General Assembly

STUDY GUIDE

DIVIDE-AND-CONQUER SORTING ALGORITHMS

The Divide-and-Conquer Method

Two algorithms use the "divide-and-conquer" method: **merge sort** and **quick sort**.
- These algorithms use **recursion** to sort data.
- They chop datasets into smaller pieces that are easier to sort than the dataset as a whole.
- Though they're a bit difficult to understand and implement, these algorithms are efficient and widely used in programming.

Merge Sort

Merge sort takes an array and splits it in half over and over again until it’s small and sorted, and then it merges those small sorted pieces back together in order. Merge sort is actually two algorithms, the **merge sort algorithm** and the **merge algorithm**. Each algorithm handles a different phase of the merge sort process:
1. Divide: The merge sort algorithm splits the array in half until it can’t anymore.
2. Conquer: The merge algorithm puts the split pieces back together.

Divide: the Merge Sort Algorithm

The merge sort algorithm divides arrays until they only have one item in them (which are fundamentally sorted). Like any recursive function, the merge sort algorithm is defined by a base case and a recursive case:

Case	Condition	Action
Base case	The array is one element or shorter.	Return the array.
Recursive case	The array is longer than one element.	Divide the array into two pieces. Call itself again on both the left and right sides you just created. Keep doing this until you have single-element arrays. The base case has been reached!

Conquer: The Merge Algorithm

The merge algorithm takes two sorted arrays, compares them, and sorts them into a results list. It keeps doing this until it creates a large, sorted array. Here is the basic process of the merge algorithm. It starts with the sorted single-element arrays created by the merge sort algorithm:
1. Start at the beginning of two arrays of items.
2. Compare the first item from each array to each other.
3. Whichever value is less, copy it to a results list.
4. Move on to the next item in the array that just gave its first element to the results list.
5. Repeat steps 1–4 until you have all of the elements from both arrays in the results list.

Merge Sort’s Time and Space Complexity

Space complexity: The merge algorithm creates a separate "results array" as the data is being merged, so it's an out-of-place sort and uses **O(N)** space complexity.

Time complexity: The two components of merge sort have two different time complexities.
- The recursive merge sort algorithm takes **O(log(N))** time, which is very efficient.
- The non-recursive merge algorithm is an **O(N)** operation.
- To get the overall complexity of merge sort, we multiply **O(log(N))** and **O(N)** to get **O(N log(N))**

Quick Sort

Quick sort is another recursive divide-and-conquer sort. Quick sort differs from merge sort in how it approaches the divide phase: merge sort divides an array into two parts at a time, but quick sort divides an array into three parts at a time:
- The pivot (a single element that is fundamentally sorted)
- The left partition, which should contain numbers lower than the pivot
- The right partition, which should contain numbers higher than the pivot

Elements are sorted by recursively calling quick sort on the array on either side of the pivot.

Visualizing Quick Sort

The concept of quick sort and partitioning is complicated. Here are a couple of fun resources you can explore if you want to dive in further:
- This tool lets you move through the algorithm on a sample data set step-by-step.
- This video helps you visualize quick sort with folk dancing.

Quick Sort's Time and Space Complexity

Space complexity: Quick sort's space complexity is**O(log(N))**, which is much lower than merge sort's.

Time complexity: Quick sort’s worst-case time efficiency is**O(N^2)**, but its average-time complexity is **Θ(log(N))**, which is the same as merge sort.
- Quick sort is usually efficient.
- Because quick sort has such a low space complexity, it tends to be a bit faster than merge sort in practice.

Merge Sort vs. Quick Sort

Merge sort:
- Better at sorting datasets with very similar values
- Useful in virtual memory environments and caching environments
- The built-in sort function in C programming languages, Java, and Python

Quick sort:
- Better in database scenarios
- Useful when additional data may arrive during or after sorting
- Used by Safari and Firefox in their implementation of JavaScript’s **.sort** function.

Preparing for Interviews

In a job interview, you might be asked to:
- Sketch out how merge sort and quick sort work
- Compare the two and describe situations in which you'd use one or the other

Use this tool to compare merge and quick sort (and other sorting algorithms).