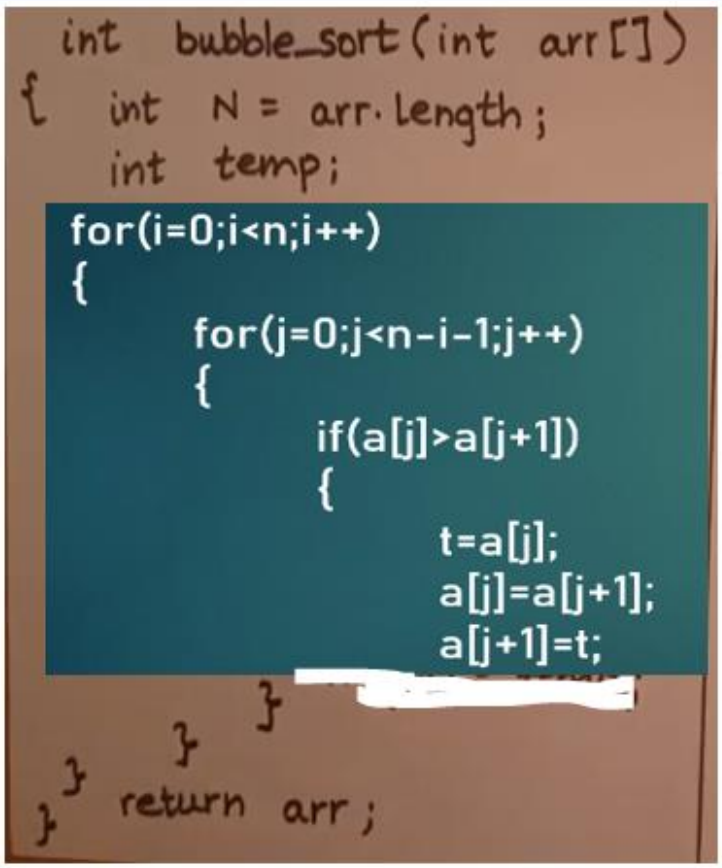
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

Bubble Sort Complexity

Time Complexity	
Best	O(n)
Worst	$O(n^2)$
Average	$O(n^2)$
Space Complexity	O(1)
Stability	Yes



Pseudocode

```
def bubbleSort(nums):
    n = len(nums)
    for i in range(n)
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                swap(arr[j] & arr[j+1])
```

ABI

Input: *arr[] = {6, 3, 0, 5}*

First Pass:

- *Bubble sort starts with very first two elements, comparing them to check which one is greater.*
  - *(6 3 0 5) –> (**3 6** 0 5), Here, algorithm compares the first two elements, and swaps since 6 > 3.*
  - *(3 **6 0** 5) –> (3 **0 6** 5), Swap since 6 > 0*
  - *(3 0 **6 5**) –> (3 0 **5 6**), Swap since 6 > 5*

ABDULLAH

Second Pass:

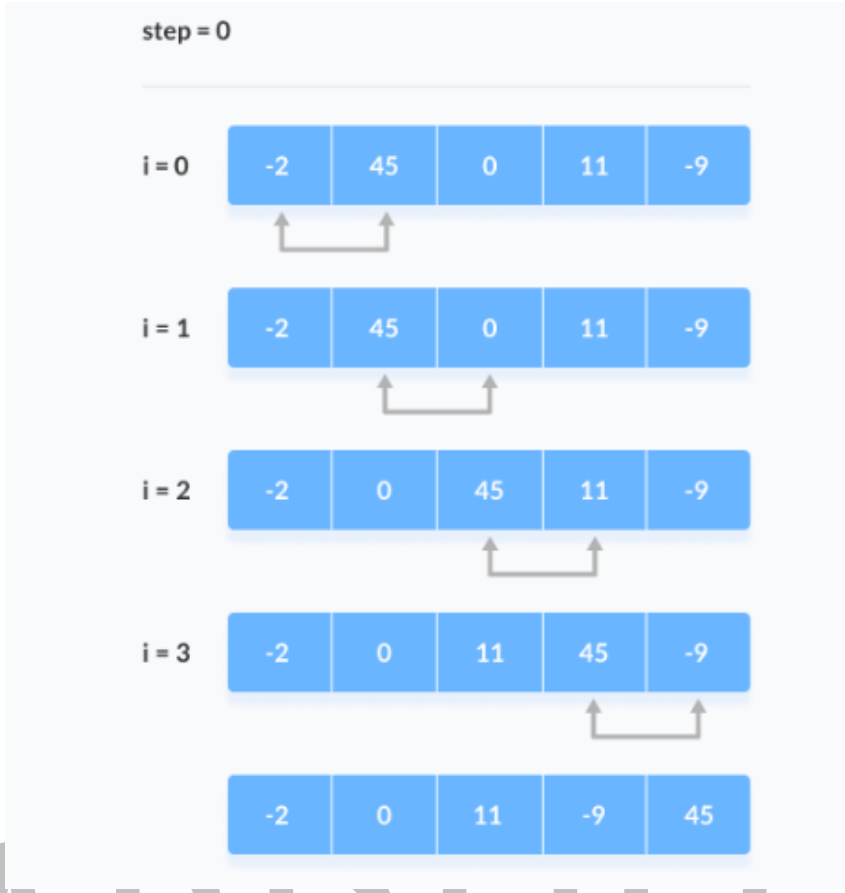
- *Now, during second iteration it should look like this:*
  - *(**3 0** 5 6) –> (**0 3** 5 6), Swap since 3 > 0*
  - *(0 **3 5** 6) –> (0 **3 5** 6), No change as 5 > 3*

Third Pass:

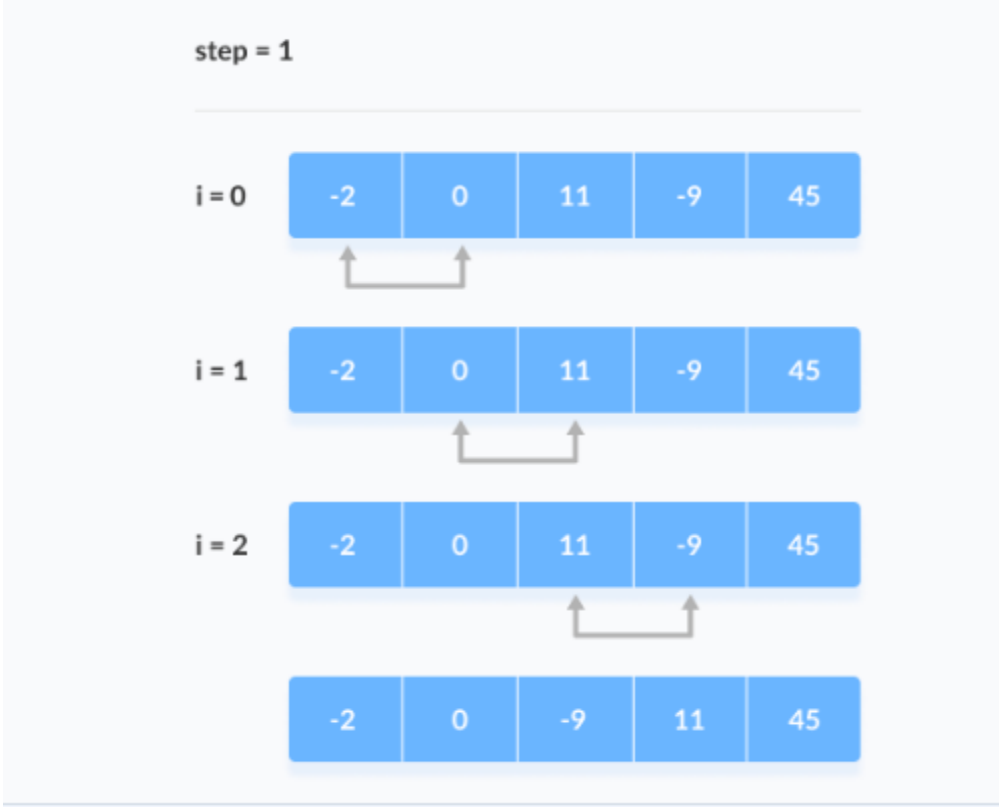
- *Now, the array is already sorted, but our algorithm does not know if it is completed.*
- *The algorithm needs one **whole** pass without **any** swap to know it is sorted.*
  - *(**0 3** 5 6) –> (**0 3** 5 6), No change as 3 > 0*

Array is now sorted and no more pass will happen.

ABDULLAH: 03343127215



ABDULLAH



ABDULLAH: 03343127215

ABDELAH

Advantages:

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

Disadvantages

- Bubble sort has a 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 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. While this is not necessarily a disadvantage, it can limit the efficiency of the algorithm in certain cases.