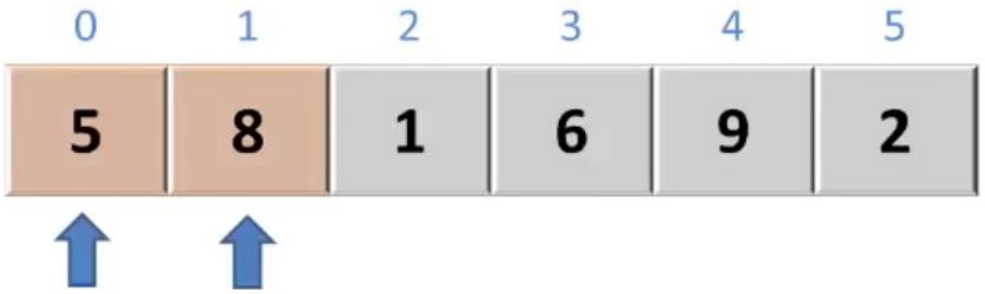
Bubble Sort

Given an array of items, sort them in increasing order

Large items bubbles at right

Bubble Sort

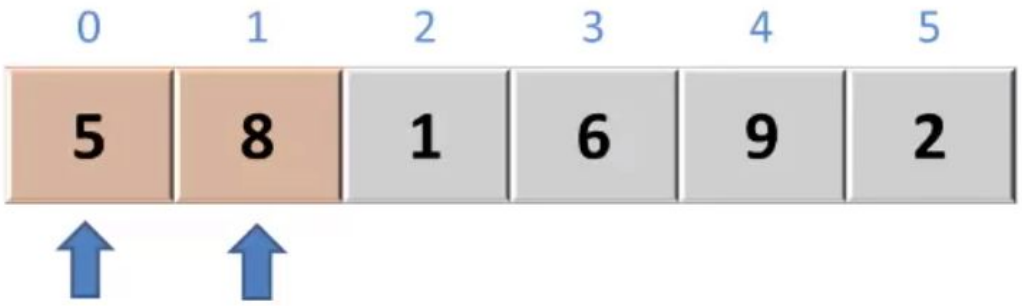


Repeatedly compares adjacent items.

Larger items “bubble” right.

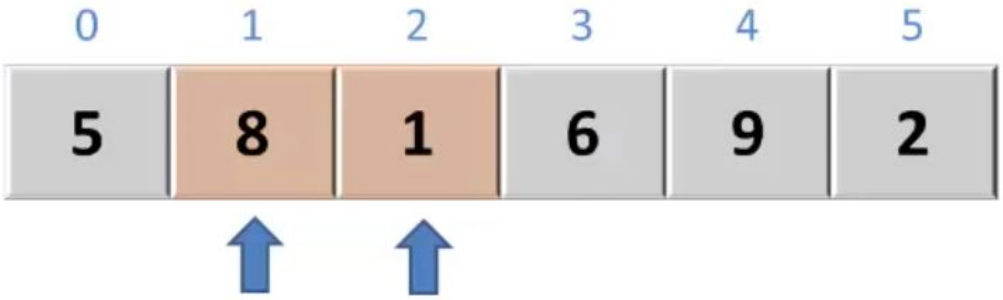
1st Iteration

Bubble Sort



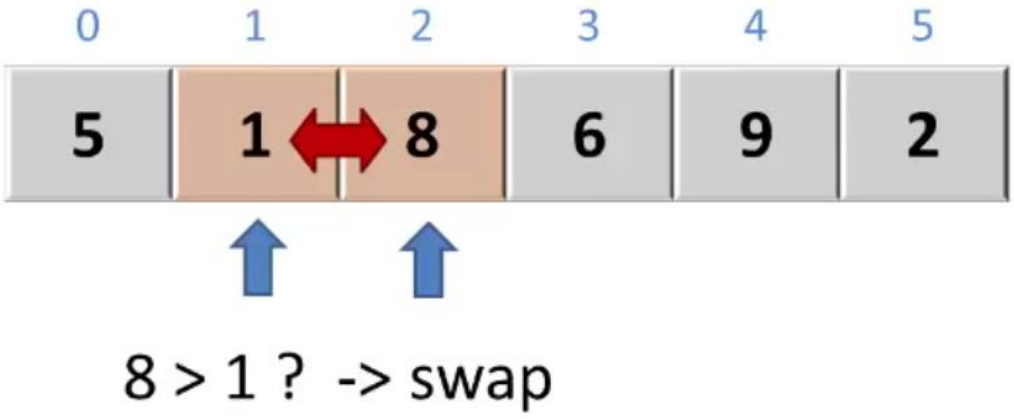
5 > 8 ? -> swap

Bubble Sort



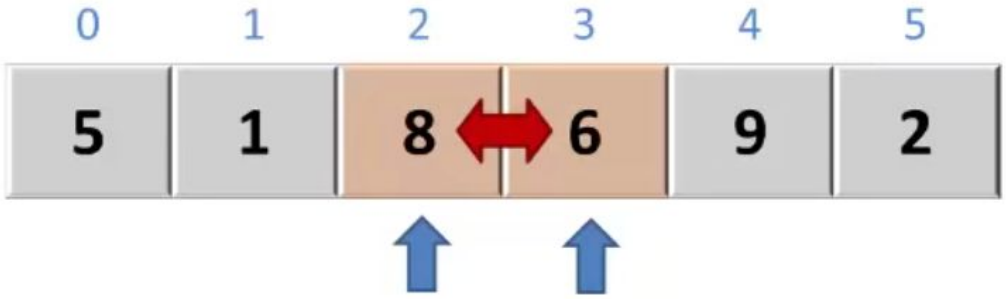
8 > 1 ? -> swap

Bubble Sort



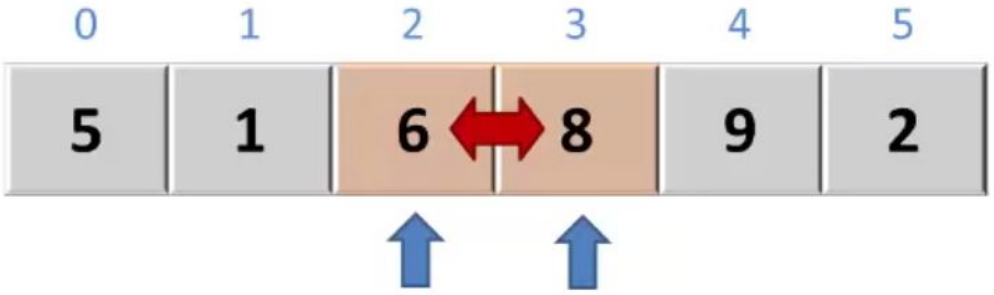
8 > 1 ? -> swap

Bubble Sort



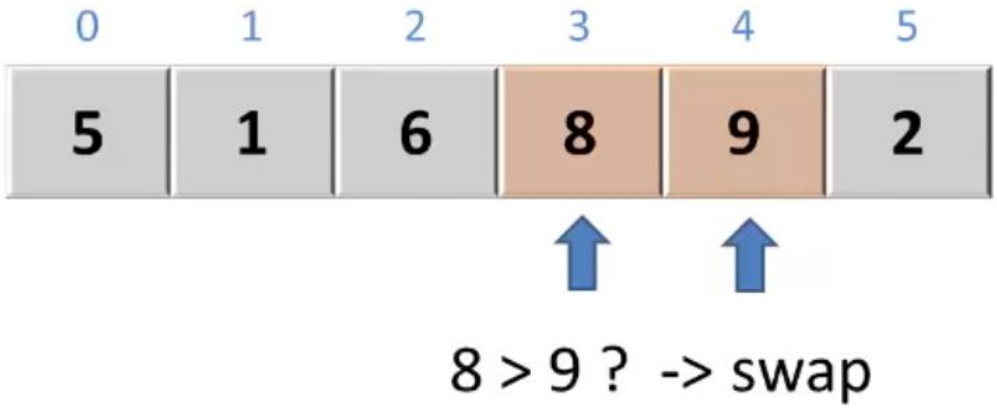
8 > 6 ? -> swap

Bubble Sort

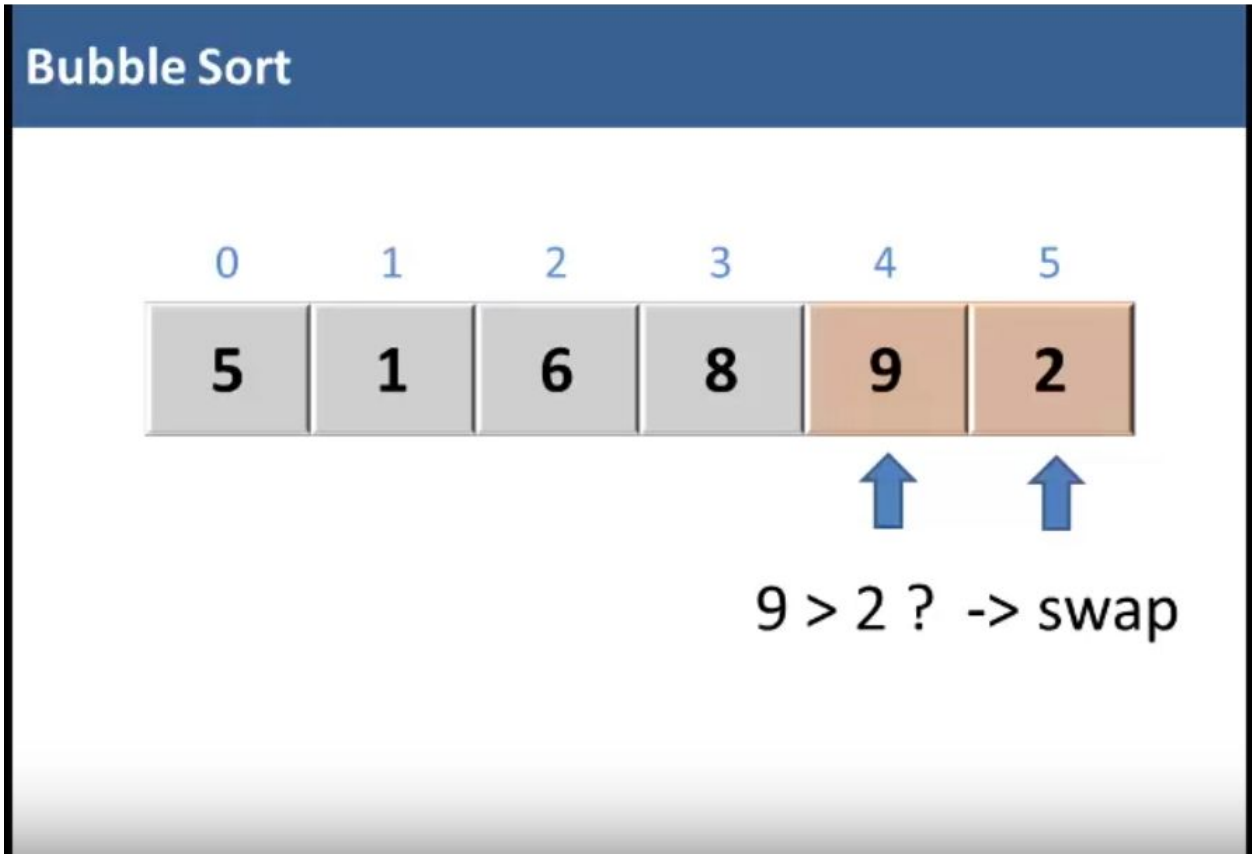


8 > 6 ? -> swap

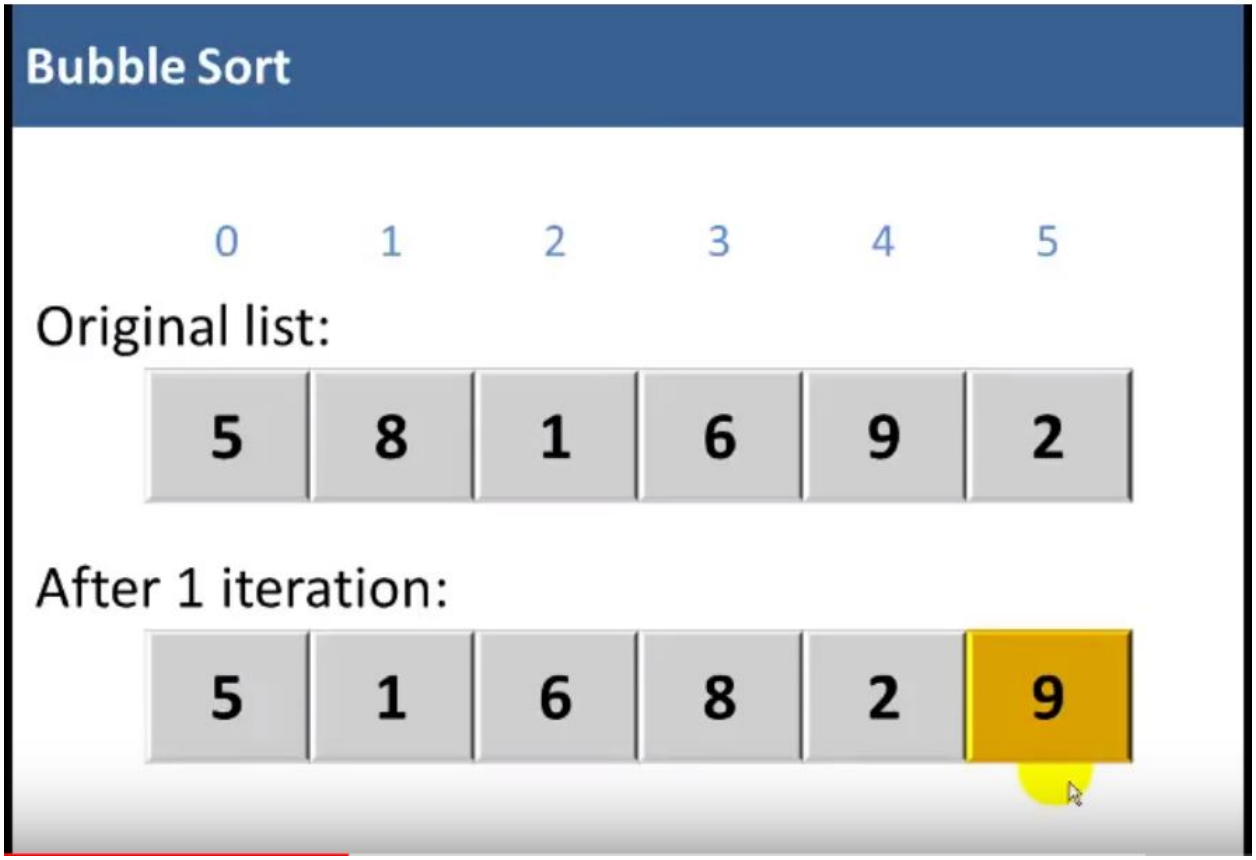
Bubble Sort



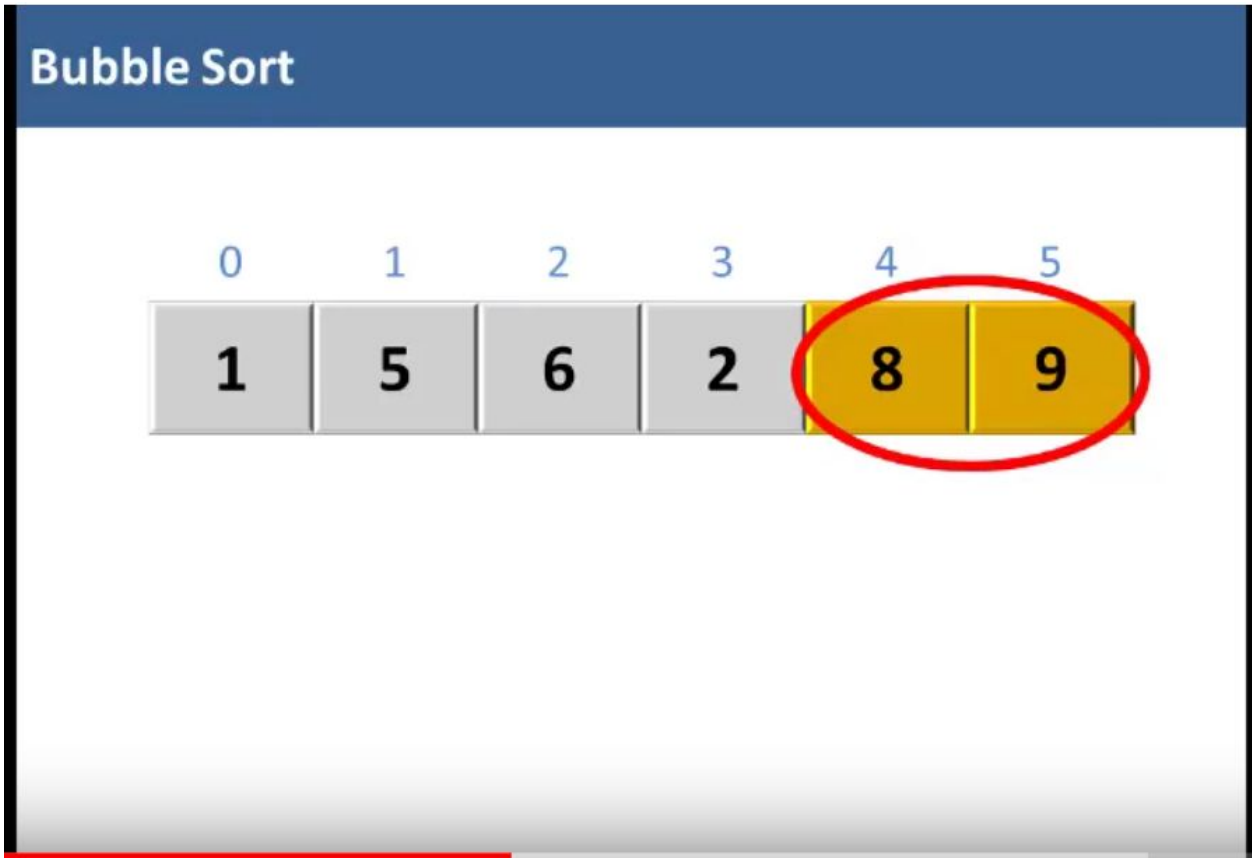
8 > 9 ? -> swap



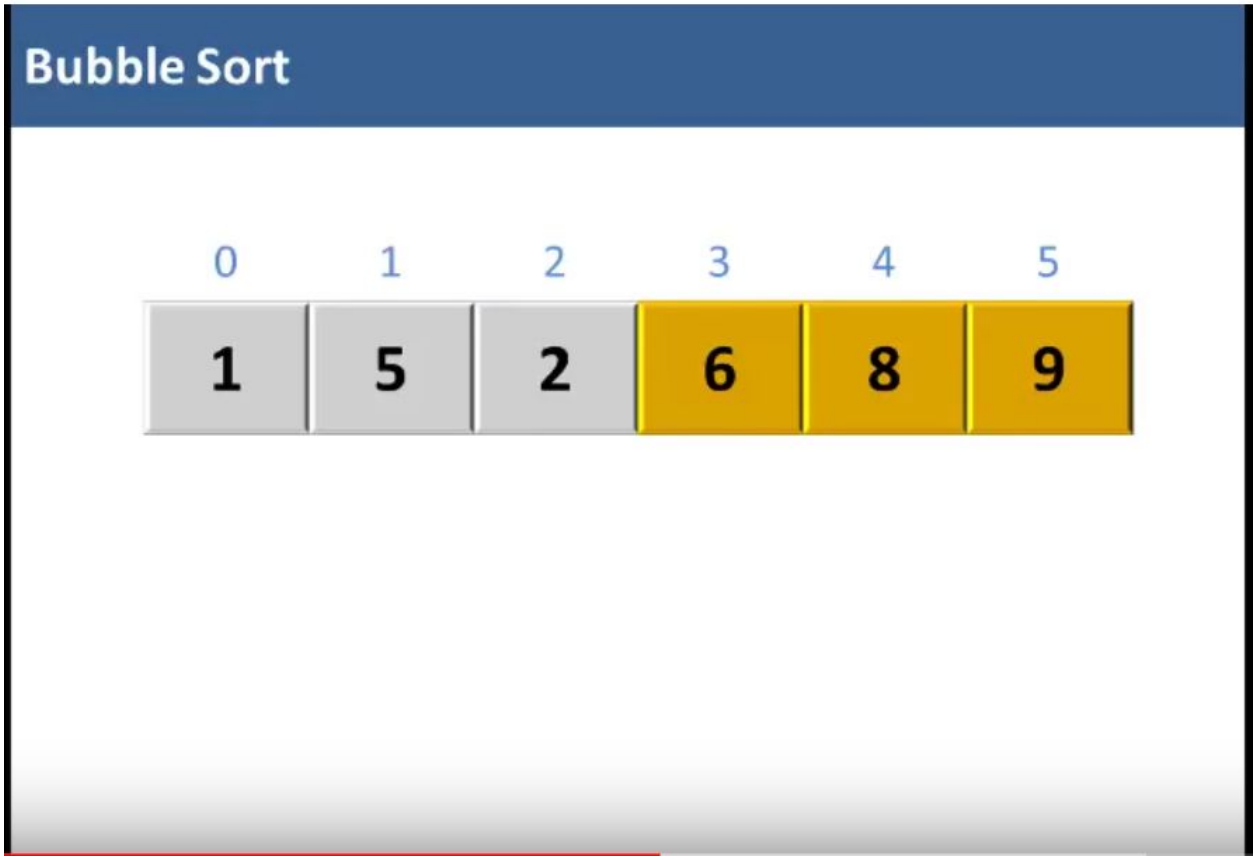
After 1st iteration, first item bubbles to right



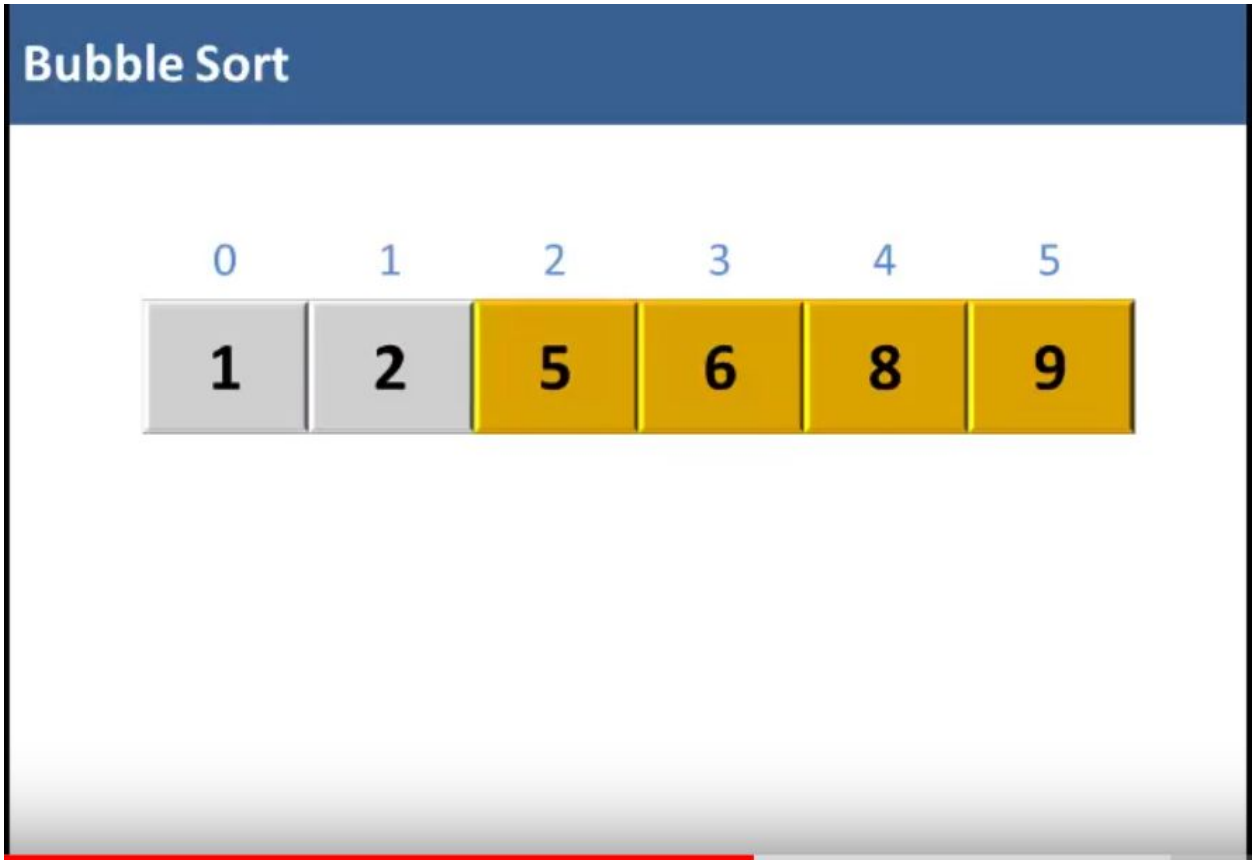
After 2nd iteration



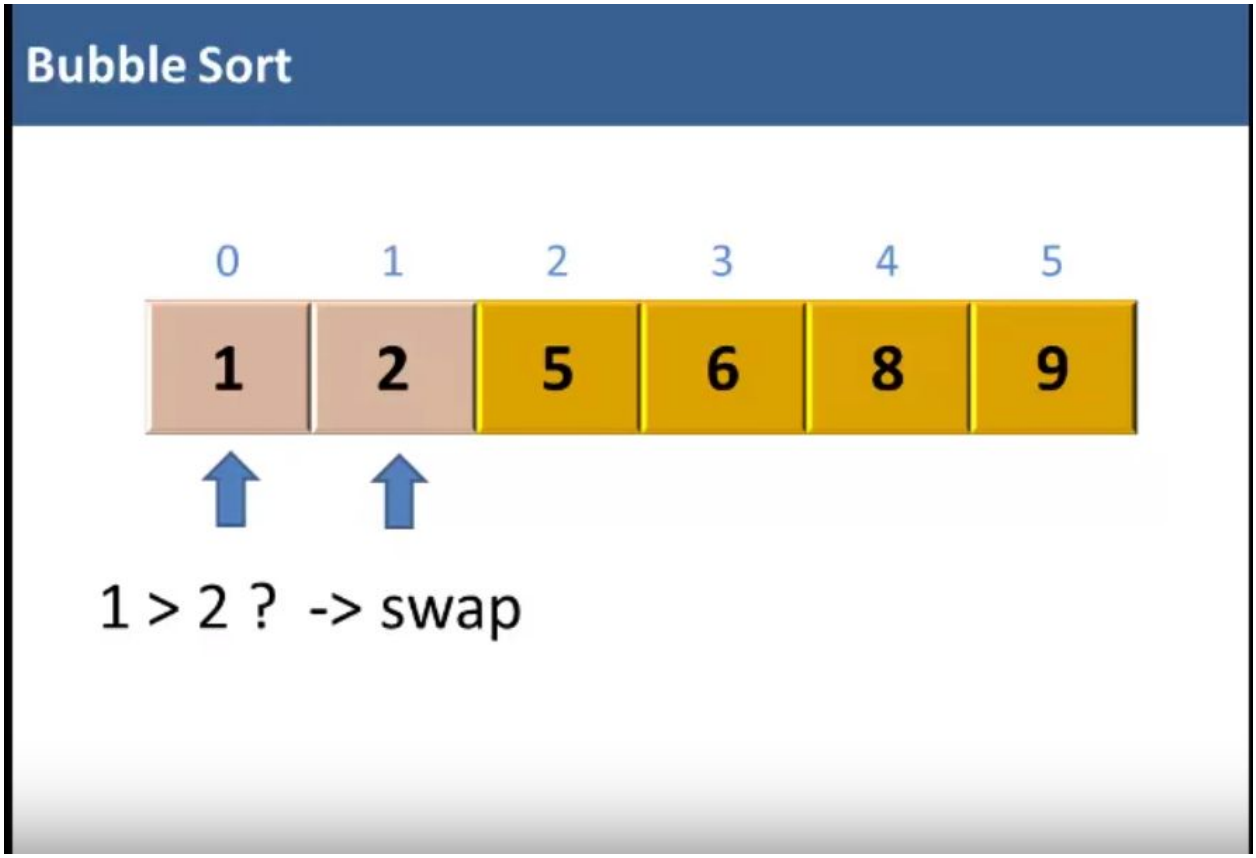
After 3rd iteration



After 4th iteration



After 5th iteration



For 6 elements, it took 5 iterations.

Code:

Bubble Sort – Java Code

```
public int[] bubbleSort (int[] list) {
    int i, j, temp = 0;
    for (i = 0; i < list.length - 1; i++) {
        for (j = 0; j < list.length - 1 - i; j++) {
            if (list[j] > list[j + 1]) {
                temp = list[j];
                list[j] = list[j + 1];
                list[j + 1] = temp;
            }
        }
    }
    return list;
```

Big o analysis:
In Bubble Sort, `n-1` comparisons will be done in the 1st pass, `n-2` in 2nd pass, `n-3` in 3rd pass and so on. So the total number of comparisons will be,

(n-1) + (n-2) + …. + 3 + 2 + 1
Sum = n(n-1)/2
I.e O(n2)

The **space complexity** for Bubble Sort is **O(1)**, because only a single additional memory space is required

Also, the **best case time complexity** will be **O(n)**, it is when the list is already sorted.

Following are the Time and Space complexity for the Bubble Sort algorithm.

- Worst Case Time Complexity [Big-O]: **O(n^2)**
- Best Case Time Complexity [Big-omega]: **O(n)**
- Average Time Complexity [Big-theta]: **O(n^2)**

- Space Complexity: **O(1)**