**SORTING**

**What is Sorting?**

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

* [Selection Sort](https://www.geeksforgeeks.org/selection-sort/)
* [Bubble Sort](https://www.geeksforgeeks.org/bubble-sort/)
* [Insertion Sort](https://www.geeksforgeeks.org/insertion-sort/)
* [Merge Sort](https://www.geeksforgeeks.org/merge-sort/)
* [Quick Sort](https://www.geeksforgeeks.org/quick-sort/)
* [Heap Sort](https://www.geeksforgeeks.org/heap-sort/)
* [Counting Sort](https://www.geeksforgeeks.org/counting-sort/)
* [Radix Sort](https://www.geeksforgeeks.org/radix-sort/)
* [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/)
* [Bingo Sort Algorithm](https://www.geeksforgeeks.org/bingo-sort-algorithm/)
* [ShellSort](https://www.geeksforgeeks.org/shellsort/)
* [TimSort](https://www.geeksforgeeks.org/timsort/)
* [Comb Sort](https://www.geeksforgeeks.org/comb-sort/)
* [Pigeonhole Sort](https://www.geeksforgeeks.org/pigeonhole-sort/)
* [Cycle Sort](https://www.geeksforgeeks.org/cycle-sort/)
* [Cocktail Sort](https://www.geeksforgeeks.org/cocktail-sort/)
* [Strand Sort](https://www.geeksforgeeks.org/strand-sort/)
* [Bitonic Sort](https://www.geeksforgeeks.org/bitonic-sort/)
* [Pancake sorting](https://www.geeksforgeeks.org/pancake-sorting/)
* [BogoSort or Permutation Sort](https://www.geeksforgeeks.org/bogosort-permutation-sort/)
* [Gnome Sort](https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/)
* [Sleep Sort – The King of Laziness](https://www.geeksforgeeks.org/sleep-sort-king-laziness-sorting-sleeping/)
* [Structure Sorting in C++](https://www.geeksforgeeks.org/structure-sorting-in-c/)
* [Stooge Sort](https://www.geeksforgeeks.org/stooge-sort/)
* [Tag Sort (To get both sorted and original)](https://www.geeksforgeeks.org/tag-sort/)
* [Tree Sort](https://www.geeksforgeeks.org/tree-sort/)
* [Odd-Even Sort / Brick Sort](https://www.geeksforgeeks.org/odd-even-sort-brick-sort/)
* [3-way Merge Sort](https://www.geeksforgeeks.org/3-way-merge-sort/)

**Bubble Sort**

* **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.

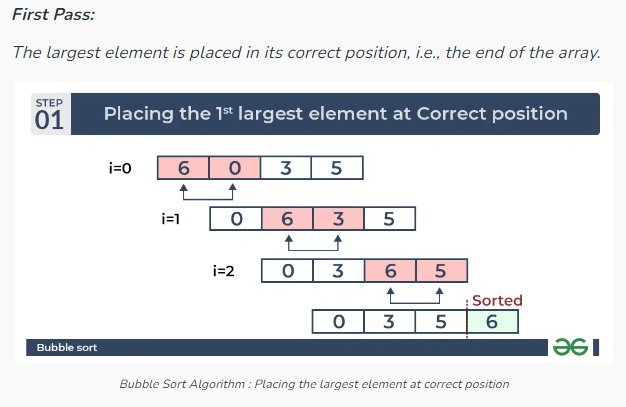
**Bubble Sort Algorithm**

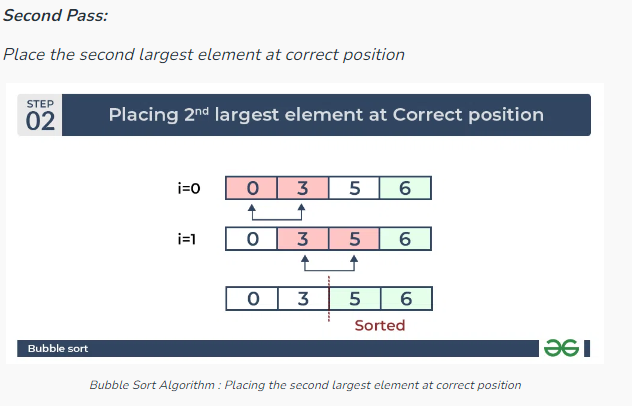
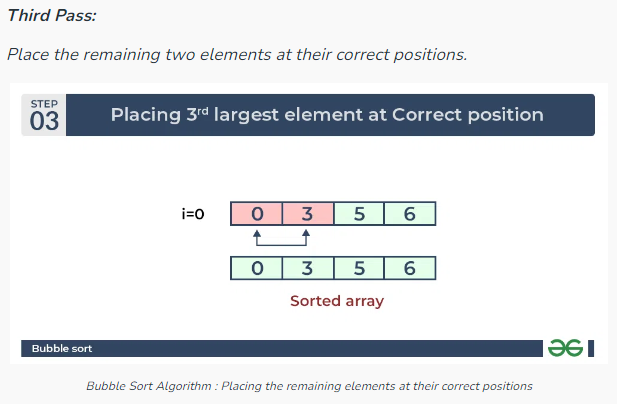
* traverse from left and compare adjacent elements and the higher one is placed at right side.
* In this way, the largest element is moved to the rightmost end at first.
* This process is then continued to find the second largest and place it and so on until the data is sorted.

## How does Bubble Sort Work?

Let us understand the working of bubble sort with the help of the following illustration:

**Input:** arr[] = {6, 3, 0, 5}





[Complexity Analysis of Bubble Sort](https://www.geeksforgeeks.org/time-and-space-complexity-analysis-of-bubble-sort/)**:**

**Time Complexity:**O(N2)  
**Auxiliary Space:** O(1)

**Advantages of Bubble Sort:**

* Bubble sort is easy to understand and implement.
* It does not require any additional memory space.
* It is a stable sorting algorithm, meaning that elements with the same key value maintain their relative order in the sorted output.

**Disadvantages of Bubble Sort:**

* Bubble sort has a time complexity of O(N2) which makes it very slow for large data sets.
* Bubble sort is a comparison-based sorting algorithm, which means that it requires a comparison operator to determine the relative order of elements in the input data set. It can limit the efficiency of the algorithm in certain cases.

|  |
| --- |
| **package** com.org.sorting;  **public** **class** BubbleSortExample\_1 {  **public** **static** **void** bubbleSort(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < n - 1; j++) {  **if** (arr[j] > arr[j + 1]) {  **int** temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  }  }  **public** **static** **void** printArray(**int** arr[], **int** size) {  **for** (**int** i = 0; i < size; i++) {  System.***out***.print(arr[i] + " ");  }  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  **int** n = arr.length;  *bubbleSort*(arr, n);  System.***out***.println("Sorted array: ");  *printArray*(arr, n);  }  } |

|  |
| --- |
| **package** com.org.sorting;  **public** **class** BubbleSortExample\_2 {  **public** **static** **void** bubbleSort(**int** arr[], **int** n) {  **boolean** swapped;  **for** (**int** i = 0; i < n; i++) {  swapped = **false**;  **for** (**int** j = 0; j < n - 1; j++) {  **if** (arr[j] > arr[j + 1]) {  **int** temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  swapped = **true**;  }  }  **if** (swapped == **false**) {  **break**;  }  }  }  **public** **static** **void** printArray(**int** arr[], **int** size) {  **for** (**int** i = 0; i < size; i++) {  System.***out***.print(arr[i] + " ");  }  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  **int** n = arr.length;  *bubbleSort*(arr, n);  System.***out***.println("Sorted array: ");  *printArray*(arr, n);  }  } |

|  |
| --- |
| **package** com.org.sorting;  **public** **class** BubbleSortExample\_3 {  **public** **static** **void** bubbleSort(**int** arr[], **int** n) {  **for** (**int** i = 0; i < n; i++) {  **for** (**int** j = 0; j < n - i - 1; j++) {  **if** (arr[i] < arr[j]) {  **int** temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  }  }  **public** **static** **void** printArray(**int** arr[], **int** size) {  **for** (**int** i = 0; i < size; i++) {  System.***out***.print(arr[i] + " ");  }  System.***out***.println();  }  **public** **static** **void** main(String args[]) {  **int** arr[] = { 64, 34, 25, 12, 22, 11, 90 };  **int** n = arr.length;  *bubbleSort*(arr, n);  System.***out***.println("Sorted array: ");  *printArray*(arr, n);  }  } |

