Stable Sorting Algorithms

Last updated: June 28, 2023

Written by: Priyank Srivastava
(https://www.baeldung.com/cs/author/priyank-srivastava)

**Sorting
(https://www.baeldung.com/cs/category/algorithms/sorting)**

1. Overview

In this tutorial, we'll learn what stable sorting algorithms are and how they work. Further, we'll explore when the stability of sorting matters.

2. Stability in Sorting Algorithms

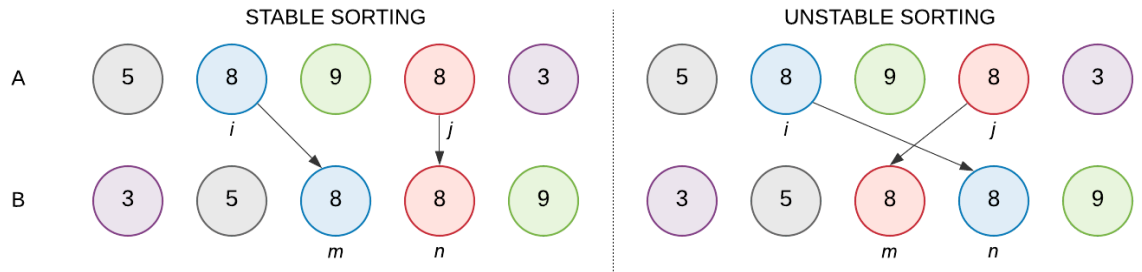
The stability of a sorting algorithm is concerned with **how the algorithm treats equal (or repeated) elements**. Stable sorting algorithms preserve the relative order of equal elements, while

unstable sorting algorithms don't. In other words, stable sorting maintains the position of two equals elements relative to one another.

Let *A* be a collection of elements and < be a strict weak ordering (https://www.boost.org/sgi/stl/StrictWeakOrdering.html) on the elements. Further, let *B* be the collection of elements in *A* in the sorted order. Let's consider two equal elements in *A* at indices *i* and *j*, i.e, *A[i]* and *A[j]*, that end up at indices *m* and *n* respectively in *B*. We can classify the sorting as stable if:

```
i < j and A[i] = A[j] and m < n
```

Let's understand the concept with the help of an example. We have an array of integers A: *[5, 8, 9, 8, 3]*. Let's represent our array using color-coded balls, where any two balls with the same integer will have a different color which would help us keep track of equal elements (8 in our case):



(/wp-content/uploads/2019/08/Stable-vs-Unstable-1.png)
Stable sorting maintains the order of the two equal balls numbered 8, whereas unstable sorting may invert the relative order of the two 8s.

3. When Stability Matters

3.1. Distinguishing Between Equal Elements

All sorting algorithms use a key to determine the ordering of the elements in the collection, called the *sort key*.

If the sort key is the (entire) element itself, equal elements are indistinguishable, such as integers or strings.

On the other hand, equal elements are distinguishable if the sort key is made up of one or more, but not all attributes of the element, such as *age* in an *Employee* class.

3.2. Stable Sorting Is Important, Sometimes

We don't always need stable sorting. Stability is not a concern if:

- equal elements are indistinguishable, or
- all the elements in the collection are distinct

When **equal elements are distinguishable, stability is imperative**. For instance, if the collection already has some order, then sorting on another key must preserve that order.

For example, let's say we are computing the word count of each distinct word in a text file. Now, we need to report the results in decreasing order of count, and further sorted alphabetically in case two words have the same count.

First step – compute the word count:

```
Input:

how much wood would woodchuck chuck if woodchuck could
chuck wood

Output:

how       1
much      1
wood      2
would     1
woodchuck 2
chuck     2
if        1
could     1
```

Second step – sort the whole list lexicographically, then by word count:

```
First pass, sorted lexicographically:

(chuck, 2)
(could, 1)
(how, 1)
(if, 1)
(much, 1)
(wood, 2)
(woodchuck, 2)
(would, 1)
```

```
Second pass, sorted by count using an unstable sort:

(wood, 2)
(chuck, 2)
(woodchuck, 2)
(could, 1)
(how, 1)
(if, 1)
(would, 1)
(much, 1)
```

As we have used an unstable sort, the second pass does not maintain the lexicographical order.

That's where the stable sort comes into the picture. **Since we already had sorted the list lexicographically, using a stable sort to by word count does not change the order of equal elements anymore.** As a result words with the same word count remain sorted lexicographically.

```
Second pass, sorted by count using a stable sort:

(chuck, 2)
(wood, 2)
(woodchuck, 2)
(could, 1)
(how, 1)
(if, 1)
(much, 1)
(would, 1)
```

3.3. Radix Sort

Radix Sort (https://brilliant.org/wiki/radix-sort/) is an integer sorting algorithm that **depends on a sorting subroutine that must be stable**. It is a non-comparison based sorting algorithm that sorts a collection of integers. It groups keys by individual digits that share the same significant position and value.

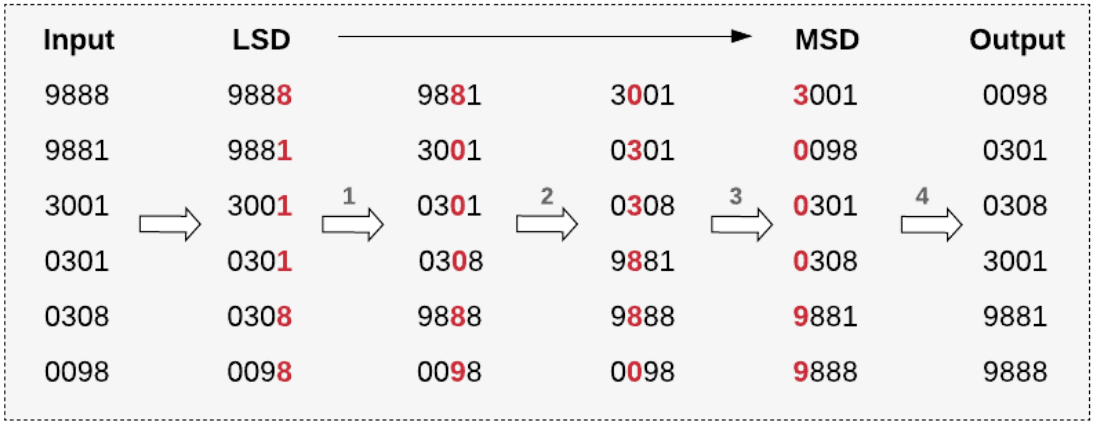
Let's unpack the formal definition and restate the basic idea:

```
for each digit 'k' from the least significant digit (LSD)
to the most significant digit (MSD) of a number:
  apply counting-sort algorithm on digit 'k' to sort the
input array
```

We are using Counting Sort (https://brilliant.org/wiki/counting-sort/) as a subroutine in Radix Sort. Counting Sort is a stable integer sorting algorithm. We don't have to understand how it works, but

that **Counting Sort is stable**.

Let's look at an illustrative example:



(/wp-content/uploads/2019/08/radix-stable.png)
Each invocation of the Counting Sort subroutine preserves the order from the previous invocations. For example, while sorting on the tens' place digit (second invocation) 9881 shifts downwards, but stays above 9888 maintaining their relative order.

Thus, Radix Sort utilizes the stability of the Counting Sort algorithm and provides linear time integer sorting.

4. Stable and Unstable Sorting Algorithms

Several common sorting algorithms are stable by nature, such as Merge Sort (/java-merge-sort), Timsort (https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Arrays.html#sort(java.lang.Object%5B%5D)), Counting Sort (https://brilliant.org/wiki/counting-sort/), Insertion Sort (/java-insertion-sort), and Bubble Sort (/java-bubble-sort). Others such as Quicksort (/java-quicksort), Heapsort (/java-heap-sort) and Selection Sort (/java-selection-sort) are unstable.

We can modify unstable sorting algorithms to be stable. For instance, we can use extra space to maintain stability in Quicksort.

5. Conclusion

In this tutorial, we learned about stable sorting algorithms and looked at when stability matters, using Radix Sort as an example.

CATEGORIES

ALGORITHMS (/CS/CATEGORY/ALGORITHMS)
ARTIFICIAL INTELLIGENCE (/CS/CATEGORY/AI)
CORE CONCEPTS (/CS/CATEGORY/CORE-CONCEPTS)
DATA STRUCTURES (/CS/CATEGORY/DATA-STRUCTURES)
GRAPH THEORY (/CS/CATEGORY/GRAPH-THEORY)
LATEX (/CS/CATEGORY/LATEX)
NETWORKING (/CS/CATEGORY/NETWORKING)