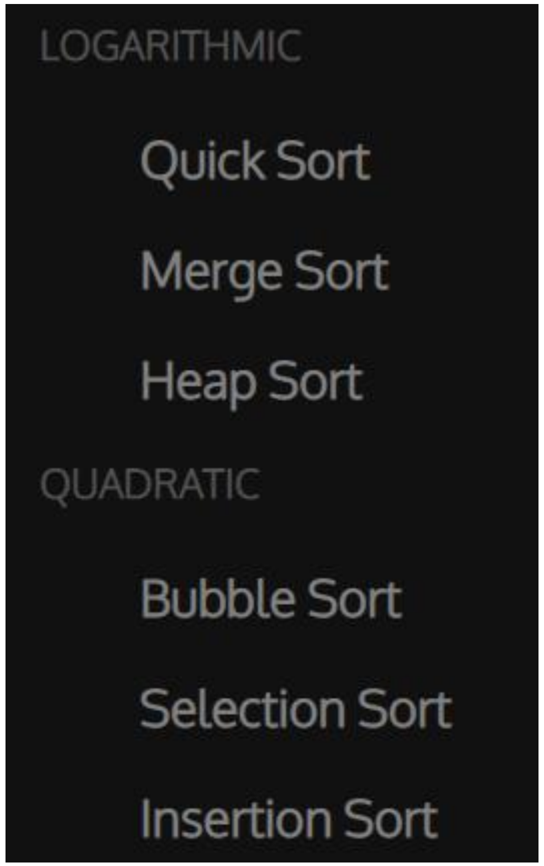
Todays Agenda

- Sorting – Introduction
- Bubble Sort
- Insertion Sort
- Selection Sort
- Quick & Merge


LOGARITHMIC

Quick Sort

Merge Sort

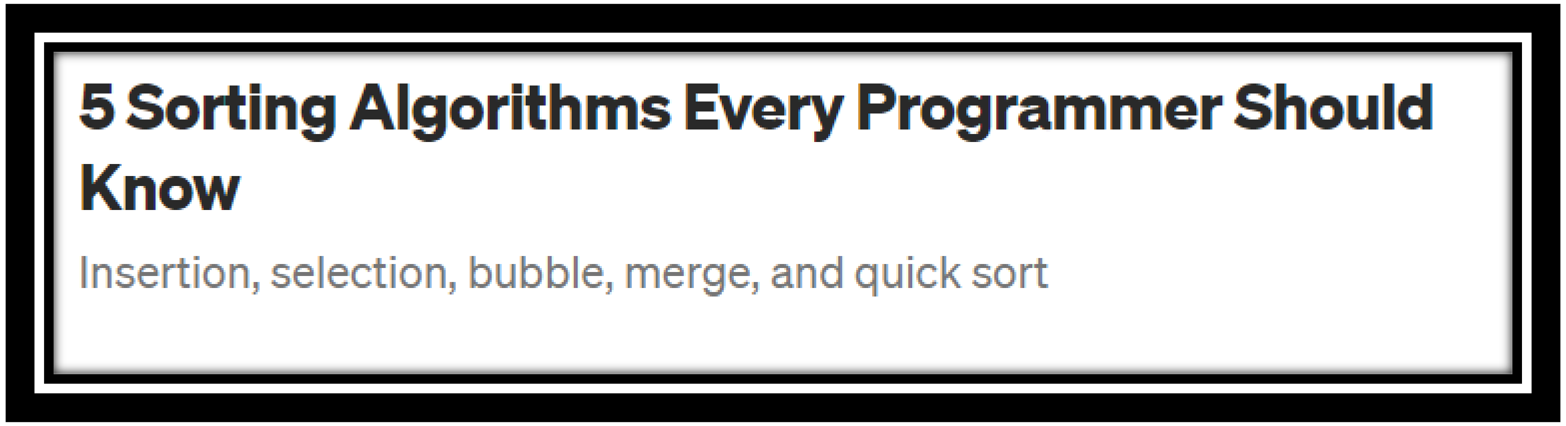
Heap Sort

QUADRATIC

Bubble Sort

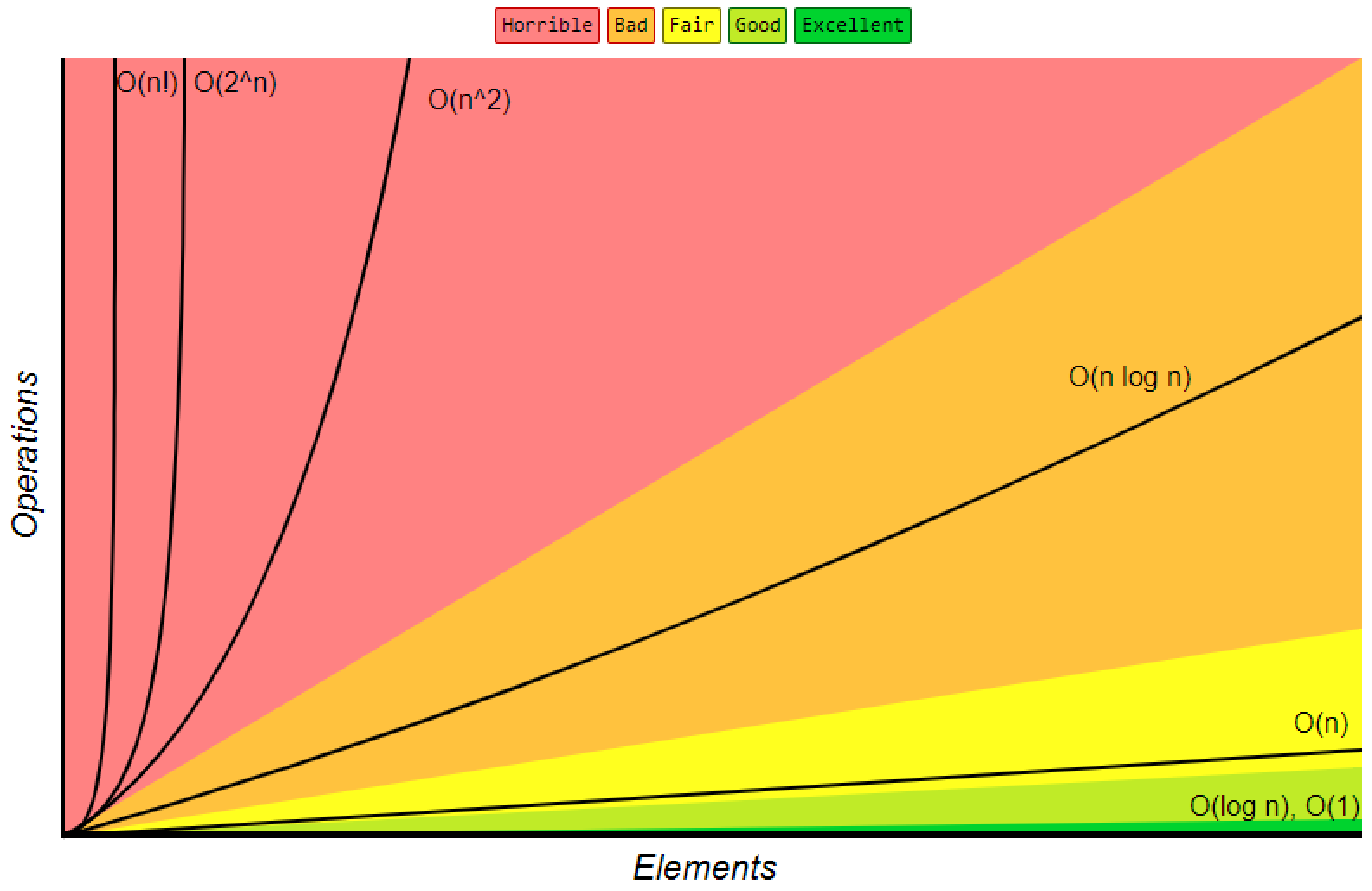
Selection Sort

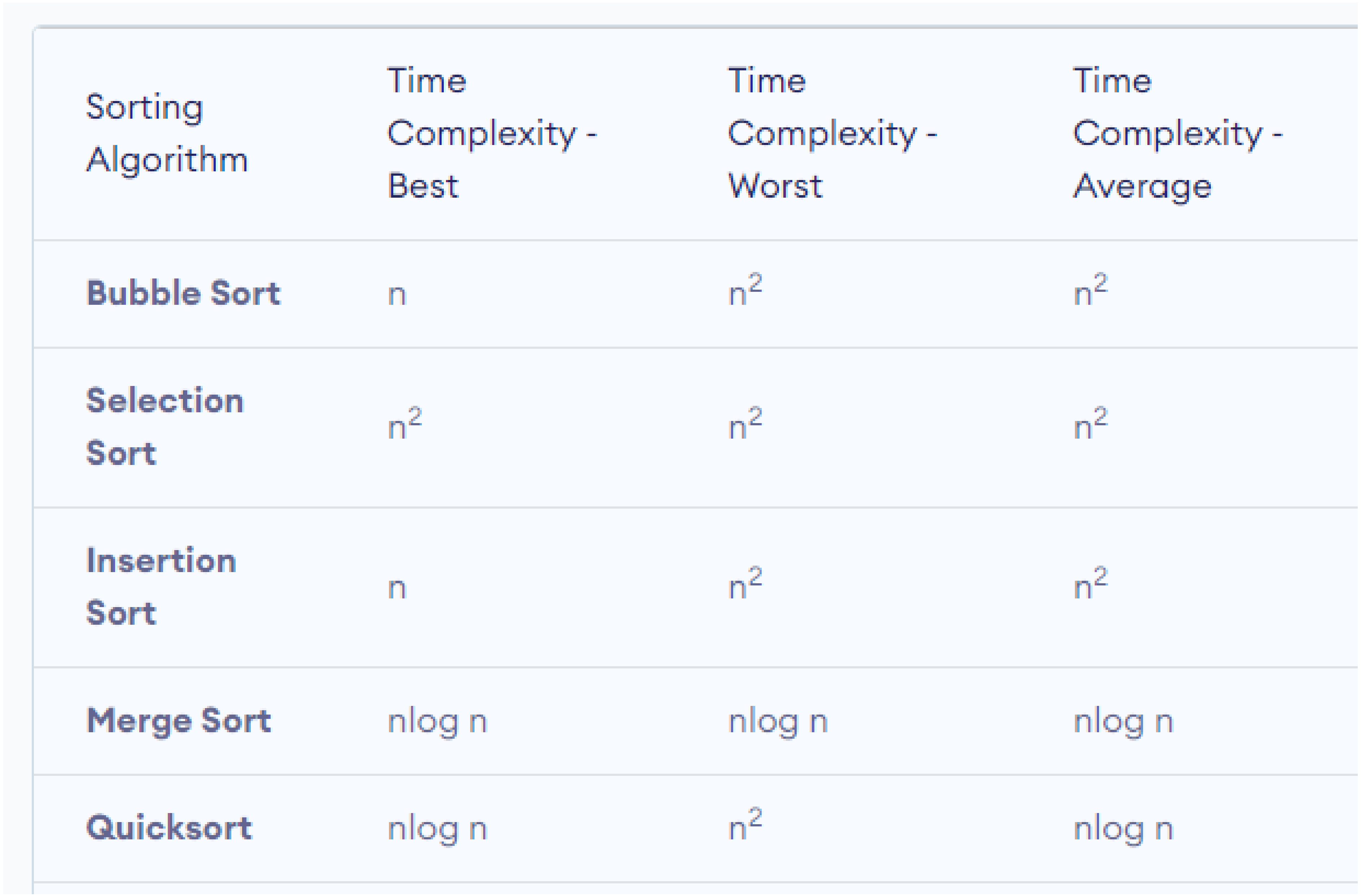
Insertion Sort


5 Sorting Algorithms Every Programmer Should Know

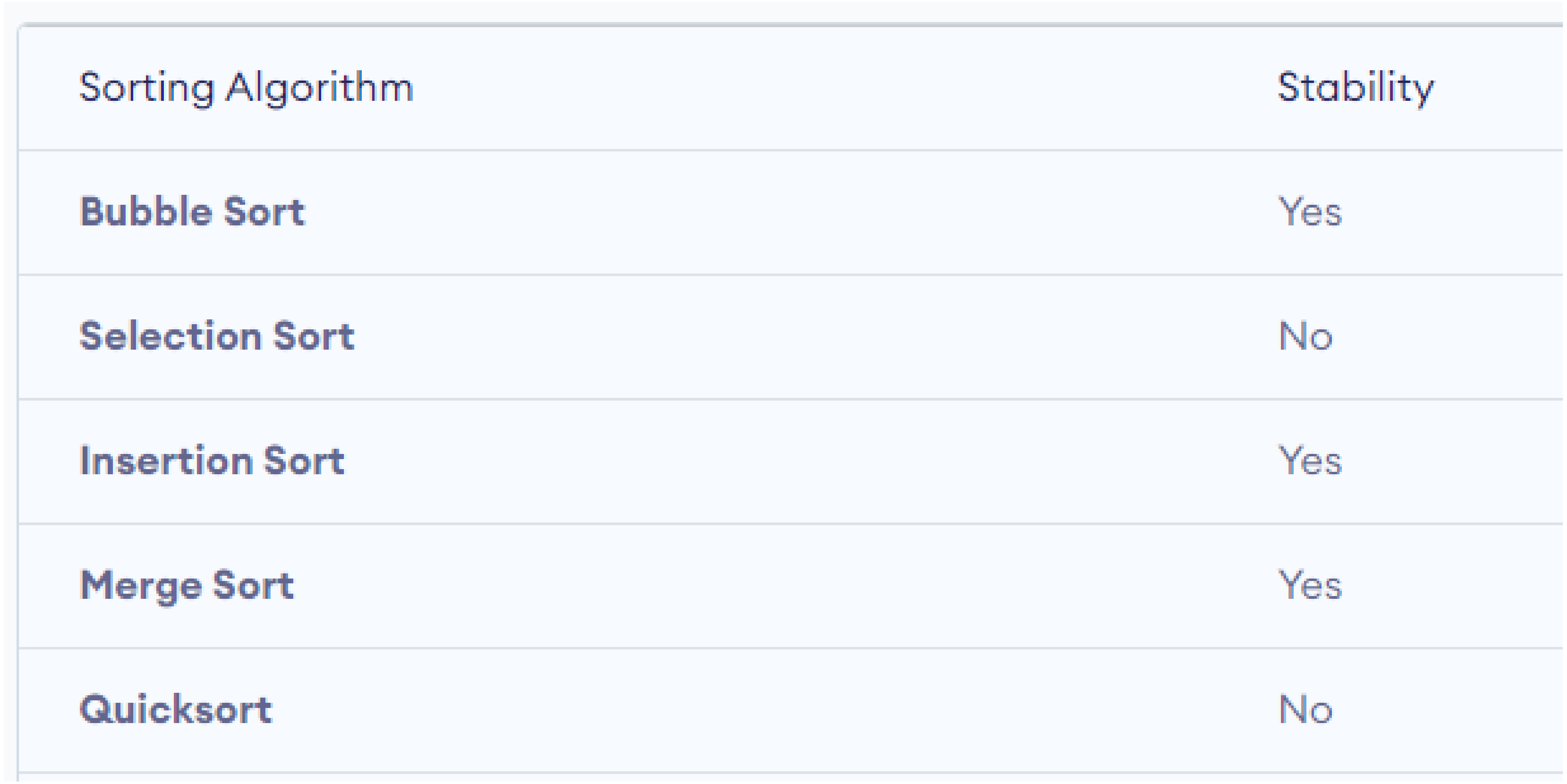
Insertion, selection, bubble, merge, and quick sort

Big-O Complexity Chart





Sorting Algorithm	Time Complexity - Best	Time Complexity - Worst	Time Complexity - Average
Bubble Sort	n	n^2	n^2
Selection Sort	n^2	n^2	n^2
Insertion Sort	n	n^2	n^2
Merge Sort	nlog n	nlog n	nlog n
Quicksort	nlog n	n^2	nlog n



Sorting Algorithm	Stability
Bubble Sort	Yes
Selection Sort	No
Insertion Sort	Yes
Merge Sort	Yes
Quicksort	No

Sorting

- Sorting refers to arranging data in a particular format.
- Sorting algorithm specifies the way to arrange data in a particular order.
- Most common orders are in numerical or lexicographical order
- Sorting algorithms can be used for collections of numbers, strings, characters, or a structure of any of these types.

Background

- The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner.
- Sorting is also used to represent data in more readable formats. Following are some of the examples of sorting in real-life scenarios −
  - ***Telephone Directory*** *− The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.*
  - ***Dictionary*** *− The dictionary stores words in an alphabetical order so that searching of any word becomes easy.*

Sorting Terminology

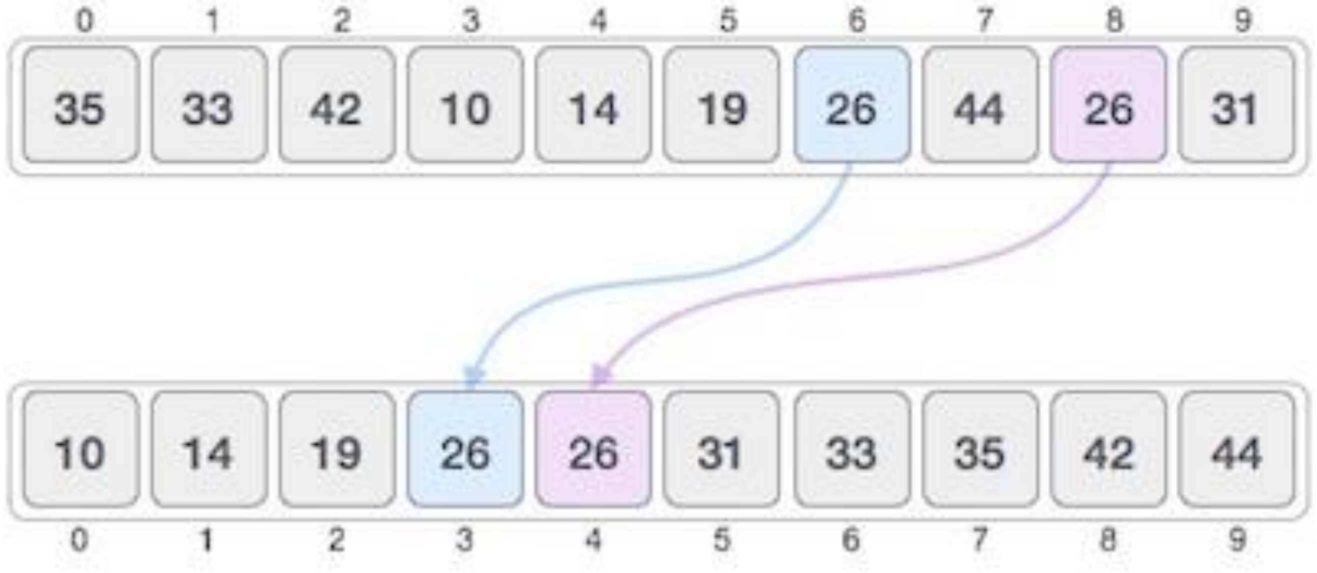
- Sorting algorithms may require some extra space for comparison and temporary storage of few data elements. These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself.
- This is called **in-place sorting**.
- Bubble sort is an example of in-place sorting.

Sorting Terminology

- However, in some sorting algorithms, the program requires space which is more than or equal to the elements being sorted.
- Sorting which uses equal or more space is called **not-in-place sorting**. Merge-sort is an example of not-in-place sorting.

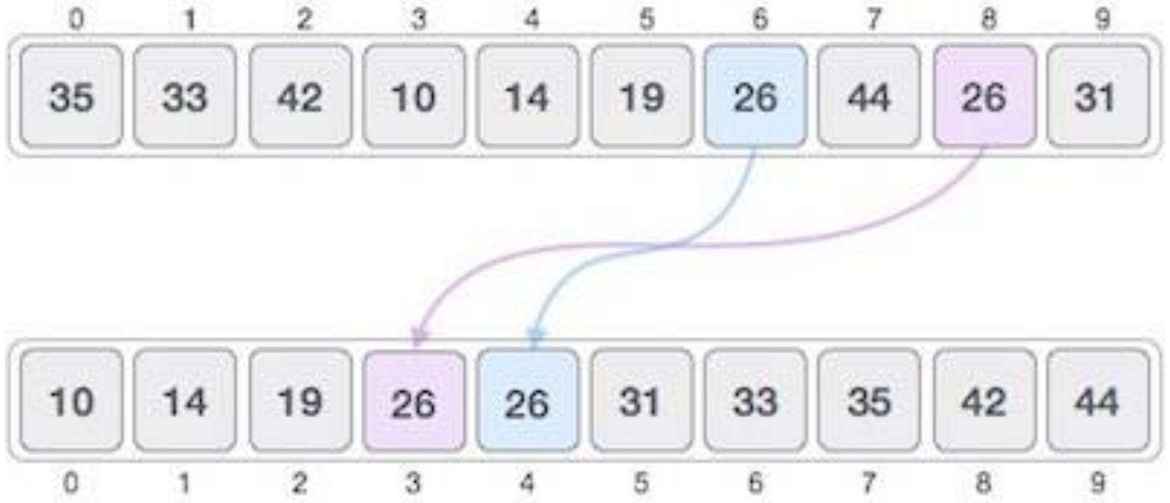
Stable Sorting

- If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called **stable sorting.**



Not Stable Sorting

- If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting.**



Adaptive and Non-Adaptive Sorting Algorithm

- if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.

Non-Adaptive Sorting Algorithm

- A non-adaptive algorithm is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.

Important Terms

- **Increasing Order**
- A sequence of values is said to be in **increasing order**, if the successive element is greater than the previous one.
- For example, 1, 3, 4, 6, 8, 9 are in increasing order, as every next element is greater than the previous element.

Important Terms

- Decreasing Order
- A sequence of values is said to be in **decreasing order**, if the successive element is less than the current one.
- For example, 9, 8, 6, 4, 3, 1 are in decreasing order, as every next element is less than the previous element.

Non-Increasing Order

- A sequence of values is said to be in **non-increasing order**, if the successive element is less than or equal to its previous element in the sequence.
- This order occurs when the sequence contains duplicate values.
- For example, 9, 8, 6, 3, 3, 1 are in non-increasing order, as every next element is less than or equal to (in case of 3) but not greater than any previous element.

Non-Decreasing Order

- A sequence of values is said to be in **non-decreasing order**, if the successive element is greater than or equal to its previous element in the sequence.
- This order occurs when the sequence contains duplicate values.
- For example, 1, 3, 3, 6, 8, 9 are in non-decreasing order, as every next element is greater than or equal to (in case of 3) but not less than the previous one.

Bubble Sort

- Bubble Sort is the simplest sorting algorithm 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

Bubble Sort

- ***Input:*** *arr[] = {5, 1, 4, 2, 8}*
- ***First Pass:***

- *Bubble sort starts with very first two elements, comparing them to check which one is greater.*
  - *(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.*
  - *(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4*
  - *(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2*
  - *(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.*

Bubble Sort

- ***Second Pass:***

- *Now, during second iteration it should look like this:*
  - *(**1 4** 2 5 8) –> (**1 4** 2 5 8)*
  - *(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2*
  - *(1 2 **4 5** 8) –> (1 2 **4 5** 8)*
  - *(1 2 4 **5 8**) –> (1 2 4 **5 8**)*

Bubble Sort

- ***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.*
  - *(**1** **2** 4 5 8) –> (**1** **2** 4 5 8)*
  - *(1 **2** **4** 5 8) –> (1 **2** **4** 5 8)*
  - *(1 2 **4** **5** 8) –> (1 2 **4** **5** 8)*
  - *(1 2 4 **5** **8**) –> (1 2 4 **5** **8**)*

Bubble Sort

- ***At pass 1 :*** *Number of comparisons = (n-1)*
  Number of swaps = (n-1)
- ***At pass 2 :*** *Number of comparisons = (n-2)*
  Number of swaps = (n-2)
- ***At pass 3 :*** *Number of comparisons = (n-3)*
  Number of swaps = (n-3)
- ***Total number of swaps = Total number of comparison***
  Total number of comparison (Worst case) = ***n(n-1)/2***
  Total number of swaps (Worst case) = ***n(n-1)/2***

Bubble Sort

- ***Worst and Average Case Time Complexity:*** *O(N^2). The worst case occurs when an array is reverse sorted.*

- ***Best Case Time Complexity:*** *O(N). The best case occurs when an array is already sorted.*

Insertion Sort

- Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Insertion Sort

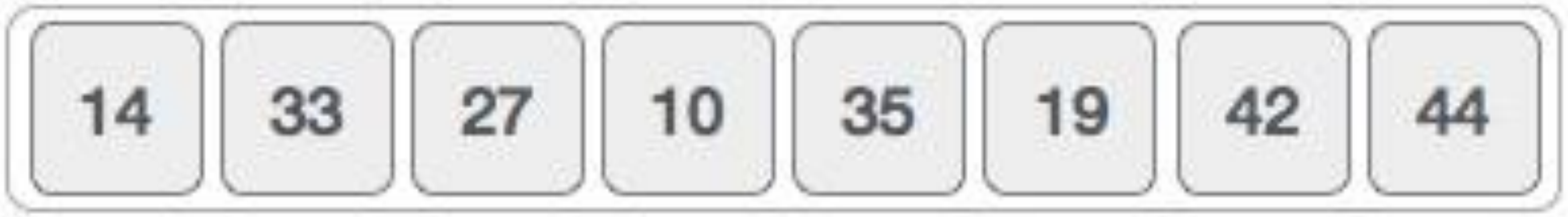
- This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted.
- An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there.
- Hence the name, **insertion sort**.

Insertion Sort

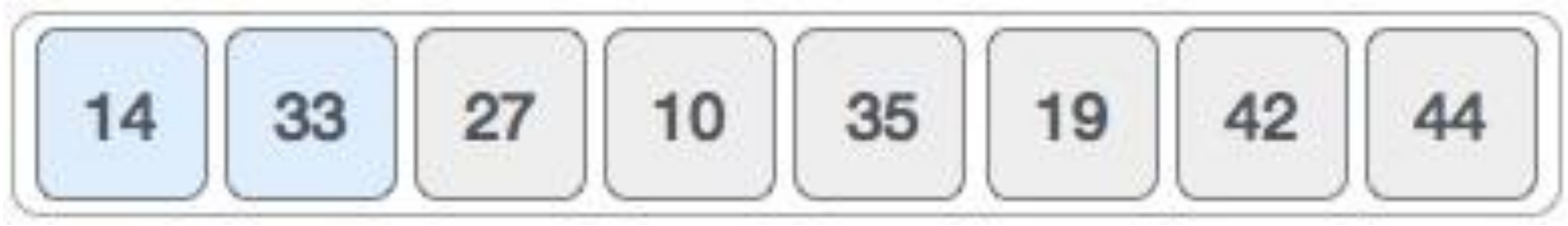
- The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).
- Use temporary variable

How it works?

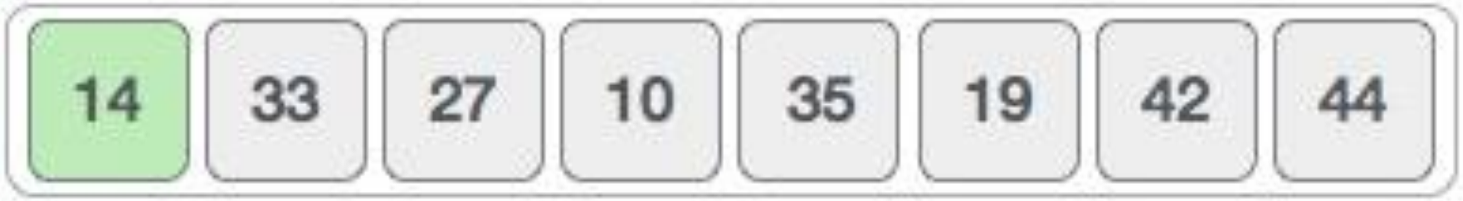
- An unsorted array for our example



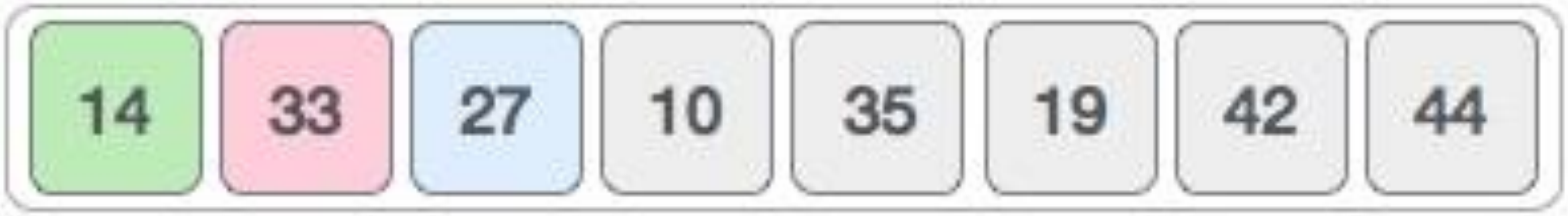
- Insertion sort compares the first two elements.



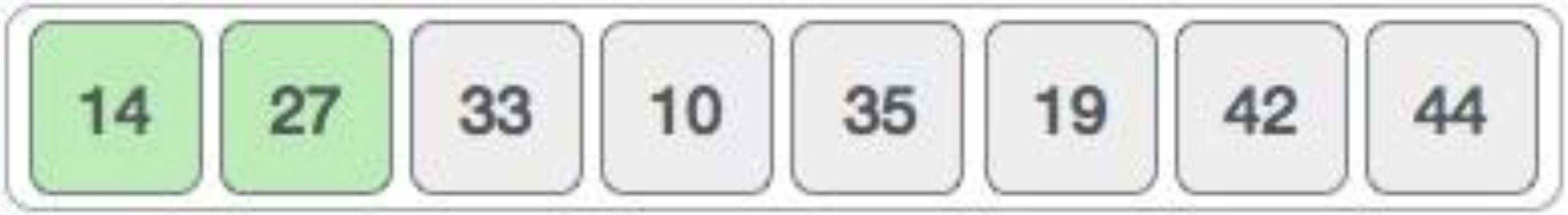
- It finds that both 14 and 33 are already in ascending order. For now, 14 is in sorted sub-list.



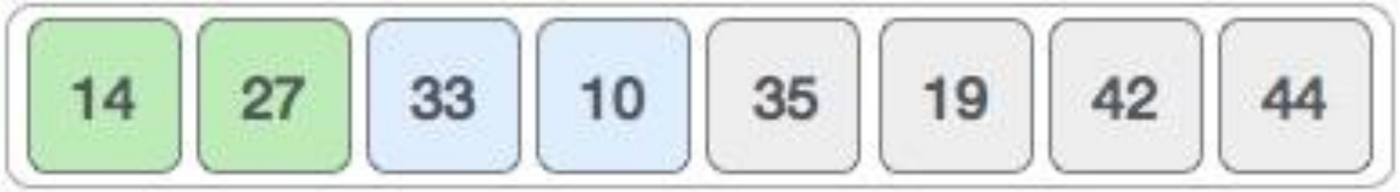
- Insertion sort moves ahead and compares 33 with 27.
- And finds that 33 is not in the correct position.



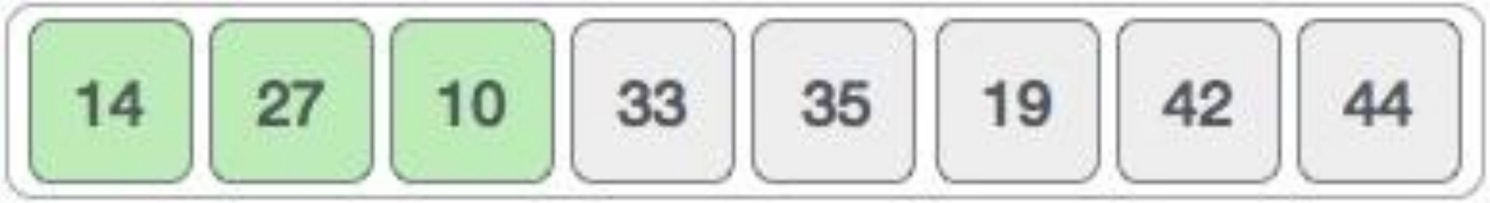
It swaps 33 with 27. It also checks with all the elements of sorted sub-list. Here we see that the sorted sub-list has only one element 14, and 27 is greater than 14. Hence, the sorted sub-list remains sorted after swapping.



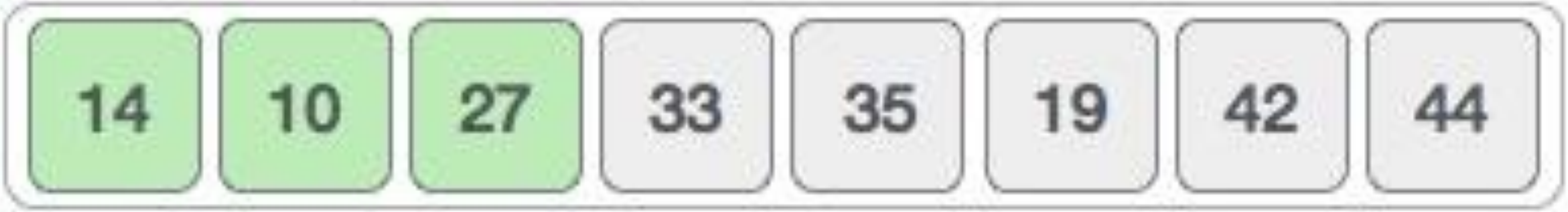
- By now we have 14 and 27 in the sorted sub-list. Next, it compares 33 with 10.



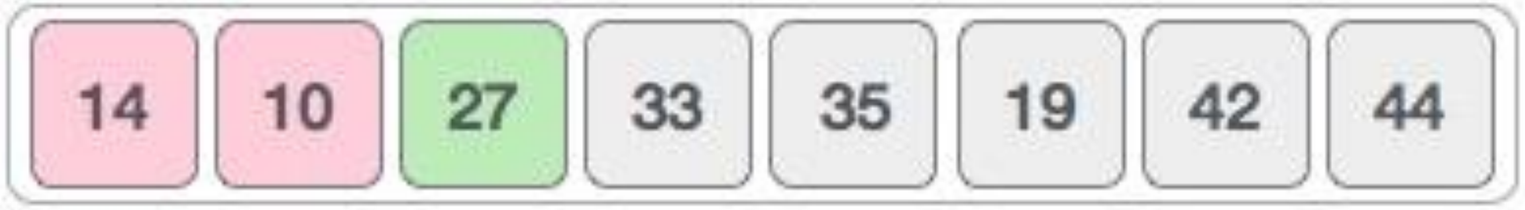
- These values are not in a sorted order.
- So we swap them.



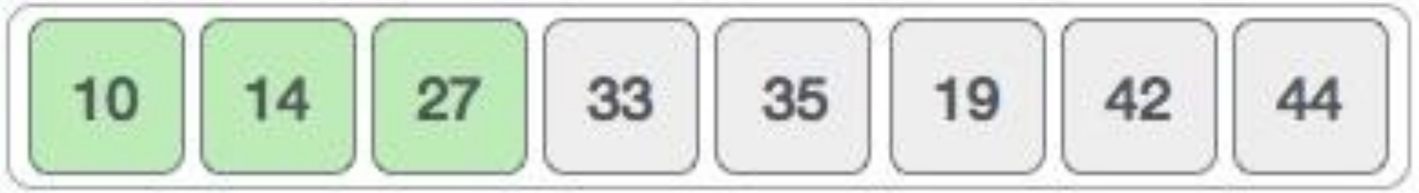
- However, swapping makes 27 and 10 unsorted.



- Again we find 14 and 10 in an unsorted order.

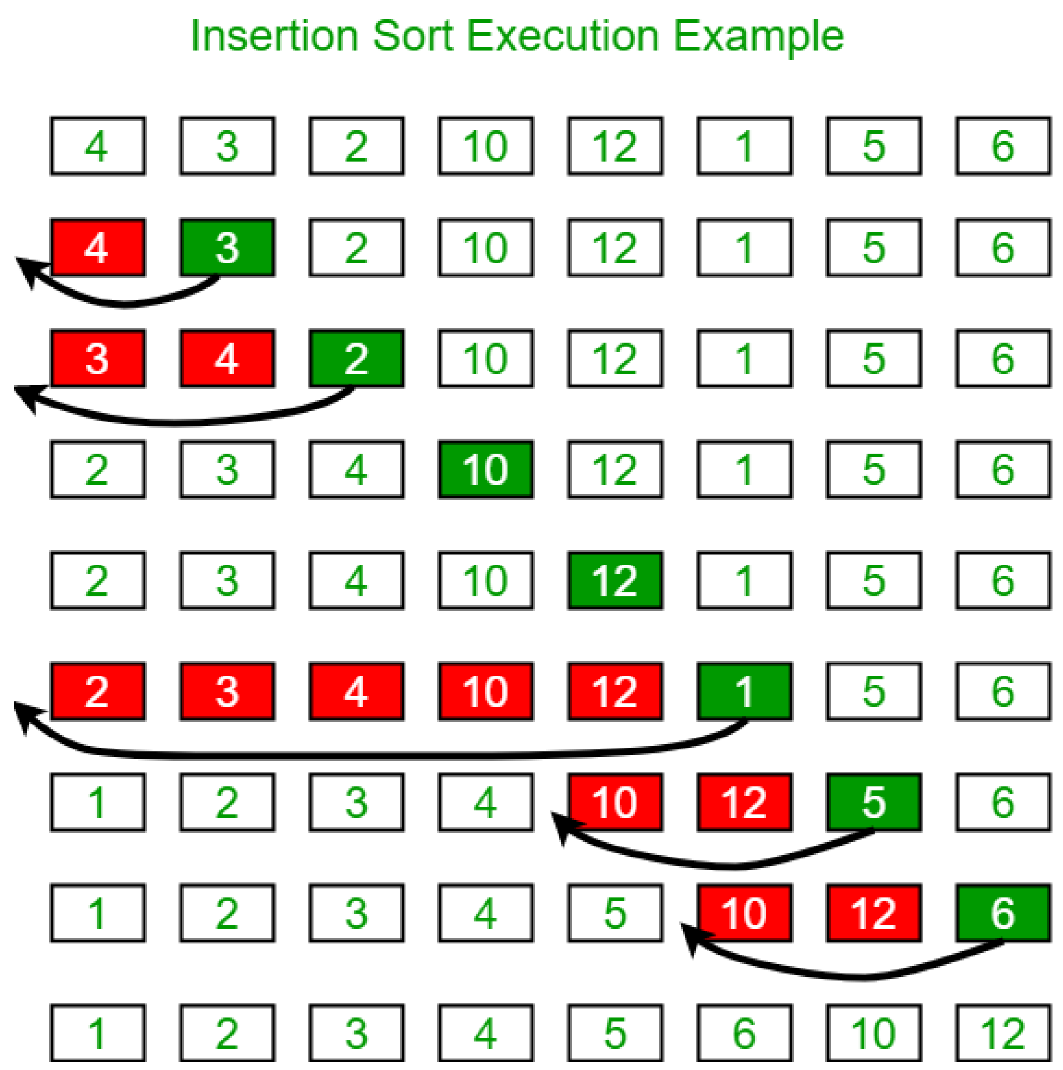


- We swap them again. By the end of third iteration, we have a sorted sub-list of 4 items.



- This process goes on until all the unsorted values are covered in a sorted sub-list.

Insertion Sort



Insertion Sort

- 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.
- **Time Complexity:** O(N^2)