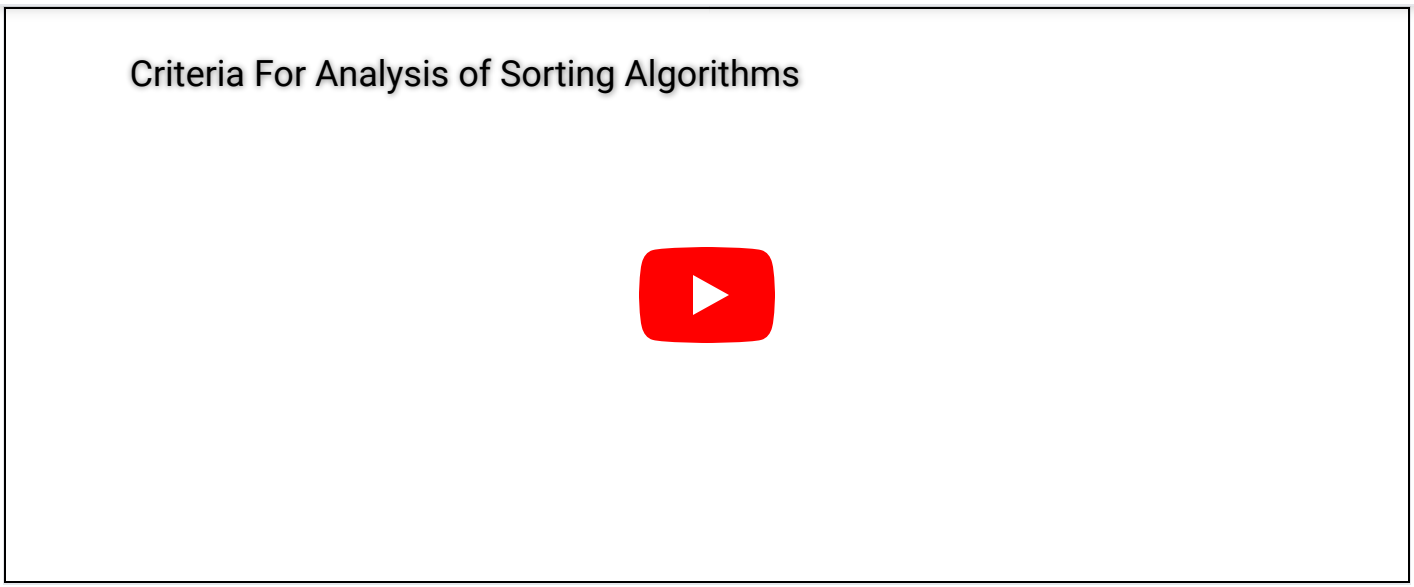


Show Course Contents

Overview Q&A Downloads Announcements

Criteria For Analysis of Sorting Algorithms

In the last lecture, we introduced to you the basics of sorting, its definition, its different types, and several examples to make you confident about its applications in real life. Today, in this lesson, you will learn how to come up with criteria for analyzing different sorting algorithms and why one differs from the other.

Before we proceed, make sure you have been through the basics. There are some old concepts, which I'll probably rush through. So, please check out the first 10-12 lectures before jumping to advance.

We will discuss each of the below-mentioned criteria in detail:

1. Time Complexity
2. Space Complexity
3. Stability
4. Internal & External Sorting Algorithms
5. Adaptivity
6. Recursiveness

Time Complexity:

- We observe the time complexity of an algorithm to see which algorithm works efficiently for larger data sets and which algorithm works faster with smaller data sets. What if one sorting algorithm sorts only 4 elements efficiently and fails to sort 1000 elements. What if it takes too much time to sort a large data set? These are the cases where we say the time complexity of an algorithm is very poor.
- In general, O(N log N) is considered a better algorithm time complexity than $O(N^2)$, and most of our algorithms' time complexity revolves around these two.

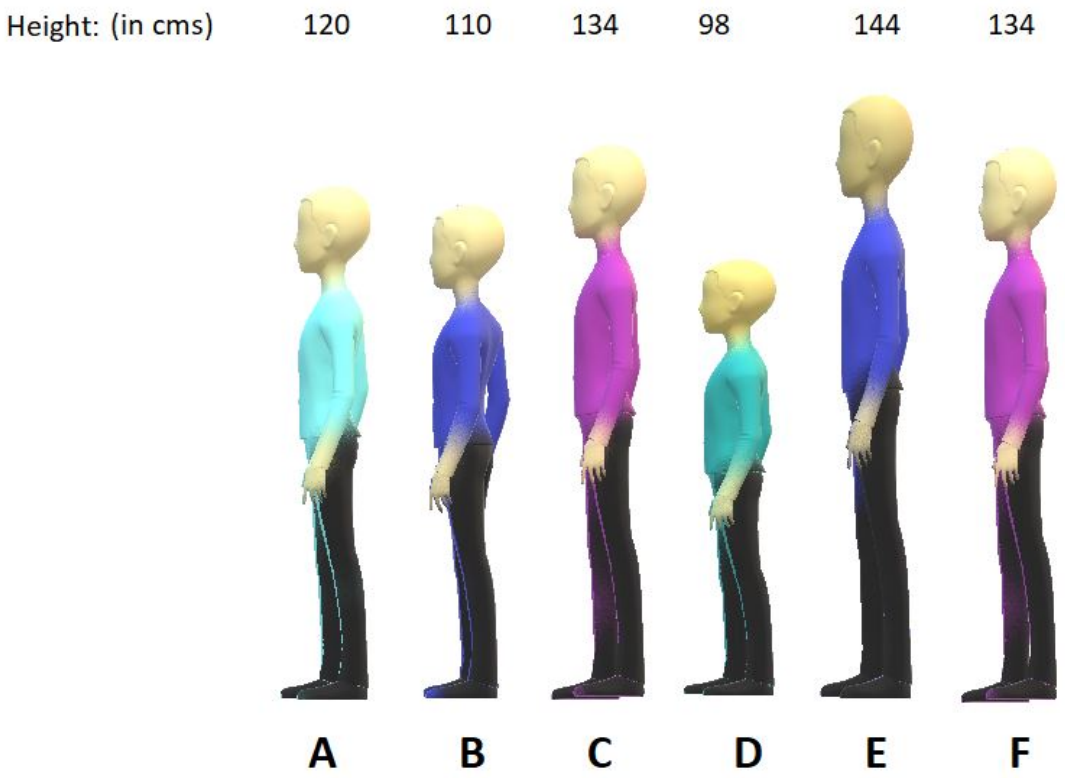
Note: Lesser the time complexity, the better is the algorithm.

Space Complexity:

- The space complexity criterion helps us compare the space the algorithm uses to sort any data set. If an algorithm consumes a lot of space for larger inputs, it is considered a poor algorithm for sorting large data sets. In some cases, we might prefer a higher space complexity algorithm if it proposes exceptionally low time complexity, but not in general.
- And when we talk about space complexity, the term **in-place sorting algorithm** arises. The algorithm which results in constant space complexity is called an in-place sorting algorithm. Inplace sorting algorithms mostly use swapping and rearranging techniques to sort a data set. One example is Bubble Sort (will be covered in the incoming videos).

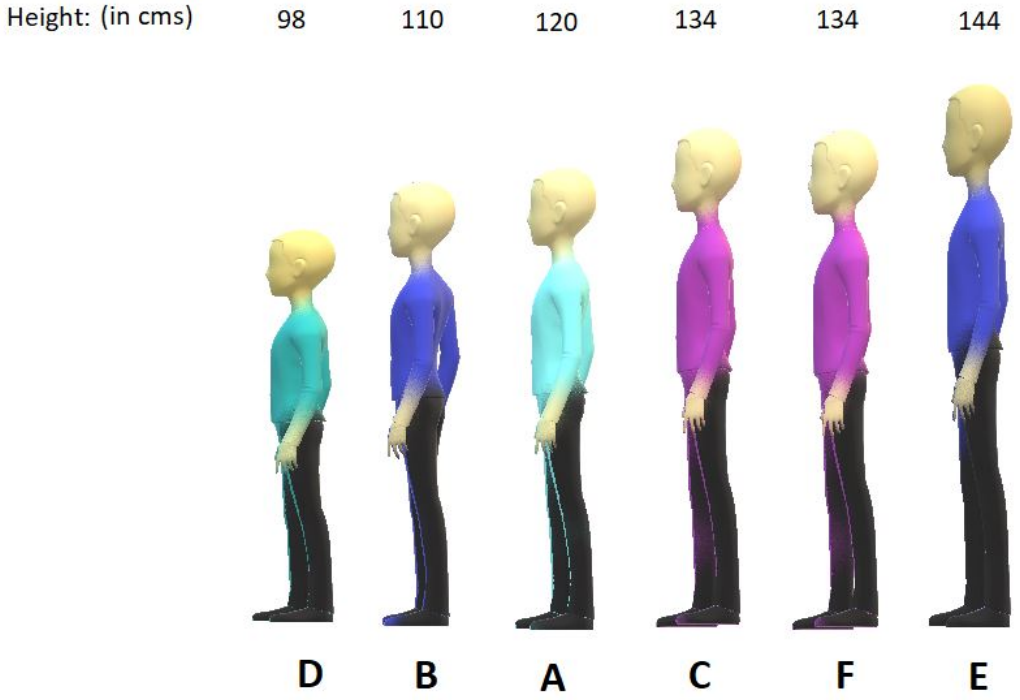
Stability:

The stability of an algorithm is judged by the fact whether the order of the elements having equal status when sorted on some basis is preserved or not. It probably sounded technical, but let me explain.
Suppose you have a set of numbers, 6, 1, 2, 7, 6, and we want to sort them in increasing order by using an algorithm. Then the result would be 1, 2, 6, 6, 7. But the key thing to look at is whether the 6s follow the same order as that given in the input or they have changed. That is, whether the first 6 still comes before the second 6 or not. If they do, then the algorithm we followed is called stable, otherwise unstable.
An illustration for your better understanding:

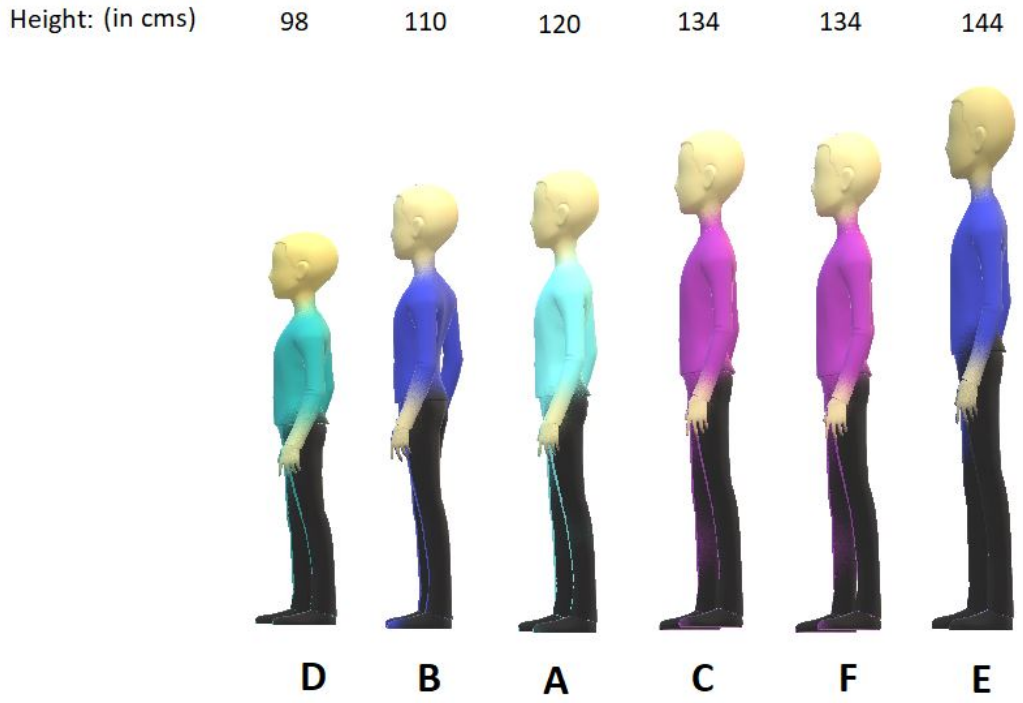


Suppose we called 6 students from a class and made them stand on the first-come basis. And then we measured their heights. And now, we used two different algorithms to assign them a position based on their increasing heights.

Sorting by algorithm A:



Sorting by algorithm B:



Algorithm A is stable, whereas Algorithm B is unstable because algorithm A preserved the order between students C and F having equal heights, and algorithm B couldn't.

Internal & External Sorting Algorithms

When the algorithm loads the data set into the memory (RAM), we say the algorithm follows internal sorting methods. In contrast, we say it follows the external sorting methods when the data doesn't get loaded into the memory.

Adaptivity:

Algorithms that adapt to the fact that if the data are already sorted and it must take less time are called **adaptive algorithms**. And algorithms which do not adapt to this situation are not adaptive.

Recursiveness:

If the algorithm uses recursion to sort a data set, then it is called a recursive algorithm. Otherwise, non-recursive.

And these were the most general criteria to analyze our sorting algorithms. If some of these terminologies were not known to you, I would recommend you first clear

these basics. You can check out my C Playlist and the first few lectures of this DSA playlist. That would surely help you. Keep practicing.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll start with our first sorting algorithm called the **Bubble Sort Algorithm**. Till then, keep coding.

Previous Next