r/cscareerquestions • 5 yr. ago
[deleted]

A /r/cscareerquestions College Survival Guide

A r/cscareerquestions College Survival Guide

With our final school year beginning, we were reflecting on how lost and confused we were when we first started university. We made a lot of mistakes (still am) along the way, but we're in a much better place after learning from them and constantly putting ourselves out there. In hopes of shining some light and helping others, we decided to make a comprehensive guide for university students – based on our knowledge/experiences - on how to start your successful CS Career (or gain the wisdom to avoid CS altogether, more on this later).

u/rishiss Background: I am a 4th year student at UC Irvine majoring in Software Engineering. I am an incoming Software Engineer at a F100 company (received return offer after interning this summer). Before that, I interned at an R & D center for space, a small cloud company, and a small IT company. I have a 3.65 GPA, won a few awards at startup competitions/hackathons, and remain pretty active in my schools CS organizations.

u/chaitu65c Background: I'm a 4th year student at UC Irvine majoring in Computer Science. I'm currently a SWE Intern at a Unicorn and just wrapped up my 2nd internship at a Live Streaming Company(you can most likely tell who they are if you browse my history LOL). Before this, I interned at my school's IT department, did research under a professor, and worked on a few small startups that other UCI students were building. I have a 3.3 GPA, won some awards along u/rishiss and was pretty active in my school's CS clubs.

Disclaimer: "But u/rishiss/ and u/chaitu65c, you don't work at a Big N, go to a target CS school, why should I take your advice?" You're absolutely right; we are, by no means, 'up there' like some other folks on this sub. And, you don't have to take our advice! Simply close this tab and do whatever else you want 😊 . Our intent is to guide and prepare uni students for a CS career they enjoy, not work at Big N or get the highest TC. Life is much more than a dick-measuring contest, and the earlier you learn that the better.

We have also created a guide with our own personal advice/stories

Please, take this advice with a grain of salt. we're not Tony Robinson or Tim Apple, we're just two random reddit users.

Table of Contents:

This guide is divided into the following sections:

- Junior Year
- Senior Year
- Searching for Internships
- Searching for Full Time Jobs
- Final Thoughts

Is CS Right For Me?

The way we see it, there's 3 types of people pursuing CS.

1. Those who know CS isn't for them – They're in it for the money, to appease their parents, for a minor/requirement, some external factor. They hated programming while taking the introductory CS course and just try to get done with their class/degree ASAP.

Advice: The majority of people who fall under this usually burn out quickly, as they aren't motivated enough to learn the material and to apply themselves. This usually leads to them cheating and getting kicked out of their major, minor, or university altogether. Even if you manage to earn a degree, we've seen a large number of these folks endure a 'pre-mature' mid-life crisis or simply get fired from their jobs. Before you even start this major, you should definitely understand that this isn't going to be easy, and you do have to put in a lot of effort to succeed. If this isn't your cup of tea, definitely look into switching into another major you like.

Some folks are really passionate about technology, but don't want to pursue an entire Computer Science major or see themselves as Software Engineers. That's completely ok! Try looking into related majors or minors. We know many students who switched from CS to majors like Informatics, Business Information Management, and Economics and are thriving in tech-related roles like Data Analytics, Product Management, UI/UX Design, and Technical Recruiting. CS is not (and should not be) for everyone, and there is no shame in having the wisdom quit and move on.

2. Those who don't know if CS is for them – Where most of the CS community is IMO. These folks (like me, u/rishiss) are riddled with something called Imposter Syndrome: "the constant feeling of not being good enough or knowing enough to do your job well."

Advice: For students, really take the time to learn and be open to anything you go through. Try sticking it out until you've taken a Data Structures course, one of the harder, more important courses out there. If you're not understanding the material or just aren't having fun with it, it's definitely ok to switch majors/careers. Otherwise, CS just might be the career for you! Give it your best shot!

Admittedly, it's hard to provide stronger insight to overcoming Imposter Syndrome, as I am afflicted by it as well. For me, my IS derives from constantly comparing myself to others and confusing inexperience with incompetency. As such, I continue to work and focus on myself and take baby steps towards smaller goals I set out for myself. Knowing that I've put the effort to improve myself by just 1% everyday has made me a lot more confident.

3. Those who know for a fact CS is for them – The diamonds in the rough. Learning and practicing CS material gives them a euphoric high.

Don't limit yourself!

Freshman Year

We recommend not taking more than 3-4 classes in your first quarter/semester, as you shouold keep an ample amount of time to go to professional/social events, make new friends and hang out with them, and pursue your interests.

We've seen a lot of freshmen (and upperclassmen) CS folks get cooped up in their dorm rooms playing video games and watching TV. We understand that these two are a passion for many, but please be cautious to not get consumed by them.

You have the privilege of pursuing higher education, making valuable connections/memories, and setting up your CS career in the trajectory you want. This year is the best year to take advantage of all that university has to offer; make the most of it.

One of the best ways to get involved in your school's/region's CS community is by joining clubs like ACM and WICS and participating in hackathons (see 'Hackathon' section below). Try pursuing internships and positions in these organizations and events as well!

One, major issue we see with freshmen (even upperclassmen) is their ignorance on all the avenues available in the CS Industry. So we've tried to narrow it down (not exhaustive).

1. Cyber Security Engineer
2. Front-End Web Developer
3. Backend Web Developer
4. UI/UX Designer
5. DevOps/Cloud/Site-Reliability Engineers
6. Mobile Engineer
7. QA Engineer
8. Product Manager
9. Data Scientist (Machine Learning/AI)
10. Embedded Software Engineer
11. Systems Administrator
12. Database Administrator (The Wizards)
13. Networking Engineer
14. Hardware Engineer
15. OS Developer
16. Video Game Developer
17. Solutions Architect/Sales Engineer/Technical Account Manager

As a freshman, definitely take the time and see if you can picture yourself doing any of the listed fields. You should open yourself to all facets of CS and not just the "hot field" like Data Science and Machine Learning.

Our recommendation is to select the top 5 fields that have piqued your interest and experiment with the field. For example, if you are interested in Mobile App Development, try learning how to build an Android app from the ground up. A simple weather app or alarm clock is completely suitable for a first project/prototype. This lets you understand what skills you would need for this field and can serve as a forecast as to what your career would look like.

You should definitely look for an internship. Ignore the people that tell you to wait until you're a junior, as it's going to be very hard to get an internship if you don't have any experience. Common places that most students don't realize are available are usually IT departments at your school and even research with professors. Researching is highly recommended as you can definitely learn more about a field you can be interested in and if you're interested in graduate school, that's going to be a letter of recommendation that you can ask for.

If you're considered a minority in Computer Science, look into first and second year internship programs as they're meant to help you succeed. Here's some programs that come to mind:

Google STEP

Microsoft Explore

Amazon Future Engineer

Uber STARInternhip

Facebook University

Another way to get internships is to research into smaller companies in your area. If the company is very small (<100 employees), consider reaching out to the CEO on LinkedIn. They might be able to help you! Also, take advantage of university recruiting websites like Handshake to see companies that directly hire from your school. More info on how to get an internship in the 'Searching for an Internship' section.

Sophomore Year

Now that you have basic programming knowledge, create your own website or GitHub account and start contributing to them with small personal projects. Nobody expects you to make a full-stack MERN project hosted and scaled on AWS at this stage. Focus instead on clean code, learning a framework or two on a language you like, and creating a small, robust feature. Grow from there!

If you weren't able to find an internship/research opportunity as a freshman, community involvement, projects, and hackathons become especially important, as they are a great way to make you stand out on your resume and to recruiters when you reapply. As you brush up on your skills, apply again, and try your luck out.

Data Structures and Algorithms

In addition, you are most likely to take a Data Structures and Algorithms course this year. Make sure you are focusing on this class and writing good notes; you will need this knowledge when interviewing for internships and full-time jobs in the near future. Here is a link to our DS and A course (in C++) for reference

full-time employee after graduating. This transition process is much easier than interviewing, and they'll usually offer you a higher compensation package if they want to convert you to a full-time employee. As you now should have knowledge of Data Structures and Algorithms, we highly recommend looking into coding interview prep sites like LeetCode and HackerRank or purchasing a prep book like CTCI or EPI (advanced).

Continue to attend hackathons, remain active in clubs/organizations, and grow your portfolio.

Classes will be much harder; expect the time for completing projects to double and the content covered to be much more difficult. We recommend taking no more than 2-3 upper-division CS courses and balancing your load with 1-2 GE classes. You should not be taking more than 16 units (assuming 4 units per course).

Start to get an idea of what field in CS you would like to pursue. Research what it takes to be successful in that field. You can do so by looking up job postings with that title on LinkedIn and looking at the requested skill set or take a look at Roadmap.sh. If you want to learn more about a related skill set and your school doesn't offer a course, consider picking up a class on udemy.com.

Senior Year

Focus heavily on your senior capstone, project classes, etc. as they're the last thing you can put on your resume before applying for full time. By now, you should have at least 3 polished, working projects on your GitHub that you can easily talk about with your recruiter. Preferably, they're aligned with the CS field you wish to enter.

If you were able to get a return offer from an internship, congrats! However, don’t immediately sign the offer. Once you have an offer, you should still try to interview at companies that you’re interested in by the deadline of the time to accept the offer. A good way of doing this is to reach out to a University recruiter for that company and explain the deadline you have. Usually, they’re really helpful and can potentially help skip interviews that you were supposed to do!

In addition, if your friends were able to intern at places you’re interested in, definitely ask for a referral or to send your resume to their recruiter. This usually reduces the risk of being ghosted by that company and increases your chances of getting hired!

Once you finally sign, definitely take the time to relax and enjoy. Just make sure you pass your classes and stay out of trouble

Classes:

What Classes should I take?

Should Already be Required:

- **Programming in Java/C++/Python (OOP)**: This is how you’re gonna start coding.
- **Boolean Algebra/Discrete Math**: Teaches you some background knowledge to CS.
- **Data Structures and Algorithms**: Teaches you some ways data is stored and retrieved. Very important as you’re going to be using them a lot.
- **Low Level Programming /C**: Teaches you what coding used to be like in the old days.

Must Take:

- **Computer Networks**: Highly recommended as it helps you understand Web protocols like HTTP, TCP, UDP, etc.
- **Operating Systems/UNIX**: Most important class. Teaches you important things such as the kernel, Threads vs Processes and Process Schedulers.
- **Databases/SQL**: It's very likely that your job as a software engineer will be to interact with databases. It's really good to understand what they are before you enter the industry.
- **Programming Languages**: Teaches you trade offs between languages like C and Python. It definitely helps when you need to pick up brand new languages!

Good to Have

- **Full-stack web dev** (pref MERN stack, our school offered both LAMP and MERN)
- **Very deep understanding of at least one language.** (You'll be surprised to learn how many students who graduate fail to do this)/
- **Semester/Year-Long Capstone course (pref working with a company) if your school supports it:** an internship where you get school credits instead of money.
- **Compilers**: Teaches you how programming languages are implemented 'under the hood.'
- **Human Computer Interaction**: If you weren't able to learn Full-stack web dev.

How do I succeed in these classes?

u/rishiss: You're more than likely coughing up hundreds, if not thousands, to attend university. It makes no sense to not take full advantage of the course and course staff.

- **Do the readings beforehand** – Dr. K explains how studying before class is an OP mechanic, also highly recommend this video on how to study by him as well
- **Attend EVERY lecture, sit at the front of the class** – I've seen a metric fuck ton of students in the back of the class with their laptops shopping, trading stocks, scrolling through Reddit, even watching lesbian hentai. By sitting in the front of the class, you're forced to stay engaged (and close your porn tabs).
- **Take notes**– People have many, different ways of taking notes; stick with what works for you.

The way I take notes: I learn from examples; I want to enter my code into the IDE to see what happens. I do a three way split; Google Docs on the left, IDE on top right and terminal (to compile, see output, make new file, etc) on bottom left. I note down the date and topic of the lecture and write questions I have in the comments on Docs. I make sure to highlight important information and possible test questions. I even share the link with friends!

- **Make a study guide, even though the professor does not give you one** – Using my Google Docs notes, I compile the highlighted information into a summarized study guide. It's a fantastic review tool. I've even shared the study guide with professors/classmates and gotten their feedback and extra information.
- **Go to office hours and become close with 1-2 professors** – Some jobs and most masters/PHD programs require letters of recommendation. While you could get reccs from your work, its great to get a

research, letters of recc, or even the opportunity to pursue a PHD under them.

- **Make friends!** - Classes are a great way to meet new people with similar interests and expand your professional network. They can especially be a saving graves if you miss a class, don't understand a topic, and need motivation to prepare for an exam. Don't be afraid to say hello!
- **START EARLY ON ASSIGNMENTS** – I can't count the number of times starting an assignment early saved my ass. Starting early gives you time to deal with the unexpected: the family emergencies, the late night hangout with friends, the memory leak on line 74. Procrastination is like playing Russian Roulette with your CS career, don't take the chance. A helpful video on procrastination

In the quarters where I followed the steps above, I never got a grade lower than an A-.

Dealing with Bad Professors

During your time in college, you're likely going to have at least one bad professor that might make it worse if you have to go to class. If that's the case, it's definitely fine to not go to class (as long as it's not mandatory). However, if you do decide not to go, you must make sure you learn the material, so you won't be behind on the coursework and studying for tests. In addition, you should be doing something productive on the side. If you don't go to class and spend the time watching Netflix or playing video games, you're losing time that you can spend on something that might be fun and can help you in the long run.

You can take Graduate Courses!?

u/chaitu65c: A highly underutilized set of courses you can take would be graduate courses. Graduate courses are usually very specialized in certain fields. If you were able to take all the undergraduate courses you wanted and still have spare classes to fill out, I'd recommend researching into taking Graduate courses! They're a good way to build out your specialization and learn new, cool stuff! In addition, if you're looking for classes to reach the required number of CS courses needed, your CS department might allow you to make the course count towards your degree!

Projects

They're super important.

How do I succeed in class projects?

- **Reading the Project Requirements:** Before starting to code, read the requirements and understand what you need to do in order to finish. Too many students ask for help that can easily be found in the requirements which wastes the student's time.
- **Learn to Debug:** Learning how to debug saves you countless hours trying to read through code you might not understand.
- **Learn to write Clean Code**: With this and being able to debug, you're going to be able to write very efficient code and to debug issues easily, thus not making you only successful in lab assignments, but also making you a better programmer.
- **Plan your work out:** This allows you to simplify the logic you are writing and this usually leads to clean code.

- **Draw Pictures:** If your project involves multiple things (AWS, Databases, Servers, etc), it's definitely a good idea to draw a picture that shows you each thing interacts with and how it comes together. This is important especially when you enter the industry and build software for companies.

Personal Projects and your CS Career

u/rishiss: Projects are your saving grace, especially if you are lacking work experience. They show technical aptitude, willingness to take initiative, and leadership. I've seen people with only projects on their resume get positions at the Big N. Projects are good ways to expand your knowledge of CS as the possibilities are endless! It is best to have a variety of projects dealing with a variety of technologies. As such, you can open yourself up to more positions and have more talking points during the interview.

I tend to edit the 'Project' Section of my resume with relevant projects and technologies. For instance, if I made a full stack web application and applied to a DevOps organization, I would highlight my AWS, CI/CD, and Terraform experiences more than my React/Node js work.

It is recommend the project is about something that motivates you and are passionate about e.g. video games, movies, books, sports, etc., as it is very easy to give up half way due to stress or lack of motivation/interest.

Like anything else in Computer Science, projects require you to break it down into smaller pieces. Start with the end in mind and draw out the intended architecture/functionalities. Start with what you know and research on the parts you don't know after that. You will be using these skills often in industry for any project/feature planning.

Spending 15-30 minutes a day is all you need to make a successful personal project. Don't make excuses and get coding!

Open Source Contributions

If you've ever noticed popular github repositories such as torvalds/linux, these are repositories where people from all over the world can report issues with it and someone can fix it. If you are able to make a contribution to a huge open source repository, it looks really good on your resume.

Hackathons

What are Hackathons?

Hackathons are large scale coding events, where students from around the area come together and collaborate - usually in teams of 4 (but you can go solo or with a partner!) - to build some software. Companies like Amazon, Northrop Grumman, Google, and Twilio sponsor awards related to best use of their technology. After 24 - 48 hours of intensive coding, participants submit their projects, whether it be an Android video game, Chrome Extension, productivity web app, etc. Submissions are shared with the companies and other hackathon organizers, where they select the best projects and award teams with swag like keyboards, gift cards, and even summer internships at their company.

questions you may have. Winning awards at these hackathons are also great resume boosters and talking points during interviews.

The biggest hackathon organizer is Major League Hacking. Visit their website, and you can see all the hackathons (remote or local) they are partnered with. Make to be on the lookout for application release dates from the hackathons and apply early.

With Covid, you may miss out on the free goodies and the in-person networking with students and professionals. However, most hackathons are accepting many more applicants due to it being virtual/remote this year.

What Should I Do At Hacakathons?

Take advantage of the resources available at hackathons. You're attending a mini CS conference and should be, besides coding, networking with professionals, learning about the different companies, attending workshops, asking technical/non-technical questions to mentors, and getting as much free shit as you can get. Besides T-Shirts, companies give out vouchers to their services, applications to their internship and full-time positions, pillows, notebooks, water bottles, sweaters, and even backpacks.

If you're looking to get an award, judges at hackathons care a lot about the pitch and the idea rather than the actual execution of the idea. Having an idea beforehand is also helpful, so you can spend your time focusing on the MVP.

Friends and Networking

u/chaitu65c: I think it's definitely useful if you have two different friend groups: One dedicated to career and Non-Career Group.

Career Group - When making a friend group dedicated to career, try to be the dumbest person in the group, you're definitely going to learn a lot from them as you soak up knowledge! Best ways of meeting friends who are career-driven can be through major specific orientation (actually how I met u/rishiss), courses, major related clubs, etc.

Non-Career Group - While having a group that motivates you for your career is important, it's also important to have another friend group that can help you relax and to enjoy your time! A really good way to find these friend groups can be anywhere from your hall to General Education courses, social clubs like Circle K, fraternities/ sororities(if that's your cup of tea) and others!

This is what has worked for us; no need to follow this exact format.

Resume

u/rishiss: Here are the few take-aways on writing a resume that gets through the ATS.

- **Make your resume accomplishment driven, not just a list of your responsibilities** – This guy puts it best TL;DR: Your bullets should be in the format -> Accomplished X by doing Y as measured by Z.

- **Make it a simple, one-column that recruiters can easily read through.** There's no need for pictures, graphics, colors, fonts, etc; the ATS can't parse this! Overall, keep it simple; the content should be carrying you.**Exceptions:** The company you are applying for is small, you're going to a career fair and you know your resume will be hand-read, you're a UI/UX person or a Graphic Designer.
- **Have at least 3 minimum (I aim for 4-5) per work experience/project**; it makes no sense when you have such amazing experience and only put two bullets.
- **If you have a GPA lower than a 3.0, do not bother keeping it on the resume.** If you have a 3.5 +, make sure to keep it.
- **Make formatting consistent.** This should be a no brainer, but I still see folks' resume with different fonts, spacing, etc. It's annoying; don't do it.
- **Focus on individual contributions and leadership, not the team.** Recruiters are looking for self-starters and leaders that can see a project throughout the life-cycle, not just another code monkey.
- **Expect to be tested on anything you put on your resume.** If you don't think you can answer questions about a skill, tech, or experience on your resume, don't bother putting it in.
- **Make sure to add these items in your resume:** Name, School, GPA (unless its less than 3.0), Work Experience, Projects, Skills (one line for languages, one line for tools/platforms are what I've seen the most), relevant links (GitHub, Website, Portfolio).
- **Take out any old or irrelevant experience.** Nobody cares about that Tic Tac Toe game you made in high school.
- **Get your resume reviewed multiple times by experienced people in tech.** r/csMajors and r/cscareerquestions also has a weekly resume roast thread that you can take advantage of.

Searching for Internships

Searching for internships in CS is really different and harder from searching for internships in other professions. CS internship interview processes are often longer and much more technical on what you have learned as a CS major. We've prepped 2-3 months beforehand on CS concepts, whiteboarding, etc.

Timeline

This timeline primarily focuses on large, non-government/defense companies or competitive startups. This also assume you are applying for a summer internship.

August - September: Applications are opened to the public. Make sure to look out for positions and apply early, as most companies admit students on a rolling basis. A site that we used often is Apply.fyi. After applying, you may receive an automated (< 48 hours) invitation to complete an Online Assessment, consisting of multiple choice and/or coding questions about Data Structures, Algorithms, and Run Time Complexity. You will have usually 1-2 weeks to complete the assessment. Please that you may be rejected if you are not able to pass 90% of the questions on the assessment: Please also note that you may be instantly rejected due to things out of your control like years of experience, cancellation of internship, internal corporate issues, and more. Don't take rejections too seriously; just keep applying!

October - November: After passing the resume screen and the OA, you will be contacted by the company's recruiter for a phone screen. During the screen, you will probably be asked a few confirmation questions about your resume, sponsorship, years of experience with X, etc. and minor behavioral questions like what

cringe/edgy), this part should be a breeze.

November - Mid January: If you made it through the two Thanos snaps, you will be invited to an onsite "Power-Day," where interviewees attend 2-4 whiteboard interviews while being grilled on their technical skills and projects. Some companies make applicants go through a panel interview, where a team of 2-5 Software Engineers grill you on technical questions and your resume. You are often pampered with free travel, food, stipends, etc.

December - February: If you were deemed a good fit by the hiring committee, you will be extended an offer to intern at the company during the upcoming summer for 10-12 weeks. Remember, nothing is final until you receive an offer letter in your inbox. Some companies may also place you on a wait-list and offer you a spot if someone were to reject their offer letter.

For government orgs, defense companies, and smaller organizations, the recruiting season starts in February/March and usually ends in April and May. After applying online and passing the resume screen, you will usually be immediately pushed to an on-site interview. Most likely, you will be interviewing with your future boss/co-worker.

Please note that internships are not only offered in the summer, they are provided in the Fall, Winter, and Spring (rare) as well. The competition for these internships is usually lower, and the process usually starts 3-4 months beforehand.

How to get the Interview

Besides following resume tips, make sure to apply to as many places as you can. To get our first internships, we recall applying to approximately 250-300 places before we secured our internship plans for that summer. Also, if you do get ghosted, don't take it personally, usually, university recruiters often spend so much time reviewing a lot of applications.

Other precautions to take to get noticed are to try attending career fairs if you can, you might be able to get an interview(worst case, free swag!). Other than that, try reaching out to upperclassmen or friends you know that interned and ask for referrals. It's one of the best ways to get noticed!

What to expect

As part of the interview process, there's 4 types of interviews that you should make sure you know.

Behavioral Interview: These interviews ask you questions about culture fit such as "Why are you a good candidate" and "Tell me about a time when you ..."

Coding/Technical Interview: These interviews ask you questions similar to what you see on Leetcode and Hackerrank. These interviews are designed to test your Data Structures and Algorithms knowledge.

System Design: System Design involves the interviewer testing your building to design a service/software and test your knowledge of understanding what things to use for the task and how you will integrate them together. You're definitely not expected to know this and it's not likely you're gonna get asked this. Places that could ask you this are Unicorns, Trading Companies and Hedge Funds, and Big Established Companies.

Some companies may adopt only one of these interviews and some may adopt all.

How to Ace the Interview

It's highly recommended that you look up the interview experiences that other students have faced so that you can potentially filter out companies with red flags and know what questions to expect. Common sources to search up on this would be Reddit (r/csMajors and r/cscareerquestions), Jumpstart (Relatively new portal for students), Glassdoor and maybe Blind (Aside from the toxic TC or GTFO culture, they do give good advice on interviews). With that said, here's some advice we have when you approach each kind of interview we've seen.

Advice on Behavioral Interviews

Use the STAR method when describing your experiences. Being quantifiable with the impact of your actions will impress the interviewer.

Advice on Technical Interviews

Begin by reviewing your notes from the Data Structures and Algorithms class. Do not proceed further until you know how to implement these DS and As from scratch with the language of your choice(If you do know python, it's recommended as there's a lot of builtin features!). After doing so, we highly recommend a book like CTCI and EPI to gain a review on programming language details and your DS and As. Then, visit sites like LeetCode to practice real questions from major companies. A Facebook Engineer completed 600 LC problems and compiled the most important ones into a list here. During the interview, make sure to talk out loud about possible approaches and tradeoffs before whiteboarding. It is perfectly acceptable (often recommended) to ask the interviewer to ask questions about the problem and get clarification. Once you have an idea in mind and have talked about it with your interviewer, begin whiteboarding. While you talk about the final idea you want to use, write out pseudo code and comments about all the steps you need to implement in order to finish coding your solution. After that, start coding. Make sure to have proper function headers, syntax, spacing, classes/structs, imports, etc. After coding your solution, give a brief explanation and attempt to make it run with less space and in less time (if your solution is not as efficient as you think it can be).

Advice on System Design

These are somewhat hard to approach if you don't have experience ever doing it. If you do have experience designing and building services in your spare time and as part of your work experience, definitely rely on your experience. An important thing is to definitely ask clarifying questions. There might be hidden requirements you didn't think about that could drastically change the way you approach the solution.

Advice on Concurrency/Low level

Understand basic principles such as Processes vs Threads (A lot of people don't know the difference!)TCP vs UDP and how to make an application thread safe. Other than that, it's recommended that you familiarize yourself with basic OS concepts such as Deadlocks, locks that you can utilize to make an application thread safe, etc.

The process for finding a Full-Time Job is usually very similar to finding an Internship. There's three main differences are:

1. **Harder Questions**. Ex: Google usually asks Leetcode Mediums to Hards + the special Leetcode Hard question that Google asks it's applicants (they create a new one every year).
2. **More Rounds of Interviewing**: For example, Microsoft makes interns do 2 rounds while New Grads do 4 rounds during the onsite part of the process.
3. **Compensation**: Interns usually get an hourly rate and, possibly, a housing stipend. New grads, however, are given a yearly salary and, possibly, a sign-on bonus, stocks, and benefits e.g. health insurance, vacation days, etc.

The process for finding a Full-time Job won't really change as much as finding an internship, but keep in mind that the bar is higher. This is probably the biggest reason why you should look into interning early; by getting an offer at the place you like, you don't need to go through the daunting process of finding a full-time role.

Get as many offers as you can this time around, so you can negotiate and select the position, company, compensation, and location that works best for you.

Negotiation

Negotiation is a really powerful tool that you can use in the interview process, even as an intern. There's a lot of guides to negotiation and we recommend Nick Singh's guide (Look at his LinkedIn and newsletters) for more.

Final Thoughts

University is a probably the most important time of your life and a foundational block of your CS Career. Like any foundation, it must be sturdy and takes a tremendous amount and energy of time to develop. Take advantage of all the resources (like this one) you can get your hands on. Definitely learn from the mistakes people have made and make sure you don't repeat the same mistakes.

'Stay hungry. Stay foolish' - Steve Jobs