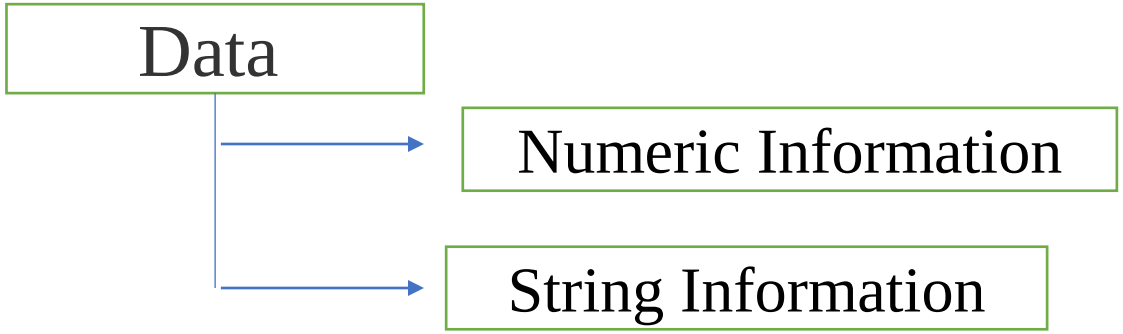
Fundamentals/Properties of Sorting

Sorting

- The arrangement of data in a preferred order is called sorting
- Sorting Algorithms are methods of reorganizing a large number of items into some specific order such as highest to lowest, or vice-versa, or even in some alphabetical order.



- Numeric Information: For a given sequence of n distinct elements <x1,x2,x3…xn> where each pair of elements can be ordered, the output is a reordering $<x_1', x_2' x_3' \ldots\ldots x_n'>$ of the given sequence such that $x_1' \leq x_2' \leq x_3' \leq \ldots \leq x_n'$ (1<=i<=n for every i). The ordering relation for numeric data simple involves arrangement of items in sequence from smallest to largest vice versa such that each item is less than or equal to its immediate successor.
- String Information: String information generally arranged in standard lexicographical order or dictionary order. This means that the words are alphabetically ordered based on their component alphabets

Sorting Algorithm

Why Study Sorting?

- It helps to make a set of data more readable.
- It makes it easier to implement search algorithms in order to find or retrieve an item from the entire dataset.
- Sorting can often reduce the complexity of a problem in order to improve time and space complexity.
- Sorting algorithms have direct applications in searching algorithms, database algorithms, divide and conquer methods, data structure algorithms, and many more.

Example: we have a list of unsorted numbers: 41 6 14 31 43 49 34 21 33

Imagine that we wanted to find the number 33, but didn't know where it was. Or, in an even more terrible scenario: imagine we want to find a number that isn't even *in* the list! In the worst case, we'd have to look through every single one of the numbers, until we got to the end, and realized that the number wasn't even in the list! In other words, it would take *linear* time.

what if it was 100 million numbers? Or 1 billion?

Sorting Algorithm

There are two **classes of sorting algorithms**

1.$O(n^2)$-**algorithms** (simpler and less sophisticated)

2. O(n log n)-**algorithms**. (advance and sophisticated)

Comparision of Sorting Algorithm complexicities as function of n

n	n^2	$nlog_2n$
10	100	33.2
100	10000	664.4
1000	1000000	9966
10000	100000000	132877

Clearly it can be observed that for higher value of n the advance methods established their superiority over simple methods.

Sorting Algorithm

Choose a sorting algorithm for a particular problem, consider the running time, space complexity, and the expected format of the input list.

Algorithm	Best-case	Worst-case	Average-case	Space Complexity	Stable?
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$	Yes
Insertion Sort	$O(n)$	$O(n2)$	$O(n2)$	$O(1)$	Yes
Bubble Sort	$O(n)$	$O(n2)$	$O(n2)$	O(1)	Yes
Quicksort	$O(n\log n)$	$O(n2)$	$O(n\log n)$	$n\log n$ best, n avg	Usually not*
Heapsort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	O(1)	No
Counting Sort	$O(k+n)$	$O(k+n)$	$O(k+n)$	O(k+n)	Yes

Sorting Algorithm

Time Complexities of Sorting Algorithms:

Algorithm	Best	Average	Worst
Quick Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Merge Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Radix Sort	Ω(nk)	Θ(nk)	O(nk)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)

Sorting Algorithm

Properties of Sorting:

Sorting algorithms are categorized on the following basis –

1. By number of comparisons: Comparison-based sorting algorithms check the elements of the list by key comparison operation and need at least O(n log n) comparisons for most inputs. In this method, algorithms are classified based on the number of comparisons. For comparison based sorting algorithms, best case behavior is O(n log n) and worst case behavior is O(n2). For example – Counting sort, Bucket sort, Radix sort, etc.

2. Depending on Space complexity/memory usage(How much memory will this algorithm need to run?)

There are two types of classifications for the space complexity of an algorithm: **In-place or Out-of-place**.

In-Place: An ***in-place*** algorithm is one that operates directly on the inputted data and changes it. The original input is effectively destroyed when it is modified the algorithm. An in-place algorithm transforms the input without using any extra memory. As the algorithm executes, the input is usually overwritten by the output, and no additional space is needed for this operation. An in-place algorithm may require a small amount of extra memory for its operation. However, the amount of memory required must not be dependent on the input size and should be constant.

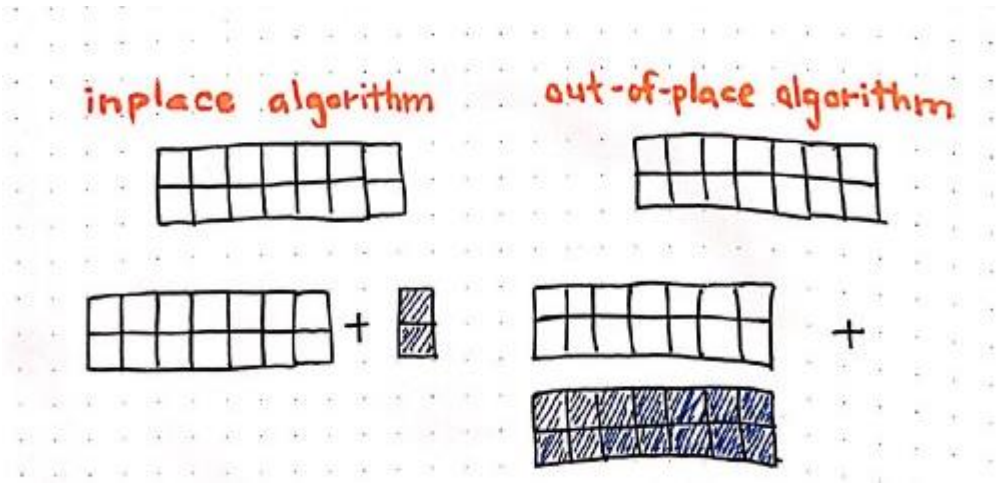
Example:insertion sort, selection sort, quick sort, bubble sort, heap sort .

All these algorithms require a constant amount of extra space for rearranging the elements in the input array.

Sorting Algorithm

Out-of-place algorithms

Unlike an in-place algorithm, the extra space used by an out-of-place algorithm depends on the input size. ***out-of-place*** algorithms don't operate directly on the original dataset; instead, the make a new copy, and perform the sorting on the copied data. This can be safer, but the drawback is that the algorithm's memory usage grows with input size.



Let's illustrate that by taking an example of reversing an array of integers :

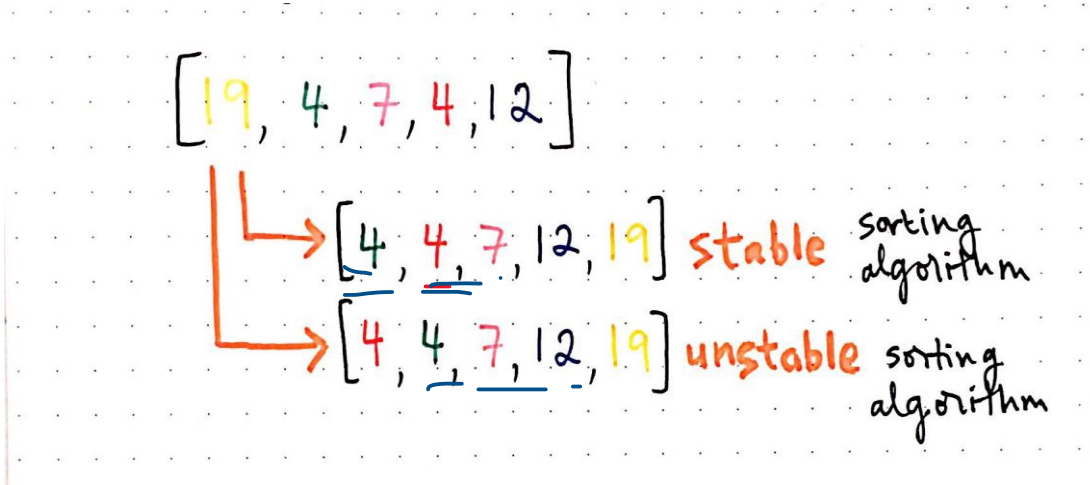
The idea is to create a new array of the same type and size, fill it with elements from the original array in reverse order, and finally copy the contents of the new array into the original one. Since this implementation requires O(n) extra space, this is not-in-place.

An in-place algorithm requires only a fixed number of integers for the auxiliary variables i, j, and temp, irrespective of the input's size. This can be done by reading the elements from both ends of the array and swapping them.

Sorting Algorithm

Stability:

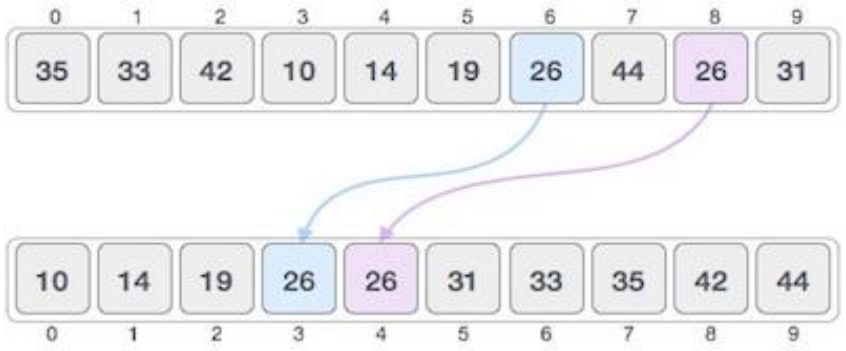
- A single dataset will have more than one element that has the same "sort key"; in other words, multiple items in a list could be considered equal in the way that they could be sorted. In the example shown here, there are two number 4's, one of which is green, and the other which is red.
- There are two ways that this list could be sorted, given the fact that there are two elements that could be sorted "equally": the green 4 could come first, the way that it did in the original list, or the red 4 could be sorted first. This is exactly what defines the stability of a sorting algorithm.



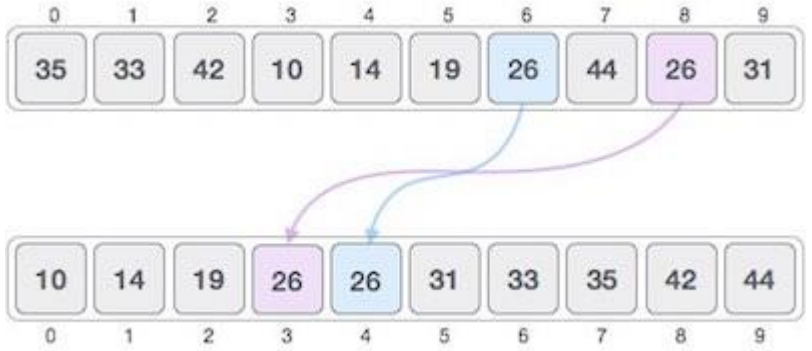
A ***stable*** algorithm is one that preserves the relative order of the elements; if the keys are the same, we can guarantee that the elements will be ordered in the same way in the list as they appeared before they were sorted. An ***unstable algorithm*** is one where there is no guarantee that, if two items are found to have the same sort key, that their relative order will be preserved.

Sorting Algorithm

1



2



What is it stable sorting or Unstable sorting?

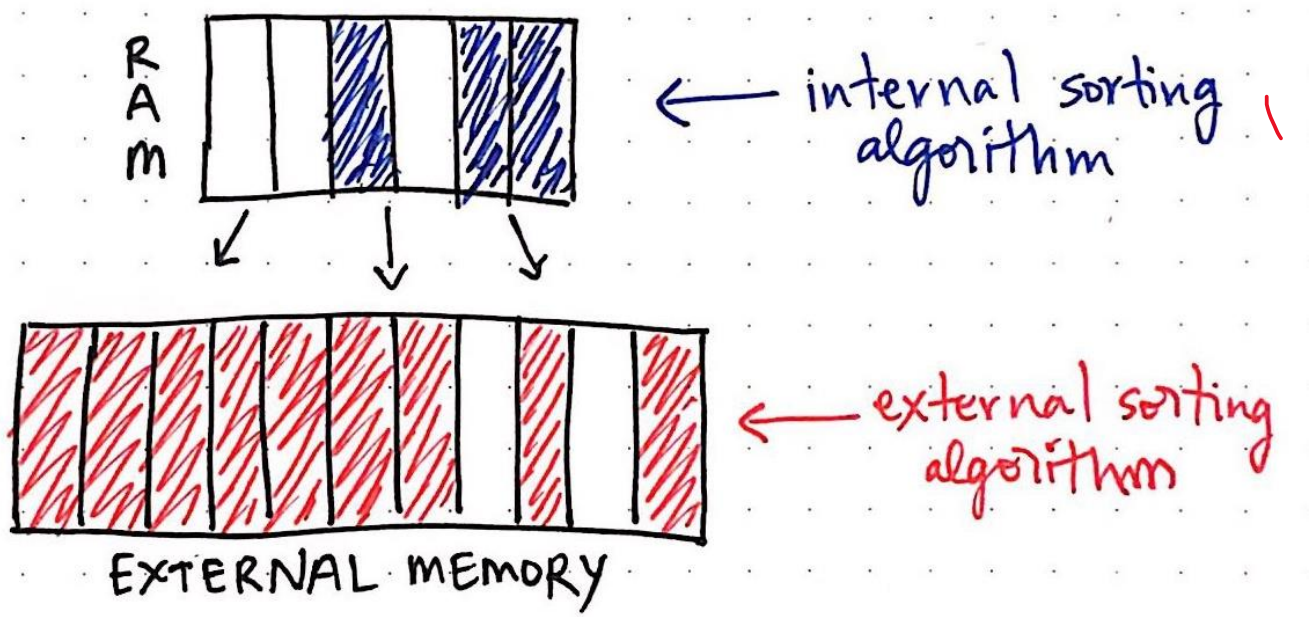
Example: bubble sort, insertion sort, merge sort ------Stable sorting
Selection sort----Unstable sort

Sorting Algorithm

Internal vs. external:

If all of the data that needs to be sorted can be kept in main memory, the algorithm is an ***internal*** sorting algorithm. Example: insertion sort, bubble sort etc

However, if the records have to be stored outside of main memory —in other words, stored in external memory, in either a disk or a tape — the algorithm is referred to as an ***external*** sorting algorithm. Example: External Merge sort

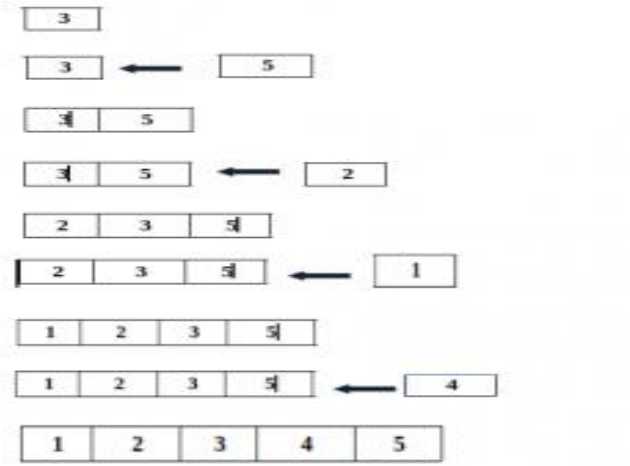


Sorting Algorithm

Online sorting and Offline sorting :

An **online algorithm** is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the beginning.

Example: Insertion sort considers one input element per iteration and produces a partial solution without considering future elements. Thus insertion sort is an online algorithm.



An **offline algorithm** is given the whole problem data from the beginning and is required to output an answer which solves the problem .

Example: The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. which requires access to the entire input; it is thus an offline algorithm.

Criteria for Choosing a Sorting Algorithm

To choose the appropriate algorithm for different data, you need to know some properties about your input data.

Selection Sort –

We can use Selection Sort as per below constraints :

- When the list is small. As the time complexity of selection sort is O(n*n) which makes it inefficient for a large list.
- When memory space is limited because it makes the minimum possible number of swaps during sorting.

Bubble Sort –

We can use Bubble Sort as per below constraints :

- It works well with large datasets where the items are almost sorted because it takes only one iteration to detect whether the list is sorted or not. But if the list is unsorted to a large extend then this algorithm holds good for small datasets or lists.
- This algorithm is fastest on an extremely small or nearly sorted set of data.

Criteria for Choosing a Sorting Algorithm

Insertion Sort –

We can use Insertion Sort as per below constraints :

- If the data is nearly sorted or when the list is small as it has a complexity of O(n*n) and if the list is sorted a minimum number of elements will slide over to insert the element at it's correct location.
- This algorithm is stable and it has fast running case when the list is nearly sorted.
- The usage of memory is a constraint as it has space complexity of O(1).

Merge Sort –

We can use Merge Sort as per below constraints :

- Merge sort is used when the data structure doesn't support random access since it works with pure sequential access that is forward iterators, rather than random access iterators.
- It is widely used for external sorting, where random access can be very, very expensive compared to sequential access.
- It is used where it is known that the data is similar data.
- Merge sort is fast in the case of a linked list.
- It is used in the case of a linked list as in linked list for accessing any data at some index we need to traverse from the head to that index and merge sort accesses data sequentially and the need of random access is low.
- The main advantage of the merge sort is its stability, the elements compared equally retain their original order.

Criteria for Choosing a Sorting Algorithm

Quick Sort –

We can use Quick Sort as per below constraints :

- Quick sort is fastest, but it is not always O(n*log n), as there are worst cases where it becomes O(n*n).
- Quicksort is probably more effective for datasets that fit in memory. For larger data sets it proves to be inefficient so algorithms like merge sort are preferred in that case.
- Quick Sort is an in-place sort (i.e. it doesn’t require any extra storage) so it is appropriate to use it for arrays.