Technical Interviews: A Survival Guide
Allen Downey

Here are some suggestions for things you can do to make a good impression in a technical interview. There are two sections: **Material** presents some topics you might want to know; **Attitude** talks about some things you can do to accentuate the positive.

Material

The E:C program at Olin is different from Computer Science programs at most schools. We think our program has many good features, but when you are preparing for a technical interview, there are two drawbacks:

1) Most E:C majors at Olin have taken fewer CS courses than CS majors at other schools.

2) Our coverage of algorithms and data structures is distributed across several classes, and de-emphasizes implementation details.

However, keep in mind:

1) Olin students take more courses in general engineering, science, design, and sometimes math than CS students at most schools, and

2) Taking more classes doesn't matter very much unless students develop the ability to apply knowledge flexibly, and (to be blunt) that is true less often at other schools than here.

That said, here is some material that sometimes comes up in technical interviews that you might want to review or, if you have not seen it in a class, learn:

Data structures

http://en.wikipedia.org/wiki/List_of_data_structure

Good ones to know: Array, Stack, Queue, Circular Buffer, Linked List, Binary Search Tree, Red-black tree, B-tree, Heap, Priority Queue, Trie, Octree, Abstract Syntax Tree, Hash Table, Graph (adjacency list and adjacency matrix representations).

For each of the standard data structures, you should know the basic operations and the order-of-growth complexity for each operation.

The Java Collections framework provides good examples of the standard data structures.

Algorithms

Searching: linear, binary, tree (including depth and breadth first).

Sorting: radix, bubble, heap, quick, merge.

Graph: minimum spanning tree, shortest path.

Software Design

You should be comfortable using UML diagrams to communicate about object-oriented design.

Low-level stuff

Bit operations: and, or, xor, shift.

How integers and floating-point numbers are represented in binary.

How characters are represented in ASCII and Unicode.

Cultural literacy

You know what is going on in the world of software. The easiest way to acquire this knowledge is to read Hacker News, Slashdot and/or Ars Technica regularly. Also read xkcd so you understand the jokes.

Attitude

Technical interviews are partially about what you know, but as important (if not more so) are

1. How you think,
2. How you communicate,
3. How you work with other people.

Don't be afraid to think out loud. Start simple and work from there. If you don't know about the specific topic you were asked about, start with something related that you know about.

For example, suppose you are asked about the pros and cons of the Befnaffle search algorithm, and you've never heard of it. You could start by saying, "Well, to evaluate any search algorithm, I would want to know its run time behavior, maybe average case as well as worst case. And I would want to know its space requirements. Then for very large data sets, I would want to know if it was amenable to out-of-core computation. Is Befnaffle search used for particular applications?"

Notice that I did two things there: (1) I demonstrated that I know *something* about search algorithms, and (2) I asked a question.

Asking questions is great because

1. It elicits more information,
2. It gives you time to think,
3. If someone tells you A, B, C, and D, and you ask about C, they assume you understood A, B and D.
4. It lets the interviewer feel smart, which makes them feel good, which makes them feel good about you.

Another general strategy for thinking out loud is “bracketing;” that is, start with upper and lower bounds, and work your way to the middle.

For example, suppose you are asked to find an optimal algorithm for a complex problem. If there is an obvious, brute-force algorithm, you could start by saying, “Well, there is an obvious brute-force algorithm...” Then explain what it is, and figure out the order-of-growth. If it’s an O(n^3) algorithm, you could start thinking about ways to improve it. If there is an obvious theoretical lower bound, you could start by saying, “Well it can’t go any faster than linear time, so let’s see if there’s a way to get there.” Now, at least, you’ve got it somewhere between O(n) and O(n^3).

Be sure to ask about the preconditions. For example, if someone asks you to search an array, ask if it is sorted. If they tell you to sort an array, ask if the size of the elements is bounded, or if there is anything else special about the inputs.

How to deal with brain teasers

Some questions are more like brain teasers -- you either get them or you don’t. Of course, it helps if you have heard it before.

1) If you have heard it before, say so. Don’t pretend to be visited by the muse of puzzlers.
2) If you have not heard it, don’t panic. Repeat the question, ask about anything that seems ambiguous, take some time to think. If the answer comes to you, great. You win.
3) If no answer comes to you, it might be because you are assuming a constraint that is not actually applicable. Think about your assumptions and ask questions like, “Am I allowed to use duct tape?,” or “I assume that I can’t cut the baby in half, right?”
4) Be a good sport. Try not to show frustration. Just do the best you can. You can ask for a hint, but don’t badger the interviewer.
5) Don’t give up too quickly, but don’t drag it on too long, either. If you don’t get it, be gracious in defeat. Smack your forehead, chuckle ruefully, and move on.

Tell the Olin story

You are different from other candidates, which means that you are better in some ways and (sorry!) less good in other ways. Emphasize the ways you are better, but be careful. You don't want to be a jerk about it. Here are some suggestions:

1) Talk about projects. Cool projects are more impressive than classes, and they give you a chance to talk about how well you handle open-ended problems, learn on your own, and work in teams. If it's appropriate, demonstrate the project. For example, if you have a portfolio of your work on your web page, maybe you can show it to the interviewer.

2) Talk about design and entrepreneurship. The foundation of the Olin curriculum is the idea that engineers need to be able to understand people, identify opportunities, formulate problems, design products and services, marshal resources, and realize solutions in the world. We use "design" and "entrepreneurship" to describe those skills, but not everyone understands those words as we use them, so you might have to practice explaining them (hint: memorize this paragraph).

3) Talk about communication and teamwork. By participating in Expo, you will have presented your work to an external audience at least 8 times, which is 8 more than students at many other schools. By participating in SCOPE, you have more experience working in teams and interacting with external clients than most other students.

4) *Demonstrate* communication and teamwork. While you are telling the interviewer how well you communicate, try to speak in complete, coherent sentences that are undecorated with "um," "you know," "basically," and "sort of." While you are talking about teamwork, demonstrate your ability to get along with people, interact comfortably, and adopt roles that are appropriate to the activity.

I hope this helps. Good luck!