

Welcome!



WWCode San Francisco - Backend Study Group

February 16, 2023

- We'll start in a moment :)
- We are **RECORDING** tonight's event
- We may plan to take screenshots for social media
- If you are comfortable, turn the video ON. If you want to be anonymous, then turn the video off
- We'll introduce the hosts & make some time for Q&A at the end of the presentation
- Feel free to take notes
- Online event best practices:
 - Don't multitask. Distractions reduce your ability to remember concepts
 - Mute yourself when you aren't talking
 - We want the session to be interactive
 - Use the 'Raise Hand' feature to ask questions
- **By attending our events, you agree to comply with our [Code of Conduct](#)**

Introduction & Agenda

- Welcome from WWCode!
- Our mission: Empower diverse women to excel in technology careers
- Our vision: A tech industry where diverse women and historically excluded people thrive at any level
- About Backend Study Group



Harini Rajendran

Presenter
Senior Software Engineer,
Confluent
Lead, WWCode SF



Prachi Shah

Host
Senior Software Engineer, Unity
Director, WWCode SF

- **Coding Interview 101**
 - **The What?**
 - **The Why?**
 - **Before**
 - **During**
 - **Housekeeping**
 - **Evaluation**
 - **Q & A**

The What?

- Technical conversation between 2 engineers
- Solving a programming problem together
- Typically 45 - 60 mins
 - Waste time getting Zoom to work (if virtual) or settling in(onsite) (5 min)
 - Introductions (5 min)
 - Question time (40-45 min)
 - Reverse question time (5 min)
- Usually done virtually (phone screen/virtual onsite) or on a whiteboard (onsite)
- You are expected to write (working) code

The Why?

Interviews are a 2 way street

- You are evaluating them as much as they are evaluating you
- They are selling themselves as much as you are selling you
- They want the not so great candidates out and great candidates in
- Important aspects of evaluation
 - Communication Skills
 - Problem Solving
 - Technical Competency
 - Testing
 - Team Player (Soft skills)
- Once they are convinced you are great, they will do their best to make sure you want to work with them because
 - Hiring is time consuming and expensive
 - Great products are built by great people
 - A's hire A's, B's hire C's

Before

1. Picking the programming language

- Companies mostly don't care
- Higher level languages that have many standard library functions and data structures
- Common options: **Python**, Java, C++, Javascript
- Don't pick low-level or not so famous languages like C, Golang, Perl, Ruby, Shell script, etc
- ^ might not be applicable if your position **REQUIRES** the knowledge of a specific programming language. Eg: iOS developer-Objective C

2. Go to the basics. Revisit your CS101

- Revisit all CS fundamentals
- OS, DBMS, Computer Architecture, etc etc
- If Masters, then revisit your speciality areas - ML, Cloud Computing, Big data, etc

3. Focus on basic Data Structure and Algorithms

- Arrays, LL, Stacks, Queue, Trees, Graphs, Maps, Heap
- Tree traversals, sorting, searching, DP
- Focus on space and time complexities
- Book that I found useful: [Elements of Programming Interviews in Java](#)
- Academic book that I have referred to in the past: [Introduction to Algorithms](#)
- Website that I extensively used: [GeeksforGeeks](#)
- WWCode Backend Study Group Data Structures 101 session [slide deck](#)

Before

4. Mastery through practice

- Practice practice practice
- Solve ds and algos problems for at least 1.5 hrs everyday for about a month
- Start from easy topics like Arrays, Linked List and progress to complex ones
- Leetcode, HackerRank, etc
- Always understand the time and space complexity of your solution
- 30 hrs at minimum and maybe upto 100 hrs. After that ROI might be less
- Found [this](#) very useful

5. Time yourself

- While practicing, track time
- First 5 mins to read and understand the question and get clarity
- Next 10 mins to finalize the approach
- Next 20 mins to code the solution
- Last 10 to 15 mins for testing, code walkthrough, time and space complexity analysis, etc

6. Build a cheatsheet with must-dos and must-remembers

- As you learn and practice, build a cheatsheet for yourself
- Time and complexities, some tips and tricks, common mistakes you make, some interesting approaches, etc

7. Mock coding interviews

- Peer mocks: Take help from friends, pramp
- Expert mocks: interviewing.io

Before

8. Prepare good self introduction

- Maintain a positive energy
- Short, direct, **enthusiastic**
- Elevator pitch style - keep it under 2 mins (Otherwise you will have less time for coding :p)
- KISS (Keep it Simple and Sweet). Don't dive in-depth into any projects
- Practice practice practice

9. Be prepared to dive deep on 1 project

- Recent project that was technically challenging
- High level architecture
- Which pieces you owned?
- What were some of the challenges?
- What was the impact?
- What did you learn from it?

10. Prepare good final questions

- Usually not used as a part of evaluation
- Company specific
- Team specific
- Tech stack
- Culture, work life balance, technical challenges, etc
- **DO NOT** ask about personal stuff, compensation, visa, immigration policies, etc.

During

1. Start with a concise self introduction
2. Ask clarifying questions
 - Underspecified and vague on purpose
 - Attention to details
 - Paraphrase the question
 - Clarify your assumptions, input ranges, format, etc
3. Don't jump into coding right away
 - Usually considered a red flag
 - Think about the solution and explain high-level approach in words
 - Consider various approaches and talk through them, listen for hints if stuck
4. Start with brute force approach and optimize with the interviewer
 - Its ok to start with brute force approach
 - Can we do better? Calls for more optimal solution
 - Caching, choosing better data structures, better algorithms, divide and conquer, etc etc
 - You can use pseudo code while discussing this

During

5. Code out the solution while talking out loud
 - Don't code in silence
 - Explain what you are trying to achieve at a high level while typing
 - Meaningful variable names (No single letter variable names like x,y,z)
 - Wherever you make a choice between 2 or more options, discuss trade-offs proactively
6. Hints
 - If you are stuck anywhere, listen for subtle hints
 - Use them to move in the right direction
 - It's okay to ask for hints but not too many
7. Break the problem into small chunks
 - Start from high-level function and keep breaking into small helper functions
 - Exhibits your structured thought process
 - Refactor if too many copy, paste chunks
8. Clean code
 - Keep your code modular
 - Define appropriate class, structs, etc as needed
 - Don't write monolithic code
 - No noodles code
 - No if-else ladders as much as possible (Consider switch case instead)

During

9. Don't bluff

- If you are unfamiliar with something, accept
- Do not pretend
- It's okay to say "I don't know" rather than making things up
- If you are unfamiliar with some libraries, you can ask to look it up (Some interviewers will allow it)

10. Pace yourself

- Timing is important
- Code at decent pace
- Target to wrap up coding by the end of 45 minutes so that you have time for testing and discussions
- If you get stuck, pause for 30-60 seconds to regroup your thoughts

11. Once you are done coding

- Immediately do not declare you are done
- Scan through the code for mistakes
- Edge cases
- Look for opportunities for refactoring and call them out
- Walkthrough the code with basic simple example
- If online, then compile and run the code and fix any errors

During

12. Time and space complexities

- Explain your code's time and space complexities
- Highlight various chunks of code and associate the complexity
- If there are trade-offs or alternatives, talk about them

13. Test cases

- Start with simple, straight forwards ones
- Positive and negative test cases
- Corner/Edge cases
- Test cases should have good code coverage


14. Fix your code

- Its okay if test cases fail
- Debug, find the issue and fix it
- Opportunity to showcase your debugging skills
- Showcasing your ability to learn from mistakes

15. End on a good note

- Ask interesting questions about the company/work/culture, etc
- Thank them for their time and the experience

Housekeeping

- Get to know the company
 - Research their products
 - Press release, blogs, linkedin, etc
 - Company specific interview questions online
 - Read reviews about the company online and see if they align with what you want (Take them with a pinch of salt)
- Get comfortable with the tools
 - If virtual, get used to the collaborative coding platform like Coderpad, etc
 - You can check with HR what they use in the interviews
 - This will be handy especially in design rounds
 - If in-person onsite, practise on whiteboard (Get a small whiteboard at home and use them during prep)
- Do not ignore the behavioural aspect
 - Communication is key
 - Maintain a positive attitude through the interview
 - Even if you are stuck, try to keep your cool 
- Warm up before the interview
 - Set up your laptop, check internet, zoom, etc
 - Keep some water next to you
 - Practice your intro if needed

Housekeeping

- Don't prepare for too long
 - Over-indexing might not be useful
 - When you know everything, you might naturally jump on to the most optimal solution (which in some cases are considered a red flag)
 - 1 to 3 months of prep time
- Rank companies and apply
 - Split companies under 3 categories
 - Not very interested in joining
 - Might consider joining
 - Dream company
 - Do not apply to dream companies first
 - Interviewing takes time and practice
 - Leave some room for failure and rejections
- Go via referral if possible
- Practice, practice, practice!!!

Evaluation (Insider scoop)

Things you are evaluated on

1. Communication - Does the candidate make clarifications, communicate their approach and explain while coding?
2. Problem Solving - Does the candidate show they understand the problem and are able to come up with a sound approach, conduct trade-offs analysis and optimize their approach?
3. Technical Competency - How fast and accurate is the implementation? Were there syntax errors?
 - a. Choice of Data Structures with time and space complexities
 - b. Knowledge in Programming language used
 - c. Clean code
4. Testing - Was the code tested for common and corner cases? Did they self-correct bugs?
5. Team Player (Soft skills) - Can they work in a team? Were they collaborative? Open to hints and feedback?

Common rating scale

- Strong hire
- Hire
- No Hire
- Strong no hire
- Unsure (This doesn't happen very often. When it does, they usually schedule more rounds or ignore that round if there is substantial feedback from other rounds)

Evaluation (Insider scoop)

Final decision

- Hiring committee
- Interview Debrief [All interviewers will get together and talk about you 😊]
- Combination of both

Backend Study Group

References:

- <https://igotanooffer.com/blogs/tech/coding-interview-tips>
- <https://www.techinterviewhandbook.org/coding-interview-cheatsheet/>
- <https://www.freecodecamp.org/news/coding-interviews-for-dummies-5e048933b82b/>
- <https://www.interviewcake.com/coding-interview-tips>
- [WWCode Data Structures 101 session](#)

Backend Study Group:

- [Presentations](#) on GitHub and session recordings available on [WWCode YouTube channel](#)
- Upcoming sessions:
 - Feb 23rd, 2023 - [System Design Series: Part 2](#)
 - Mar 1st, 2023 - [Careers in Cyber Security](#)
 - Mar 16th, 2023 - [System Design Series: Part 3](#)

Women Who Code:

- [Technical Tracks](#) and [Digital Events](#) for more events
- Join the [Digital mailing list](#) for updates about WWCode
- Contacts us at: contact@womenwhocode.com
- Join our [Slack](#) workspace and join `#backend-study-group!`

You can unmute and talk or use the chat

