

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Following are some of the examples of sorting in real-life scenarios –

Telephone Directory – The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.

Dictionary – The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

In-place Sorting and Not-in-place Sorting

Sorting algorithms may require some extra space for comparison and temporary storage of few data elements. These algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself. This is called in-place sorting. Bubble sort is an example of in-place sorting.

However, in some sorting algorithms, the program requires space which is more than or equal to the elements being sorted. Sorting which uses equal or more space is called not-in-place sorting. Merge-sort is an example of not-in-place sorting.

Stable and Not Stable Sorting

If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called stable sorting.

Stable Sorting

If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they

appear, it is called unstable sorting.

Unstable Sorting

Stability of an algorithm matters when we wish to maintain the sequence of original elements, like in a tuple for example.

Adaptive and Non-Adaptive Sorting Algorithm

A sorting algorithm is said to be adaptive, if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.

A non-adaptive algorithm is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.

Important Terms

Some terms are generally coined while discussing sorting techniques, here is a brief introduction to them –

Increasing Order

A sequence of values is said to be in increasing order, if the successive element is greater than the previous one. For example, 1, 3, 4, 6, 8, 9 are in increasing order, as every next element is greater than the previous element.

Decreasing Order

A sequence of values is said to be in decreasing order, if the successive element is less than the current one. For example, 9, 8, 6, 4, 3, 1 are in decreasing order, as every next element is less than the previous element.

Non-Increasing Order

A sequence of values is said to be in non-increasing order, if the successive element is less than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 9, 8, 6, 3, 3, 1 are in non-increasing order, as every next element is less than or equal to (in case of 3) but not greater than any previous element.

Non-Decreasing Order

A sequence of values is said to be in non-decreasing order, if the successive element is greater than or equal to its previous element in the sequence. This order occurs when the sequence contains duplicate values. For example, 1, 3, 3, 6, 8, 9 are in non-decreasing order, as every next element is greater than or equal to (in case of 3) but not less than the previous one.

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where n is the number of items.

How Bubble Sort Works?

We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble Sort

Bubble sort starts with very first two elements, comparing them to check which one is greater.

Bubble Sort

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

Bubble Sort

We find that 27 is smaller than 33 and these two values must be swapped.

Bubble Sort

The new array should look like this –

Bubble Sort

Next we compare 33 and 35. We find that both are in already sorted positions.

Bubble Sort

Then we move to the next two values, 35 and 10.

Bubble Sort

We know then that 10 is smaller 35. Hence they are not sorted.

Bubble Sort

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –

Bubble Sort

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –

Bubble Sort

Notice that after each iteration, at least one value moves at the end.

Bubble Sort

And when there's no swap required, bubble sorts learns that an array is completely sorted.

Bubble Sort

Now we should look into some practical aspects of bubble sort.

Algorithm

We assume list is an array of n elements. We further assume that swap function swaps the values of the given array elements.

```
begin BubbleSort(list)
```

```
  for all elements of list
```

```
    if list[i] > list[i+1]
```

```
      swap(list[i], list[i+1])
```

```
    end if
```

```
  end for
```

```
  return list
```

```
end BubbleSort
```

Pseudocode

We observe in algorithm that Bubble Sort compares each pair of array element unless the whole array is completely sorted in an ascending order. This may cause a few complexity issues like what if the array needs no more swapping as all the elements are already ascending.

To ease-out the issue, we use one flag variable swapped which will help us see if any swap has happened or not. If no swap has occurred, i.e. the array requires no more processing to be sorted, it will come out of the loop.

Pseudocode of BubbleSort algorithm can be written as follows –

```
procedure bubbleSort( list : array of items )
```

```
    loop = list.count;
```

```
    for i = 0 to loop-1 do:
```

```
        swapped = false
```

```
        for j = 0 to loop-1 do:
```

```
            /* compare the adjacent elements */
```

```
            if list[j] > list[j+1] then
```

```
                /* swap them */
```

```
                swap( list[j], list[j+1] )
```

```
                swapped = true
```

```
            end if
```

```
        end for
```

```
        /*if no number was swapped that means
```

```
        array is sorted now, break the loop.*/
```

```
        if(not swapped) then
```

```
            break
```

```
        end if
```

end for

end procedure return list

Implementation

One more issue we did not address in our original algorithm and its improvised pseudocode, is that, after every iteration the highest values settles down at the end of the array. Hence, the next iteration need not include already sorted elements. For this purpose, in our implementation, we restrict the inner loop to avoid already sorted values.