Interviewing

I ASK ALL PROSPECTIVE EMPLOYEES THIS QUESTION TO TEST THEIR REASONING.
www.dilbert.com scottadams@aol.com
YOU HAVE ONE FOX AND TWO CHICKENS THAT YOU NEED TO GET ACROSS A RIVER. YOU CAN ONLY TAKE ONE AT A TIME IN THE ROW-BOAT. THE FOX WILL EAT THE CHICKENS IF LEFT ALONE.
11-27-03 © 2003 United Feature Syndicate, Inc.
I'D BUY LIVESTOCK INSURANCE, THEN BARBECUE THE CHICKENS AND BLAME THE FOX.
CAN YOU START TODAY?

CSE 403

"… in my old age I treat Dilbert less as farce and more as documentary." – Patrick McKenzie

For and from you

- Those who have done this before—Think about what you want to add, because I’ll be asking throughout.
- Others—You’re going to be doing this soon, so pay attention!

The Process

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/Skype interview(s)

Timeline: 3 weeks to 3 months

Any other processes?

Studying for Technical Questions

- DO Practice real questions
  - Strongly prefer breadth of knowledge to depth
  - Do the questions—don't just read the answers
  - Read interview books and websites
  - Share with friends (but beware NDAs)
- DO use paper or a whiteboard, not a computer
- DO switch off with a friend being "interviewer" and "interviewee"

Interviewers look here, too!

Studying for Technical Questions

- DO know the obvious topics
  - Depth-first and bread-first traversals of DAGs
  - Implementation/operations/traversals/running times for hash tables, binary (search) trees, arrays, singly-linked lists, and heaps
  - Quicksort, merge sort
- DO pick a good language and know it well
  - Most companies let you pick any language (tip: try Python)
  - Use language features that make things easy (e.g., list slices and generators in Python)

Studying for Technical Questions

- DON’T learn the complex data structures
  - Your interviewers haven’t written an AVL tree since college, if ever
  - Most questions feature traversals of arrays/strings, singly-linked lists, grids (two-dimensional arrays), and DAGs
- DON’T trust the recruiter to tell you about questions and topics to expect
  - Each interviewer selects their own questions

Studying for Technical Questions

- DO learn the patterns in solutions. Examples?
  - Dynamic programming on 2**n solutions
  - Slow-pointer/fast-pointer traversal of linked lists
  - Heaps are common
  - Range constraints (last five minutes, ages) often imply easy constant-space solutions
  - To determine if two strings are anagrams, sort their characters

Chat for a couple minutes with the people around you!

Studying for Technical Questions

- Other DOs or DON’Ts for preparing?

Studying for Non-Technical Questions

- Practice nugget-first/situation-action-result
  - How did you lower costs or increase profits?
- Focus on recent experience
- Prepare for typical questions. Examples?
  - Tell me about a project you're working on.
  - Tell me about a recent programming challenge you faced.
  - Why do you want to work here?
  - Tell me about a recent conflict with a teammate.

This is your basic job description, so keep it in mind.

The Night Before the Interview

- Study the company
  - What do they build? What tools do they use? How do they present themselves? Organization structure?
- Study the position
  - Different companies assign different responsibilities to roles with the same title.
- Plan your trip so you can be comfortably on-time
  - Where is the building? How will you get there and back? How long will it take to get there? Who will you ask for? Dress comfortably and slightly better than their average employee.

In the Interview—Psychology

- You have to make the interviewer like you—be charismatic.
- If you're tense, it will show in your attitude and your answers, so stay calm!
  - Postpone important interviews until after you've had practice with other companies
- Some interviewers pick a question they know you can't solve just to see how far you get and how you handle the stress

Tackling a Technical Question

1. Write the problem on the whiteboard. Ask clarification questions.
2. Talk through an algorithm. No code yet!
3. Write the problem on the whiteboard. Ask clarification questions.
4. Step through at least one non-trivial test case. Fix bugs carefully and methodically.

Tackling a Technical Question

1. Write the problem on the whiteboard. Ask clarification questions.
   - If you’ve done this exact problem, say so. Be prepared to describe the solution.
   - Guarantees that you understand the question.
   - Questions: What about symbolic links in file systems? Does this maze have an exit?
     - Others?

Tackling a Technical Question

2. Talk through an algorithm. No code yet!
   - Always mention obvious-but-inefficient solutions. They’re great fallbacks, and show that you *can* solve the problem.
   - You’re never totally stuck. You can always solve at least part of the problem, so focus on that!

No matter what, **stay positive**. Laugh about your confusion!

Tackling a Technical Question

3. Write code at a moderate pace (it will feel slow).
   - It’s okay to forget some syntax or an API—just say so.
   - Use good decomposition: “Gee, I wish I had a function that …”. (Don’t implement helpers yet, and only if the interviewer wants you to!)

It’s worth saying again: No matter what, **stay positive**. Laugh it off!

Tackling a Technical Question

4. Step through at least one non-trivial test case. Fix bugs carefully and methodically.
   - Don't be careless—make sure to completely understand the source of the problem before trying to fix it.

I said this already, but no matter what, **stay positive**. Laugh off the mistakes!

Tackling a Technical Question

- What has worked for you in interviews?

Now it’s your turn

- Demonstrate your insight and passion
  - How do you overcome problem X given your problem Y (scale, distributed systems, tools, deployment, etc.)
  - I’m interested in learning X. Did you come to this company with a background in X already, or are there opportunities to learn it?

Now it's your turn

- Evaluate whether you want to work there. Good questions to ask?
  - Can you imagine getting along with your interviewer?
  - Is there opportunity for advancement and movement between projects?
  - How much time would you spend in meetings per week? Coding per day?
  - What is the ratio of developers to testers to product managers?

After the Interview

- Relax—there’s no point worrying and you cannot accurately judge your performance.
- If you haven’t heard anything in a week, you can send a polite email.
- You can ask them to hurry up or give you more time to align with other companies’ schedules. Be polite!

After the Interview

- You got the job
  - Negotiate. It’s expected! Remember—a lot of money to you is peanuts to them.
- Or you didn’t
  - Don’t take it personally.
  - Companies encourage you to “try, try, try again”.

Bibliography and Other Resources

- *Cracking the Coding Interview* (Gayle Laakmann)
- *Programming Interviews Exposed* (Mongan, Giguere, and Kindler)
- *Elements of Programming Interviews* (Aziz, Prakash, and Lee)
- "Don't Call Yourself a Programmer". Blog post. Patrick McKenzie (Kalzumeus Software).
- "How to Get a Job at Google". Thomas L. Friedman (New York Times).
- "UW CSE Recruiting Policy for Employers". UW CSE.