

Sorting Algorithms

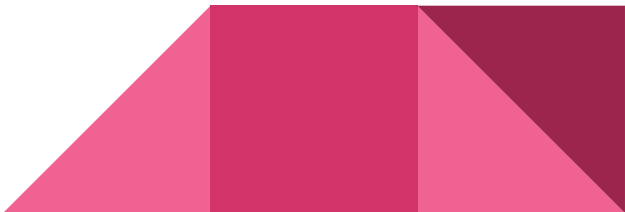
Part 1

Do Now

Given the following list of numbers : -2 , 45 , 0 , 11 , -9

Describe a strategy to put these numbers in order from smallest to largest

Note: Your strategy should work for any list of numbers, including a list that has some values repeated



Bubble Sort

Bubble Sort is one of the most widely discussed algorithms, simply because of its lack of efficiency for sorting arrays.

If an array is already sorted, Bubble Sort will only pass through the array once.

However, the worst case scenario has a time complexity of $O(N^2)$, which is extremely inefficient.



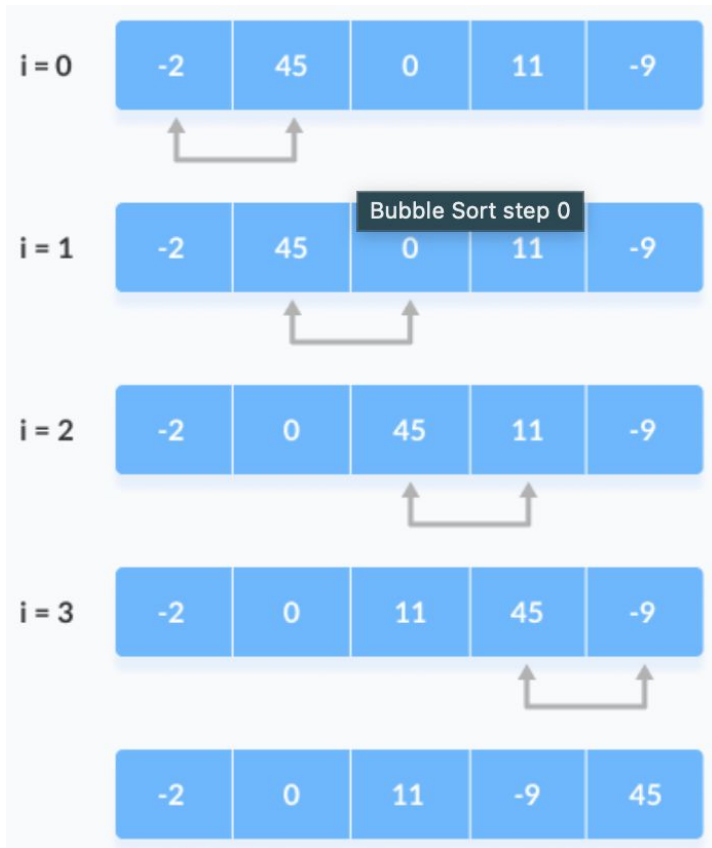
How does Bubble Sort work?

This algorithm compares two adjacent elements and swaps them until they are in the intended order.

As the algorithm progresses we have a sorted partition and an unsorted partition
(This is a logical partition, we do not have two arrays)



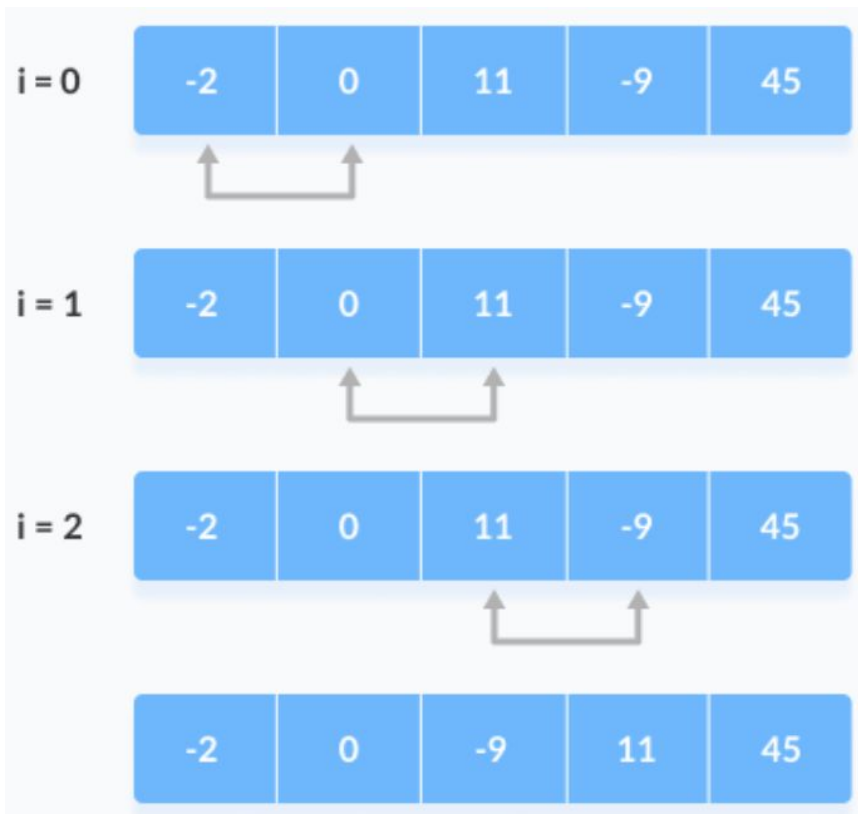
Bubble Sort - Step 0



1. Starting at index 0, compare the first and the second elements.
2. Swap the element if the first one is greater than the second one.
3. Next, compare the second and the third elements and swap them if the order is not correct.
4. Keep doing the process until your algorithm reaches the last element.

A logical sorted partition starts forming with the largest element placed at the end.

Bubble Sort - Step 1



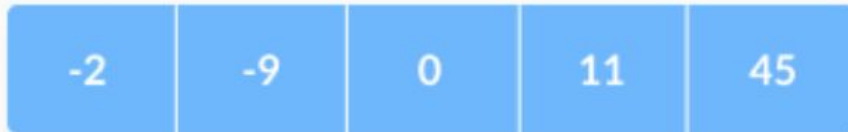
Repeat the comparing and swapping for the remaining iterations (unsorted partition).

Bubble Sort - Step 2

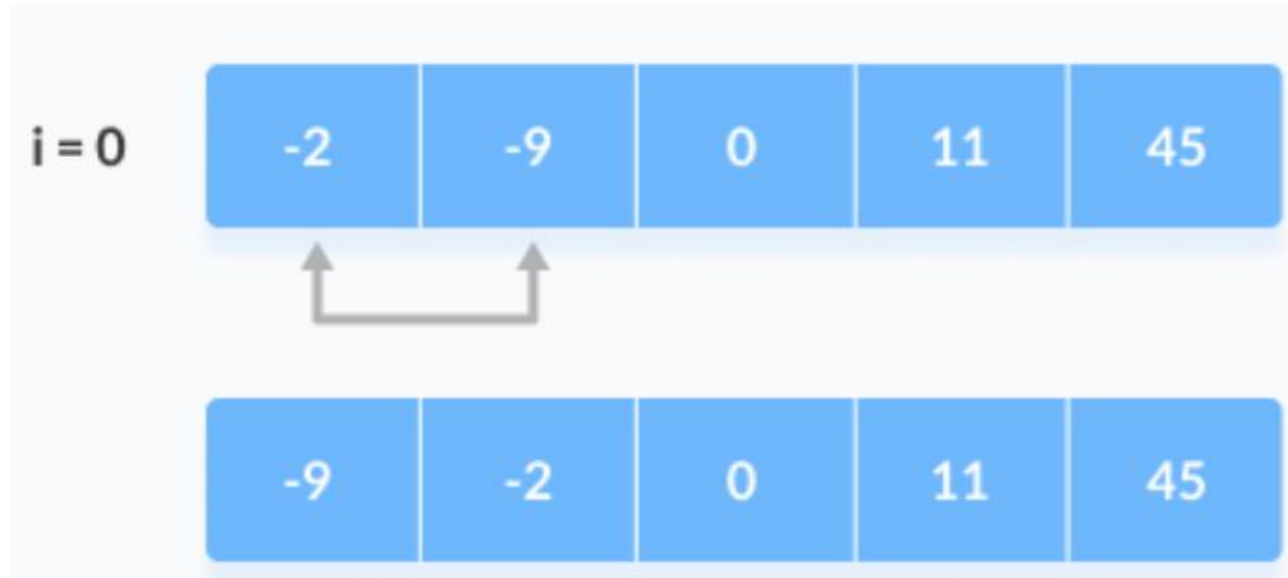
i = 0



i = 1



Bubble Sort - Step 3



Time Complexities

- **Worst Case Complexity: $O(n^2)$**

If we want to sort in ascending order and the array is in descending order then the worst case occurs.

- **Best Case Complexity: $O(n)$**

If the array is already sorted, then there is no need for sorting.

- **Average Case Complexity: $O(n^2)$**

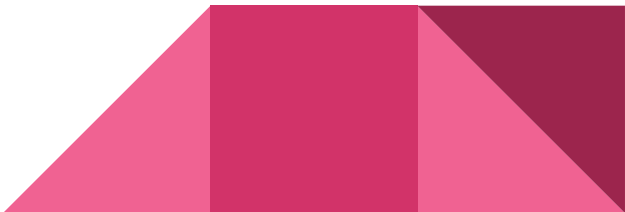
It occurs when the elements of the array are in jumbled order (neither ascending nor descending).



Bubble Sort Advantages

- Bubble sort is easy to understand and implement.
- It's an adaptive sorting algorithm. The order of elements affects the time complexity of the sorting (sorted array $O(n)$).

Bubble Sort Disadvantages

- Time complexity of $O(n^2)$ which makes it very slow for large data sets.
 - It is not efficient for large data sets, because it requires multiple passes through the data.
- 

Bubble Sort Applications

- It is often used to introduce the concept of a sorting algorithm because it is very simple.
- This algorithm is not efficient for real life applications unless:
 - Complexity does not matter
 - Short and simple code is preferred



How does Insertion Sort work?

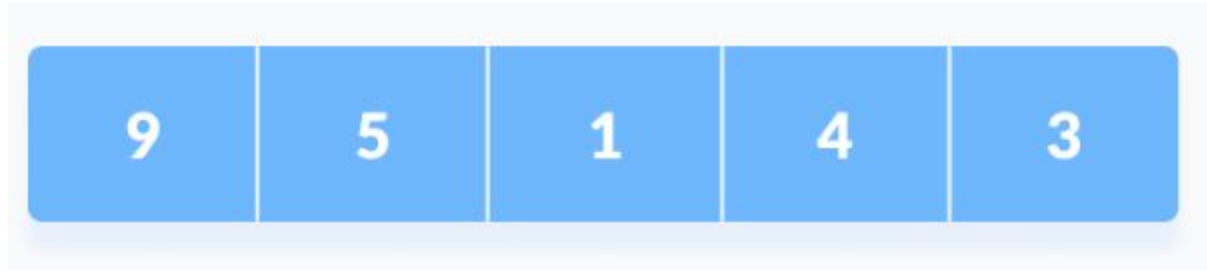
Insertion sort places an unsorted element at its suitable place in each iteration.

This algorithm assumes that the element at position 0 is sorted. This element becomes part of the logical sorted partition.

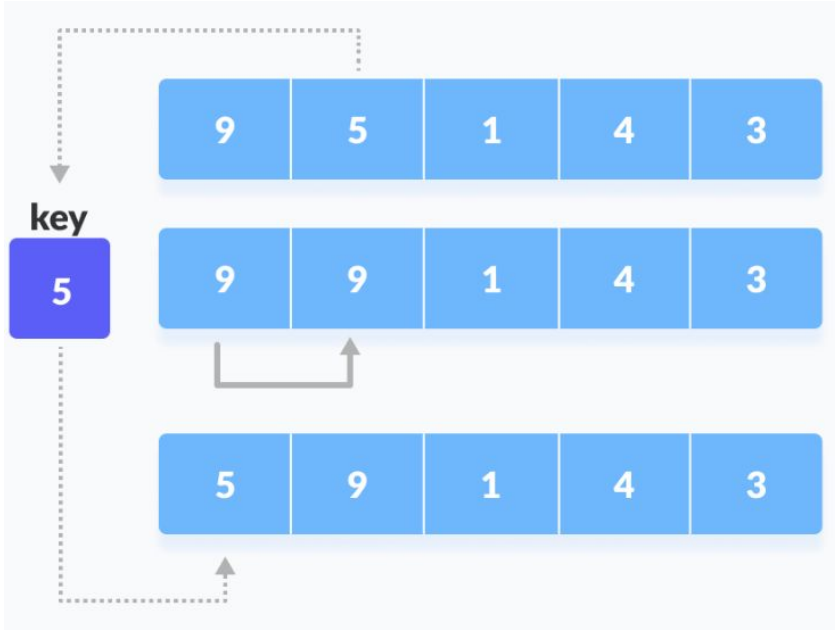
Then, the other elements are compared to their previous neighbor in the logical sorted partition. If the current element is smaller than its previous neighbor, the algorithm should compare it to the elements before and swap elements when the desired position for the current element has been found.



Insertion Sort - Initial Array



Insertion Sort - Step 0

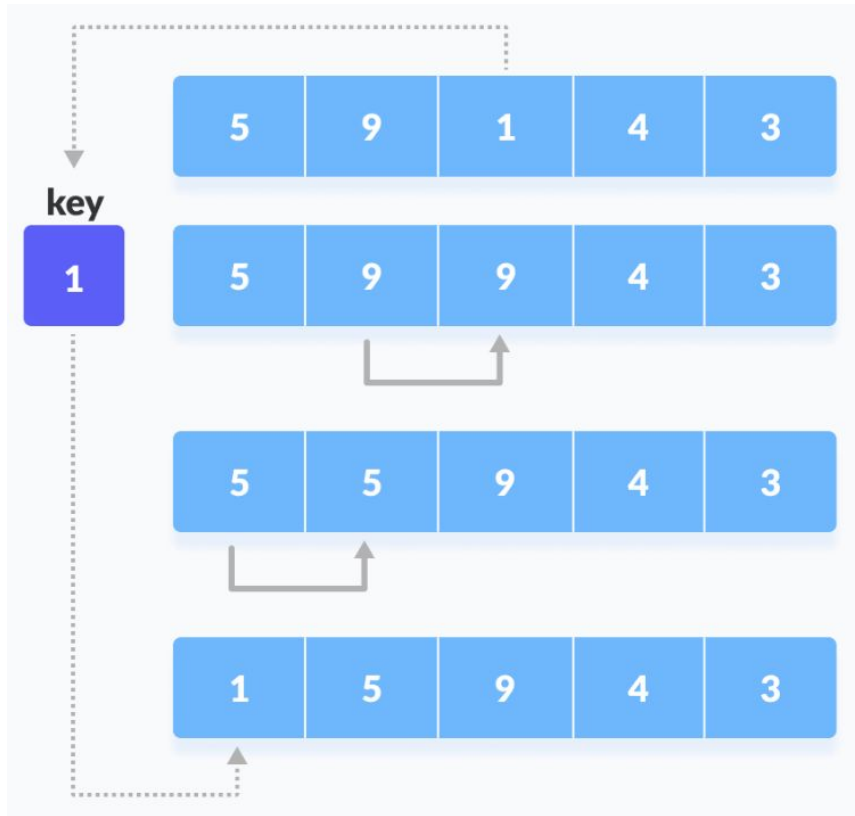


This algorithm assumes that the element at position 0 is sorted. This element becomes part of the logical sorted partition.

The element at position 1 is stored in an extra variable **key**.

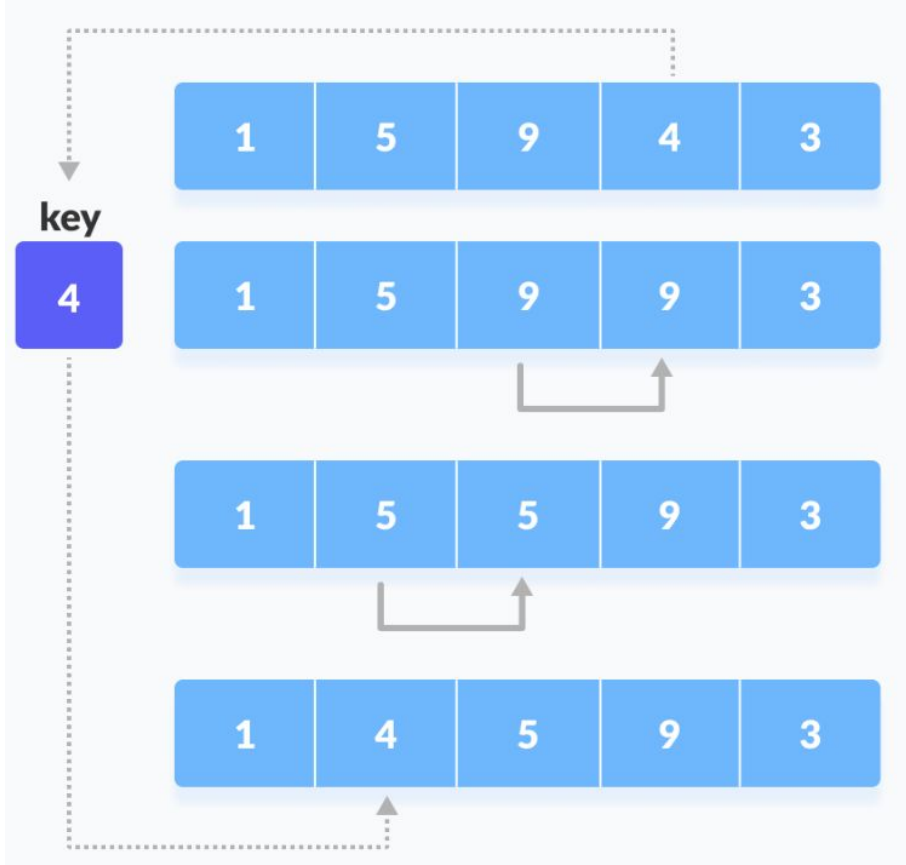
Compare **key** with the first element (sorted partition). If the first element is greater than **key**, then **key** is placed in front of the first element.

Insertion Sort - Step 1

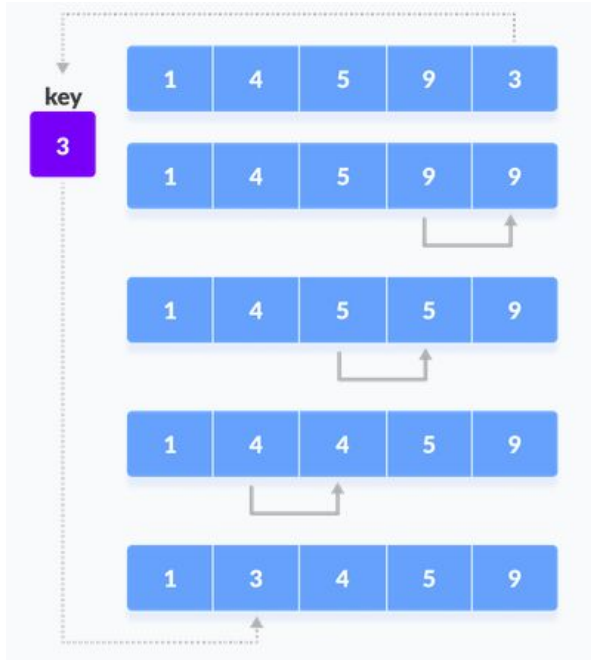


Now, let's take the third element and compare it with the elements in the logical sorted partition (elements on the left). Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

Insertion Sort - Step 2



Insertion Sort - Step 3



Time Complexities

- **Worst Case Complexity: $O(n^2)$**

If an array is in ascending order, and you want to sort it in descending order. Each element has to be compared with each of the other elements so, for every n^{th} element, **$(n-1)$** number of comparisons are made.

Total number of comparisons = $n*(n-1) \sim n^2$

Having the same Big-O as the Bubble sort doesn't mean identical performance.

- **Best Case Complexity: $O(n)$**

When the array is already sorted, the outer loop runs for n number of times whereas the inner loop does not run at all. So, there are only n number of comparisons. Thus, complexity is linear.

- **Average Case Complexity: $O(n^2)$**

When the elements of an array are in jumbled order (neither ascending nor descending).



Insertion Sort Applications

This algorithm is used if:

- The array has a small number of elements
- There are only a few elements left to be sorted



Insertion Sort Advantages

- It can start the sorting process even if the complete data set isn't available.
- Insertion sort makes fewer comparisons compared to the Bubble Sort.

Insertion Sort Disadvantages

- Quadratic worst-case time complexity.
- Work slowly on large datasets.



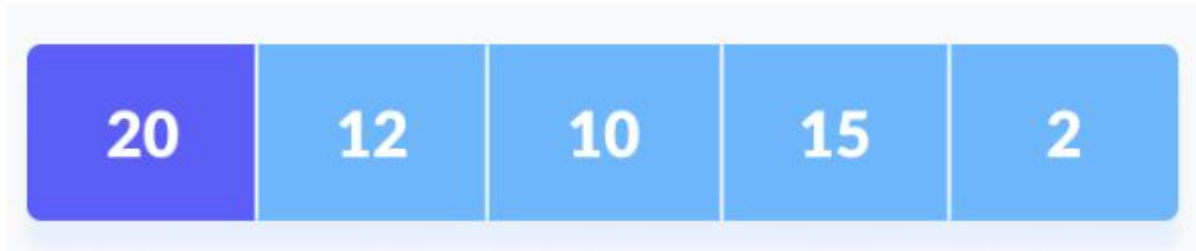
Selection Sort

This algorithm selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.



Selection Sort - Initial Array

Set the first element as **minimum**:



Selection Sort - Comparing



Compare **minimum** with the second element. If the second element is smaller than **minimum**, assign the second element as **minimum**.

Compare **minimum** with the third element. Again, if the third element is smaller, then assign **minimum** to the third element otherwise do nothing.

Repeat the process until the last element.

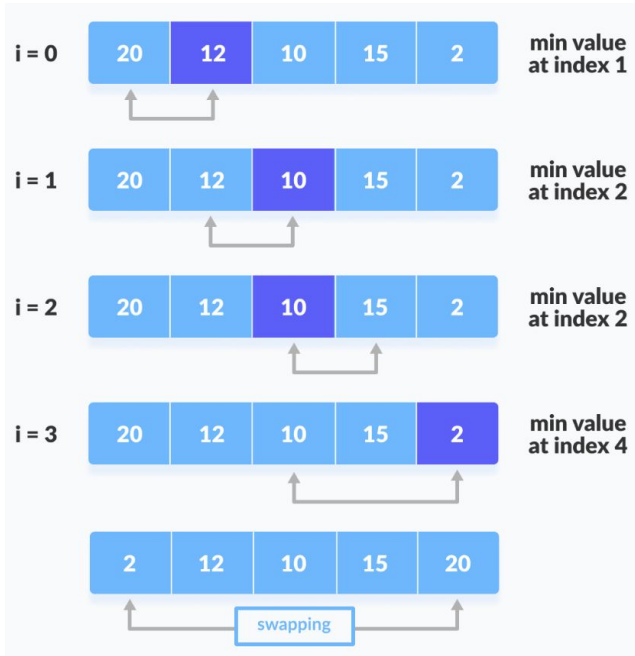


Selection Sort - Swapping

After each iteration, **minimum** is placed in the front of the unsorted list which becomes the logical sorted partition.



Selection Sort - Step 0



For each iteration, indexing starts from the first unsorted element.

Repeat the comparing minimum element and swapping processes until all the elements are placed at their correct positions.

Selection Sort - Step 1

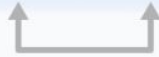


Selection Sort - Step 2

i = 0



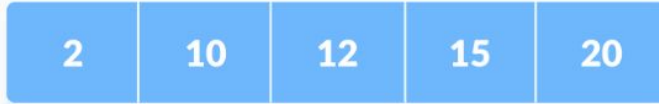
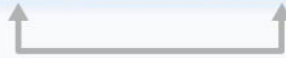
**min value
at index 2**



i = 2



**min value
at index 2**



already in place



Selection Sort - Step 3



Complexity

Number of comparisons: $(n - 1) + (n - 2) + (n - 3) + \dots + 1 = n(n - 1) / 2$ nearly equals to n^2 .

Complexity = $O(n^2)$

Also, we can analyze the complexity by simply observing the number of loops. There are 2 loops so the complexity is $n * n = n^2$.



Time Complexities

- **Worst Case Complexity:** $O(n^2)$

If we want to sort in ascending order and the array is in descending order then, the worst case occurs.

- **Best Case Complexity:** $O(n^2)$

It occurs when the array is already sorted

- **Average Case Complexity:** $O(n^2)$

It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

Space Complexity

Space complexity is $O(1)$ because an extra variable is used.

