ChatGPT

Top 30 Stack Problems for Google Interview Preparation

Preparing for a Google interview with limited time means focusing on high-yield problems that cover a broad range of stack applications. Below is a curated list of 30 stack-related LeetCode problems (mostly Medium/Hard) that are known to be **popular in interviews at top tech companies (including Google)** [1] [2]. These are grouped by category to ensure variety in concepts and to avoid solving very similar problems repeatedly. Each problem is annotated with its difficulty and the key idea or why it's important.

1. Balanced Parentheses and Valid Strings

Stack is often used to validate or manipulate strings with parentheses/brackets – a classic interview theme [2]. These problems test understanding of stack for matching pairs and handling extra characters:

- **32. Longest Valid Parentheses (Hard)** – Find the length of the longest well-formed parentheses substring. Requires using a stack or two-pass scan to track valid segments. This problem is frequently asked at Google [3] and other top companies [2] because it tests edge cases in stack usage (empty stack, resetting start indices, etc).
- **678. Valid Parenthesis String (Medium)** – An extension of the classic valid parentheses problem where the string can contain `*` wildcards that can act as either `(`, `)` or an empty string. This problem demands careful use of two stacks or greedy counts to account for flexibility in matching, providing depth in understanding beyond the basic valid parentheses check.
- **1249. Minimum Remove to Make Valid Parentheses (Medium)** – Remove the minimum characters to make a parentheses string valid. A straightforward use of a stack (or counter) to track unmatched parentheses and remove them. It ensures you understand how to use stacks to process and rebuild strings and is a practical variant of the balancing problem [4].
- **856. Score of Parentheses (Medium)** – Evaluate a string of balanced parentheses where `()` has a score of 1 and `AB` has score `A+B`, `(A)` has score `2*A`. This problem is solved by pushing interim scores onto a stack. It teaches how a stack can be used to evaluate a custom expression (here, nested parentheses scoring) in a single pass.

2. Monotonic Stack for Array Problems

Many array problems use a **monotonic stack** technique, where the stack is maintained in increasing or decreasing order. Mastering these is crucial, as *FAANG companies love to ask monotonic stack questions* [5]. This category includes finding next greater elements, span problems, and others where stack helps track previous smaller/greater elements efficiently:

- **42. Trapping Rain Water (Hard)** – Compute water trapped between bars. While a two-pointer approach exists, the stack solution uses a decreasing stack to find boundaries for water trapping. This has been asked in Google interviews [6] and is a classic problem to test understanding of using stacks to handle range calculations (each bar traps water until a taller bar is seen) [7].

- **84. Largest Rectangle in Histogram (Hard)** – Find the largest rectangular area in a histogram. A classic monotonic stack problem: we maintain an increasing stack of indices, and when a bar breaks the trend, we calculate areas. This problem is notoriously tricky and often appears in Google interview prep [8] for testing algorithmic thinking with stacks.
- **907. Sum of Subarray Minimums (Medium)** – Sum of minimum value of every subarray in an array. The efficient solution uses two monotonic stacks to find, for each element, how far it extends as the minimum to left and right [9]. It's an excellent example of using stacks to count contributions of elements to a global sum, and covers a more advanced application of the histogram technique.
- **503. Next Greater Element II (Medium)** – Given a circular array, find the next greater element for each element. This extends the basic Next Greater Element problem by handling wrap-around using a stack (by iterating twice). It ensures you can manipulate stack logic for cyclical scenarios.
- **739. Daily Temperatures (Medium)** – For each day's temperature, find how many days until a warmer temperature. This is a straightforward monotonic decreasing stack problem and a common interview question to test basic stack application for Next Greater-like patterns.
- **853. Car Fleet (Medium)** – Compute how many car fleets reach an endpoint given positions and speeds. While not obviously a stack problem, the typical solution sorts cars by start position and uses a stack to count fleets by comparing arrival times from the end. This problem has appeared in Google interviews [10] and tests one's ability to apply a stack in a greedy simulation context (monotonically decreasing arrival times represent fleets).
- **735. Asteroid Collision (Medium)** – Simulate collisions between asteroids moving in opposite directions. A stack is used to model the sequence of collisions: moving through the array, use the stack to handle when a new asteroid meets the previous ones. It's a good test of careful stack-based simulation (pushing and popping based on conditions).
- **456. 132 Pattern (Medium)** – Determine if there's a 132 pattern in an array (a subsequence `ai < aj > ak` with `ai < ak < aj`). Uses a tricky decreasing stack approach to keep track of possible "ak" values while scanning from right. This problem is included to deepen understanding of creative stack usage for pattern finding in arrays. It's considered a hard-to-think-of solution, reflecting Google's fondness for challenging problems that require insight.

3. Expression Evaluation and Nested Structures

Stacks are invaluable for evaluating expressions (infix or postfix) and parsing nested structures. Google interviewers may ask you to evaluate arithmetic expressions or interpret nested data formats, which these problems cover [11]:

- **224. Basic Calculator (Hard)** – Implement a basic calculator to evaluate a string expression with `+`, `-`, and parentheses. This question is known to be "irritating" but important [11]. It tests your ability to use stacks to handle parentheses and maintain the current evaluation context (e.g., pushing results and signs when entering parentheses). Mastering this ensures you can handle expression parsing – a skill sometimes tested in interviews.
- **150. Evaluate Reverse Polish Notation (Medium)** – Evaluate a postfix expression given as a list of tokens. This is a straightforward stack application: push numbers and pop for operations. It's a fundamental example of stack usage for computation and often a warm-up interview question. (It also reinforces understanding of how compilers or calculators evaluate postfix expressions [12].)
- **341. Flatten Nested List Iterator (Medium)** – Given a nested list of integers and lists, create an iterator that flattens it. This problem uses a stack to perform a controlled depth-first traversal, flattening on the fly. It teaches how to use stacks for custom iteration and is useful for

understanding how to traverse nested structures without recursion (which is relevant when designing iterators or dealing with JSON-like data).

4. Stack Applications in Trees and Linked Lists

Stacks can replace or supplement recursion in tree traversal and can assist in non-trivial linked list operations. Google has been known to ask tree traversal and reconstruction problems that involve explicit stack usage:

- **114. Flatten Binary Tree to Linked List (Medium)** – Convert a binary tree into a "flattened" linked list in-place (following pre-order traversal order). While it can be done recursively, using an explicit stack to guide the traversal is a great way to demonstrate non-recursive tree processing. It shows understanding of tree structure manipulation using a stack to store nodes to revisit.
- **173. Binary Search Tree Iterator (Medium)** – Design an iterator over a BST yielding nodes in sorted (inorder) order. The typical implementation uses a stack to simulate the controlled recursion (traversing to the leftmost node and backtracking). It's a common interview question that tests ability to use a stack for on-demand traversal of a tree.
- **331. Verify Preorder Serialization of a Binary Tree (Medium)** – Check if a given preorder traversal (with null markers) is valid for some binary tree. One solution uses a stack to verify the structure by replacing complete subtrees with a marker. This problem is more of a logic puzzle using a stack to simulate tree node consumption, and it broadens your understanding of how stack can validate structural correctness of serialized data.
- **445. Add Two Numbers II (Medium)** – Add two numbers represented by linked lists (most significant digit first). The straightforward approach is to use two stacks to reverse the order (since we want to add from the least significant digits). This problem is included to demonstrate using stacks for linked list problems where reversing or backtracking is needed without actually modifying the list. It's a practical pattern that can appear in various linked list arithmetic or comparison tasks.

5. Stack in String Manipulation

Beyond parentheses, stacks are handy for various string manipulation tasks – especially when the operation involves removing characters or undoing actions. These problems ensure you can apply a stack to process strings character by character:

- **71. Simplify Path (Medium)** – Simplify a Unix-style file path (with `.` and `..`). A stack is used to handle navigation: push directory names, pop for `..`. This is a solid example of using a stack for string parsing and was featured in some stack-focused learning resources [13]. It's straightforward but very practical.
- **388. Longest Absolute File Path (Medium)** – Given a string representation of file system paths with newline and `\t` as delimiters, find the longest file path. The solution uses a stack to keep track of current directory lengths at each depth. This problem is a good test of how stacks can help maintain hierarchical context while parsing a string. It's known to be tricky and has appeared in interviews for its creative use of stacks to accumulate lengths.
- **402. Remove K Digits (Medium)** – Remove k digits from a number string to form the smallest possible number. This is essentially a monotonic stack problem on characters: you push digits and remove a previous digit when a smaller digit comes after (if removal quota allows). It combines greedy thinking with stack usage on strings, and ensures you understand how to apply monotonic stack patterns to character sequences (common in making lexicographically smallest strings).

- **316. Remove Duplicate Letters (Hard)** – Remove duplicates from a string so that the result is the smallest in lexicographical order (keeping the relative order of characters). This classic problem uses a stack to build the result while ensuring each character appears once. You push characters, and pop them if a smaller character that appears later could take their place (while ensuring not to lose needed characters). It's a favorite for testing greedy algorithms with stack (and was asked by companies like Google for its nuanced logic).
- **1190. Reverse Substrings Between Each Pair of Parentheses (Medium)** – Reverse every substring enclosed in parentheses in a given string. The straightforward approach is using a stack: whenever you see `)`, pop characters until the matching `(` and then push them back (effectively reversing that segment). It's a good example of stack usage for string transformation problems and tests careful implementation of stack operations during parsing.

6. Advanced and Miscellaneous Stack Problems

This final group contains diverse problems that use stacks in less obvious or more complex ways, covering everything from interval timing to custom data structures. These ensure depth in your preparation, touching on design and complex algorithms:

- **636. Exclusive Time of Functions (Medium)** – Given logs of function enter/exit times in a multithreaded environment, determine exclusive time for each function. A stack is used to track active functions; each time a function starts, the previous one is paused. This problem simulates call stack behavior and is good for understanding how stacks can model real-life nested processes (a scenario that has shown up in interviews to test one's ability to simulate systems using data structures).
- **85. Maximal Rectangle (Hard)** – Given a binary matrix, find the largest rectangle of 1's. This extends the histogram rectangle problem (#84) to 2D by using it on each row's histogram of consecutive 1's. Every row, a stack is used to compute largest area considering that row as the base. It combines dynamic programming (building heights) with the stack algorithm for largest rectangle, adding depth to your knowledge. It's included to ensure you can apply stack techniques in multi-dimensional contexts.
- **1130. Minimum Cost Tree From Leaf Values (Medium)** – Given an array, combine adjacent leaves (numbers) to form a binary tree with minimum possible sum of product costs. A known greedy solution uses a **monotonic decreasing stack** [9] to decide which leaves to combine first (always combine the two smallest adjacent leaves). This problem is valuable as it's not immediately obvious as a stack problem – it tests your ability to discover a stack-based strategy to minimize a cost function, a type of optimal substructure problem sometimes seen in Google interviews.
- **2487. Remove Nodes from Linked List (Medium)** – Remove all nodes from a singly linked list that have a node with a greater value to their right. This can be solved with a monotonic **decreasing stack** (iterating from left, maintaining a stack of nodes whose values are in decreasing order). It's essentially the Next Greater Element concept applied to a linked list. This problem ensures you can apply stack patterns in linked list contexts (not just arrays), demonstrating understanding of the underlying concept beyond a specific data structure.
- **946. Validate Stack Sequences (Medium)** – Given two sequences (pushed and popped), check if they could correspond to a sequence of stack push/pop operations. This is a direct simulation of a stack: iterate through the push list and use a stack to match the pop sequence. We include this because it tests understanding of the stack operational constraints and is a neat logical problem (often asked to verify that a candidate understands stack mechanics well).
- **895. Maximum Frequency Stack (Hard)** – Design a stack-like data structure that supports push, pop, and pop-the-most-frequent operations. This is an advanced design problem where you use multiple stacks or a hash-map of stacks to track frequencies [9]. It has appeared in interview

rotations for its blend of a stack with a hash map or heap. Including this problem exposes you to designing complex data structures with stacks under the hood, without being purely implementation-heavy (the concept is the main challenge).

Each of these problems targets a different aspect of stack usage, from classic stack algorithms to creative adaptations. By practicing this set, you'll cover **all important stack concepts** – validating parentheses, parsing expressions, monotonic stacks for spans and ranges, simulating systems or recursion, and even designing new data structures – which collectively have been asked in Google interviews and other top tech companies [1] [5]. Good luck with your preparation!

Sources: The selection and categories are informed by common interview problem lists and known favorites of top companies. For example, balanced parentheses problems are noted to be frequently asked at FAANG companies [2], and monotonic stack problems are specifically highlighted as a favorite in tier-1 company interviews [5]. Several of the listed problems (e.g. *Longest Valid Parentheses*, *Trapping Rain Water*, *Largest Rectangle in Histogram*, *Car Fleet*) are well-documented as Google interview questions [6] [8] [10], underscoring the relevance of this focused practice set.

---

[1] [4] [5] [11] [12] [13] GitHub - ctanishq/Stack-for-Coding-Interviews: Master the Stack data structure for coding interviews. A minimal set of 15 problems, progressing from easy to hard.
https://github.com/ctanishq/Stack-for-Coding-Interviews

[2] [3] Longest Substring With Balanced Parentheses Problem
https://interviewkickstart.com/blogs/problems/longest-valid-parentheses

[6] [7] Leetcode 42 — Trapping Rain Water (Google Software Engineer Interview Question) | by cslintee | Medium
https://medium.com/@cslintee/leetcode-42-trapping-rain-water-google-software-engineer-interview-question-92df4055b738

[8] Largest Rectangle in Histogram Interview Question for Google - Taro
https://www.jointaro.com/interviews/questions/largest-rectangle-in-histogram/?company=google

[9] 2487. Remove Nodes From Linked List - LeetCode Solutions
https://walkccc.me/LeetCode/problems/2487/

[10] Car Fleet: 853 - AWESOME google interview question - YouTube
https://www.youtube.com/watch?v=TPSiTAFhszA