INTERACTIVE TEACHER DIRECTION ASSESSMENT STANDARD

# Sorting Algorithms and Running Time

**Learning Objective:** Students will demonstrate understanding of several sorting algorithms and how different algorithms that solve the same problem can have different runtimes.

**Standards addressed:**

* 9-12S.AP.12: Implement searching and sorting algorithms to solve computational problems
* 9-12S.AP.13: Evaluate algorithms in terms of their efficiency
* 3-5.AP.10: Compare and refine multiple algorithms for the same task and determine which is the most appropriate

## Glossary

**Algorithm** - set of rules or steps for solving a problem

**Runtime** - the length of time that a program takes to run

## Introduction

Have you ever thought about the way we sort things? For example, when you put some books in alphabetical order, what steps do you go through? You might just start putting books in place without giving it a lot of thought.

But how might a computer sort things? As you’ve seen by now, computers need very specific instructions to carry out a task. In the programs you’ve created, you’ve broken down problems into steps so that the computer can follow those steps to do something.

So how might you give a computer instructions for sorting things? Well, just like a human could go about sorting things in different ways, there are different ways for computers.

You may have heard the word “algorithm” before. An **algorithm** is just a set of steps designed to solve a problem. Programmers write these steps in code to create programs to solve problems.

| **Video**  To illustrate how sorting algorithms work and why it matters which you use, have students watch the following video:  <https://www.youtube.com/watch?v=WaNLJf8xzC4> |
| --- |

As it turns out, sorting is a more difficult problem than it may seem at first! Computers usually only compare two things at a time. The different sorting algorithms are based on the sequence of comparisons.

There are many different types of sorting algorithms, and they can have different runtimes in different situations. The **runtime** of a program is exactly what it sound like–how long it takes to run!

Usually you want to choose a more efficient algorithm with a shorter runtime. The tricky part is that now algorithm has the fastest runtime in every situation!

## Demo:

Pair the students up before this demonstration.

To further illustrate the variety of sorting algorithms and how their runtimes differ in different contexts: <https://www.toptal.com/developers/sorting-algorithms>. Have the pairs of students talk about which ones seem to be faster and slower.

Focus on the sorting algorithms you have already discussed: bubble sort, insertion sort, and quick sort. Introduce the selection sort.

Students may wonder why anyone would use the selection algorithm at all, seeing as it is clearly one of the slowest in every situation. You can point out and explain the discussion section here: <https://visualgo.net/bn/sorting> (don’t look at radix sort, random quick sort, and counting sort). Same thing with this, have the students work together to understand the patterns that each sorting algorithm uses to be able to fully sort the list of elements (link was changed because the previous one only had a simulation for selection sort. This new link covers all of the types of common sorting methods, which helps scaffold their learning because it shows the code for more coding-literate students, lets you insert your own values, and change the speed of the demonstration. It’s much more flexible compared to the previous link for learning).

## Unplugged Activity:

Have students try it out by sorting strips of paper based on length and arrange them from shortest to longest. Have one partner keep count of the number of comparisons. Students can use an eraser for a marker.

Firstly, the students should try to sort the strips of paper however they want, counting the number of comparisons they make before the whole set of strips is sorted.

Then have them do the same thing, except with the two algorithms below (have them watch the link for a more comical demonstration of the algorithm). Video links are added as a way to scaffold their learning by providing more resources for them to use to understand these sorting algorithms.

**Insertion Sort (**[**https://www.youtube.com/watch?v=ROalU379l3U**](https://www.youtube.com/watch?v=ROalU379l3U)**)**:

1. Start with the paper strips in random order.
2. Set a marker for the sorted section.
3. Repeat the following steps 4 through 6 until the unsorted section is empty
4. Select the first unsorted strip
5. If the strip to its right is shorter, swap it to the left
6. Advance the marker to the right one position
7. Stop

**Selection sort (**[**https://www.youtube.com/watch?v=Ns4TPTC8whw**](https://www.youtube.com/watch?v=Ns4TPTC8whw)**)**:

1. Start with the paper strips in random order.
2. Set a marker for the unsorted section at the front of the list
3. Repeat steps 4 - 6 until one number remains in the unsorted section
4. Compare all unsorted numbers in order to select the smallest one
5. Swap this number with the first number in the unsorted section
6. Advance the marker to the right one position
7. Stop

After trying both methods, students should compare the performance of these two algorithms as well as the performance of these algorithms with what they originally did to sort the strips. Once they are done, students should try to articulate their own sorting algorithm.

Remind them that they should only be comparing two strips at a time!

## Assessment:

Short answer questions:

1. What is an algorithm?
2. Why might a programmer choose one sorting algorithm over another if the end result is the same?
3. Briefly explain the main differences between the selection sort and the quicksort algorithms.
4. Briefly compare the runtimes of each of the algorithms by listing them from slowest to fastest. The algorithms to compare are: bubble sort, quick sort, selection sort, insertion sort, and merge sort.

## Reflection:

1. As we rely more and more on computing devices, algorithms shape the world we live in today. Other than runtime, how might the algorithms programmers use affect the outcomes they have? Think about things that you might sort that are more complex than just numbers, strips of paper, or weights. (If students still struggle to think of examples, mention social media).

## Extra Resources:

Sorting weights:

<http://csunplugged.org/sorting-algorithms/>

A Scratch Project that Illustrates Bubble Sort:

<https://scratch.mit.edu/projects/17768900/>