Bubble Sort in Java

Bubble Sort is the elementary sorting algorithm for sorting various data structures.

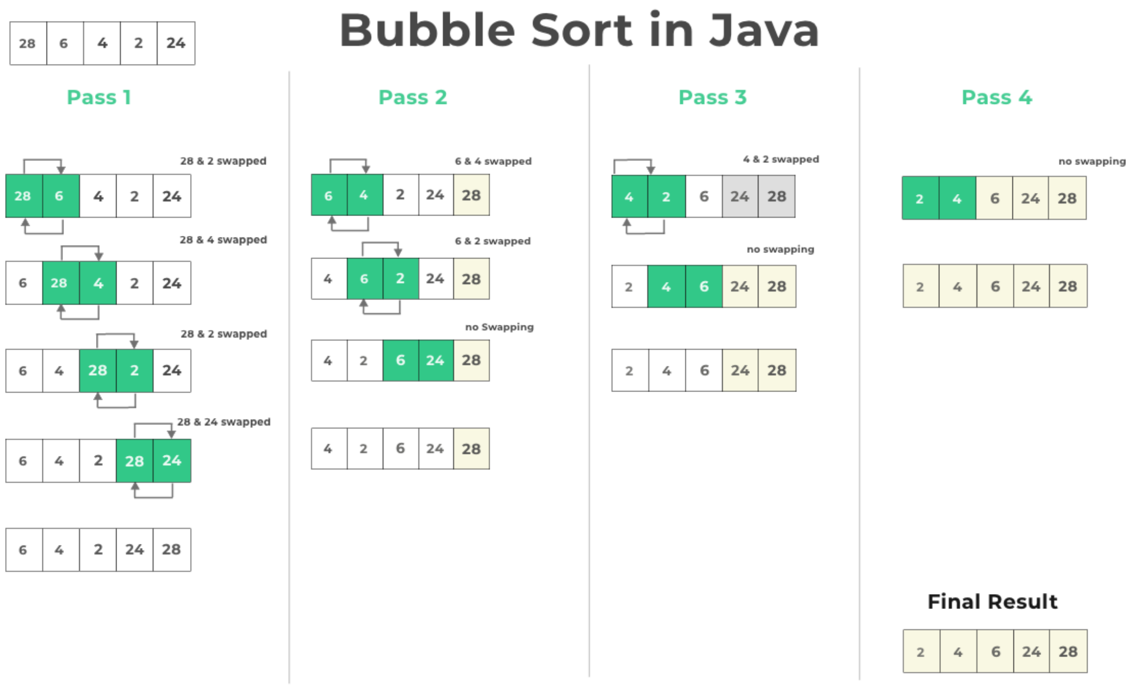
It is a comparison-based sorting algorithm in which each element is compared with the next element, and is swapped if those elements are not in the correct order.

- The algorithm starts with comparing the first pair of elements.
- If the first element is smaller than the second element, then the elements are swapped
- This algorithm is not suitable for large data sets

Time Complexity	O(n2)
Best Case	O(n)
Worst Case	O(n2)
Space Complexity	O(1)
Avg. Comparisons	n(n-1)/2

Implementation of bubble sort in Java

- First, we will take starting two elements from the list.
- Then we will compare those elements.
- If these elements are found in unsorted order we will sort them.
- Else we will compare next to elements.
- We will repeat the previous steps until we get our sorted array.



Pass 1

- (**28 6** 4 2 24) -> (**6 28** 4 2 24) : Swapped 28 & 6 since 28 > 6
- (6 **28 4** 2 24) -> (6 **4 28** 2 24) : Swapped 28 & 4 since 28 > 4
- (6 4 **28 2** 24) -> (6 4 **2 28** 24) : Swapped 28 & 2 since 28 > 2
- (6 4 2 **28 24**) -> (6 4 2 **24 28**) : Swapped 28 & 24 since 28 > 24

Algorithm for Bubble Sort in Java

- **Step1:** Repeat step 1 to 4 for i=0 to n
- **Step2:** For j=0 to n
- **Step3:** if(arr[j]>arr[j+1])
- **Step4:** swap(arr[j],arr[j+1])
- **Step5:** End

```
package Sort;

public class Bubble1 {
  static void bubbleSort(int a[]) {
    int len = a.length; // calculating the length of array
    for (int i = 0; i < len - 1; i++)
      for (int j = 0; j < len - i - 1; j++)
        if (a[j] > a[j + 1]) // comparing the pair of elements
        {
          // swapping a[j+1] and a[i]
          int temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
        }
  }

  /* Prints the array */
  static void printArray(int a[]) {
    int len = a.length;
    for (int i = 0; i < len; i++)
      System.out.print(a[i] + " "); // printing the sorted array

    System.out.println();
  }

  // Main method to test above
  public static void main(String args[]) {
    int arr[] = { 28, 6, 4, 2, 24 };

    bubbleSort(arr);// calling the bubbleSort function

    System.out.println("Sorted array");

    printArray(arr); // calling the printArray function
  }
}
```

The issue with this algorithm was that it will always run all passes and iterations. Even if the arrays are already sorted, nearly sorted or not sorted.

We need to optimize the this algorithm so that if the array becomes sorted at some time. We do not implement the algorithm further and print the array right then and ther.

💡

Arr [] = {1, 2, 3, 4, 5, 6, 7}
As you can see the array is sorted already, but our algorithm will run all iterations and passes anyways

💡 Arr [] = {1, 2, 3, 4, 5, 10, 7}
The above array is nearly sorted, just 10 out of place. However, in pass 1 itself the array will get sorted but previous algorithm will still run all the passes

💡 **How it is solved?**
We can solve the above issue by bringing another boolean variable called swapped.
If in any pass no swapping happens, it means the array has become sorted and we do not need to run any further passes.

```
package Sort;

public class Bubble2 {
  static void bubbleSort(int a[]) {
    int len = a.length; // calculating the length of array
    for (int i = 0; i < len - 1; i++) {
      // for optimization when array is already sorted & no swapping happens
      boolean swapped = false;
      for (int j = 0; j < len - i - 1; j++) {
        if (a[j] > a[j + 1]) // comparing the pair of elements
        {
          // swapping a[j+1] and a[i]
          int temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
          // swapping happened so change to true
          swapped = true;
        }
      }

      // if no swaps happened in any of the iteration
      // array has become sorted so stop with the passes
      if (swapped == false)
        System.out.println("Array is Sorted:");
        break;

    }
  }

  /* Prints the array */
  static void printArray(int a[]) {
    int len = a.length;
```

```
      for (int i = 0; i < len; i++)
        System.out.print(a[i] + " "); // printing the sorted array

      System.out.println();
    }

    // Main method to test above
    public static void main(String args[]) {
      int arr[] = { 1, 2, 3, 4, 5, 6, 7 };

      bubbleSort(arr);// calling the bubbleSort function

      System.out.println("Sorted array");

      printArray(arr); // calling the printArray function
    }
  }
```

Bubble sort would need lesser memory then other sorting techniques.

Its, time complexity is $O(n^2)$,

When you have very large items then, it becomes even more slow thanks to $O(n^2)$

The algorithm will be slowest when the array is sorted by in reverse

Best Case : O(n), when its already sorted