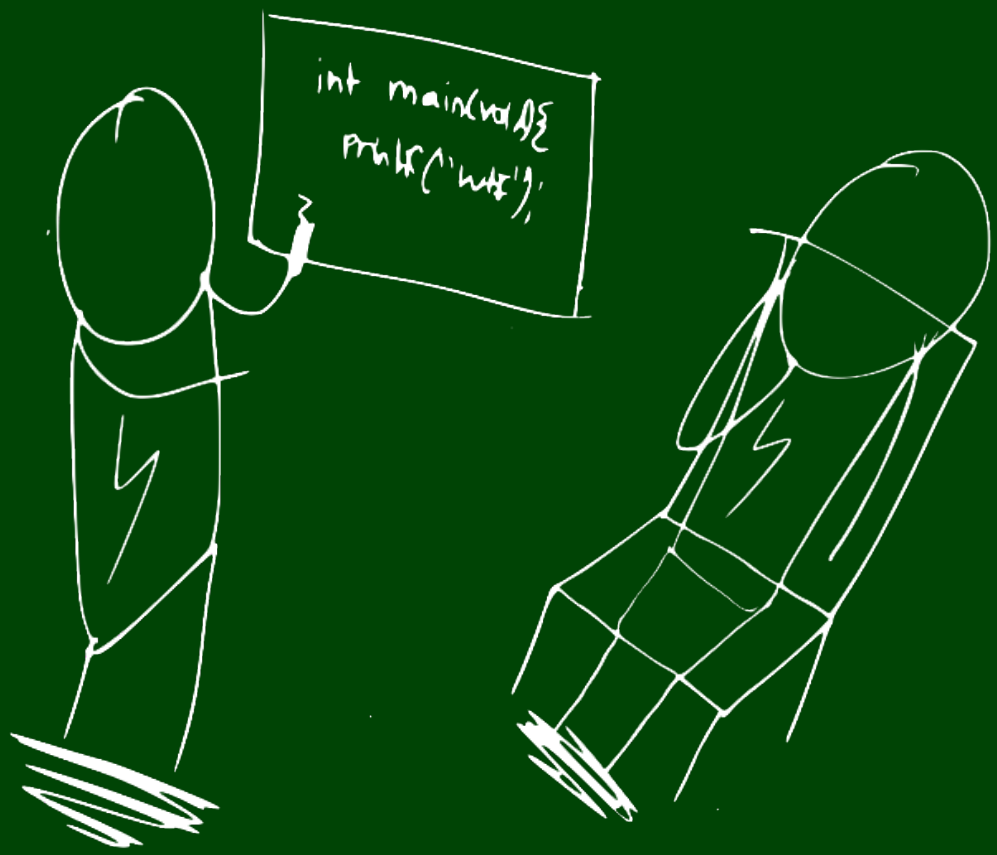
CS 3

Introduction to Software Design

Interviews



1. Recruiter/resume screen

1. Recruiter/resume screen

2. Preliminary interview(s)
   - Non-technical phone screen
   - 1-2 technical phone interviews
   - 1-2 technical on-campus interview
   - 1-2 online programming challenges

1. Recruiter/resume screen

2. Preliminary interview(s)
   - Non-technical phone screen
   - 1-2 technical phone interviews
   - 1-2 technical on-campus interview
   - 1-2 online programming challenges

3. On-site interview day(s)
   - 2-7 in-person technical interviews

Overview

1. Recruiter/resume screen
2. Preliminary interview(s)
   - Non-technical phone screen
   - 1-2 technical phone interviews
   - 1-2 technical on-campus interview
   - 1-2 online programming challenges
3. On-site interview day(s)
   - 2-7 in-person technical interviews
4. Follow-up technical phone/video interview(s)

Outline

- Always remember: they are there to recruit you!
- At job fairs, it's usually engineers–these are your peers!
  - Otherwise, they are often non-technical
- The process is **people** oriented
- Be positive!

- Always remember: they are there to recruit you!
- At job fairs, it's usually engineers–these are your peers!
  - Otherwise, they are often non-technical
- The process is **people** oriented
- Be positive!
- Be a reasonable person...

The Five Areas (Steve Yegge)

1. **Coding.** The candidate has to write some simple code, with correct syntax, in C, C++, or Java.
2. **OO design.** The candidate has to define basic OO concepts, and come up with classes to model a simple problem.
3. **Scripting and regexes.** The candidate has to describe how to find the phone numbers in 50,000 HTML pages.
4. **Data structures.** The candidate has to demonstrate basic knowledge of the most common data structures.
5. **Bits and bytes.** The candidate has to answer simple questions about bits, bytes, and binary numbers.

The Five Areas (Steve Yegge)

1. **Coding.** The candidate has to write some simple code, with correct syntax, in C, C++, or Java.
2. **OO design.** The candidate has to define basic OO concepts, and come up with classes to model a simple problem.
3. **Scripting and regexes.** The candidate has to describe how to find the phone numbers in 50,000 HTML pages.
4. **Data structures.** The candidate has to demonstrate basic knowledge of the most common data structures.
5. **Bits and bytes.** The candidate has to answer simple questions about bits, bytes, and binary numbers.

What I'm looking for here is a total vacuum in one of these areas. It's OK if they struggle a little and then figure it out. It's OK if they need some minor hints or prompting. I don't mind if they're rusty or slow. What you're looking for is candidates who are utterly clueless, or horribly confused, about the area in question. –Steve Yegge

Their Questions

- Tell me about a time when you...
  - collaborated by dividing tasks, delegated, pair programmed
  - had a conflict with a teammate
  - couldn't solve a homework problem
- Tell me about the hardest bug you've ever had
- Tell me about a project you're working on
- Why do you want to work here?

What To Expect: Behavioral Questions

Their Questions

- Tell me about a time when you...
  - collaborated by dividing tasks, delegated, pair programmed
  - had a conflict with a teammate
  - couldn't solve a homework problem
- Tell me about the hardest bug you've ever had
- Tell me about a project you're working on
- Why do you want to work here?

Your Questions

- What's the most exciting part of your job?
- What's the most interesting problem you've solved at X?
- ...anything that demonstrates you actually care about the company
- ...anything else you can think of!

Code-wise, I personally am mainly looking for: (1) can you come up with a reasonable solution (and identify unreasonable solutions), and (2) can you turn your algorithm into code. –Microsoft Engineer

Code-wise, I personally am mainly looking for: (1) can you come up with a reasonable solution (and identify unreasonable solutions), and (2) can you turn your algorithm into code. –Microsoft Engineer

Focus on doing things that convey "positive signal", and less so on getting the "right solution". E.g. write moderately well-refactored code from the start, talk through your plan out loud, think about edge cases and testing your code... –Dropbox Engineer

Code-wise, I personally am mainly looking for: (1) can you come up with a reasonable solution (and identify unreasonable solutions), and (2) can you turn your algorithm into code. –Microsoft Engineer

Focus on doing things that convey "positive signal", and less so on getting the "right solution". E.g. write moderately well-refactored code from the start, talk through your plan out loud, think about edge cases and testing your code... –Dropbox Engineer

Themes

- Interviewers care about bugs in your code
- Interviewers care about your design
- Interviewers like "iterative questions"
- Interviewers expect you to generate test cases and run through them line-by-line

Essential Topics To Study

- DFS/BFS
- Binary Search
- Standard Data Structures: arrays, stacks, queues, trees, hash tables, graphs
- Sorts: merge sort, quick sort
- Recursion
- Dynamic Programming

Outline

Approaching Behavioral Questions

- An interviewer who doesn't like you is a wasted opportunity
- Your attitude **matters!**
- Don't spew "canned" responses
- Be excited both when you're talking and when they're talking
- Take break time to "re-center" if you're shaken

- Talk through **everything**!
- Ask for clarification on edge cases
- Do not discount the "trivial" brute force solution
- When the interviewer gives you a hint. . . do not ignore it
- Iterate on your solution
- If you've seen the problem, say so. But expect to code it anyway.
- Simple errors are usually forgivable
- Run **every line** of your code through your test cases

- Talk through **everything**!
- Ask for clarification on edge cases
- Do not discount the "trivial" brute force solution
- When the interviewer gives you a hint. . . do not ignore it
- Iterate on your solution
- If you've seen the problem, say so. But expect to code it anyway.
- Simple errors are usually forgivable
- Run **every line** of your code through your test cases
- . . . Keep Talking . . .

Outline

Let me tell you a story. . .

- Get used to coding without a computer

- Get used to coding without a computer
- The interview is **timed**; do not underestimate this!

- Get used to coding without a computer
- The interview is **timed**; do not underestimate this!
- Do not do your first few interviews at places you care about

Generic Advice

- Get used to coding without a computer
- The interview is **timed**; do not underestimate this!
- Do not do your first few interviews at places you care about
- Be reasonable

Google Criteria

- Does the candidate know how to use data structures and algorithms?
- Is the candidate's code well designed?
- Does the candidate write idiomatic code?
- Does the candidate's code have non-trivial bugs?
- Google explicitly does not evaluate the questions you ask.

Google Criteria

- Does the candidate know how to use data structures and algorithms?
- Is the candidate's code well designed?
- Does the candidate write idiomatic code?
- Does the candidate's code have non-trivial bugs?
- Google explicitly does not evaluate the questions you ask.

Google Process

- Campus Interview / Phone Screens
- On-site Interviews with "effectively random" engineers
- Engineers submit feedback to recruiter
- "Hiring committee" collates feedback and makes decision
- "Executive committee" reviews decision
- You get hired!

Microsoft Criteria

- Adaptability
- Collaboration
- Customer Focus
- Drive for Results

Microsoft Criteria

- Adaptability
- Collaboration
- Customer Focus
- Drive for Results

Microsoft Process

- Campus Interview / Phone Screens
- On-site Interview with **members of the team you'd be working on**
- Your interviews are usually with multiple teams that have been pre-chosen by Microsoft
- The team decides very quickly

Dropbox looks for **combining** data structures/algorithms with systems knowledge. It's not enough to know one or the other.

Example Questions

- Design a database schema for X

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X
- Implement a regular expression matcher

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X
- Implement a regular expression matcher
- Determine if a linked list has a cycle

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X
- Implement a regular expression matcher
- Determine if a linked list has a cycle
- Determine if two strings are anagrams of each other

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X
- Implement a regular expression matcher
- Determine if a linked list has a cycle
- Determine if two strings are anagrams of each other
- Determine if an element exists in an unbounded stream with unknown size in $\mathcal{O}(\lg n)$ time

Example Questions

- Design a database schema for X
- Design an OOP hierarchy for X
- Implement a regular expression matcher
- Determine if a linked list has a cycle
- Determine if two strings are anagrams of each other
- Determine if an element exists in an unbounded stream with unknown size in $\mathcal{O}(\lg n)$ time
- Implement semaphores

Outline

So . . . , you didn't get it

Companies prioritize NOT making a **bad** hire
over making a **good** hire!