Bubble Sort Algorithm

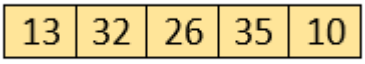
```
1. begin BubbleSort(arr)
2.    for all array elements
3.       if arr[i] > arr[i+1]
4.          swap(arr[i], arr[i+1])
5.       end if
6.    end for
7.    return arr
8. end BubbleSort
```

Working of Bubble sort Algorithm

Now, let's see the working of Bubble sort Algorithm.

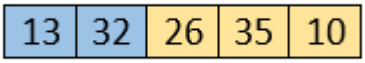
To understand the working of bubble sort algorithm, let's take an unsorted array. We are taking a short and accurate array, as we know the complexity of bubble sort is **$O(n^2)$.**

Let the elements of array are -

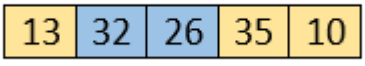


First Pass

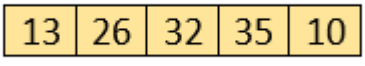
Sorting will start from the initial two elements. Let compare them to check which is greater.



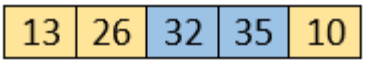
Here, 32 is greater than 13 (32 > 13), so it is already sorted. Now, compare 32 with 26.



Here, 26 is smaller than 36. So, swapping is required. After swapping new array will look like -

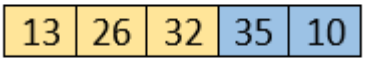


Now, compare 32 and 35.

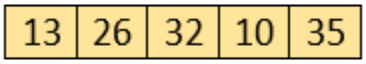


Here, 35 is greater than 32. So, there is no swapping required as they are already sorted.

Now, the comparison will be in between 35 and 10.



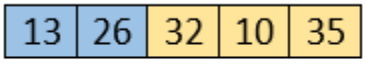
Here, 10 is smaller than 35 that are not sorted. So, swapping is required. Now, we reach at the end of the array. After first pass, the array will be -

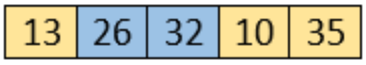


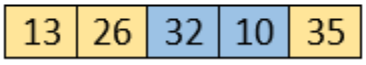
Now, move to the second iteration.

Second Pass

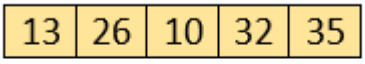
The same process will be followed for second iteration.

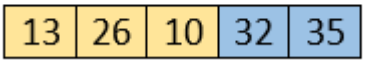






Here, 10 is smaller than 32. So, swapping is required. After swapping, the array will be -

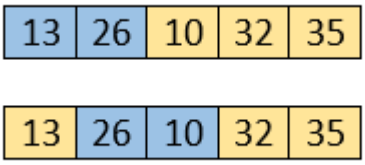




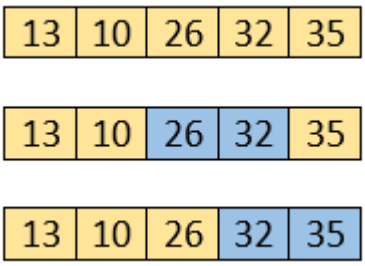
Now, move to the third iteration.

Third Pass

The same process will be followed for third iteration.



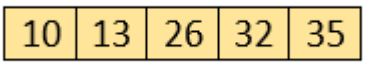
Here, 10 is smaller than 26. So, swapping is required. After swapping, the array will be -



Now, move to the fourth iteration.

Fourth pass

Similarly, after the fourth iteration, the array will be -



Hence, there is no swapping required, so the array is completely sorted.

```cpp
// Bubble sort in C++

#include <iostream>
using namespace std;

// perform bubble sort
void bubbleSort(int array[], int size) {

  // loop to access each array element
  for (int step = 0; step < size; ++step) {

    // loop to compare array elements
    for (int i = 0; i < size - step; ++i) {

      // compare two adjacent elements
      // change > to < to sort in descending order
      if (array[i] > array[i + 1]) {

        // swapping elements if elements
        // are not in the intended order
        int temp = array[i];
        array[i] = array[i + 1];
        array[i + 1] = temp;
      }
    }
  }
}

// print array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    cout << "  " << array[i];
  }
  cout << "\n";
}

int main() {
  int data[] = {-2, 45, 0, 11, -9};

  // find array's length
  int size = sizeof(data) / sizeof(data[0]);

  bubbleSort(data, size);

  cout << "Sorted Array in Ascending Order:\n";
```

```
  printArray(data, size);
}
```

Optimized Bubble Sort Algorithm

In the above algorithm, all the comparisons are made even if the array is already sorted.

This increases the execution time.

To solve this, we can introduce an extra variable `swapped`. The value of `swapped` is set true if there occurs swapping of elements. Otherwise, it is set **false**.

After an iteration, if there is no swapping, the value of `swapped` will be **false**. This means elements are already sorted and there is no need to perform further iterations.

This will reduce the execution time and helps to optimize the bubble sort.

Algorithm for optimized bubble sort is

```
bubbleSort(array)
  swapped <- false
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
      swapped <- true
end bubbleSort
```