Technical Interviews

Presented by Michael & Brad

A poorly lit technical interview

Types of interview questions

- Most interviews in the tech industry tend to stray away from traditional interview questions (Tell me about your strengths, where do you see yourself in 5 years, why do you want to work here, etc)
- Instead, they focus on two types of interview questions:
  - Behavioral: Questions about your past work and personal experiences and how you handled challenging situations.
  - Technical: Questions about your specific stack, general comp-sci questions, or abstract problem solving questions.

Important Advice

- The most important thing to remember: **These questions are designed so that you wont see the solution immediately.** This is the scary part. You will probably want to panic and think you will never ever figure out the solution.
- You have to fight through that feeling and approach the problem just like any other problem you have faced while programming.
- Often times you just need a place to start, and once you really begin to focus on the problem and get over the nerves, the solution will begin to show itself.

Avoid Silences

- If you are stumped or shocked by the question at first, counter the question with questions of your own. Ask for more details or clarification. Ask if there are any edge cases that you need to account for. A lot of the time their answers will greatly benefit you.
- The person interviewing you is usually from the dev team you will be working on if you get the job. They want to see how you think. So talk through the problem as you work through it. Don’t just silently write or type things.

Be Resourceful

- During a technical interview, you will always have either a computer, a pad of paper, and/or a whiteboard at your disposal. So use them!
- Most interviewers will make it clear if you are allowed to consult google or the documentation before the interview begins. If they don't and you need those resource, ask if its okay.
- Once you start writing code on your computer, or writing pseudo code on the paper or whiteboard, it will help put you in problem solving mode. You will start thinking, okay what properties do I need? what methods? what kind of objects should I use? These are the things the interviews want to see.

Pseudo Code

- Pseudo code allows you to write out and describe your algorithm without having to worry about the exact syntax of a language.
- Once you have written out your algorithm in pseudo code, it should be very easy to translate into working code. A trick is to pretend that someone else is going to use your pseudo code as instructions to write the code.
- Structurally, it should be very similar to how you would write the real code. Specifically, indent things when they would normally be indented (like in conditionals and for loops.)
- Writing pseudo code will help transition your mind into problem solving/coding mode.

Pseudo Code Examples

Pseudocode Editor

Edit Tools

```
If within boundaries of search
      Calculate midpoint
      Get value at midpoint as integer
      If value at midpoint is what we are looking for
            Return midpoint
      Else if value at midpoint is too big
            Look to the left
      Else if value at midpoint is too small
            Look to the right
      ENDIF
ENDIF
Return -1 when number wasn't found
```

Algorithm

```
for i = 1:n,
     swapped = false
     for j = n:i+1,
          if a[j] < a[j-1],
               swap a[j,j-1]
               swapped = true
     → invariant: a[1..i] in final
     break if not swapped
end
```

PSEUDOCODE

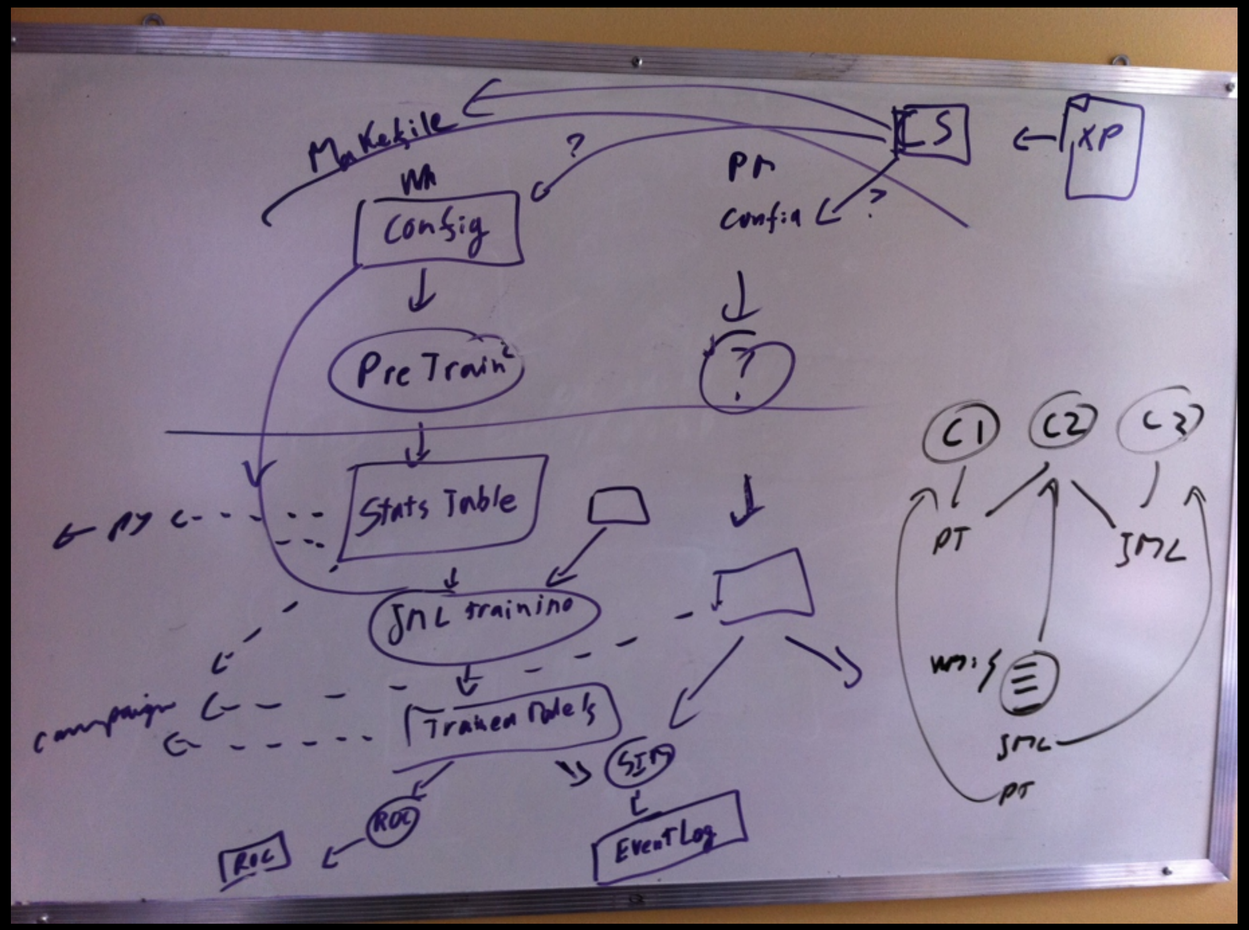
```
set total to zero

get list of numbers

loop through each number in the list
   add each number to total
end loop

if number more than zero
   print "it's positive" message
else
   print "it's zero or less" message
end if
```

PsueN0 code



- Symbols can be repeated
  - Not V, L, D (just I, X, C, M)
  - No more than 3 occurrences
- Symbol followed by ≤ same
- Otherwise, subtractive
  - Not V, L, D (only I, X, C, M)
  - Only two symbols

Phone Screening

- You will usually go through a technical phone interview before being invited for an in person interview.
- This will either be completely over the phone, or another dev will watch your screen through a screen sharing app while you work through a few problems.
- Sometimes they will have you logon to a proprietary online coding website, and you will be forced to code in a web language you don't know. If its a language you have never seen or used before, ask plenty of questions about syntax and structure. They may be doing this on purpose to throw you off your game.

Types of technical questions

- Stack Specific iOS questions (yay!)
- Abstract object oriented questions (yay!)
- Data structures and algorithms (oh crap)

Stack Specific questions

- Meant to test your knowledge and experience with the language and frameworks used in the job you are applying for.
- This is the most common type of questions our students have received.
- Can be either an abstract pattern question (MVC, Delegation,etc), or they will have you build a feature in Xcode.

iOS Examples

- What is a delegation pattern used for?
- Explain MVC
- What is a protocol used for?
- Explain categories/extensions
- How would you implement Core Data in an app?
- How do you get multiple view controllers on one screen?
- Implement a collection view.
- How do block/closures work and what are they good for?
- How would you go about accessing web services/API’s asynchronously?
- ARC

Abstract problem solving

- These questions are meant to test your overall problem solving skills and your ability to think on the fly.
- They are purposely weird to see how you squirm under uncomfortable circumstances.
- Ignore the strangeness and focus on how you would solve this problem as if it were your actual job.

Abstract Examples

- Build me an elevator
- How many windows are there in seattle?
- You're shrunken to the size of a quarter and you are inside a blender that is about to turn on. What do you do?

Data structure and algorithm questions

- Meant to test your fundamental computer science knowledge and your ability to find solutions under pressure.
- 99% of the time the solution will not come to you immediately.
- It isn’t a disaster if you don't find the correct solution, as long as you don’t give up and you are trying things that make sense.
- Always be thinking of big O when doing these types of questions. How efficient is my solution? Can I improve the efficiency in some way?

Data structure and algorithm examples

- Write out 0 through 10 in binary, base 3, and base -3.
- Create a function to find the angle between the hands of a clock
- Write a function that takes two ordered arrays and combines into one ordered array.
- How do you remove duplicates from an array?
- Implement an array from scratch using TDD
- What data structure would you use to add two polynomials together?
- how would you add two polynomials?