CODING INTERVIEW ASSESSMENT RUBRIC

FAANG-Level Evaluation for Algorithms & Implementation

1. OVERVIEW

Coding interviews assess problem-solving ability, technical depth, and clear thinking under pressure. This is NOT about memorizing solutions.

Expected Interview Flow

1. Clarify	2-3 min	Ask about inputs, outputs, constraints, edge cases
2. Examples	2-3 min	Work through examples by hand
3. Approach	5-7 min	Discuss approach and complexity BEFORE coding
4. Code	15-20 min	Write clean code while explaining
5. Test	5-7 min	Test with cases, find and fix bugs
6. Optimize	5-10 min	Discuss improvements and answer follow-ups

Pass/Fail Thresholds

	STRONG PASS	Optimal solution, clean code. Ready for FAANG.
	PASS	Working optimal solution. Minor issues. Hire signal.
	BORDERLINE	Working but suboptimal or with bugs. More practice needed.
	FAIL	Incomplete or major issues. Significant gaps.
	STRONG FAIL	Cannot solve problem. Rebuild foundations.

Critical Rule: Any dimension below 3.0 = automatic FAIL.

2. SCORING DIMENSIONS

2.1 Problem Understanding & Clarification

FAANG engineers NEVER start coding without complete clarity on requirements and edge cases.

Jumps into coding without clarification. Wrong assumptions.	Few questions. Misses critical constraints. Partial understanding.	Asks relevant questions. Identifies most edge cases. Confirms understanding.	Strategic questions revealing deep thinking. All edge cases identified.	Exceptional clarification uncovering hidden complexity.

Feedback Templates:

- **Score 1-2:** You started coding without clarifying. ALWAYS ask: input constraints, output format, edge cases (empty, single, duplicates, negative), time/space priorities.
- **Score 3:** Good questions but missed [edge case]. Build a checklist: null/empty, single element, all same, sorted/unsorted, duplicates, overflow.
- **Score 4-5:** Excellent clarification. You identified [non-obvious edge case]. This systematic approach is exactly what FAANG wants.

Follow-up Questions:

- What happens with empty input?
- Can there be negative numbers?
- Are there duplicates?
- What if no valid answer?
- Optimize for time or space?

2.2 Algorithm Design & Approach

FAANG expects candidates to discuss approach BEFORE coding and justify choices.

Cannot identify any approach. Flawed algorithms.	Only brute force. No optimization. Doesn't discuss before coding.	Identifies optimal approach. Explains clearly. Discusses before coding.	Quickly finds optimal. Compares multiple approaches with tradeoffs.	Instantly recognizes patterns. Multiple optimal solutions. Deep insight.

Feedback Templates:

- **Score 1-2:** Study these patterns: Two Pointers, Sliding Window, Hash Maps, Binary Search, BFS/DFS, Dynamic Programming. Know when to apply each.
- **Score 3:** Working approach but missed optimal. Key insight was [technique]. When you see [indicator], consider [optimization].
- **Score 4-5:** Excellent design. You recognized [pattern] and identified [technique] as optimal. Strong analytical thinking.

Follow-up Questions:

- Walk me through your approach before coding?
- Time and space complexity?
- Is there a more efficient way?
- Why this data structure over alternatives?

2.3 Complexity Analysis

FAANG expects rigorous Big-O analysis with understanding of tradeoffs.

Cannot analyze complexity. No Big-O understanding.	Basic but often incorrect. Can't analyze recursion.	Correct for straightforward cases. Identifies dominant terms.	Rigorous analysis including recursion. Discusses tradeoffs.	Expert analysis. Best/ average/worst case. Cache considerations.

Feedback Templates:

- **Score 1-2:** Master fundamentals: O(1), O(log n), O(n), O(n log n), $O(n^2)$, O(2^n). For space: count allocations and recursion stack.
- **Score 3:** Time correct but [missed space / didn't account for X]. Remember: recursion = O(depth) stack. Hash maps = O(n) space.
- **Score 4-5:** Excellent analysis. Correctly identified [time] and [space] with clear reasoning.

Follow-up Questions:

- Time complexity?
- Space complexity including stack?
- Best possible for this problem?
- Can you reduce space?

2.4 Code Quality & Implementation

FAANG expects clean, readable, production-quality code.

Code doesn't run. Syntax errors. Unreadable.	Runs but buggy. Poor naming. Hard to follow.	Correct and readable. Reasonable naming and structure.	Clean, well-structured. Excellent naming. Helper functions. Production-ready.	Elegant, concise. Perfect style. Could merge to production as-is.

Feedback Templates:

- **Score 1-2:** Focus on: meaningful names, consistent formatting, helper functions, testing as you write. Practice without IDE.
- **Score 3:** Works but could be cleaner. Rename [x] to [descriptive], extract [logic] to helper. Easier to debug and maintain.
- **Score 4-5:** Excellent quality. Your use of [good practice] made it very readable. This is FAANG standard.

Follow-up Questions:

- More descriptive variable name?
- Extract to helper function?
- How handle [edge case]?
- Walk me through this section?

2.5 Testing & Debugging

FAANG engineers test thoroughly and debug efficiently.

No testing. Cannot identify bugs. Random debugging.	Only happy path. Misses obvious bugs. Inefficient debugging.	Tests multiple cases including edge cases. Reasonable debugging.	Proactive comprehensive testing. Quick bug identification. Systematic debugging.	Exhaustive testing. Identifies bugs by inspection. Expert debugging.

Feedback Templates:

- **Score 1-2:** ALWAYS test: empty, single, two elements, normal case, large input, problem-specific edges. Trace line-by-line.
- **Score 3:** Tested some but missed [case]. Build checklist: min input, max constraints, boundaries, all same, sorted/reverse.
- **Score 4-5:** Excellent testing. You caught [bug] proactively. Systematic debugging of [issue] was impressive.

Follow-up Questions:

- Walk through with [test case]?
- What about empty input?
- Returns [wrong] for [input] - find the bug?
- What test cases before shipping?

2.6 Communication & Thought Process

FAANG interviews are collaborative - silent coding is penalized.

Silent throughout. Cannot explain. Ignores interviewer.	Only speaks when asked. Unclear. Misses hints.	Explains adequately. Thinks aloud. Responds to hints.	Excellent communication. Natural think-aloud. Incorporates feedback.	Outstanding. Makes complex simple. True collaboration.

Feedback Templates:

- **Score 1-2:** Practice thinking aloud: narrate what you're considering, verbalize when stuck, explain decisions. Interview is conversation, not exam.
- **Score 3:** Adequate but be more proactive. Don't wait to be asked - explain approach, verbalize tradeoffs, flag uncertainty.
- **Score 4-5:** Excellent communication. You explained [approach], reasoned about [decision], incorporated hint about [topic]. This is what FAANG wants.

Follow-up Questions:

- What are you thinking?
- Walk me through reasoning?
- Have you thought about [hint]?
- Why did you choose [decision]?

2.7 Problem-Solving Under Pressure

FAANG problems are designed to be hard - resilience matters.

Panics when stuck. Gives up. Frustrated.	Gets flustered. Slow to adapt. Visible stress.	Maintains composure. Adapts with hints. Manages pressure.	Stays calm. Quickly adapts. Smooth recovery. Positive attitude.	Thrives under pressure. Elegant recovery. Maintains enthusiasm.

Feedback Templates:

- **Score 1-2:** When stuck: pause, breathe, verbalize what you know, consider simpler versions, ask for hints. Practice under time pressure.
- **Score 3:** Handled pressure but showed stress at [moment]. Build resilience: practice with timers, do mock interviews.
- **Score 4-5:** Excellent composure. When [challenge], you stayed calm and found a path forward. Essential for FAANG.

Follow-up Questions:

- What have you tried?
- Think of simpler approach?
- Ignore [constraint] for now?
- Which part most unsure about?

3. ALGORITHMIC PATTERNS

Two Pointers	Sorted arrays, pairs, palindromes	O(n), O(1)
Sliding Window	Subarrays with constraints	O(n), O(k)
Hash Map	Frequency, lookup, two-sum	O(n), O(n)
Binary Search	Sorted data, search space	O(log n), O(1)
BFS	Shortest path, level-order	O(V+E), O(V)
DFS	Path finding, tree traversal	O(V+E), O(V)
Dynamic Programming	Optimal substructure	Varies
Backtracking	Permutations, combinations	O(n!) or O(2^n)
Heap	K-th element, merge K lists	O(n log k), O(k)
Monotonic Stack	Next greater/smaller	O(n), O(n)
Union-Find	Connected components	O(α(n))
Trie	Prefix matching	O(word len)
Topological Sort	Dependencies	O(V+E), O(V)

4. RED FLAGS

Starts coding without discussing approach	Problem Understanding capped at 2.5
Cannot write syntactically correct code	Code Quality capped at 2.0
Wrong complexity analysis (claims O(n) for $O(n^2)$)	Complexity capped at 2.0
Silent for extended periods	Communication capped at 2.5
Refuses hints or feedback	Communication capped at 2.0
Gives up without trying alternatives	Problem-Solving capped at 2.0
Cannot trace code with simple example	Testing capped at 2.0

No testing before declaring done	Testing capped at 3.0
Cannot explain own code	Communication capped at 2.0

5. ASSESSMENT TEMPLATE

Problem Understanding	[X.X]
Algorithm Design	[X.X]
Complexity Analysis	[X.X]
Code Quality	[X.X]
Testing & Debugging	[X.X]
Communication	[X.X]
Problem-Solving Under Pressure	[X.X]
OVERALL AVERAGE	**[X.X]**

Approach: [Algorithm used]
Time: [Big-O]
Space: [Big-O]
Optimal?: [Yes/No]
Verdict: [PASS/FAIL]

6. LLM INSTRUCTIONS

Interview Flow

- Present problem clearly with examples
- If coding immediately, interrupt: 'Walk me through approach first'
- Ask for complexity BEFORE coding
- If silent >30s, prompt: 'What are you thinking?'
- After code, ask to test with specific cases
- If correct but suboptimal: 'Can we do better?'

Hint Ladder

- Level 1 (subtle): 'Think about constraints differently'
- Level 2 (pattern): 'Consider using [data structure]'
- Level 3 (direct): 'What if we [technique]?'
- Level 4 (explicit): 'Try [approach]. Implement that.'

Hint penalties: L1-2: -0.25, L3: -0.5, L4: -1.0 and caps at 3.5

Scoring Rules

- 3.0 = meets expectations, not good
- 5.0 requires: optimal, clean, no bugs, excellent communication, no hints
- Incomplete but correct approach: 3.0-3.5
- Complete brute force: max 3.5