How to interview for an engineering role at Dropbox (for Industry Candidates)

Hi! We are very excited you're interviewing at Dropbox, and we want to help you succeed. Maybe you're new to interviewing, or out of practice? Or maybe you're an experienced technical interviewer, but want to know more about what Dropbox looks for in particular and what we consider important? If so, this doc is for you!

We evaluate candidates in two main areas:

- team fit + engineering values, and
- software engineering, which is evaluated using a few different interview types such as coding, architecture, all-around, and deep-dive.

The sections below will help explain what we look for in each of these areas.

PART I

Team fit & Engineering Values

We want to learn as much as we can about your background and what drives you, and we know that we can't possibly hope to find out everything about you from your resume. So if you have anything (no really, anything) that you think will help us understand your abilities as an engineer, please let us know!

Some examples of things that greatly help us understand your technical abilities are

- Products you have worked on in the past
- Apps, websites, or other side projects that you built for fun
- GitHub projects or other open source contributions
- Papers that you have published
- Links to your blog or any other technical writing

Our interviewers will try to make time in every interview to chat about your achievements so that we can understand your unique strengths as an engineer. We hope you'll be excited and ready to tell us about your past work.

We also want to hear about how you go about tackling complex problems, and how you work with your team. We're a diverse and multi-talented engineering team, but we share a common set of engineering values which describe how we like to work together. Our interviewers are trying to understand in what ways you align with them. Here are our engineering values:

1. Relentlessly focus on impact. Iterate, prioritize, and build products that solve real user needs.
2. Own it. Do whatever it takes to see a project through to completion. Deliver on promises and own up to failure.
3. Pursue simplicity and sweat the details. Design and develop products anyone can use.
4. Build a culture you want to be part of ten years from now. Collaborate and help your team. Seek out feedback and provide constructive feedback to others.

Finally, please feel free to ask your interviewers any questions you have about the Dropbox product, the long-term vision, our team, and anything else that you want to know about. We really want to encourage you to ask good questions, even if they're hard! The interview is not just for us to learn about you; it's also a great opportunity for you to learn about us and gather information on how Dropbox fits you.

PART II

Technical interviewing at Dropbox

Most of your interviews will be technical and involve a coding component. In a typical 60-minute interview, you will design and code up a solution to a multi-part coding question.

While some problems test domain-specific knowledge such as architecture design or concurrency, all of our questions require a strong foundation in coding and algorithms. You should know strings, arrays, stacks, queues, linked lists, hash tables, and binary trees like the back of your hand: when to use them, how to use them, and the time/space complexities of each of their operations. You should also know what using these data structures looks like in your interview language of choice (see the “Interview mechanics” section for more on languages).

You should be prepared to implement clean, correct code, and be familiar with general algorithmic techniques like greediness, divide-and-conquer, and recursive backtracking. You should be familiar with the algorithms covered in an introductory college algorithms class, such as depth-first search and breadth-first search on a graph, searching and sorting arrays, and tree traversal.

You should be ready to convince the interviewer of both the Big(O) performance bounds and the correctness of your code. But correctness isn’t the only goal: it is also essential to demonstrate that you can write clean, well-decomposed code that other engineers could understand and build upon in a production setting. An important part of this is being able to write idiomatic code in your interview language, using commonly accepted patterns and style.

Here's a useful but non-exhaustive list of CS topics you might encounter. Note that we'll avoid asking domain-specific questions to candidates who don't have the background for it. So if you have limited background in operating systems, don't worry--we won't ask you a hard systems question.

- **Memory management** pointers, garbage collection, memory leaks, memory hierarchy
- **Concurrency** locks, semaphores, condition variables, thread pools. (Make sure you know how to use your language's concurrency primitives!)
- **Software engineering** design patterns, problem decomposition, separation of concerns, designing clean APIs
- **Advanced algorithms** memoization, dynamic programming

Technical prep materials

This section contains some reading suggestions and links that we think are helpful for preparing for coding interviews (and life in general!)

Books

The following are some classics that every coder should be aware of. There's definitely no need to read any of them cover-to-cover, but it can be valuable to sample from one or more of them if you have particular areas you want to brush up on.

- **Algorithms** *Programming Pearls* by Jon Bentley. (You can also use Wikipedia as a resource on the algorithms and data structure topics listed above.)
- **Software engineering** *Design Patterns: Elements of Reusable Object-Oriented Software* by the Gang of Four, *The Pragmatic Programmer* by Hunt and Thomas
- **Style** *Code Complete 2* by Steve McConnell
- **Interviewing** *Cracking the Coding Interview: 150 Programming Questions and Solutions* by Gayle Laakmann McDowell, *Elements of Programming Interviews: The Insiders' Guide* by Tsung-Hsien Lee.

Online resources

In addition, it can be very helpful to practice on coding exercises and sample interview questions, especially if you haven't interviewed for a software engineering role in several years or otherwise might be rusty:

- http://www.careercup.com/page (Big collection of interview questions)
- http://www.geeksforgeeks.org/ (Even bigger collection of interview questions)
- http://codekata.com/kata/codekata-intro (Several practical coding exercises)
- http://geeksquiz.com ("Computer science mock tests for geeks")

Interview mechanics

This section contains some advice for navigating a real interview at Dropbox.

Before the interview

Coding interviews are done on either a laptop or whiteboard. Make sure to practice writing code on a whiteboard beforehand! Working on a whiteboard is tricky for most people, so some preparation in writing clearly and organizing your ideas can help your interview performance a lot.

Right after you hear the problem

Ask any clarifying questions you have and make sure you totally understand the problem. Then, brainstorm and talk through some approaches with your interviewer. If your interviewer seems satisfied with a certain approach, go for it! Don't worry about an excessively general solution right away-just focus on getting a basic correct implementation. Your interviewer may add additional wrinkles or requirements later.

But don't start coding too early. A leading cause of bad interviews is when candidates jump right into code before they have a clear shape of the solution in their head. A little bit of thinking beforehand can go a long way. Don't be afraid to draw pictures or work through simple examples before starting. Always try to start with a clear high-level plan in mind.

While solving the problem

Make sure to talk through the issues you're thinking about! Effective communication is very important and can help your interviewer give you useful feedback about your approach.

Try to break the problem down and find good abstractions. Keep momentum on your side by using helper functions provided by your language (or making up helper functions if you don't remember them exactly). Often, your interviewer will let you delay or skip the implementation of helper functions, as long as they're simple-just make sure to communicate the assumptions you are making and what you are doing, and they'll let you know if you should dive into the implementation of your helpers.

Don't worry about having 100% perfect code and variable names. We understand that coding on a whiteboard is hard and we adjust our expectations accordingly. If you can get to a solution that works, is understandable, and has reasonable performance, you'll be in great shape.

Feel free to use any programming language or libraries you're comfortable with, as long as you can clearly communicate to your interviewer what's going on. It's easiest for us if you're fluent in at least one major language like Python, C++, or Java, but if you have to choose between a less common language you're an expert in and a mainstream language in which you have less experience, choose the former.

Before you say "I'm done"

Take a step back and do some sanity checks. Work through some simple examples and make sure you can convince yourself (and your interviewer) that you haven't missed any edge cases. If you've cut any corners in the interest of time, let your interviewer know how you might do things differently in a production-quality setting.

Good luck! We look forward to seeing you in the interview.