# BUBBLE SORTING

Bubble sort is a type of sorting algorithm you can use to arrange a set of values in ascending order. If you want, you can also implement bubble sort to sort the values in descending order. A real-world example of a bubble sort algorithm is how the contact list on your phone is sorted in alphabetical order.

EXAMPLE:

**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
  + ( 30 **6** **5** ) –>  ( 30 **5 6**), Swap since 6 > 5

**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.

# MERGE SORTING

Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being Ο(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

EXAMPLE:

To understand merge sort, we take an unsorted array as the following −

Unsorted Array

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

Merge Sort Division

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

Merge Sort Division

We further divide these arrays and we achieve atomic value which can no more be divided.

Merge Sort Division

Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.

Merge Sort Combine

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Merge Sort Combine

After the final merging, the list should look like this −

Merge Sort

Now we should learn some programming aspects of merge sorting.