Coding Prep Guide

What to Expect

- 45-min interview with 1-2 coding questions
- You'll be tested on your problem-solving and core CS fundamental skills (theory, algorithms, data structures, design patterns, recursions, binary tree questions, etc.), as well as questions specific to your background
- You'll need to think of an **efficient, optimized and bug-free solution** to code up quickly and concisely in whatever language you code best in

How to Prepare and Helpful Tips

- Review the Interview Flow Chart on the following page
- Click here to practice (FB Code Lab)
- We highly encourage you to watch the following videos (**password**: FB_IPS)

  Cracking the Facebook Coding Interview – The Approach

  Cracking the Facebook Coding Interview – Problem Walkthrough
- Take hints from your interviewer to showcase your thought process
- Avoid solutions with a lot of edge cases or huge if/else if/else blocks
- Think about different algorithms and algorithmic techniques (sorting, divide-and-conquer, dynamic programming/memorization recursion)
- Review data structures, particularly the ones used most often (Array, Stack/Queue, Hashset/Hashmap/Hashtable/Dictionary, Tree/Binary Tree, Heap, Graph, Bloom Filter etc.)
- Don't worry about rote memorization such as runtimes or API/native calls; it's good to know how to figure out approximate runtimes on the fly but the code you write is more important
- You will be asked about memory constraints on the complexity of the algorithm you are writing (and its running time - O(N^2) to O(N) etc.)

Typical Questions Asked

- Imagine it is your first day here at Facebook. What do you want to work on? What features would you improve on?
- Why are you interested in Facebook? (Know about our technical environment, projects, challenges, etc. and be excited!)
- What are the most interesting projects you have worked on and how might they be relevant to the Facebook environment?
- What are some of the biggest professional challenges you have faced and how you have overcome them?

CRACKING THE CODING SKILLS

Created By Gayle Laakmann McDowell

Best Conceivable Runtime (BCR)

BCR is the runtime you *know* you can't beat. For example, if asked to compute the intersection of two sets, you know you can't beat O(|A|+|B|).

5 Approaches

- **BUD:** Look for bottlenecks, unnecessary work, duplicated work.
- **DIY:** Do It Yourself
- **Simplify & Generalize:** Solve a simpler version.
- **Base Case & Build:** Solve for the base cases then build from there.
- **Data Structure Brainstorm:** Try various data structures.

1 Listen

Pay very close attention to any info in the problem description. You probably need it all for an optimal algorithm.

2 Example

Most examples are too small or are special cases. **Debug your example.** Is there any way it's a special case? Is it big enough?

BUD Optimization

Bottlenecks

Unnecessary Work

Duplicated Work

3 Brute Force

Get a brute-force solution as soon as possible. Don't worry about developing an efficient algorithm yet. State a naive algorithm and its runtime, then optimize from there. Don't code yet though!

4 Optimize

Walk through your brute force with **BUD optimization** or try some of these ideas:

- Look for any unused info. You usually need all the information in a problem.
- Solve it manually on an example, then reverse engineer your thought process. How did you solve it?
- Solve it "incorrectly" and then think about why the algorithm fails. Can you fix those issues?
- Make a time vs. space tradeoff. Hash tables are especially useful!

5 Walk Through

Now that you have an optimal solution, **walk through your approach in detail**. Make sure you understand each detail before you start coding.

6 Implement

Your goal is to **write beautiful code**. Modularize your code from the beginning and refactor to clean up anything that isn't beautiful.

7 Test

Test in this order:

1. Conceptual test. Walk through your code like you would for a detailed code review.
2. Unusual or non-standard code.
3. Hot spots, like arithmetic and null nodes.
4. Small test cases. It's much faster than a big test case and just as effective.
5. Special cases and edge cases.

And when you find bugs, **fix them carefully!**

What You Need To Know

1. **Data Structures:** Hash Tables, Linked Lists, Stacks, Queues, Trees, Tries, Graphs, Vectors, Heaps.
2. **Algorithms:** Quick Sort, Merge Sort, Binary Search, Breadth-First Search, Depth-First Search.
3. **Concepts:** Big-O Time, Big-O Space, Recursion & Memoization, Probability, Bit Manipulation.

Exercises:

- Implement data structures & algorithms from scratch.
- Prove to yourself the runtime of the major algorithms.

Books by Gayle

Do not...

- **Do not** ignore information given. Info is there for a reason.
- **Do not** try to solve problems in your head. Use an example!
- **Do not** push through code when confused. Stop and think!
- **Do not** dive into code without interviewer "sign off."

What We Assess

You can answer the questions in the language that you feel most comfortable with. However, if you are experienced with a multitude of languages, it might be best to code in the one that provides the most optimal/speedy solution to the problem. It's also important that you think out loud/provide a narrative as you go through the problem so the engineer has insight into your thought process.

We assess the coding portion based on a few things:

1. Communication - Does the person ask clarifying questions, or do they just dive in to the code? Asking clarifying questions will
help the interviewer understand your problem solving skills, and will also help you create a better solution. Interviewers comment on this frequently.

2. Speed – Meaning speed of thought and the ability to clearly explain your thought. Can the engineer we're assessing move fast?
It is very important that you practice coding quickly, as we see interviewers commenting on this frequently as well.

3. Quality – Optimized functional working code. Is the code clean/bug free? If not, it’s okay to fix bugs and optimize the solution,
but this may affect your ability to solve more than one coding problem within 45 minutes.
Concepts to know:

CS Fundamentals to Review

Before the interview, we strongly encourage you to have a full understanding of the following techniques (although not limited to) that can be applied to many different kinds of questions and also can assist in solving the problems in the most speedy and efficient manner: algorithms, design patterns, data structures, recursions, binary tree questions.

Also, it's important to reacquaint yourself to core computer science concepts & general software engineering skills (study the basics) and topics related to the scale of our environment.

Algorithms to study:

- Merge Sort
- Quick Sort
- Breadth-first search
- Depth-first search
- Binary Search

Data Structures to study:

- Arrays/ArrayLists
- Hash Tables
- Trees
- Graphs
- Stacks/Queues
- Heaps

Concepts to know:

- Big O
- Recursion
- Memoization/Dynamic Programming

Useful Links

How to Crush Your Coding Interview

Coding Practice

Interview Advice from a Facebook Engineer

What to Expect During the Recruiting Process

Facebook Engineering Page

Engineering Blog

Research Publications

Open Source at Facebook

Latest Facebook News