General Assembly

STUDY GUIDE

INTRODUCTION TO SORTING

What Is Sorting?

Sorting in computer science is no different than sorting in real life: it involves taking an unordered collection of stuff and putting them into some sort of order.
- Sorting is the foundation for much more complex tasks (e.g., tasks involving search, scheduling, or optimization).
- There are dozens of sorting algorithms: bubble sort, merge sort, quick sort, heap sort, radix sort, cube sort, bucket sort, shell sort, tree sort, insertion sort, selection sort, counting sort, and so on.
- Each sort has its advantages and disadvantages. It turns out that some of the most intuitive ways to sort are pretty inefficient.

Two Types of Sorting

There are two major types of sorting algorithms: **comparison sort** and **distribution sort**. One isn't better than the other; they're suited for different types of data sets.
- **Comparison sort** compares two items at a time and decides which one to put first.
- **Distribution Sort** uses some property of the item itself to decide where it fits. Most of the time, this means dividing the items into groups based on some characteristic of the data itself.

Stability

Stability is an important characteristic of sorting.
- A sorting algorithm is **stable** if it preserves the initial order of a collection.
- Some sorting algorithms are stable while others aren't.
- Stability lets you sort data by multiple factors (e.g., sorting a deck of cards by number and suit).

Comparing Sorting Algorithms

We can compare the efficiency of sorting algorithms using Big O Notation. Recall that, in software development, Big O measures the efficiency of an algorithm as its input increases.
- **Time complexity** refers to the amount of time an algorithm takes to run.
- **Space complexity** refers to the amount of memory or RAM an algorithm needs to run.

Big O measures look at the "worst-case scenario": the maximum amount of time it could take for your algorithm to run or the maximum amount of space it would require.

Time Complexity of Sorting Algorithms

The variable **N** is the size of the input: the number of items being sorted. As we move from left to right across the table, we're moving from very efficient runtimes like O(1) and O(log(N)) to extremely inefficient runtimes like O(N!).

Input **(N)**	**O(1)**	**O(log(N))**	**O(N)**	**O(N^2)**	**O(N!)**
1	1	1	1	1	1
10	1	3	10	100	3,628,800
40	1	5	40	1,600	8.16e+47
80	1	6	80	6,400	80!
600	1	9	600	360,000	600!

Space Complexity of Sorting Algorithms

Space complexity measures how much extra space our algorithm requires. The space complexity of sorting algorithms is described in one of two formats: **in place** or **out of place**.

Type	Big O Complexity	Description
In place	**O(1)** or **(Olog(N))**	The space the algorithm takes is predictable and does not depend on the size of the input.
Out of place	**O(N)** or **O(N^2)**	The algorithm uses one or more extra arrays to sort the values.

The Worst, The Best, and The Average

Big O, which represents worst-case performance, is the most common way to compare algorithms, but we can discuss best and average cases, too.
- **Best-case performance** is called "Big Omega", or Ω.
- Big Omega describes how fast an algorithm would run if it performed the fewest possible actions on a data set (e.g., if it tried sorting an array that was already sorted).
- **Average-case performance** is called "Big Theta", or Θ.
- Big Theta describes the typical runtime of an algorithm.

Choosing a Sorting Algorithm

The perfect sort doesn't exist: there isn't one "best" algorithm, because different sorts are good at different things. When choosing a sorting algorithm, you'll need to consider:
- Efficiency (measured by time and space complexity)
- Stability (whether or not the items in the set stay in order)
- Sorting method (comparison, distribution, or a combination of the two)
- The size and structure of your data (sorted, unsorted, very large, etc.)

Sorting Algorithm Cheat Sheet

Keep these comparisons in mind as you learn about different sorting algorithms in the lessons that follow.

Sort	Best Case	Average Case	Worst Case	Space	Stability	Sorting Method
Bubble sort	**Ω(N)**	**Θ(N^2)**	**O(N^2)**	**O(1)**	Stable	Comparison
Insertion sort	**Ω(N)**	**Θ(N^2)**	**O(N^2)**	**O(1)**	Stable	Comparison
Bucket sort	**Ω(N+K)**	**Θ(N+K)**	**O(N^2)**	**O(N+K)**	Stable	Distribution
Radix sort	**Ω(NK)**	**Θ(NK)**	**O(NK)**	**O(N+K)**	Stable	Distribution
Merge sort	**Ω(log(N))**	**Θ(log(N))**	**O(log(N))**	**O(N)**	Stable	Comparison
Quick sort	**Ω(log(N))**	**Θ(log(N))**	**O(N^2)**	**O(log(N))**	Unstable	Comparison