# Landing a Gig in the Tech World

## Introduction to Tech Recruiting

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeline</td>
<td>4</td>
</tr>
<tr>
<td>Popular Companies</td>
<td>5</td>
</tr>
<tr>
<td>Available Positions</td>
<td>5</td>
</tr>
<tr>
<td>Popular Locations</td>
<td>5</td>
</tr>
<tr>
<td>Application Material</td>
<td>5</td>
</tr>
<tr>
<td>Major Requirements</td>
<td>6</td>
</tr>
</tbody>
</table>

## Before you apply

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make Connections</td>
<td>6</td>
</tr>
<tr>
<td>Make a LinkedIn</td>
<td>6</td>
</tr>
<tr>
<td>Research alumni using the Amherst Alumni directory to find their email addresses:</td>
<td>6</td>
</tr>
<tr>
<td>Cross-reference alumni from the Amherst Alumni directory with LinkedIn:</td>
<td>7</td>
</tr>
<tr>
<td>Cold-emailing alumni:</td>
<td>7</td>
</tr>
<tr>
<td>Phone call/Skype/FaceTime/Hangouts:</td>
<td>8</td>
</tr>
<tr>
<td>Follow up:</td>
<td>8</td>
</tr>
<tr>
<td>Build your Resume</td>
<td>9</td>
</tr>
<tr>
<td>Draft your Resume</td>
<td>10</td>
</tr>
<tr>
<td>Only list CS accomplishments:</td>
<td>10</td>
</tr>
<tr>
<td>Use technical terms:</td>
<td>10</td>
</tr>
<tr>
<td>Bullet points should be structured as: what you did, why you did it, metrics that show your solution worked.</td>
<td>10</td>
</tr>
<tr>
<td>Use metrics!</td>
<td>10</td>
</tr>
<tr>
<td>Be diverse with languages/frameworks.</td>
<td>11</td>
</tr>
<tr>
<td>Use tech-industry specific Buzzwords.</td>
<td>11</td>
</tr>
<tr>
<td>Use emphatic action words when describing what you did.</td>
<td>11</td>
</tr>
<tr>
<td>Your resume should be one page long.</td>
<td>11</td>
</tr>
<tr>
<td>In your education section, include your GPA, Amherst major(s) and your high school diploma.</td>
<td>11</td>
</tr>
<tr>
<td>Proofread!</td>
<td>11</td>
</tr>
<tr>
<td>Draft a Cover Letter</td>
<td>12</td>
</tr>
<tr>
<td>Make a Github</td>
<td>12</td>
</tr>
<tr>
<td>Make a Personal Website</td>
<td>12</td>
</tr>
<tr>
<td>Go to Hackathons, Career Fairs, Talks and the CS Club</td>
<td>12</td>
</tr>
<tr>
<td>Make a list of companies you are going to target</td>
<td>13</td>
</tr>
<tr>
<td>Be Motivated</td>
<td>13</td>
</tr>
</tbody>
</table>
Applying
Tailoring your applications 13
Referrals 13
Online Portals 14

Interviewing 14
Resume Screening 15
First Interview 15
Online Coding Challenges 15
Phone Screen 16
  Introductions (10-12 minutes): 16
  Coding Problem (35-40 minutes): 16
  Final questions (10-12 minutes): 17
  Thank-you Note 18
Number of Rounds 18
On-Site Interviews 18
Variations 19

Rejection 19

Receiving an Offer 19

Post-Offer 20

Interview Prep Material 21
Personality Questions 21
  The spiel question 21
  The motivation questions 21
  The resume questions 22
  The experiential learning questions 22
Technical Preparation 23

Interview & Coding Guidelines 24
Problem-solving approach 24
  Ask questions to clarify the problem. 24
  Run through an example 25
  Coming up with a solution 25
  Iterate through 2 and 3 until the interviewer is happy 25
Pseudocode 26
Edge Cases 26
Coding Tips 27
Checking and Testing your code 27
Running your code 28
Modifying the problem 28

**General Tips for the process** 30
Start early 30
Don’t memorize solutions 30
Encountering an interview problem you’ve done before 30
Don’t panic if you see a hard question 30
Practice Interviewing with other people 30
Schedule Interviews when you’re at your best 31
Unconventional Internship opportunities 31

**Good Luck!** 31

**Sources** 32
Landing a Gig in the Tech World

The technology industry is a hard one to get into, but once you’re there, it can be extremely rewarding. By rewarding, I don’t mean just lucrative salaries ($100K+) but also an enjoyable experience, replete with free snacks, catered gourmet meals, 5pm beer Fridays, over-the-top perks like gym subscriptions, ping-pong and beaches, depending upon where you get employed. The tech sector is also the fastest growing sector of the US economy. The world has a net shortage of engineers, so it is hard to go wrong with a career in tech right now.

Here’s a tour of Google’s office to show you what working in tech may look like: https://www.youtube.com/watch?v=HBe6XUFvtRY
Note their office gym is better than Amherst’s!

The point of this Info Pack is to help you get a job in Technology. I remember how painful it was to go through the entire technical recruiting process alone, relying on external resources like peer feedback, articles, alumni connections and luck to guide me. Moreover, coming into the process, I lacked the social and knowledge capital needed to do well. I entered Amherst with very little knowledge and job access, not having had rich parents, siblings or a priori social networks to guide me through the journey. This document serves as a “giving back” to my communities. I want to help share my knowledge with you so that you can succeed at these amazing opportunities too. So, let’s get started.

Introduction to Tech Recruiting

The recruitment cycle serves three purposes: 1) to allow recruiters to connect with capable candidates from schools over the US, 2) to channel these candidates through a series of interviews for an internship or full-time position, and 3) to enable recruiters to maintain contact with talented candidates for future recruitment cycles.

Timeline

Online application portals usually open mid-summer to early fall. Banks that offer engineering internships, like Goldman Sachs and JP Morgan, tend to be the first ones to accept applications, starting in early July. Most other companies will publish their postings latest by October. Almost all interviewing is over by February, soon after which offers and rejections are delivered. Generally, the earlier you apply for the job, the better.

Popular Companies

Google, Facebook, Microsoft, Amazon and Apple are the Big 5. They recruit very heavily every year and are often the toughest jobs to get. Other big, but less “heard of” companies are also in the intern market but aren’t where newbies to the process intuitively gravitate toward. Examples of this latter category are Salesforce, Cisco, Intuit and Oracle. On the other hand, medium sized reputed companies like Airbnb, Lyft, Uber, Snap and Palantir are fast becoming extremely attractive places to work due to great
growth and work experiences. Some in this category may also be subsidiaries of larger companies. Audible, for example, is an Amazon company. Similarly, Instagram and WhatsApp are Facebook products. The other end of the spectrum comprises start-ups. Startups don’t necessarily follow a structured recruiting cycle - they just kinda wing it. Often, opportunities won’t even be listed on their websites, even though there could be space for a “word of mouth” valuable summer experience. Generally, the larger the startup, the more likely they will offer an engineering role. Many non-traditionally tech companies have also begun offering positions. Examples are banks, trading firms, healthcare companies and even transportation and logistics enterprises. The specifics may vary but the trends are fairly similar, with firms like JP Morgan, Goldman Sachs, Fidelity, General Motors, Tesla and SpaceX and Bank of America in this category.

Available Positions

The available positions fall into 3 broad roles: 1) Engineering, 2) Product management and 3) UI/UX design/research opportunities. Engineering roles include anything from Front-end to Back-end to Full-Stack coding, Data Science to even iOS/Android development. Product management generally involves some coding along with project development skills, while the final category is almost exclusively a research category. All discussion hereafter will regard Software Engineering roles, though much of the following information will also be pertinent to Product management and research.

Popular Locations

Tech really is everywhere in the US. Silicon Valley in California is unquestionably the hub, with startups a dime a dozen in the locality. Seattle has recently begun challenging the Valley’s monopoly over the tech world, with the Microsoft and Amazon headquarters located in the vicinity. And New York, of course, is the omnipresent competitor. South CA may be a good option, just as Boston and Austin, but you will find jobs everywhere really, from Hartford, CT to Raleigh, NC.

Application Material

Most online applications will ask for a resume, a cover letter and some information about your work eligibility. The resumes are expected to be tailored to the tech industry, the cover letters are expected to be well-researched and all information provided is expected to be truthful. It’s better to tell the truth because even if you lie and get the job, background verification will catch you. **The easiest way to get rejected is to apply online on a company site.** Instead, you should ask a contact who already works at the company to refer you for the position you would like to be considered for. More on this later. In addition to these documents, it is recommended to set up a descriptive LinkedIn page and maintain a Github for all your code. A personal website, if you are able to create one, would also be helpful for recruiters.

Applying for a tech job through **Handshake**, however, is worth your time. Handshake is the Loeb Center’s career management platform, and fields postings from companies who have specifically selected Amherst as one of their recruiting targets. Those companies will then track the pipelines of applicants coming in through the Handshake process and consider them separately than those coming in through their main company website.
Major Requirements
A job in Tech is one of the few that actually has major requirements. Math, Statistics and Computer Science (or any other type of engineering) are usually the only acceptable majors. You could have another non-Tech major too, like Philosophy, but unfortunately, many job listings will disallow purely humanities majors from applying.

Before you apply
Applying is a tumultuous process and it’s extremely repetitive. Researching and shortlisting companies, updating and re-sending your resume, writing cover letters and ultimately, interviewing for multiple rounds for multiple companies is monotonous. Nonetheless, successful applicants spend the summer preparing for this process in multiple different ways. The goal of this section is to tell you what you should be working on throughout the summer to put yourself in the best possible position for an internship/job the next summer.

Complete your Handshake Profile
One of the easiest first steps you can take, to make sure you don’t miss any Amherst-specific recruiting or networking opportunities, is to complete a robust profile in Handshake. The more complete your profile, the more tailored the platform can work for you. For instance, if you “favorite” target tech employers in Handshake, you can receive and email ping when that company posts a new position. Furthermore, recruiters search out undergraduate candidates across schools, across the entire platform, using search criteria based on profile data. So help the system help you by creating as detailed a profile as possible.

Make Connections
Connections are indispensable to an effective recruiting experience. Getting to know people at different companies has many advantages, often depending on their roles in those companies. Any person working at the company should be able to refer you for an internship/job posting. Employee referrals will guarantee you, at the very least, a talk with a recruiter and a reading of your resume. Why is this important? Simply because there are too many people competing for a limited number of highly demanded jobs. Applying boldly on online application portals is often pointless because 1) some companies never get back to you, 2) other companies will reject you outright without reading your application because there are simply too many applicants for a limited number of spots and 3) almost all companies have programs that screen your resume for keywords, and that are unlikely to do justice to your resume than had it been read by a human. Knowing an alumnus who works at the company you are applying to and who can refer you for the position is the best way to apply. Certain Amherst alumni also work as university recruiters for these companies. These are the wonderful people who not only refer you to jobs but also tell you what and how to prepare for interviews, as well as what the company is looking for in candidates.
Making connections isn’t instrumentalizing, neither is it using people to get a job. Alumni are friends and building relationships with them it a good thing - particularly when you have an underprivileged background. They love helping Amherst kids, too. The following subsections detail how to make and use connections in the tech world. In case you’re still wondering why you would need to network to succeed, here and here are some convincing Loeb Center resources.

1. **Make a LinkedIn:**
   There are tons of guides for this. You can use mine as a template.

2. **Research alumni using the Amherst Alumni directory to find their email addresses:**
   There are multiple ways you can search for alumni who work in tech on the alumni directory. First up, you could simply type the company’s name into the search box and retrieve a list of alumni who work there. Clicking on their profiles reveals their email addresses. You could also search by alumnus name, if you know (of) someone who graduated earlier and has a gig. You could also search directly by industry (software engineering), or filter your results based on class year, major, hometown and other parameters. A good strategy is to create an Excel document with separate spreadsheets for each company you are targeting and populate them with alumni and their email addresses. If you want to be sneakier, code a script that does all this for you (and include it as a side project on your resume).

3. **Cross-reference alumni from the Amherst Alumni directory with LinkedIn:**
   Sometimes, alumni will not update their profiles on the directory. This means that even if they do work in tech, there’s no simple way to know using just the Alumni Directory. That’s why you should cross-reference with LinkedIn. On LinkedIn, search for People who work at XYZ company and who went to Amherst. Make sure you have the email addresses for all the profiles that show up using the LinkedIn search by searching for people’s names directly.
4. Cold-emailing alumni:

Once you have found their addresses, you now need to cold-email people asking them to help you. Here’s a good template for this purpose:

Dear xyz,
I hope you are well!
I obtained your contact from the Amherst College Alumni Directory – I am a sophomore/rising junior from <place> with a Computer Science <and other> major at Amherst.
I am really passionate about technology and plan to pursue a career in the industry after I graduate; in particular, software engineering really interests me. I am excited about the work <xyz company> is doing and am interested in learning more about your work with and contributions to the company. I am sure your schedule is demanding, but I would really appreciate the opportunity to talk with you about it and opportunities within for interning for <period of internship, for example summer 2019>.
Would you be available for a short call over phone or <other acceptable service>? Please let me know whatever times work for you.
Thank you for your time and hope to hear from you soon!
Best Regards,
<name>.

Send out hundreds of these. I always found it easiest to send emails in batches - they don’t take a lot of effort once you have a template and you can simply copy paste the required fields. Pay attention! Don’t send an email to someone who works at Lyft asking them how to get an internship at Uber. Also, don’t send an email to a Microsoft employee asking them to do a Google Hangouts with you, or a Google employee asking them to chat over Skype. Loyalty to
company products matters and a good first impression is clé. Diligence counts. In my experience, most alumni respond and almost all are very willing to help. Don’t press them - they are busy people and it can take up to a week for replies to roll in.

5. **Phone call/Skype/FaceTime/Hangouts:**

Be prepared to introduce yourself, present your main projects and your motivations to work in tech in under a minute to your interlocutor. Biographical spiels and marketing yourself will prove crucial throughout the process, so it’s important you learn now. Normally, the person you are talking to will respond with a spiel about themselves at this point, after which you should ask questions about the company, about the internship and about interviewing. Read up on the company your interlocutor works at before you go into the chat. Example questions are: what is the company culture like? What does a typical day at work look like for you? Do you like your job? What team do you work on? What are some bad things about the company? (not the first question you should ask) How does the internship process work? Do you have tips for interviewing? **At the end, you should ask for a referral.** The request should go something like: “I really like what I’m hearing about <company> from you and I think this would be a great opportunity for me. I’d love to apply for an internship, but I was wondering whether you would be kind enough to refer me for the position.” They don’t refuse.

6. **Follow up:**

Send a thank you note to your conversant after you talk to them. Tell them you will keep in touch with them and update them on where you end up. Post-talk remerciements are crucial in the entire interviewing process - you want to thank everyone who takes the time out to entertain you.

You could also directly message University Recruiters or alumni on LinkedIn. If you decide to do this, here is a good template to use:

```
Hi <xyz>!
I found your profile while I was searching for <xyz company> Recruiters on LinkedIn. I am a Junior Computer Science major at Amherst College and <I created xyz project in Android that I then launched on the App Store. I have also coded an AI in TensorFlow that can translate Urdu.> I am interested in working at <xyz company> as a software engineering intern during summer 2018; can I send you my resume?
Thanks!
<name>.
```

This summer, I encountered friends who landed amazing jobs in technology by cold-emailing recruiters. Recruiter emails are hard to get if they’re not given to you, but you can get your hands on them by using a Gmail extension called ‘Clearbit Connect.’ CC predicts email addresses to a very high degree of accuracy; so if you find a recruiter’s name, you can use CC to generate her email address & send her your resume directly.

Direct messaging people on LinkedIn is generally not very effective. Recruiters rarely respond, alums are often too busy to do so and you might just end up wasting your time and LinkedIn credits.
The above steps detail the standard procedure on making connections. Nonetheless, you should employ your social network to the maximum to learn about opportunities. Talk to your coach, professors, friends, family and even family friends. Generally, the more people you can reach out to, the more likely you are to come across an opportunity.

**Build your Resume**

Companies are looking for drive. They want to hire students excited about tech, curious about new technologies, and exhilarated by engineering - not someone lukewarm about CS. Building your resume is a great way to show technology excites you. This involves acquiring as much experience coding, working on projects (coursework and non-coursework related), TAing, learning new languages and contributing on open-source projects. You don’t have to do all, but you should do some to be competitive. An internship the previous summer is great - if you don’t have a lot of industry experience, that’s fine too. Side projects are a fantastic way to build your resume. They show motivation, determination and curiosity - qualities that recruiters are looking for. Moreover, they provide great talking stock for interviews. To complete a side project involves learning one or more new languages (normally by online courses), practicing them, using them to build something cool and then publishing your code on Github. You could create something in a language you already know - that’s completely fine too. Your side project doesn’t have to be the next Tinder. Keep it simple, straightforward and executable. Side projects are also really helpful because they diversify your programming language portfolio. Courtesy to the Amherst COSC department, you would know only Java if you didn’t learn new languages outside the classroom. A programmer who knows only Java isn’t a great programmer.

It’s daunting to learn a new language. **Keep going!** You’ll get the hang of it eventually, and it gets easier the more languages you learn. [Codecademy](https://www.codecademy.com) is a fantastic resource to learn new languages. You can take well-organized courses in HTML, CSS, Javascript, Python etc with supervised approaches. Many languages and frameworks have websites that offer detailed tutorials on how to use them. Google’s [Tensorflow](https://www.tensorflow.org) guides users on how to build AI from scratch. [W3Schools](https://www.w3schools.com) is an exceptional resource on HTML, Javascript, CSS and the like. [Angular.io](https://angular.io) is wonderful to learn the Angular JS framework. Other sites like [scotch.io](https://scotch.io) offer courses on entire frameworks, like the MEAN stack (MongoDB, ExpressJS, Angular and NodeJS). Most capable programmers have a solid grasp of HTML, CSS, Javascript (at least jQuery), SQL and at least one backend language like Java.

Here are some side project suggestions to get you started:

1) Learn Python and build a simple deep learning AI model using Tensorflow to identify cats.
2) Learn the MEAN stack and build a simple web application to display quotes by Rousseau.
3) Learn Reactnative and build a simple To-Do list mobile app.
4) Learn the Java Spring framework and build a web application that has a user login/logout feature.

Keep an eye out for Tutorship or Teaching Assistant opportunities in the CS department. Typically, TAing a higher level class is more impressive than TAing COSC-111. Take courses with big final projects so you
can list them on your resume later. If you can get involved in open-source frameworks, that is really impressive - there are many on Github and you can even create and publish your own! Volunteer for coding or testing opportunities offered on campus (for professors for example), and if you’re daring enough, launch your own entrepreneurial ventures here. Amherst needs more creativity!

You can list side projects that are work-in-progress on your resume too. Although finished projects uploaded on Github are more concrete because no one can claim you’re bluffing, incomplete side projects can show initiative too and are fair game as long as you can talk about them.

**Draft your Resume**

The resume is where you actually market yourself. The goal is to convince recruiters to give you an interview, and there are ways to describe your accomplishments impressively.

1) **Only list CS accomplishments:**
   Recruiters don’t want to read that you edited the Amherst Student or play hockey. If you built a hockey website though, that’s worth mentioning.

2) **Use technical terms:**
   Say you deployed a server using ExpressJS and built a front-end in Angular that you then connected to a MongoDB instance to retrieve user login data for a news website; you are appealing to technical people so feed them meals they like.

3) **Bullet points should be structured as: what you did, why you did it, metrics that show your solution worked.**
   For example: created a web scraper using NodeJS and CheerioJS to scrape lyrics of all Taylor swift songs from AZlyrics.com. 30+ songs and over 10,000 words were stored in a MySQL store to be displayed on taylorswift4pres.us, a Taylor Swift fan page.

4) **Use metrics!**
   Metrics tell recruiters the scale and success of your project. They can be anything from database sizes to speeds of algorithms, number of users to outreach figures. For example: developed a modification to Dijkstra’s algorithm that improved running time by 15% on graphs with a “grid” property. Metrics allow you to quantize your project - what can be measured can be improved.

5) **Be diverse with languages/frameworks.**
   You don’t want all your projects to be in Java. Make sure to represent some JavaScript, Scala, Go, R, Android etc.

6) **Use tech-industry specific Buzzwords.**
   When screeners screen your resume, they look for certain keywords; a resume with more buzzwords is more likely to get an interview. Some in the tech industry are: “big data” “machine
learning” “artificial intelligence” “full stack” “ReactJS” but they will often differ from company to company.

7) **Use emphatic action words when describing what you did.**

For example, say: “architected SQLite database back-end of Pie Bill Gates, an Android game” instead of “responsible for designing back-end of Pie Bill Gates, an Android game.” Action words add energy to your resume and recruiters will want to read more of it.

8) **Your resume should be one page long.**

If you don’t have enough technical material to fill one page, keep working on side projects until you do. If you have more than a page, select what you think are your most substantial achievements. What accomplishments qualify? Internships, TAing, Tutoring, Projects done in class or on your own, Languages and skills you know, Open-source contributions, startups or tech-related stuff in general.

9) **In your education section, include your GPA, Amherst major(s) and your high school diploma.**

In your contact details, put down your phone number and email. Mailing addresses are unnecessary. Keep personal details like race and hometown out of the resume - resumes are to list your achievements. Recruiters will get to know you personally once they decide you are capable.

10) **Proofread!**

Even honest spelling and grammar errors make you appear careless and half-hearted about the job, which sucks, because you aren’t! Standardize formatting (line space, margins, bullet points) throughout your resume to make it appear professional. It’s best to handle resumes in pdf because Word documents are known to change formatting when opened on different computers.

**Here** are some great resume templates to get you started drafting your resume. I used template 4 although 2-5 are fairly similar. In my resume, the sections are: Education, Work Experience, Projects & Languages and Skills. If you need more help drafting a resume, read **this** and **Cracking the Coding Interview**.

You will likely go through multiple drafts throughout the interviewing process; this is normal because you should always keep improving your resume by adding newer languages and achievements.

**Draft a Cover Letter**

Cover Letters aren’t as important as the resume in the tech industry in the grand scheme of things, but it’s always good to have one. **Here** are some good templates to get you started. The biggest mistake is sending the same cover letter to multiple companies. You should always research every company, find out exactly what it does, how it does it and why it does it, and explain why you want to work for a company like that. Cover letters are meant to detail your motivation to join the company in whatever
role they are offering, why the role appeals to you and how you can add value to the company. One thing I did to ease tailoring cover letters to companies was to use the same basic template; I researched and added company-specific information to my cover letter later.

Make a Github

A good software engineer always has a portfolio of her projects. Github is where such portfolios are created and accessed. The website allows you to create repositories (projects) where you can add code and other files. Although it’s meant for version control, Github is an excellent tool because it allows recruiters to study the quality of the code you write as an engineer. Recruiters can assess project structure and code size, the way you use existing frameworks and your potential as a coder. Having a Github is also proof you aren’t lying on your resume. For at least every project you mention on your resume, you should have a corresponding repository on Github. The more the better. You should also make it a habit to employ Github features (push, pull etc) whenever you work on a programming project; in software, version control is a really useful skill to have.

Make a Personal Website

Personal Websites are the least important of all pre-recruiting tools. You should do this only if you have finished all the above and still have time. A nicely designed one could even be a side project. Nonetheless, websites are really helpful marketing tools. On personal websites, you can gather all information about yourself - links to your LinkedIn and Github, your resume, languages you are proficient in and other interests (like linking a blog you write on) - to give people a holistic image of you as a person.

Go to Hackathons, Career Fairs, On-Campus Employer Info Sessions and the CS Club

Hackathons are amazing opportunities to learn new technologies. Career Fairs and Talks can put you directly in contact with a recruiter or someone important in the company, so that you already have a head start. The Grace Hopper Conference is one such opportunity to meet people and find internships. Other conferences, like Start at a Startup, provide similar networking opportunities. The Greylock Techfair and Internapalooza are popular internship fairs in San Francisco. Moreover, virtually every growth-stage and established tech company hosts a Open House during the summer. Open Houses are designed to attract undergraduate students to employment opportunities at the specific company; paired with approachable recruiters, Open Houses provide amazing opportunities to gain insight into the company’s mission and success. Open Houses are normally by application, so it pays to be proactive in seeking them out. On another note, several startups (Jumpstart.me, stella.ai and Nav Talent) are in the business of intelligently matching undergraduate engineers to internships. Once you upload your resume to their portals, their Machine Learning models purport to match you to reputed company recruiters - this could be an effective way to get interviews. The Amherst CS Club often hosts workshops on programming techniques and languages that you could use to build your resume. The ‘Hackathon Hackers’ Facebook group is a great place to meet people who are hiring. Even SURF in CS may be a good resume builder. Once again, the onus is on you, the prospective applicant, to seek out opportunities and go out of your comfort zone to do so. Reach out to that one friend’s dad’s brother-in-law and you might stumble onto a dormant pot of gold.
There are local career fairs worth considering, as well. Smith College maintains an open invitation to Amherst students for their career fairs (typically in October and February each year), which draw a range of employers looking for software engineers. UMass also has a large tech internship and jobs fair in September, completely open to students from all the five colleges.

The Loeb Center also hosts many tech companies throughout the year - Google, Facebook, LiveRamp, HubSpot and Audible have all regularly made visits to campus to make focused connections with Amherst College students. A little known fact is that if you attend a Google session on campus, their algorithm will recognize you when you apply for a position, and help sort your resume to the top of the pile. Completing your Handshake profile and indicating tech as an interest area ensures your account will alert you to employers coming to campus or other relevant events.

**Make a list of companies you are going to target**

This is useful to track your progress throughout the recruitment cycle. In my list, I included any and all companies I could think of. At this stage in the process, it pays to not be too selective and to have an open mindset about where you could end up.

**Be Motivated**

You’re amazing. Remember this throughout the entire recruitment process, especially when people tell you you’re not a great coder, that your achievements suck or that they carefully read your application but after consideration, decided not to move further with it. A bad interview or a rejection sucks, and it’s the easiest thing in the world to beat yourself up after it. But becoming an engineer is hard. The world is extremely competitive and people are ruthless; if you’re interviewing, you’re already there.

Having said this, it’s important to work hard. **Grind.** Prepare, be on top of all of your shit, be willing to go out on a limb, be creative with your approaches and be consistent. When you feel like giving up, ask why you started in the first place. As Nietzsche put it: “He who has a why can bear almost any how.” If your motivations are strong and your intentions pure, then you’re going to succeed.

But don’t be arrogant either. Tons of extremely capable but extremely full-of-themselves engineers get rejected every year. Companies do want good engineers - but first and foremost, they want **good people.** Be humble, remember where you came from, smile and sell yourself confidently. The world owes you nothing. It’s a privilege to be here in the first place.

Out of the 40 companies I applied to, I got just 2 offers. Trust me, they were worth it.

**Applying**

Applying is a boring and repetitive process. There may be discrepancies from company to company, but the process is largely identical.
Tailoring your applications

Different companies are looking for different skills. At Pivotal, for example, knowledge of testing/test-driven development is valued. Google likes to see knowledge of open-source libraries. Palantir, on the other hand, values system design highly. While it is hard to drastically change your resume and cover letter for every company you apply to, it’s not recommended to use exactly the same material for every application. At the very least, you should address company specific work, values, and mission in your cover letter. Altering your resume for each company is a little trickier, especially considering it is hard to find out what recruiters are looking for. But if an alum discloses company preferences, do attempt to emphasize any projects/achievements in those domains. When I was researching companies, I took notes on company culture, mission, work, tech involvement, technologies used etc. for each company that I later consulted during the interviewing stage. These subjects provide excellent material to ask questions about later on.

Referrals

When someone at a company refers you, your resume is flagged in the company employment system. This means you are guaranteed at least a human read of your resume and a recruiter is likely to reach out to you. Congratulations, you are already in a stronger position than most other candidates. However, you will likely still have to apply through the company’s online portal to get your resume into the system to begin with.

Online Portals

Company internship and job listings normally include a description of the role, responsibilities of the employee and some expectations of her. They often also include a list of requirements for potential candidates, such as major, class year and programming skills. A software engineer intern position might require candidates to be proficient in either or more of Java, Python, C++, Go or Scala. A full-stack intern position will likely involve an HTML, CSS and Javascript requirement. If you are involved in side projects, you will likely acquire many of the required skills. If you meet the requirements (and have found a referrer - recommended!) you should go ahead and apply. Apply early! Keep an eye out for when application portals open (August for many companies) and submit your application as soon as you find a referrer.

Most application portals are pretty straightforward. After having uploaded your resume and cover letter and inputted basic personal information, you will then be asked some additional optional questions. These questions will concern race and citizenship status, military, disabilities and gender. Fill these carefully; if you’re an international student you should declare you will need future sponsorship. Do not lie. Even if you do, in the best case, you get the job but get caught lying during a background check, after which the company will alert authorities and you will be unlikely to land a job in tech ever; in the worst case, you get caught immediately and the company once again alerts the relevant authorities. Not only is it not worth lying, but lying is wrong. You want to land a job honestly.
After having submitted your application, you should follow up with your referrer. There may be discrepancies in the application process from company to company, like sending your application material directly to your referrer instead of applying online, but the process is largely identical.

**Interviewing**

Welcome to the most stressful stage of the entire process.

What can you expect once you have applied? Well, nothing really for the first week. It takes time to process applications, especially when there are thousands of them. This is why it’s so important to apply as soon as the opportunity opens up - which you can only do if you have begun the entire process early in the summer.

To explain how the process works from here on, I’ll describe the life-cycle of Ali Khan’s application to a company called Merkrith.

**Resume Screening**

After Merkrith receives Ali’s application, it is screened; normally by a computer, but likely by a human if he has been referred. If Ali passes the initial test, his resume is assigned to someone called a “University Recruiter.” This person’s job is to reach out to Ali and schedule the interviews. Some reasons why Ali might get rejected at this preliminary stage are that his resume doesn’t include key buzzwords, his expertise does not match up with Merkrith’s needs or that he is just too weak of a candidate.

**First Interview**

Alexa, Ali’s recruiter, then reaches out to him either by phone or email. The first interview is normally an informal chat between the two. Alexa simply wants to know Ali better. He will be asked to present himself and talk about his resume for a minute, questioned why he wants to work at Merkrith, what excites him about the company and why he is interested in tech to begin with. Some oddballs may also be thrown at him: Alexa might want to know the hardest tech problem he has faced and how he solved it, she might inquire about a specific side project on the his resume and she might even ask him what his goals in life are. These questions are known as “behavioral” questions in the tech world; they test Ali’s personality and not his coding abilities. To do well, Ali should research Merkrith (or revise his notes on it from when he wrote his cover letter) and read the [Personality questions of this guide](#). If Ali does well, Alexa arranges for him to go through the first round of technical interviews. These could either be an online coding challenge or a phone screen.

**Online Coding Challenges**

If Ali is sent an online coding challenge, he will receive an email with a link to it. Generally, challenges are hosted on HackerRank or HackerEarth and are 1-2 problems long with a limited amount of time to complete them (often around 2 hours). These problems are mostly coding questions; for each, Ali will be given a problem statement and will then not only have to come up with a solution, but also code and test it in an online coder text environment. HackerRank/HackerEarth provides several test cases for Ali in
order to test his solution, but code that provides the right answer isn’t normally good enough - it needs to be fast and minimize space usage too. Thus, he is likely to spend time brainstorming different possible solutions and code the most efficient one. Sometimes, not all the problems Ali encounters on online challenges will be coding questions. They may be Multiple Choice Questions, for example: what is the running time of HeapSort? or what is the worst case running time of a search operation on a Binary Search Tree? Ali might even encounter a personality question (why do you want to work for Merkrith) in addition to coding problems. Nonetheless, all problems will be language agnostic. This means that Ali can choose to code his solution in any language he likes - at least 10 different languages are supported by the online platform.

If time is running out and Ali realizes he will not finish, he should submit his solutions with comments detailing what code he would have written. He should then reach out to his recruiter, Alexa, telling her what happened. If he finds his solution is unacceptably inefficient and any attempts to write more efficient code give errors in test cases, he should submit the inefficient but correct solution, leaving his other attempts in comments. Every attempt to show recruiters you tried should be made. Recruiters are people - they are nice and understanding - and it’s in their interest that you succeed (because they make money), so they will help you.

I always tried to get out of coding challenges because it was always tougher for me to take a test by myself than do a technical interview with someone. If you can get out of it, I suggest getting out of it.

Once Ali has completed the online challenge, he will either be notified he has been selected for the next round of interviews within a week, or likely never receive a response. The latter means he didn’t do well enough. He should nonetheless email his recruiter and confirm if he hasn’t heard back within a week. The next round will be a phone screen.

Phone Screen

If Ali makes it to this round, he will undergo a technical phone interview or a phone screen or a ‘phone call’ (they’re all the same). Alexa will coordinate to arrange a call between him and a software engineer at Merkrith, normally over phone, but if Ali is abroad, then over Skype, FaceTime, Hangouts or other VOIP services - recruiters are usually very flexible. The phone screen will be 45 minutes to an hour in duration. Here is a breakdown of that time:

1. **Introductions (10-12 minutes):**
   The interviewer, Fei-Fei, will introduce herself. She will describe her title and position at Merkrith, what team she works on and her role in the organization. Then she will ask Ali, the candidate, to introduce himself. Ali will be expected to give his spiel - present himself, his resume and his motivations for joining the tech industry. Fei-Fei will have access to Ali’s resume and will ask questions about his accomplishments. What was the hardest bug in the Taylor4Pres project, for example? Or why did Ali maintain a server text file to store user information instead of using a database? She will also ask the question Ali will hear at every stage of the recruiting process: “why do you want to work here?” Typically, a response extolling the company culture,
attitudes, engineering philosophy and values should suffice, though Ali should strive to be more creative and company-oriented. Once satisfied, Fei-Fei will present the coding problem.

2. **Coding Problem (35-40 minutes):**

Fei-Fei and Ali will navigate to an online text editor, like CollabEdit or CoderPad. These editors are like Google Docs for code: while Ali types, Fei-Fei sees (and can change) anything he writes in real time. The online editors, like HackerRank and HackerEarth, provide multi-language support and a console to run and test programs. Fei-Fei will initiate by pasting a problem statement into the editor. She will briefly present the problem before handing the reins to Ali.

Ali will be expected to clarify the problem, understand it, propose an acceptably fast solution to it, code his solution and test and then run his code on sample cases. Although we will discuss strategies on performing well later, Fei-Fei is judging Ali’s process of reasoning just as much as his ability to arrive at an efficient and correct solution. Ali is expected to communicate vocally throughout this part of the interview - he must not get caught up in his thoughts, but explain how he is thinking about the question and what potential solutions he has reached yet. It is Fei-Fei’s job to assist Ali throughout the reasoning process. Often, she will give hints, so Ali’s best interests lie in listening to her. Ali will likely come up with at least one correct solution (a brute force solution) and this is an excellent place to start. Nonetheless, Fei-Fei will push him to do better. Once Ali has provided an adequate algorithm, Fei-Fei might test him further: she may change problem parameters and ask him how he would adopt his algorithm to solve the modified problem. She will most definitely inquire about the running time and space requirements of both algorithms.

Ultimately, Ali must produce some code. If he doesn’t have enough time, he should even put down pseudocode, though this isn’t an ideal situation. Setting Ali aside for a second, I always knew whether I had done well on a question or not. I remember interviews where I aced the question and I remember those that aced me up. A lot of it depends on luck - on a good interviewer and the type of question she asks. I’ve had interviewers who couldn’t care less, and I’ve also had interviewers who couldn’t stop giving me hints. I’ve also been asked pretty straightforward questions and I have been posed questions that I basically didn’t know where to begin solving. We’re all human and intelligence is a genetic lottery; I lost little sleep and saved my strength for the right moments.

3. **Final questions (10-12 minutes):**

This is the most underrated part of every phone screen.

After Ali completes the assigned problem, Fei-Fei will give him 10-15 minutes to ask her any questions about Merkrith or her job at the company. Ali should have questions pre-prepared for this time. Once again, his notes on Merkrith will come handy here. In a phone screen, it’s typically not enough to ask nebulous questions like “what are Facebook’s goals for the next 5 years?” Ali should instead direct his questions to things like technologies and programming workflows, company structure, engineering philosophy and different open-source projects.
Examples of great questions are: “what is Merkrith’s tech-stack?,” “why did Merkrith use Angular and not React on its front-end? [in general, why did the company use x instead of y?]” “what is the engineering team structure at Merkrith?” “do engineers have autonomy to switch teams or pitch projects they think would be valuable to Merkrith?” etc. Ali should write down Fei-Fei’s responses because they will come handy should he proceed to the next round of interviews.

With this, the phone screen comes to an end on Ali’s end. Fei-Fei, on the other hand, will assess Ali’s performance and give him scores on a range of qualities: code quality, thought process, correctness and efficiency, amongst others. The feedback will be then passed on to higher authorities.

4. Thank-you Note

It is imperative Ali send thank you notes after every interview. Thank-you notes are a subtle way of showing he’s grateful for the opportunity. A good attitude goes a long way, especially after a bad interview. An ideal time for sending one is the evening after the interview.

Number of Rounds

The number of interviews varies greatly. Some companies have 5 before they give applicants a decision; others take no more than three rounds. In my year, Facebook and Uber had only two. The premise for having multiple rounds is the same though - to filter out weak candidates and ensure only the best pass through. Tech has been often been criticized for its stringent recruiting philosophy: essentially, interviews are designed to not let any “bad” applicants pass onto the next round, instead of selecting for good candidates. In fact, smart and capable engineers get rejected all the time; nonetheless, the process is designed so that companies can ensure anyone who can’t handle the pressure of understanding, solving, coding and testing the solution to a random problem on any given day in 30 minutes doesn’t move forward.

If Ali performs well, Alexa, his recruiter, will coordinate to schedule the following rounds, which will also be phone screens. If Ali makes it through those too, he is obviously a well-prepared candidate, and will typically need to surmount one more hurdle before he gets a decision for his software engineering intern application to Merkrith.

On-Site Interviews

It’s just like it sounds - for an on-site, Ali will get invited to an all-expenses paid multi-day trip to Merkrith’s HQ! - and there he will go through at least three more sets of interviews. Typically, on-site interviews are structured similarly to phone screens. Introductions are followed by a coding problem which is followed by concluding questions. What sets the on-site round apart though, is that interviewers are looking to test certain topics in particular - like system design and scalability. This will be talked about in the Interview Prep section.
The biggest challenge for Ali is that interviews at Merkrith will be back-to-back. There are usually no breaks and no respite; it’s like a mile-long sprint to an elusive end. Ali can take restroom or water breaks between interviews (which are helpful to revitalize) but 3 hours solving mind-straining computer science problems will be a struggle for anyone. Once the morning is over, Ali will have lunch with an engineer. This is a good time to destress; it is an informal setting and an ungraded engagement and Ali should ask questions about the company that genuinely pique his curiosity.

Classically, the afternoon could go two ways. Ali could either be taken to a “Merkrit 101” -- a presentation about the company, its product/service, a demo of its product and company goals. This is traditionally a bad sign. It means Ali didn’t do well and that Merkrith is done with him. Or, he could be taken to a manager who will talk more about what it’s like to work at Merkrith and answer any final questions he has. This is a good sign - it means the company has found value in Ali and is looking to hire him. Afternoon on-site decisions are highly contentious: companies have historically followed this bipartite structure, indirectly conveying their decisions on the spot. But there are always variations, and hiring managers are now beginning to change the offhanded afternoon decisions scheme.

In any case, Ali will be sent a rejection or offered a place at Merkrith within a week of his on-site. It is hard to know who decides and how. Often, feedback from all on-site interviews is accumulated and presented to a hiring committee that includes at least one person who has never met Ali. Alexa, Ali’s recruiter, will convey the final decision to him. If he doesn’t hear back in a week, Ali should reach out to Alexa.

Variations

No two companies have identical recruiting procedures. There will always be variations in the process from company to company - some may have no on-site and only phone screens; others may have an on-site directly after a single coding challenge. Once again, recruiters are flexible with arranging interviews - I had several “on-site” interviews over Google Hangouts while I was abroad. It’s their job to make sure candidates have the best possible shot!

Rejection

What happens if you get rejected? Nothing - you focus on the next company you’re interviewing with.

A rejection isn’t a statement about your worth, nor even your ability as a coder - it’s just an assessment of how well you have prepared for the recruiting process. Just because you didn’t get a job at Google, doesn’t mean you’re a bad coder - it means you didn’t play the process as well as others in the applicant pool. This is actually excellent news for anyone who wants to become an engineer. It means that the more smartly you prepare (which means carefully reading this manual), the more likely you are to get a top-notch internship/job. In other words, the genetic intelligence lottery isn’t that big of a deal.

But it’s also important not to get too attached to a single company. The world isn’t perfect and you will always get some rejections, even if you are a superstar. You might think a certain company is the
“perfect” fit for you but there are always, always other companies that have similar values, do similar work and have a similar culture. The world isn’t perfect but there are alternatives.

I applied for 40 positions and only got two offers. But that’s also because I bombed a ton of my interviews because I didn’t know how to prepare for them. Hopefully, my experience and advice distilled into this document will help you get many more than I did.

Receiving an Offer

It feels great; partly because you’ve been offered a lot of money, mostly because your efforts have paid off!

Different companies pay differently - some pay by the hour, others pay a fixed amount per week, etc - but the average pay is northward of $40 per hour, with increased overtime pay. Most companies give a “relocation fund” which is basically money you need to move and live elsewhere. Then, there are perks: catered gourmet meals (better than Val!), office dogs, free snacks, ping pong tables, gym subscriptions and even covered commuting costs. The length of internships may vary from 10 to 12 weeks - some go higher - but a 12 week internship is the ideal length for a summer.

If you find yourself with the privileged problem of having multiple offers on the table, it is worth negotiating with companies for better offers. You can use one offer as leverage, asking for increased pay/benefits/stock options (though it is rare for interns to get stocks) from another company. However, make sure that if you are using that leverage to ask for a better offer from an employer, you need to be willing to accept the offer if they make it to you, so it is best to use this tactic just with your first choice company.

Post-Offer

Once you are satisfied with an offer, you sign a contract with the company. It is understood that contracts are final and candidates can’t go on interviewing or negotiating with other companies. It is unethical to renege on an employment agreement or continue to pursue an opportunity after accepting a position with someone else. You are just starting to build your reputation in a field, and people move around a lot between companies in tech, so protect your integrity no matter what. If you have accepted a position through a recruiting process facilitated by the Loeb Center (including applying for a job through Handshake), reneging on an agreement or continuing to interview after accepting an offer is considered a violation of the code of conduct and you may lose privileges in Handshake or with other Loeb Center resources.

Then begins the background check process. For them, you will need to submit documents verifying previous employment, academic degrees and your citizenship status in the US. Most background tests check out and companies then begin an onboarding process for new employees/interns. In the first stage, the new hire is assigned to a team in the company. Team allocation is a lengthy and convoluted
process that differs significantly among companies. Usually, the recruiter stays in touch after the offer, asking the accepted candidate for technology stack or team preferences which are evaluated in conjunction with team demands. Interns are then accordingly assigned. The second stage of the process involves learning more about the incoming hire: personality, interests, pastimes etc. to try and connect the candidate with employee groups that she will get along with. Finally, the last stage involves bringing new hires up to speed with the codebase the team uses, something that normally occurs during the first week of the internship.

Even though interns are allocated relocation and housing stipends, the onus is usually on them to find housing. A good way to do this is through Facebook housing groups, for example the San Francisco Housing and Sublets group. Company Facebook intern groups are also hotspots for housing news and roommates.

International students on F-1 visas should begin the process of applying for OPT. In fact, sometimes it is smarter to apply for it even before you have received your offer. You don’t need employer sponsorship to apply for OPT. Because OPT approval takes 90 days, it is recommended to apply by the first week of March for jobs beginning in the first week of June; note that this may be before all internship decisions (especially for startups) have been released. Be in touch with Amherst’s Center for International Student Engagement for guidance on this process.

Stay in touch with the people who referred you throughout the process! They have built a relationship with you and you want to guard them as valuable future resources too and updating them about your interviewing experience is one way to maintain a rapport. Also, keep exploring new languages, frameworks, and even new CS fields that make you curious. Join the AI club, for example. As long as you keep learning things because you want to learn, you’re in good shape to become an engineer.

**Interview Prep Material**

After reading the description of the entire interview process, there’s still (at least) one burning question left to answer: what, exactly, are the interview questions about? What kinds of problems are asked and what technical topics are the most important? This section is the response.

**Personality Questions**

Personality questions comprise all non-coding, non-CS and non-technical questions.

1. **The spiel question**

   The most important personality question is: “Tell me more about yourself.” This is the spiel question. To adequately answer it, think chronologically. Share your background with the interviewer, when you took your first CS course, how involved you have been, what you like about CS, why you want to join the industry, and tell some things you have done to show your interest in CS. Here was my spiel: “So I’m actually from Pakistan and I’d never taken a single CS course before college. I took my first CS course, Intro to CS, during my first semester at Amherst just out of curiosity. I ended up loving it so much I decided I was going to major in Computer
Science. I really enjoyed the way I could teach computers to solve problems and I craved the challenge of finding ever-better answers to problems that were posed to me. Over the next two years, I practically completed the major. In fact, I was also the TA for the Data Structures course offered and the tutor for Intro to CS. But my involvement with Computer Science hasn’t only been inside the classroom. I’ve taken courses online and have learnt a bunch of technologies on my own. Last winter I learnt web development and created a website called xyz. This summer, I learnt the MEAN stack and I created a web application that queries a database to retrieve a random quote by a French philosopher. I am also currently interning in BookingPal, Inc. a startup based in Irvine, CA,” after which the interviewer would usually jump into questions she had about my internship or other projects on my resume. The spiel doesn’t have to be perfect, but writing one may help if you’re not too much of a speaker.

2. The motivation questions

“Why do you want to work at Merkrith?” “Why do you want to become a software engineer?” These are classical motivation questions. Good answers to these will be of the form:
   a) Some kind of personal experience or values
   b) The experiences or values an opportunity at Merkrith/in tech would afford you
   c) why b) aligns with a).

For example: “I believe understanding data well can change the world. Moreover, I think many world problems, like monitoring child trafficking, can be solved if we had the tools to understand data well. Palantir does this kind of work. Its data analysis tools, Palantir Foundry and Palantir Gotham, allow organizations all over the world to understand data leading to successes like the effective distribution of disaster relief during Hurricane Sandy. I want to be part of Palantir’s data revolution. Palantir’s values align with mine and I would like to work there because I want to contribute to its mission.” You can spin this 3-step process into a solid answer to any motivational question. Note that it is especially impressive if you mention specific technical products the company is involved in. Recruiters will realize you actually care about the company.

3. The resume questions

“Tell me more about taylor4pres.com. Why did you use Angular and not React to program it?” or “Why did you create beliebersmya55.net?” An answer template for such questions is the following:
   a) Describe a problem
   b) Describe why technology could solve that problem
   c) Explain why you did b) to solve a)

For example: “Amherst has an extremely pretty campus, but every year the admissions office finds a shortage of student workers to give tours on campus. This gets counter-productive because so many prospective students interested in Amherst lose out on the opportunity of a guided tour that unveils the college’s history and culture. I thought technology would be a great way to solve this problem. What if I could make an app that gave an augmented reality tour of Amherst College? That way any prospective student could tour anytime she wanted and we would never have a shortage of tours on campus because people could simply download the
app. So, I created MammothRide.” Responses don’t have to be mind-blowing or


glass-shattering; you just need to come up with parts and string together a convincing answer.

Knowing your resume well is a good way to prepare for these questions in advance.

4. The experiential learning questions

These are questions like “What was the hardest thing about MammothRide?”, “Describe a time


when you have worked on a team and there has been a conflict. How did you resolve the


conflict?”, “What is the hardest bug you have ever faced?” and “What are some of your


weaknesses?” To best answer experiential learning questions, you should take some time out to


truly reflect on some of the experiences you have had as a CS major. In COSC-171 was John, your


partner, too busy playing squash to contribute to the final project? What about Rager’s final


project for COSC-112? Were you able to easily fix hard bugs that came up in the games? You


could respond to EL questions by describing the problem, explaining how you tackled it (did you


use the Eclipse debugger, did you find common time with John etc.) and conclude by showing


how your problem solving strategy actually worked. These obviously aren’t the only types of


experiential learning questions an interviewer could ask, but the questions are usually


structured identically: “what was the ___est ___ you encountered and how did you resolve it?”


A good answer to the “weaknesses” question isn’t something cocky like “My biggest weakness is


that I don’t have any.” I always replied that my biggest weakness was “being too perfectionist


about code and constantly obsessing over it, which was counterproductive in the long run.”


Being open helps.


Technical Preparation

To begin with, there’s no language compulsion during interviews. It’s not like Google is only accepting


interviewers who can code in Go, and rejecting everyone who doesn’t know the language. Interviewees


are free to code in the language they are most fluent in; they just have to tell interviewers what that


language is.


The vast majority of interview questions, especially those in online challenges and phone screens, will


involve **Data Structures and Algorithms**. They won’t normally be questions about DS & A but a deep


knowledge of both is required to successfully solve them. Here’s the kind of question you normally


won’t see on an interview: “Describe how insertion sort works.” Here’s the kind that you will see: “Write


an algorithm that, given a list of the ages of a population, provides you the median of the ages in O(n)


time.”


Some data structures you should be an expert on are Lists (Array Lists, singly Linked Lists, doubly Linked


Lists and circular Linked Lists), Stacks and Queues, Heaps (minimum and maximum), Priority Queues,


Trees (Simple trees, Binary Trees, Binary Search Trees and the basics of Red-Black Trees), Hash Tables


(Chaining and Open Addressing), Sets, Maps and Graphs (Adjacency List and Matrix representations).


You should also be familiar with the Collection interface and the Iterator interface, the Comparator


interface and the Comparable interface, and the Collections.java and the Arrays.java static classes. Here
is an excellent website that teaches you the different implementations of the Data Structures in the Java SDK. I learnt the ins and outs of DS from the CLRS book (Introduction to Algorithms, 3rd Edition), which is the CS bible for DS & A. Implementing Data Structures by yourself also really helps grasp their workings.

Knowledge of fundamental algorithms is also required to do well in interviews. Some algorithms you should know how to write are sorting algorithms (insertion sort, merge sort, quick sort, heap sort, tree sort, radix sort, counting sort and bucket sort), tree algorithms (search, add, remove, inorder traversal, preorder traversal and postorder traversal), map algorithms (get, put, containsKey, remove) and graph algorithms (Breadth-first search, Depth-first search, graph traversal and possibly Topological sort). You don’t need to know very specific algorithms like the Bellman-Ford algorithm for shortest paths or Prim’s algorithm or Dijkstra’s algorithm etc.

You should also have a good grasp on programming techniques and some other algorithmic concepts too, like runtime notation (Big-Oh, Omega and Theta, upper, lower and tight bounds and amortized runtime), runtime space requirements, graph runtimes and space needs, Inheritance, Access Modifiers, recursion, memoization and dynamic programming, brute force, and have some idea of greedy programming. Other important topics are Bit Manipulation, Memory Limits (of ints, doubles, floats etc) and Strings. System Design, Scalability and Testing, on the other hand, are hot subjects for on-site interview problems.

These DS & A and programming techniques should be all you need to know to tackle most, if not all interview questions. Practically all of this material is covered in COSC-211 and part of COSC-311. In fact, the Amherst CS major prepares students uncannily well for the technical interview. If you revise your notes from those classes, you should be in a very good position for interviewing. Nonetheless, this Github post and Cracking the Coding Interview can help you prepare for whatever isn’t covered by the CLRS (like Scalability and Testing techniques).

Once you have mastered the material, it’s time to practice it. Learning all these topics abstractly is one thing; applying them is a whole other ball game. This information pack won’t include practice questions, but there are sites that provide excellent ones, like HackerRank, HackerEarth and Leetcode, which even allow you to filter based on Data Structures so that you can develop expertise in a certain area. Cracking the Coding Interview, the “bible” of tech recruiting, also has many questions on different topics. A random Google search will return tons of practice resources too.

Doing all of this will get you a long way. If you study well, you will be a decent candidate. To set you apart from the rest of the field, though, requires avoiding common interview pitfalls. To ace the technical interview, you don’t just need the right answer - you need to interview well too. Recall the section on the phone screen: Ali is evaluated based on many criteria other than the correct solution. The next section teaches you how to interview for the tech industry; how to communicate, how to code and how to ace the interview itself.
Interview & Coding Guidelines

This section explains how to approach and solve the Coding Problem during an interview. To best illustrate different strategies, I'll run through with an example problem:

“Write an algorithm that, given a list of the ages of a population, provides you the median of their ages.”

Problem-solving approach

1. Ask questions to clarify the problem.

   The problem is most likely not what you think it is. So to understand the problem (which is crucial to actually solving it), you must ask your interviewer questions to clarify it. For example: “How am I going to be given the list? Is it an array or an ArrayList?” or “Will there be repeats?” or “Will the list be sorted?” and even “Will any of the numbers be negative?” It is important not to make any assumptions. You should question every aspect of the problem to be sure you understand the question identically to your interviewer. In fact, misunderstanding the problem is the biggest reason why candidates get rejected. Interviewers are happy to keep answering, so don’t worry about offending them by asking too many questions. Some key questions to ask are:
   a) Input data types and output data format: “What should the program output?”
   b) Some assumptions about the input you may have: “Is the list of ages ordered in any way?”
   c) What the output is supposed to be: “So, by median, I am thinking of the value at the middle of the data, once its ordered. Is this what you are thinking of too?”
   d) Any “edge” values -- Null or empty inputs. “What happens if the list is empty?” “What happens if the list is null?”
   e) Any other restrictions: “Can I modify the input list or do I need to create a copy of it?”

   Once you have clarified the question, it helps to explain the question back to the interviewer to see if you completely understand it. The amount of time invested asking questions pays big dividends, even if this entire process seems uncomfortably long. Note that the questions you ask will (obviously) depend upon the problem, but they will generally fall into one of the five categories above.

2. Run through an example

   On the online coder, write out a possible input and then trace the steps you would have to take to get to the correct answer. For example: “So I’m going to run through an example here. Suppose we have the list [10,0,4,35,60], the median would be to order the list → [0,4,10,35,60] and then find the middle-most value of the list, which is 10 in this case.” Running through an example is important for many different reasons. Firstly, it is confirmation you have correctly understood the problem. Secondly, it helps you come up with your first working algorithm. Thirdly, even though you are using the classical case of the problem at this stage, you understand the special cases of the problem you could encounter. For example, what would have happened if the length of the list was even? Or what if the list had been empty?
3. **Coming up with a solution**

Essentially, all coding questions require you to come up with some kind of algorithm to solve a problem, but it’s not always intuitive how to produce those algorithms. Running through an example usually provides you the first solution. It doesn’t matter if this first algorithm is an inefficient brute-force solution; what is important is that you have at least a correct solution that you can convey to the interviewer. You should then discuss the tradeoffs of this solution (the runtime and the space efficiency). Using quicksort, for instance, our first solution gives a runtime of $O(n\log n)$ and space requirement of $O(n)$. Brute-force is an excellent, even necessary, starting point. But companies aren’t looking for candidates with brute-force solutions and your interviewer will push you to find a more efficient solution.

4. **Iterate through 2 and 3 until the interviewer is happy**

Now you are at the point where you need to come up with more efficient solutions to the original problem. Here are some strategies I used to come up with successful algorithms (source: CtCI).

   a) Data Structure brainstorm
      Think of all the Data Structures you know. Is there one that would make the problem easy? Could adding all the ages to a Sorted Set (a Tree) help, for example?

   b) Graph approach
      Is there a way to model this problem as a graph, with, for example, every age becoming a vertex and an edge between consecutive ages? Would a BFS/DFS give you a meaningful solution? Think about the problem: “Given a list of the ages of people, find the sequence of 5 consecutive ages that has the least sum” - a graph approach could help you with this new problem.

   c) Recursion/Dynamic Programming/Base case and build
      Is the problem an optimization problem, or is there an optimal substructure to the problem. Does the solution to this problem build on solutions to identical smaller sized problems? If so, recursion/dynamic programming could help you. Consider, “given a list of ages, find the non-consecutive sequence of length 3 that has the maximum sum.” This is a dp problem.

   d) The similar problem approach
      Have you encountered a similar problem elsewhere whose solution could be adapted to this new problem? A problem similar to our example could be: “given a list of strings, find the string that lies alphabetically in the middle.”

Thinking out loud is crucial during this stage. Interviewers grade you on how well you communicate your reasoning too - so make sure to constantly convey what’s going on in your head. If you really need to think in silence, tell the interviewer “Ok, I’m going to think about this problem for a moment, but I promise to explain whatever I come up with.” An interviewer’s biggest pet peeve is a black-box: someone who is silent and whose reasoning process they have no idea about. If you’re struggling, the interviewer will give you hints. This normally looks like a nudge in the right direction, for example: “Think about different sorting algorithms. Is there one
that could give a faster runtime?” **Pay attention to these hints.** They are intended to lead you to the algorithm your interviewer is looking for - in our example, using counting sort provides us a O(n) algorithm.

Remember that any algorithm you propose must be correct - correctness is vital. An incorrect but very efficient algorithm is of no use.

5. **Pseudocode**

Coming up with an algorithm your interviewer is happy with means you’re halfway there. Now, you need to actually code this solution. An excellent way to begin is by writing pseudocode. Good pseudocode should capture the high level steps of your algorithm. The good thing about pseudocode is that you can easily go back and edit it if you find errors, something that is hard to do with actual code. In fact, going back and editing during actual coding leaves a negative impression on the interviewer. Writing pseudocode also gives you increased clarity about your algorithm. The more code-like your pseudocode is, the easier to convert it into actual code it later becomes. While pseudo-coding, make sure you are handling input and outputting in their correct formats.

6. **Edge Cases**

What happens when you receive a null or empty input? What happens in special cases of the input, for example when all the ages in the input array are identical? Not thinking about edge cases is the most common candidate pitfall; code written by heedless candidates can rarely account for the full range of possible inputs, **something which is expected of good coders**. So, it is worth taking a few moments to examine your algorithm and see if it fails to handle special cases of the input. You will often need to add a few lines of code to handle them. A further step is to ask whether you can optimize your solution for special cases. In our instance, for example, we could check if all the values in the array are the same and if they are, we would just output the value at the first index.

7. **Coding Tips**

Coding is the process of converting your pseudocode into compilable, executable and correct industry-level code. You basically need to rewrite your pseudocode into Java (or whatever language you are using). A key requirement is that **your code compiles**. Writing code that compiles is easier said than done, however, especially since online editors do not have autocomplete features like the Eclipse editor. To get the hang of closing your braces and putting your semicolons, you should practice coding on a whiteboard or on paper, especially since most on-site interviews entail coding on a whiteboard. I remember writing code in Google Docs and then copying that code to Eclipse to see where I had made errors. Revising syntax and structures and being fluent in the ins and outs of the programming language of your choice (something that comes with practice) are other excellent ways to write industry-level code. Code should be neat and logical; you should always use braces for if-else and while statements, even if the bodies of these structures only have one line of code. Concision also matters - if you can write
one boolean that covers 5 different if clauses, you should do that. Think about modulising tasks - don’t hesitate to create different methods to subfactor the work done in the main algorithm. In phone screens, you will almost always be writing one method, that you should give a short but precise name, like “medianAge.” If you find yourself writing classes, always use private fields and getter and setter methods. Don’t reinvent the wheel - be familiar and confident enough with the Java API to use in-built methods. Use the Collections.sort() method, for example, instead of writing a merge sort for ArrayLists. If you aren’t sure whether a particular method exists, ask the interviewer.

8. Checking and Testing your code

Once done coding, you should tell your interviewer you will check your code to make sure it compiles before your move forward. Common compiler errors: missing return statements, unclosed braces, missing semicolons and undeclared variables. You should also run through your code with a test input at this point to make sure it gives the correct output. This process should be “testing by hand” - you should literally trace the path of operations a test input would take when it runs through the program. **Attention!** Perform the operation your code actually performs on the input and not what you think it does.

9. Running your code

Some interviewers won’t ask you to do this, but the final step of the interview usually involves thinking of test cases and executing your code on them. How to come up with interesting test cases? By analyzing the problem at hand. A set of good test cases will include a range of possibilities: the standard case, the edge case, the null case and a large-sized case. In our median ages problem, here’s a good set of test cases: [], null, [0], [9,9,9,9], [1,2,3,4], [34,98,5,2,56,78], [4,3,5,7,1]. The cases are hard-coded and then passed to the program, which is executed. The output appears in the console. Fingers crossed…!

It’s completely fine if your program messes up, as long as you go back to the code and figure out what is going wrong. I’ve been in this situation many times, and it’s actually quite amusing: many times even the interviewer doesn’t know what the issue is. Then begins a collective search for the bug. 2 people are usually enough to find it...usually.

10. Modifying the problem

Often, the interviewer will revisit the problem to modify a question parameter. In our case, once finished with the original problem, the interviewer might say: “What if instead of calculating the median of the entire list, I asked you to calculate the median of the 5 largest ages? How would you do that?” The modifications might sometimes be challenging too, but interviewers want to see how you handle changes in the question conditions, something which comes up all the time in real-world engineering. You won’t have to write any code for this part but simply explain how to alter your existing algorithm to solve the modification, eg: “after counting sort, I would take the median of the last five elements in the List.”
Following all 10 of these steps might seem overkill - it’s actually not. Each of them checks a box on the interviewer’s feedback form. You should pace yourself through the stages according to what feels right - I always preferred to spend time writing pseudocode because it made coding easier; I know very capable interviewees who prioritized testing though. Here’s a handy problem-solving guide sheet I used while interviewing:
Problem Solving Approach

1) Read the question & note the details
2) Ask questions — think: what interviewer is trying to teach us?
   → Ask: are there any other assumptions/information given?
3) Talk about: get input/output systems.
4) Work through an example: → a straightforward way
5) Come up with a working solution & ask the interviewer.
6) Discuss the feasibility of the solution.
7) Decide by 4, 5 & 6 until you reach a plan the interviewer is happy with.

- Example:
  1) Base case held?
  2) Data structure maintained, that makes the problem easier.
  3) Graph approach.
  4) Dynamic programming.
  5) Brute force

8) Begin by writing header & then think about edge cases ⇒ null inputs, or bad inputs
   ⇒ “interesting” inputs
   Code: → simple variable names
   → Don’t reinvent the wheel (ask interviewer for help)
   → Code fast (like)
   → Don’t check for wrong!!! OR test first!!!!
9) Test: check if code samples (methods, variables, return type, types)
   → Work through a few test cases.

Modularisation of Tasks!!! Do it now! → Refactor code
   1) Delay implementation of helper functions
With the conclusion of our problem-solving approach, let's focus on general tips to help you throughout the process.

**General Tips for the process**

1. **Start early**
   Start now. You can’t hang around for another month before you begin the process because that makes a potential offer harder to get and pushes it back by a month too. The earlier you get started, the higher your priority on xyz company’s candidate list, and the more likely you are to get an interview. Reach out to people, prepare your resume, begin side projects and revise your data structures to get ahead of the game.

2. **Don’t memorize solutions**
   It’s not worth it. It takes too much time with very little payoff and you’re extremely unlikely to encounter a similar/the same problem. Which takes us to the next tip:

3. **Encountering an interview problem you’ve done before**
   What happens if you get asked a problem and you’ve come across exactly the same one before, either in a previous interview or while practicing? Should you confess you’ve solved it before or should you ace the problem anyway? This is a hard question with very divisive answers. There are those who claim owning up is the right thing to do. It tells the interviewer you’re an honest and integral person. Besides, even if you breeze through the problem with little difficulty, the interviewer would realize that you had encountered the question before. Then there are those who affirm that encountering an interview problem you’ve done before is a sign of how well you’ve prepared; you own the right to guarding your silence because you’ve studied really hard and deserve to do well on the interview. There’s no consensus here, but the former school of thought seems the more honorable to me. Nonetheless, this is only in the case where you’re seeing exactly the same problem again. If you encounter a similar problem, it’s completely fair to go ahead and solve it (**in fact, it’s an important problem-solving strategy**).

4. **Don’t panic if you see a hard question**
   Breathe. You’ve worked hard - leave it to your mind and you’ll come up with a solution. Just follow the routine of working through an example and using the different problem-solving strategies and you’ll find an algorithm that works.

5. **Practice Interviewing with other people**
   Like squash or painting, interviewing is a skill that’s best honed by practice. Hit up your peers (and that person you like who’s doing tech recruiting) and schedule interviews with each other. Then grade each other and provide feedback - learning how well (or poorly) you interview from someone else often dispels any false notions you’ve created about yourself. You can even find interview buddies online, through [Pramp](https://www.pramp.com), a platform that simulates coding interviews.
6. **Schedule Interviews when you’re at your best**

   I’m a morning person, so I scheduled all my interviews early in the day. It didn’t matter that my interviewers had to jump on calls at 9am; whether or not I was going to the next round depended largely on my performance, not on theirs. Make sure you have a decent block of time (around 2 hours) and pick a day/time when you’re not too stressed with school. Interview in a quiet place! Don’t go to 1st floor Frost; pick one of the study rooms on A Level or resign yourself even to your dorm room - silence is crucial to concentrate and the interviewer doesn’t want to hear your friends gossiping either.

7. **Unconventional Internship opportunities**

   Most internship programs, like those offered by the Big 5, are intended for Juniors in college. This is also true for medium to large sized companies, and even banks. However, several programs target Freshmen and Sophomores directly, like the Microsoft Explore and Google Engineering Practicum Intern program. Other companies sometimes have diversity-based recruiting programs, which may be a huge boon if you qualify. It’s worth doing some research into this.

   **Good Luck!**
Sources

Cracking the Coding Interview, Gaayle Laakman McDowell 3rd Edition. (good for practice questions and introductory readings but not an in-depth guide).

Getting a Gid - Cassidoo https://github.com/cassidoo/getting-a-gig (helpful for an overview of the process, not an in-depth guide either).