Sorting Algorithm

# Sorting Algorithm

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions:

* The output is in nondecreasing order (each element is no smaller than the previous element according to the desired total order);
* The output is a permutation (reordering but with all of the original elements) of the input.

Examples:

* Telephone Directory
* Dictionary

# Classification of sorting algorithms

## Time complexity

Let x be the maximum number of comparisons in a sorting algorithm. The maximum height of the decision tree would be x. A tree with maximum height x has at most 2x leaves.

n!  <= 2x

Taking Log on both sides.

log2(n!) <= x

Since log2(n!) = Θ(nLogn), we can say

x = Ω(nLog2n)

Therefore, any comparison based sorting algorithm must make at least nlog2n comparisons to sort the input array, and Heapsort and merge sort are asymptotically optimal comparison sorts.

## Space complexity or Memory usage

**In-place Sorting**: Sorting algorithms do not require any extra space and sorting is said to happen in-place, or for example, within the array itself

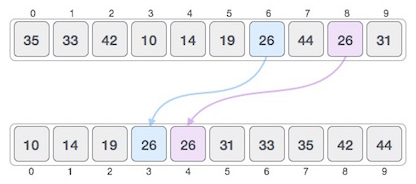
Example: Bubble sort

**Not-in-place Sorting**: Sorting algorithm requires space which is more than or equal to the elements being sorted

Example: Merge-sort

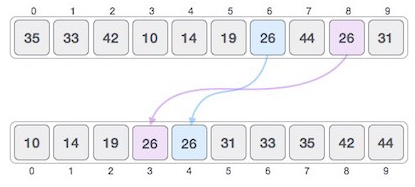
## Stability

**Stable sorting**: If a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear



Example: Bubble Sort, Insertion Sort, Merge Sort, Count Sort etc.

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



Example: Quick Sort, Heap Sort etc.

**Can we make any sorting algorithm stable?**

Any given sorting algo which is not stable can be modified to be stable. There can be sorting algo specific ways to make it stable, but in general, any comparison based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal keys.

**Can we make any sorting algorithm stable?**

Any given sorting algorithm which is not stable can be modified to be stable. There can be sorting algorithm specific ways to make it stable, but in general, any comparison based sorting algorithm which is not stable by nature can be modified to be stable by changing the key comparison operation so that the comparison of two keys considers position as a factor for objects with equal keys.

## Internal or external sort

**Internal Sort**: All records in main memory or RAM

**External Sort**: Records are on disk/tapes

## Recursive or non-recursive

**Recursive**: Implemented using recursive function

Example: Quick sort, Merge sort

**Non-recursive**: Implemented using non-recursive function

Example: Insertion sort, Selection sort

## Adaptive and Non-Adaptive Sorting Algorithm

**Adaptive**: 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.

**Non-Adaptive**: 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

## Increasing Order

A sequence of values is said to be in increasing order, if the successive element is greater than the previous one.

Example: 1, 3, 4, 6, 8, 9

## Decreasing Order

A sequence of values is said to be in decreasing order, if the successive element is less than the current one.

Example: 9, 8, 6, 4, 3, 1

## 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.

Example: 9, 8, 6, 3, 3, 1

## 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.

Example: 1, 3, 3, 6, 8, 9

# Comparison of algorithms

<https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms>

|  |  |  |  |
| --- | --- | --- | --- |
| ****Algorithm**** | ****Time Complexity**** | | |
| Name | **Best** | **Average** | **Worst** |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | Ω(n^2) | θ(n^2) | O(n^2) |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | Ω(n) | θ(n^2) | O(n^2) |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | Ω(n) | θ(n^2) | O(n^2) |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n^2) |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) | Ω(n+k) | θ(n+k) | O(n^2) |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) | Ω(nk) | θ(nk) | O(nk) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Memory | Stable | Method | Other notes |
| [Selection Sort](http://geeksquiz.com/selection-sort/) | 1 | No | Selection | Stable with O(n) extra space, for example using lists |
| [Bubble Sort](http://geeksquiz.com/bubble-sort/) | 1 | Yes | Exchanging | Tiny code size. |
| [Insertion Sort](http://geeksquiz.com/insertion-sort/) | 1 | Yes | Insertion | O(*n* + *d*), in the worst case over sequences that have *d* [inversions](https://en.wikipedia.org/wiki/Inversion_(discrete_mathematics)). |
| [Heap Sort](http://geeksquiz.com/heap-sort/) | 1 | No | Selection |  |
| [Quick Sort](http://geeksquiz.com/quick-sort/) | log n on average worst case space complexity n  Sedgewick variation is log n worst case | Typical in-place sort is not stable; stable versions exist | Partitioning | Quicksort is usually done in-place with O(log n) stack space |
| [Merge Sort](http://geeksquiz.com/merge-sort/) | A hybrid [block merge sort](https://en.wikipedia.org/wiki/Block_sort) is O(1) mem | Yes | Merging | [Highly parallelizable](https://en.wikipedia.org/wiki/Merge_sort#Parallel_merge_sort) (up to O(log *n*) using the Three Hungarians' Algorithmor, more practically, Cole's parallel merge sort) for processing large amounts of data. |
| [Bucket Sort](https://www.geeksforgeeks.org/bucket-sort-2/) |  |  |  |  |
| [Radix Sort](https://www.geeksforgeeks.org/radix-sort/) |  |  |  |  |

# Popular sorting algorithms

## Simple sorts

1. Insertion sort
2. Selection sort

## Efficient sorts

1. Merge sort
2. Heapsort
3. Quicksort

## Bubble sort and variants

1. Bubble sort
2. Shellsort
3. Comb sort

## Distribution sort

1. Counting sort
2. Bucket sort
3. Radix sort