A word on technical interview problems

Context before we dive in

The problems we are about to see are the genre of problem commonly asked in software engineering interviews at large technology companies.

What they test well:

- Whether you can identify the right data structure for a problem
- Whether you can reason about time and space complexity
- Whether you can translate an idea into clean code under pressure

What they do not test:

- General engineering ability
- Intelligence
- Creativity, communication, or system design

Think of them like the SAT: a contentious standardized genre with known patterns. Learning the patterns is a skill in itself, separate from the underlying computer science.

Why we cover them:
They are a genuinely useful exercise in applied data structure selection. Each one rewards the same question:

What is my dominant operation, and which structure makes it fast?

Problem 1: Balanced parentheses

LeetCode #20: Valid Parentheses (Easy) `leetcode.com/problems/valid-parentheses`

Problem: Given a string of brackets, return `true` if every opening bracket is closed in the correct order.

`"{}()[]{}"`	**true**
`"({[]})"`	**true**
`"(}"`	**false**
`"([)]"`	**false**

Think about it:

- What information do you need to remember as you scan left to right?
- When you see a closing bracket, what do you need to check?
- What order do you need to access that information?

Hint: which structure fits?

You are processing characters one at a time. When you see `(`, you need to remember it. When you later see `)`, you need to check the *most recently opened* unmatched bracket.

Most recently opened = last in, first out.

Structure: `Stack`

Complexity target:
$O(n)$ time, $O(n)$ space

Try it on LeetCode before next class.

Problem 2: k-th largest element

LeetCode #215: Kth Largest Element `leetcode.com/problems/kth-largest-element-in-an-array`

Problem: Given an array and integer k, return the k-th largest element.

`[3,2,1,5,6,4], k=2` → **5**
`[3,2,3,1,2,4,5,5,6], k=4` → **4**

Think about it:

- What is the simplest approach, and what is its complexity?
- Can you do better than sorting the whole array?
- What if new elements kept arriving in a stream — could you still answer efficiently?

Hint: which structure fits?

You want to maintain the k largest elements seen so far. When a new element arrives, you need to quickly check whether it belongs in that group — and if it does, evict the smallest of the k.

Quickly find and remove the smallest of k elements = min-heap of size k.

Structure: `Min-heap` of size k

Complexity target:
$O(n \log k)$ time, $O(k)$ space

Try it on LeetCode before next class.

Problem 3: Two sum

LeetCode #1: Two Sum (Easy) `leetcode.com/problems/two-sum`

Problem: Given an array and a target, return the indices of the two numbers that add up to the target.

`[2,7,11,15], target=9` → **[0,1]**
`[3,2,4], target=6` → **[1,2]**

Think about it:

- The naïve approach checks every pair — what is that complexity?
- As you scan element x, what are you really looking for?
- If you had already seen `target - x`, where would you find it instantly?

Hint: which structure fits?

For each element x you scan, you need to check in $O(1)$ whether `target - x` has already appeared. If you store every element you have seen — along with its index — you can answer that question instantly.

$O(1)$ lookup by value = hash table.

$\Downarrow$

Structure: `Hash table` (value → index)

Complexity target:
$O(n)$ time, $O(n)$ space

Try it on LeetCode before next class.

Problem 4: Valid anagram

LeetCode #242: Valid Anagram (Easy) `leetcode.com/problems/valid-anagram`

Problem: Given two strings `s` and `t`, return `true` if `t` is an anagram of `s` — that is, if `t` uses exactly the same characters in the same quantities.

`s="anagram", t="nagaram"` → **true**
`s="rat", t="car"` → **false**

Think about it:

- What information do you need to compare about the two strings?
- Sorting both strings would work — what is that complexity?
- Can you do it in $O(n)$ without sorting?
- What if you counted each character in `s`, then subtracted counts using `t`?

Hint: which structure fits?

You need to track how many times each character appears in each string. A hash table maps each character to its count. Scan `s` to build the counts, scan `t` to subtract them — if everything reaches zero, they are anagrams.

Count by key, $O(1)$ per update = hash table.

Structure: `Hash table` (character → count)

Complexity target:
$O(n)$ time, $O(1)$ space (only 26 letters)

Try it on LeetCode before next class.

Problem 5: Merge two sorted lists

LeetCode #21: Merge Two Sorted Lists (Easy) `leetcode.com/problems/merge-two-sorted-lists`

Problem: Given the heads of two sorted linked lists, merge them into one sorted linked list and return its head.

`1→2→4` and `1→3→4` $\rightarrow$ `1→1→2→3→4→4`
`[]` and `[0]` $\rightarrow$ `0`

Think about it:

- You have seen how to merge two sorted arrays — how is this similar?
- Do you need to allocate a new list, or can you rewire the existing nodes?
- At each step, which list's current node is smaller?
- What happens when one list runs out before the other?

Hint: which structure fits?

This is the merge step from merge sort, but on linked lists. You maintain two pointers, one into each list. At each step you pick the smaller node, attach it to your result, and advance that pointer. No new nodes needed, you are just rewiring `next` pointers.

Pointer manipulation at known locations = linked list operations.

Structure: `Linked list` (in-place pointer rewiring)

Complexity target:
$O(n+m)$ time, $O(1)$ space

Try it on LeetCode before next class.