Intro to Technical Interviews

CS-HU 390

Nilab & Amit

Recap of Last Week

- Resumes
- Cover letters
- Networking
- Elevator Pitch

Technical Interviews Truths

- Talented and smart candidates can fail interviews
- It is okay to ask lots of questions
- It is okay to struggle with questions and accept hints
- The interviewer is on your side

What the Interviewer is Looking for

- Can this candidate :
    - Communicate
    - Solve problems
    - Learn & Teach
    - Strong CS fundamentals
    - Express their solution through code
    - Ask questions instead of making assumptions

- Can this candidate keep up with the demands of a developer position?

What the Interviewer is NOT Expecting from you:

- That you know all the answers
- That you solve the problem immediately
- That you code perfectly

What You Need to Know

Data Structures: Hash Tables, Linked Lists, Stacks, Queues, Trees, Tries, Graphs, Vectors, Heaps.

Algorithms: Quick Sort, Merge Sort, Binary Search, Breadth-First Search, Depth-First Search.

Concepts: Big-O Time, Big-O Space, Recursion & Memoization, Probability, Bit Manipulation

Cracking the Coding Interview - The 7 Steps

1. Listen
2. Examples
3. Brute Force
4. Optimize
5. Walk Through
6. Implement
7. Test

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?

2. Example

Q: intersection size: find # of elements in common between two sorted, distinct arrays

[1, 12, 15, 19]

[2, 12, 13, 20]

*** too small**

*** same index [12]**

[1, 12, 15, 19, 20, 21]

[2, 15, 17, 19, 21, 25, 27]

*** not same size**

*** not a special case [15, 19, 21]**

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

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!

4. Optimize with BUD

B - bottlenecks

U - unnecessary work

D - Duplicate work

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, clean, and readable 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!

Examples:

15-minute solid technical phone interview:

https://www.coursera.org/lecture/cs-tech-interview/appendix-full-length-mock-phone-interview-M5aGs

Homework

- Resumes final version due date - **Sept 5th, 2019**
- Think of a time when you demonstrated **leadership**
- **Read pages 60-81** Cracking the Coding Interview 6th ed