

# **Technical Interview Prep and Final Exam Review**

**Welcome back to CS 2100!**

**Prof. Rasika Bhalerao**

# Final exam topics

- Properties (Q2)
  - `_ / __`
  - `@property / *.setter`
- Lists (Q2)
  - List comprehension
  - Sorting / filtering
- Abstract methods (Q3)
  - `@abstractmethod`
  - Instantiating classes
- Inheritance (Q3)
  - What is inherited
  - Overwriting inherited methods
  - `super()`.
- UML diagrams (Q3)
  - Abstract class / method
  - `+ / -`
  - Relationships
- Privacy (Q3)
  - (Un)intended recipients
  - Tradeoffs
- Iterator (Q4)
  - Iterable / Iterator protocols / interfaces
- CCE (Q4)
  - Detecting / mitigating coupling
  - Enhancing cohesion / encapsulation
- Comparable (Q4)
  - Comparable protocol
  - Inconsistencies
  - Using `< , >`, etc.
- Recursion (Q4)
  - Tracing / finding bugs
  - Relative efficiency
- MSTs (Q4)
  - Definition
  - Kruskal's Algorithm

# Prof. Jeongkyu Lee's Technical Interview Prep



Northeastern  
University  
New York City

Last updated: 11-20-2025

# The Technical Coding Interview



What It Is (and Isn't) + Tips & Tricks

Prof. Jeongkyu Lee

*Contributors since 2022: Eunji, Yiling, and Nick*



## Agenda

1. Overview of Job Application Process + Tips
2. How to Pass Your Interviews
3. Study Tips and Resources
4. Mock Technical Interview @Lee's class

Focusing on **Tech Coding Interview** Today!!!



## Agenda

1. Overview of Job Application Process + Tips
2. How to Pass Your Interviews
3. Study Tips and Resources
4. Mock Technical Interview @Lee's class

## Overview of Job Application Process



Written  
Application



Online  
Assessments



Technical/Behavioral  
Interviews



Offer!

## Overview of Job Application Process



Written  
Application:  
Resume

- Before sending in any applications, you need to **make** and **optimize** your resume
- Most companies use AI or other software to filter out majority of applicants before any human actually sees it, so resume needs to be able to pass this filter before anything else
- Use an **ATS-friendly** resume template (personal recommendation is [Jake's Resume](#))
  - Formatting should be **clean** and **minimal**, no pictures or hyperlinks, should fit on **one page**
  - Content should be **described concisely** (no wasted words or lines) and **quantified** (specific numbers describing performance / impact). Also consider **bolding** key words and numbers

## Overview of Job Application Process

- How to optimize your resume content?
- Each experience / project should have 2-3 bullet points describing concisely what you did, how you did it, and what impact you had, preferably with numbers (!!!)
- What to avoid:
  - Vague phrases like "Responsible for...", "Worked on...", "Contributed to...". Just say exactly what you did instead!
  - Buzzwords like "team player", "detail-oriented", "utilized best practices". Your descriptions should convey these!



Written  
Application:  
Resume

Recommended format for each bullet point:

**did X by doing Y, resulting in Z**

where Z is quantitative impact

## Overview of Job Application Process

- Example of **bad** resume line:

Responsible for improving the performance of the company's web application using best practices

- Why is it bad?
- Example of **good** resume line:

Reduced page load time by **60%** (from 5s to 2s) by implementing **lazy loading, code splitting, and migrating from REST to GraphQL**, improving conversion rate by **12%**

- Why is it better?
- Tip: LLMs are great at editing / evaluating resumes and finding weaknesses in technical descriptions



Written  
Application:  
Resume

## Overview of Job Application Process



Written  
Application:  
Applying to Jobs

- First thing to internalize: this is a **numbers game**
- Even the most qualified candidates with the best profiles get mostly ignored and rejected, so don't take any of it personally! Just take it as feedback that you can use to improve for future opportunities
- The goal is to maximize your chances, but nothing is ever guaranteed or even likely

## Overview of Job Application Process



Written  
Application:  
Applying to Jobs

- Look on [Github repos](#), [Handshake](#), [Northeastern job fair](#), [Khoury Career Peer Advisor program](#)
- Use school name to connect with alumni at companies you're interested in (over LinkedIn or email or other contact)
- Go to company recruiting events and job fairs to talk to recruiters!!! These are huge for getting interviews!
- Ask friends/classmates/alumni/TAs for **referrals**
- Start applying as early as **one year before** the position you're applying to starts!
  - For summer internships, start as early as the previous August.
- Track everything in a spreadsheet



## Overview of Job Application Process



Online  
Assessment

- Many companies will first send an **online assessment**, which is an **asynchronous virtual test**
- Usually 2-4 problems in 1-3 hours, typically Leetcode easy-medium, but some can go up to hard
- Goal is to pass all test cases by any means necessary
  - Do not worry about style or documentation
- In many cases (not all), runtime is less important than correctness
  - Get a correct solution first, see if it passes all test cases, then try to optimize if you have time
- Common platforms: [Hackerrank](#) , [CodeSignal](#), etc.

## Overview of Job Application Process



Technical /  
Behavioral  
Interviews

- You've gotten the interview, congrats! You're halfway there!
- Can have anywhere from **1-6 rounds of interviews**
- Some companies want you to write runnable code, others do not care as long as approach is correct
  - Be comfortable with syntax either way!
- More companies are starting to ask **object-oriented design** questions and test your understanding of **fundamental CS concepts**.
- Rest of the lecture will be dedicated to explaining how to pass interviews

## Overview of Job Application Process



Technical  
(coding)  
Interviews

- Interviewer poses a problem, which you have to solve by writing code
- Should be **collaborative**: ask clarifying questions (!!!), use your interviewer to help you along, explain thought process before coding, ask "Does that make sense?", "Does this look good?" to make sure you're on the same page
- Can be multiple parts that build off of each other, so need to be thorough but also time-efficient



## Overview of Job Application Process



Technical  
(Coding)  
Interviews

- Usually over **shared coding environment** (such as [CoderPad](#))
- Some companies want you to write runnable code, others do not care as long as approach is correct
  - Be comfortable and consistent with syntax either way!
  - You will usually be allowed to look up documentation or ask questions about language specifics
- Pick a programming language you like and master it!
  - **Python** is usually easiest for coding interviews but some roles / companies may be language-specific
  - Most important thing by far: **PRACTICE, PRACTICE, PRACTICE!**

## Overview of Job Application Process



Behavioral  
Interviews

- Two types: recruiter calls (introductory interview, low stakes) and engineering manager interviews (usually final round, high stakes)
- Questions are about your previous experiences, like "Tell me about a time you were put on a tight deadline", "Describe a situation where you had to work with a difficult team member", or "If you could go back and change one thing about a previous experience, what would it be?"
- Use **STAR framework** to tell a story:
  - Situation: what were the circumstances of the story?
  - Task: what was the task you had to complete?
  - Action: what did you do to complete the task?
  - Result: what was the result of completing the task?
- Like your resume, the more specific the better!

## Overview of Job Application Process



Offer!

- Congratulations!
- Make sure to negotiate if you can!
- Take advantage of offer deadline to make other companies speed up their process



## Agenda

1. Overview of Job Application Process + Tips
2. **How to Pass Your Interviews**
3. Study Tips and Resources
4. Mock Technical Interview @Lee's class

## How to Pass Your Interviews: Practice



### Learn the basics first (and learn them really really well!)

- There are thousands of algorithms and patterns out there, but for the vast majority of coding interviews, only a select few are relevant, so you should **practice** those until they become as easy as breathing
- Luckily for you, the most common patterns are also the easiest to learn and fundamental to other patterns

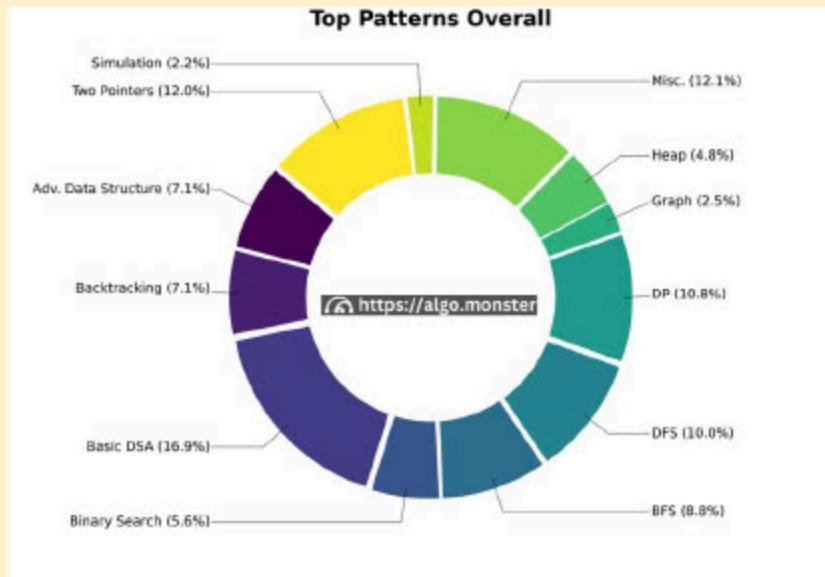


Chart describing frequencies of interview patterns. Source:  
<https://algo.monster/problems/stats>

## How to Pass Your Interviews: Practice



### Patterns to master

- **Arrays/Strings/Matrices:** two pointers, sliding window, general fluency with manipulating these data structures
- **BFS/DFS:** these have reusable templates that you should memorize, learn to recognize when a problem is actually BFS/DFS in disguise (ex. [Open the Lock](#), [Word Ladder](#), [Word Search](#)). Learn backtracking as an extension of DFS. Many of these are also solvable by union-find, which also has an easily memorizable template.
- **Sorting/Searching:** binary search, sorting, and heaps. Often serve as the solution itself, but also very frequently used in optimizing brute force solutions.

Topic	Difficulty to Learn	Return on Investment
Two Pointers	Easy	High
Sliding Window	Easy	High
Breadth-First Search	Easy	High
Depth-First Search	Medium	High
Backtracking	High	High
Heap	Medium	Medium
Binary Search	Easy	Medium
Dynamic Programming	High	Medium
Divide and Conquer	Medium	Low
Trie	Medium	Low
Union Find	Medium	Low
Greedy	High	Low

Chart describing ROI of interview patterns.

Source: <https://algo.monster/problems/stats>

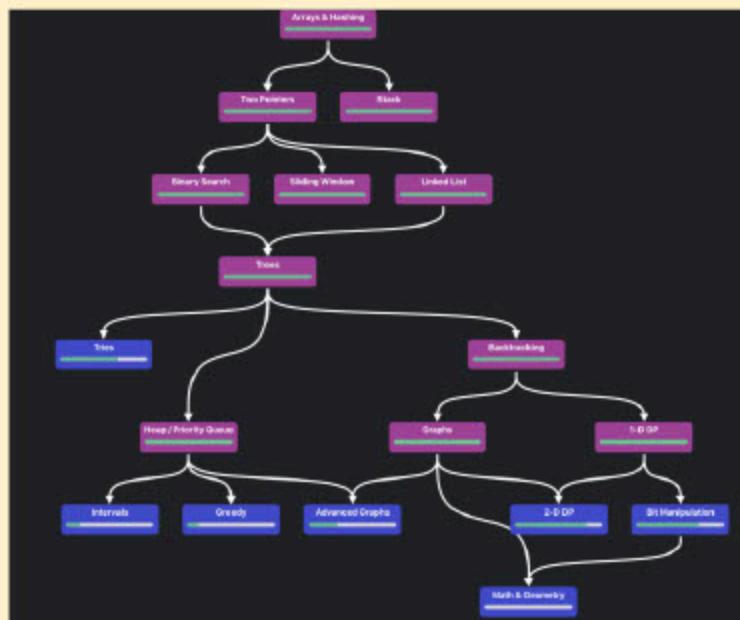
Other patterns (DP, greedy, topo. sort, weighted path-finding, bit manipulation) show up but are less common and depend on the ones listed. So focus on and master the basics first!

## How to Pass Your Interviews: Practice



### How to learn and practice these concepts?

- Learn topics in order: no point in trying a 2D DP problem if you haven't learned recursive backtracking yet
- [neetcode.io \(highly recommended\)](#) has a great roadmap for the order you should learn topics
- For each topic, first learn the relevant algorithm template/pattern from a textbook, online explanation, problem solution etc. Try a few problems with that reference, then try a few without it. **Come back to problems over multiple days to review and really internalize the topic!**
- You should aim to be able to print out a perfect BFS/DFS, binary search, union-find, etc. template at will with no help, no hesitation, and no mistakes. Then you can say you've learned the concept.



Example of neetcode roadmap progress. Start at top and go down. Source: <https://neetcode.io/roadmap>

## How to Pass Your Interviews: Practice



### How to learn and practice these concepts? (cont.)

- As you're learning a topic, make sure you can [explain what every single line](#) in the template or code does
- Also make sure to [learn, memorize, and understand time and space complexity](#) of each algorithm (!!!). Interviewers will often ask how you got that answer, so should know derivations
- Learning the topics and templates are only half the battle: also need to learn how to recognize [which template is relevant and how to apply it](#)
  - Sometimes this is obvious, sometimes this is actually the whole problem
  - Doing the leetcode daily (random problem every day) is good practice for this
  - Some people like to use [flowcharts](#), but from my personal experience, these are overkill and create overly cumbersome mental overhead. By practicing this way topic-by-topic, you will gain an intuition for when to use each pattern.
  - For example, if the problem is about finding the "total number of ways" to do something, you should recognize that this is a backtracking problem

## How to Pass Your Interviews: Practice



- **Note:** this advice is specifically for **technical coding interviews**. There are many different types of technical interviews (system design, quant trading / research interviews, role / domain-specific interviews), each of which deserves its own specialized prep, but **the underlying principles and habits are the same for all of them**
- Be **deliberate** and **organized** in your studying, **group problems by topics** and master each one before moving on, **review problems regularly** to internalize their solutions (don't just solve it and forget it), and **stay consistent (!!!)**



## How to Pass Your Interviews: Performance

**You've optimized your resume, gotten the interview, practiced hard and smart for months, and now you're in the interview. Now what?**

## How to Pass Your Interviews: Performance

### Introduction (5 min)

- Prepare a *brief* (<2 min) speech about yourself
  - Name, major, class year, concentration (if applicable)
  - Past internship experiences, personal projects, research interests
  - Quick sentence saying why you're interested in this role / company
- Interviewer will also introduce themselves. Look interested!
- Remember to smile and be friendly! When they ask "How are you?", don't just say "I'm good.", say something unique like "I just finished with classes for the day so I'm doing well, and I'm excited to be here now! How are you?"

**Keep it short! The technical interviewer does not take this into account for your performance, but you want to leave a good first impression!**

## How to Pass Your Interviews: Performance

### Coding Questions (40-50 min, 1 to 2 problems)

- Interviewer will either give you the problem in text or explain it verbally: either way, take a minute or two to yourself to understand the problem, thinking about which topics are relevant to this problem
- Before anything else, **ask clarifying questions (!!!)**
  - Ask about input/output formats and their validity, ask about error handling, ask about any potential ambiguous behavior, ask for any extra examples (this is good because it might make the interviewer reveal a trick input or edge case they were saving for later)
  - Even if questions may seem "stupid" or "obvious", better to ask questions and not need the answer than to not ask and end up needing the answer later
  - Makes you look "collaborative" and "communicative" to interviewer, so free points without even solving the problem
- When you have a solution idea, **don't start writing code**: describe it in words, ask your interviewer "Does that make sense / Does that sound good?" and get their approval, then start writing code
  - Even if you can only come up with a brute-force naïve solution, that is ok! Better to have a working suboptimal solution than no solution

## How to Pass Your Interviews: Performance

### Coding Questions (40-50 min, 1 to 2 problems) (cont.)

- After writing solution, briefly explain time and space complexity
- Be prepared to walk through how your solution solves an example input (this is where understanding every single line of your code is valuable)
- Interviewer will have follow-up questions: these may be another part building off of it, in which case you repeat this process, or it is an optimization
  - If you have to optimize your solution, think about any **redundancies** in your code: identify any spots where you make unnecessary operations or store any unnecessary data
  - Think about alternative ways or algorithms to achieve the same thing, evaluate if those would be more efficient in any way
  - These are often very difficult, so **talk through your thought process!** Use your interviewer for help, ask them questions to see if you're on the right track
  - There are a few "canonical" optimizations you should memorize (ex. Preprocessing with prefix sums, linear search -> binary search, recursive backtracking for DP -> memoized recursive search for DP, 2D matrix DP -> 1D dynamic array DP), but most of the time it will depend on the specific problem

## How to Pass Your Interviews: Performance

### Questions (5-10 min)

- Interviewer may be evaluating your culture fit or enthusiasm for the role
- Come prepared with a few questions:
  - "How did you come to work at Company X?"
  - "What is the culture of the team like?"
  - "How does Company X decide what project or team is the right fit for me?"
  - "What is a brief day in the life of [role I'm interviewing for]?"

**Never end the interview with "I have no questions"! Ask what you are genuinely curious about, or one of these!**



## Agenda

1. Overview of Job Application Process + Tips
2. How to Pass Your Interviews
- 3. Study Tips and Resources**
4. Mock Technical Interview @Lee's class



## Study Tips and Resources



LeetCode

Most valuable resource! Lots of interview questions you can run! Do 3 a night during interview season!

As previously mentioned, very good for organizing topics and learning in order! Neetcode 150 also has most common problems for each topic.

**NeetCode**

**AlgoMonster**

Good for first learning templates, also can use flowchart if you want. However, I recommend developing the templates and logic yourself to maximize understanding!

## Study Tips and Resources



Personal recommendation for textbook resource! Lots of common problems and patterns, explains solutions well. Good as supplementary reference!



Lots of practice questions and explanations of useful topics! Hard to use as primary resource but a good refresher on data structures / algorithms!

### ELEMENTS OF PROGRAMMING INTERVIEWS



ADNAN AZIZ  
TSUNG-HSIEN LEE  
AMIT PRAKASH



## Study Tips and Resources

- **Study with your friends!** Do mock interviews for each other
  - Will be easier once you have done a few real ones and know what they're like!
- Your first interview (usually) does not go well!
  - Interviewing is like any skill - it comes with practice
  - Don't get discouraged :)
- No amount of resources replaces solid data structure and algorithm knowledge
  - Pay attention in class and learn the material!

Good Luck!



## Agenda

1. Overview of Job Application Process + Tips
2. How to Pass Your Interviews
3. Study Tips and Resources
- 4. Mock Technical Interview @Lee's class**

## **Components of the Mock Tech Interview**

**First 2 mins**

### **Introduction**

- Candidate and interviewer introduce themselves

**1 ~ 2 mins**

### **Warm-up Question (Optional for Mock)**

- Intent is to “break the ice” and to build candidate’s confidence

**10 ~ 12 mins**

### **Technical Question**

- The interviewer will present the basic premise of the technical question
- This is a conversation that both the candidate and the interviewer engage in in order to problem solve collaboratively

**Last 2 mins**

### **Wrap-up**

- Interviewer asks other questions about their experience
- Candidate asks the interviewer questions

## As the interviewer (TA, Section Leader) ...



- Behaviors Interviewers hope not to see
  - **Coding right away** - Immediately writing before discussing or understanding the problem with you.
  - **Staying silent** - While you're trying to understand how they think.
  - **Ambiguous variable and function names** - Trying to make the code perfect, instead of understandable.
  - **Not testing** - They do not go through their code line-by-line to make sure it's doing what they thought.
  - **Ignoring hints** - Thinking you're there to judge.

What to notice from your candidate

## As the candidate (You, students) ...

- Ideal behaviors you strive to display
  - **Asking clarifying questions** - "Just to make sure I understand the question..."
  - **Example input and output** - "If I have input ABC, I should get output XYZ?"
  - **Break problems down** - Identify sub-steps. Use helper functions. If you don't know how to start, you start small.
  - **Testing** - You go through your code line-by-line as you write it. Double-checking functions using example input and output.
  - **Engaging in conversation** - You know good problem solving involves a lot of back-and-forth.
  - **Consider corner cases** - You ask yourself how will your code handle unexpected input?



How to wow your interviewer

Prof. Lee's Last Comments...



Burn one's bridges

背水之陣



# Mock interview (Reflection poll)

## Question: Design a Recipe Management System with Popularity Rankings

- Design a system for managing recipes in a cooking app
- Create an abstract `Recipe` base class with abstract methods for calculating preparation time and ingredient cost
- Implement at least two concrete recipe types (e.g., `BakedRecipe`, `SautéedRecipe`)
- `Recipe`s should be comparable based on their popularity score

## Follow-up questions from interviewer

- Ask if there is any coupling and how to mitigate it
- Ask whether all data is encapsulated (and why)
- Ask whether classes are cohesive (and why)

# Mock interview (Reflection poll)

## Question: Design a Playlist Manager

- Design a `Playlist` class that manages a collection of songs
- Each song has a title, artist, and duration (in seconds)
- `Playlist` should be `Iterable` and have:
  - List of songs
  - Read-only `total_duration` property that calculates the sum of song durations
  - `song_count` property that is the number of songs

## Follow-up questions from interviewer

- Ask how to modify to support adding / removing songs
- Ask how to provide option to play songs in "shuffle" (random) order (efficiently)

# Mock interview (Reflection poll)

## Question: Design an Office Cable Network Layout

- You're planning the network cable layout for an office building
- You have a list of possible cable connections between rooms, where each connection has an associated installation cost (based on distance and difficulty)
  - List of tuples: [ ('Room101', 'Room102', 150), ('Room102', 'Room103', 200), ('Room101', 'Room103', 400), ... ]
- Design a function that determines the minimum cost way to connect all rooms to the network, ensuring every room has connectivity without redundant cables
- Then, write a recursive function that, given the resulting network, determines all rooms that would lose connectivity if a specific room's network equipment failed

**Poll:**

- 1. What is your main takeaway from today?**
  
- 2. What would you like to revisit next time? (Co-op panel)**