Technical Interview Preparation in Summer (TIPS)

Lecture 3: Introduction to Technical Interviews & Pre-Coding Techniques

Outline for Tonight

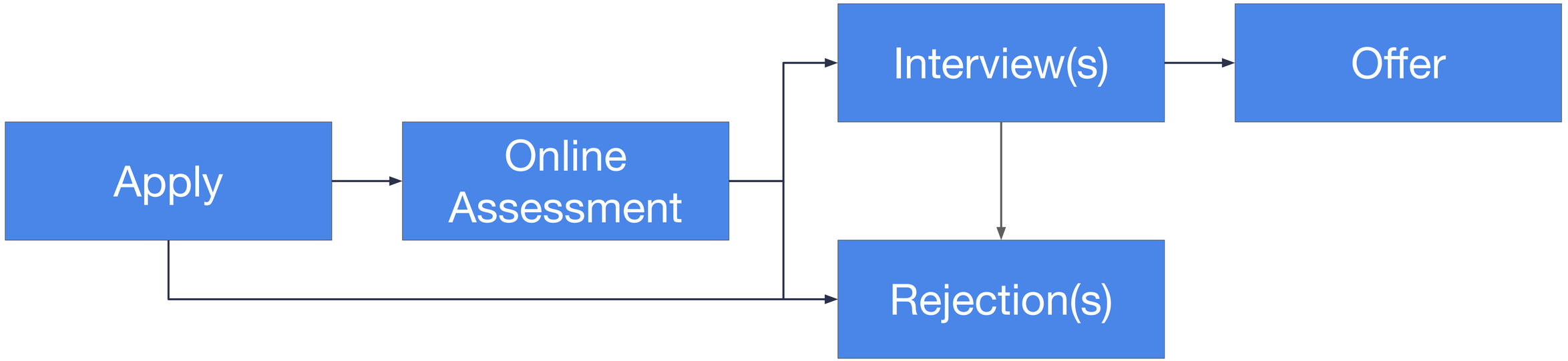
- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

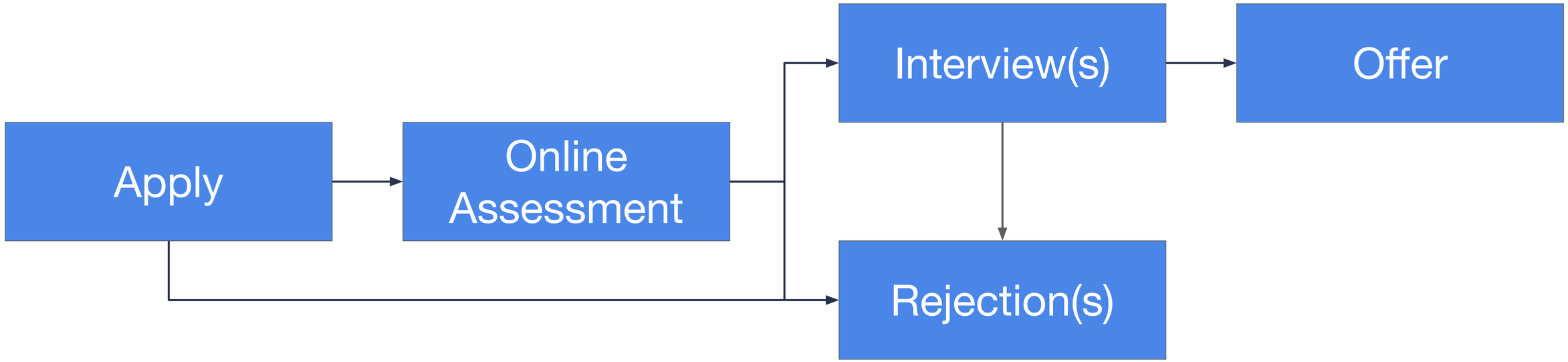
DISCLAIMER

By no mean is TIPS an “absolute guide” on how things will be!

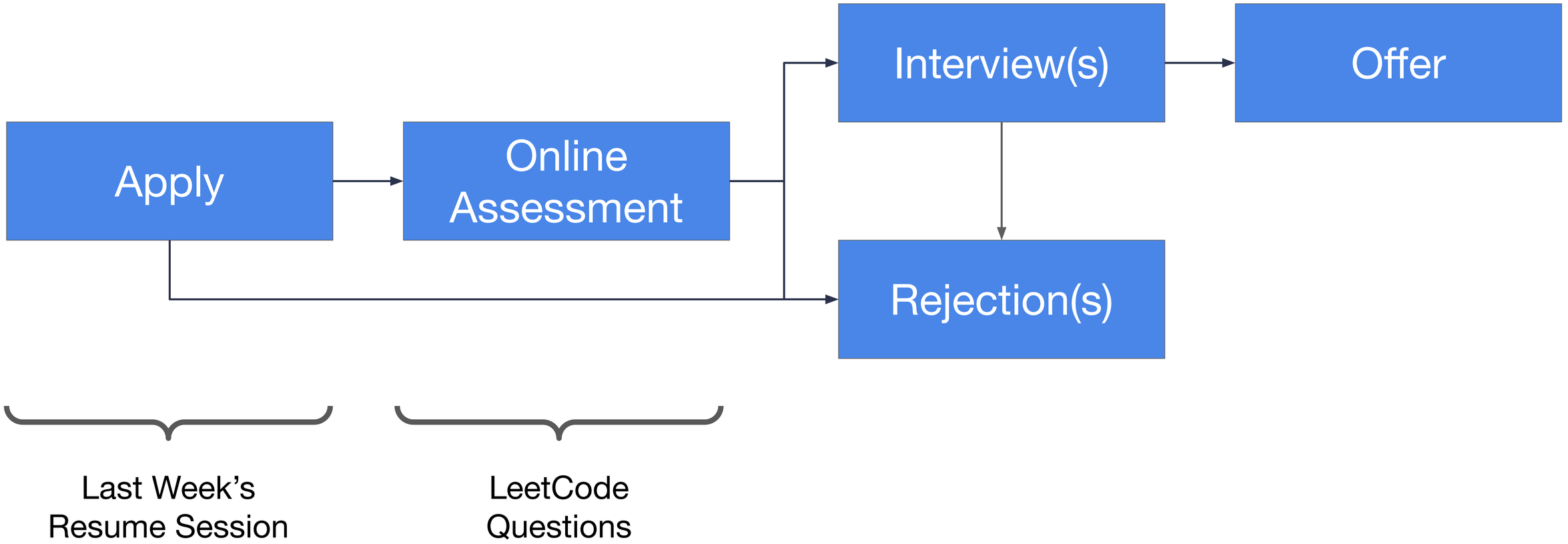
Every company is different, and every interview is different.

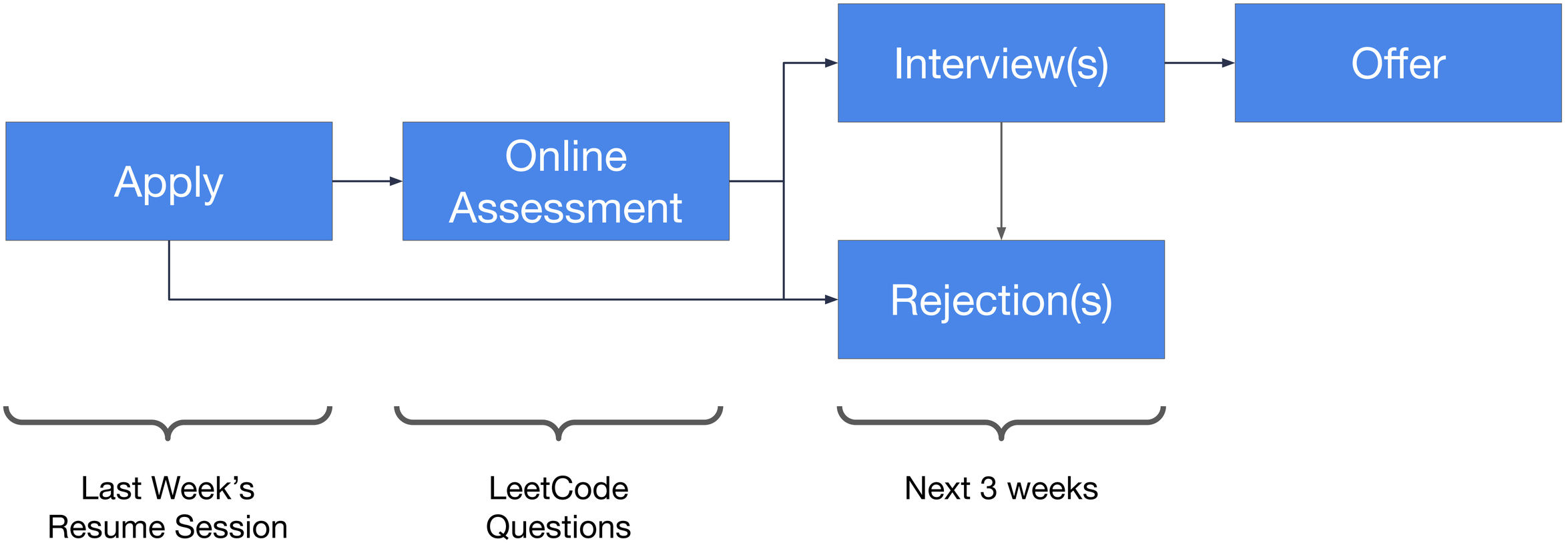
BUT many of the things that we share will be quite **transferrable**.


Apply
Online Assessment
Interview(s)
Offer
Rejection(s)



Last Week's
Resume Session


Apply
Online Assessment
Interview(s)
Offer
Rejection(s)
Last Week's Resume Session
LeetCode Questions


Apply
Online Assessment
Interview(s)
Offer
Rejection(s)
Last Week's Resume Session
LeetCode Questions
Next 3 weeks

Outline for Tonight

- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

What’s the technical interview all about?

At most top tech companies, algorithm and coding questions form the largest component of the interview process.

Generally, each interview lasts about 45 minutes to an hour, and you will be asked one to two questions.

Your job is to not just solve the problem, but solve it **together** with the interviewer.

You also need to write **real code**! Pseudocode will not be accepted.

What your interviewer will look out for

Analytical skills: How well you solved the problem, if you needed any help, how optimal your solution is, if you considered any tradeoffs.

Coding skills: Were you able to translate your algorithm into clean code?

CS knowledge and fundamentals: Whether you have a strong foundation, whether you know relevant technologies.

Experience: Have you made good technical decisions before, how your portfolio is like.

Culture fit / Communication skills: Will you fit in at the company and with the team?

What your interviewer will look out for

They’re looking out for your data structures and algorithms foundation. Your goal is not to memorise, but to implement.

Common library functions are fair game, i.e. you need to know your language well.

Specific questions about concepts are rare, unless your role needs that concept or you claim to be an expert.

Why coding questions? It seems like even good candidates might not do well. And do I really need to know my DS&A to do well for my job?

- They want to know how you think about and tackle hard problems.
- They want to know if you can talk about your ideas and actually implement them!
- They would love to see how you consider certain tradeoffs.
- Basic knowledge on data structures and algorithms is more important than you'd think.
- Most importantly, they themselves have considered their tradeoffs: they'd rather have false negatives than false positives.

Types of Interviews

"Closed-book": Solve a problem without reference to any API.

"Open-book": Allows you to reference certain API. Rather niche interviews that test you on specific skills, e.g. ability to code out a program that calls their API.

No execution: No code compilation / execution allowed, e.g. Google uses Google Docs.

Execution allowed: Code out a program that can run and produce some results. Generally not common.

Online (Phone): Use of an online collaborative editor OR screenshare your IDE.

Physical (Whiteboard): Code on a whiteboard.

Types of Interviews

"Closed-book": Solve a problem without reference to any API.

"Open-book": Allows you to reference certain API. Rather niche interviews that test you on specific skills, e.g. ability to code out a program that calls their API.

No execution: No code compilation / execution allowed, e.g. Google uses Google Docs.

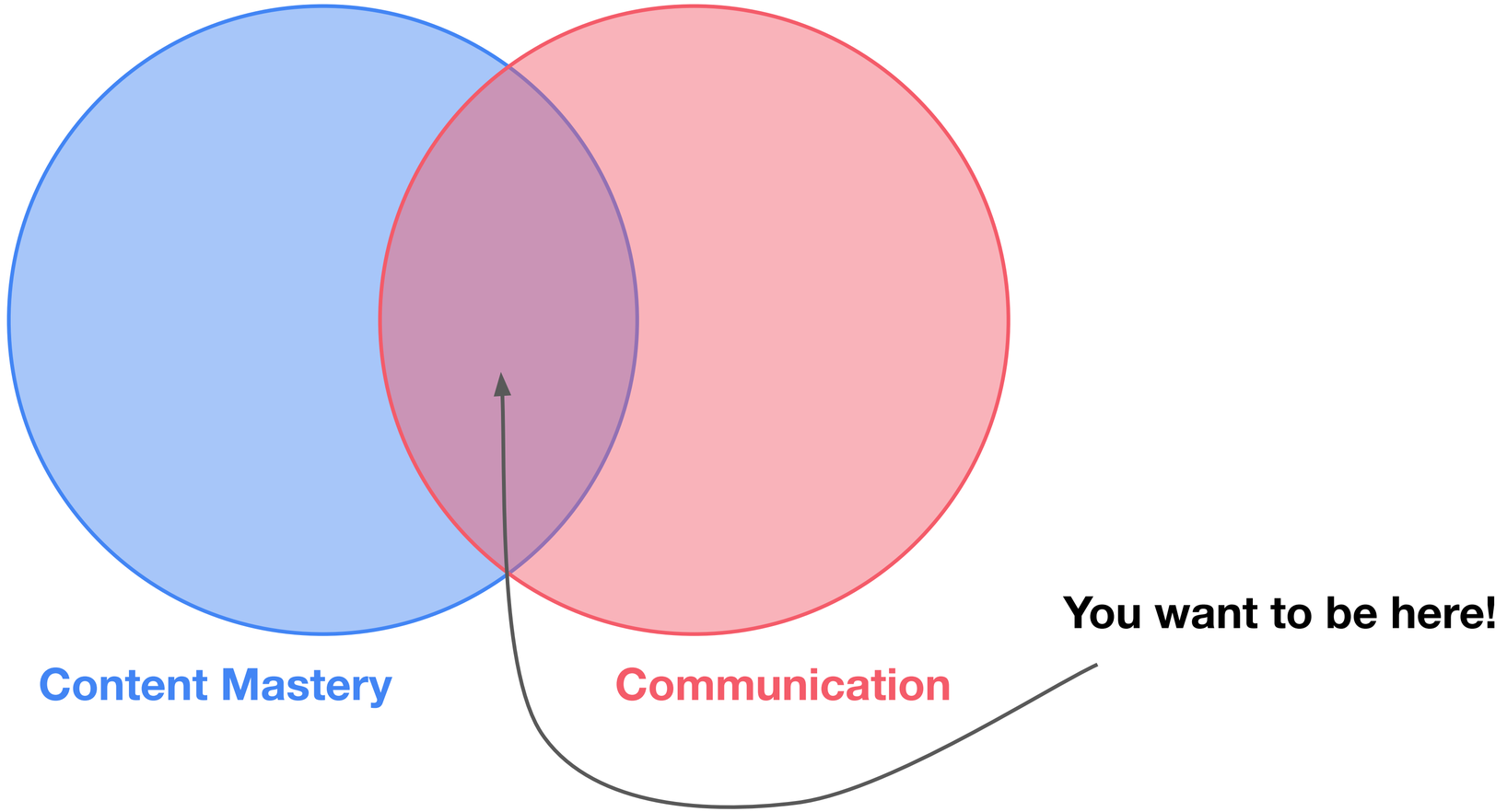
Execution allowed: Code out a program that can run and produce some results. Generally not common.

Online (Phone): Use of an online collaborative editor OR screenshare your IDE.

Physical (Whiteboard): Code on a whiteboard.

The Key to Being Successful

…is mastering the intersection of
content mastery and communication



Outline for Tonight

- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

Prepare a Good Personal Space

Find a quiet space free from distractions. Bonus if it looks good.

Check that your internet connection is good. **If you're in school, consider using a hotspot.**

Plan for all situations so that you're ready!

Double Check Your Technology

Install any drivers, etc.

Test your camera and microphone.

No better way to do this than to do a mock interview!

Understand the Code Editor / Platform

If the interview allows you to use an IDE of your choice, make sure it's configured well for your language of choice. Practice coding while zoomed in.

Check if there are any whiteboard tools you can use, e.g. there's one in Coderpad. Some interviews may use Google Drawings. Consider having a real whiteboard with you.

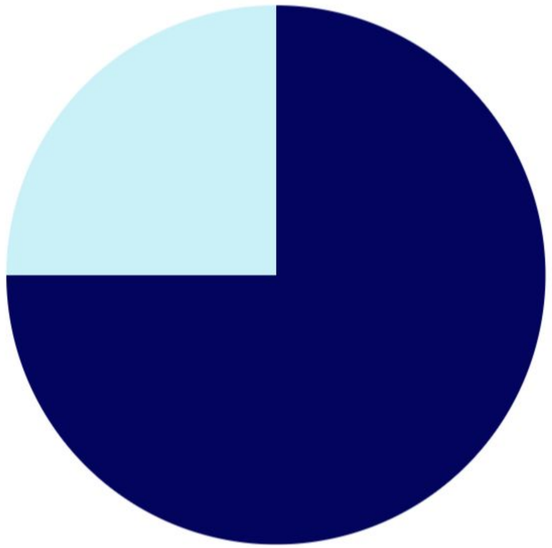
If you're using some non-code text editor, spend some time to set it up (either before or quickly at the start of the interview). You may want to:

- Turn off auto caps
- Turn off word substitutions
- Turn off list/link detection
- Use spaces instead of tabs

Outline for Tonight

- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

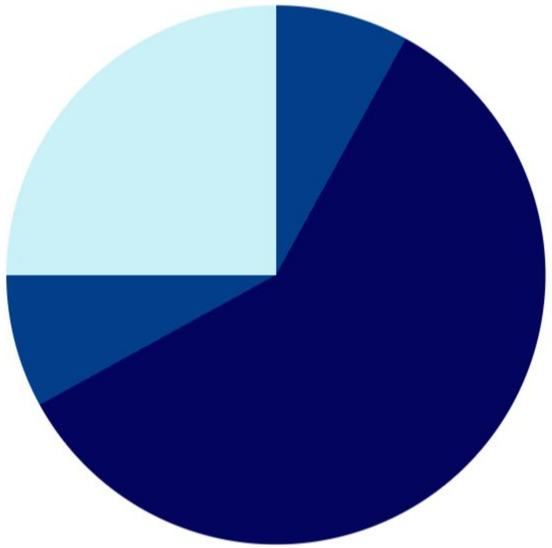
Assuming you have 2 questions to solve, that’s 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

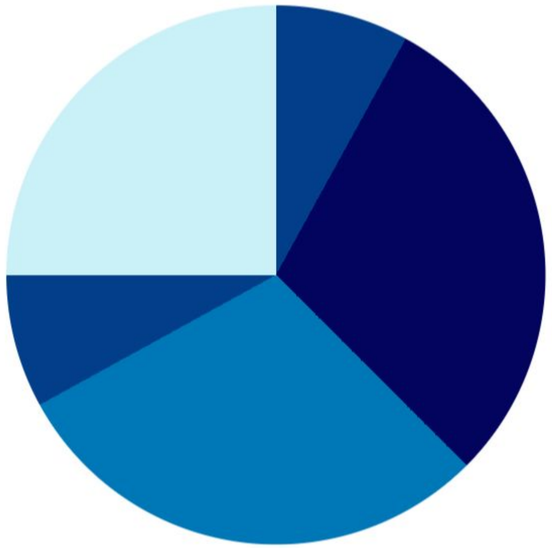
Assuming you have 2 questions to solve, that's 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

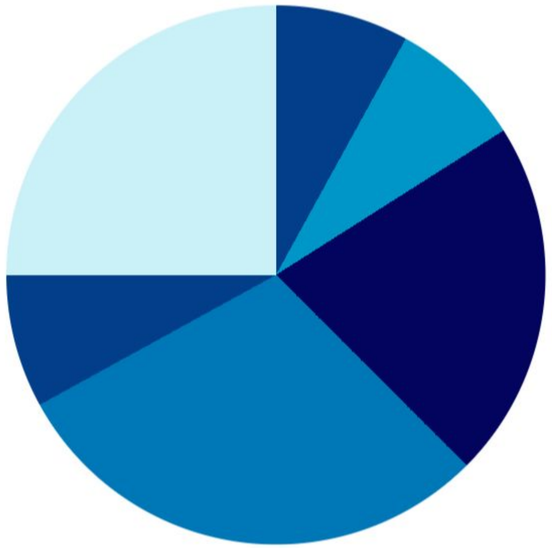
Assuming you have 2 questions to solve, that’s 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

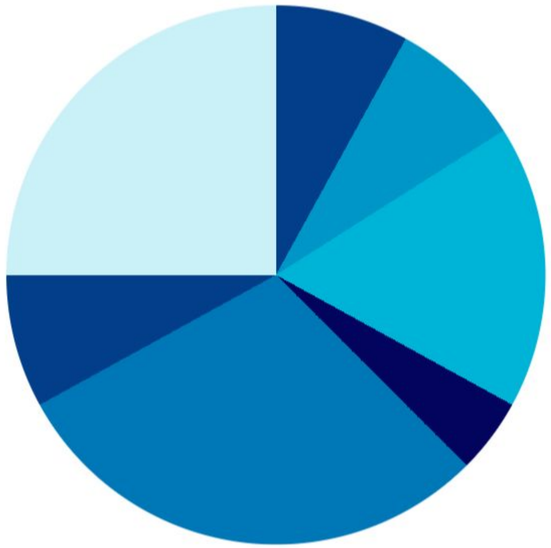
Assuming you have 2 questions to solve, that’s 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

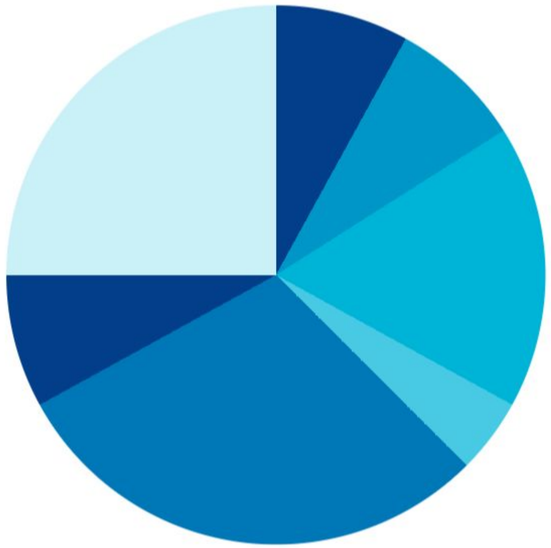
Assuming you have 2 questions to solve, that's 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Working the Clock



You have 45 mins.

Give yourself 5 mins for intro at the start and another 5 for questions at the end.

Assuming you have 2 questions to solve, that’s 17 mins each.

For each question, you should spend 5 mins before writing code…

10 mins actually coding…

and 2-3 minutes testing your solution.

Why 2 questions?

Generally, interviewers like to ask one warmup question + one harder question OR ask a question then introduce a twist to it as a second question.

However, even if you're not aware of how many questions will be asked, it's good to try to solve as if you will be doing two. Sometimes, interviewers will decide on how many questions to ask depending on how fast you do them.

And the more questions you can do, the more information you're giving them on how you tackle hard problems, consider tradeoffs, and communicate.

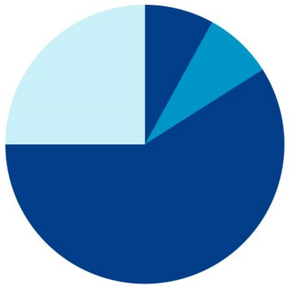
Will I be penalised if I don’t get through 2 questions?

There is no need to panic if you don't think you can finish the questions in time.

A good interviewer will actually try to stretch your mind during the interview to see your limits.

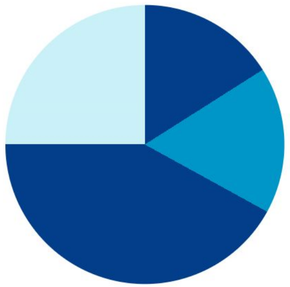
But do be strategic if you feel that time is running out. Do things that best showcase your abilities.

Overview of Techniques



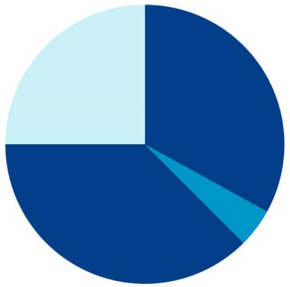
The first 5…

- Proactive Communication
- Designing Your Algorithm



The next 10…

- Maximising Collaborative Editors
- Talking As You’re Coding



The last 2-3…

- Handling Mistakes
- Testing Your Code

Outline for Tonight

- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

Proactive Communication

Proactive

/pro’aktiv/

(adj) controlling a situation by causing something to happen rather than responding to it after it's happened.

Helps you to get what you need, get organised, and set the tone.

Don't let the first 5 minutes impede the next 30 minutes!


DO NOT
START CODING
IMMEDIATELY

Proactive Communication

There are three parts to this:

1. Listen Carefully
2. Ask Questions
3. Draw an Example

1. Listen Carefully

Generally, interviewers do not provide any information that's not useful!

You need to listen out for keywords or unique information. For example:

"*Given two arrays that are sorted...*"

You probably need to know that the data is sorted! The optimal solution for the sorted arrays is likely different from the optimal solution for non-sorted ones.

1. Listen Carefully

If your solution isn't making use of all information that you've been given, you're unlikely to reach the optimal solution.

The best way to remember everything may be to type out all the keywords that you hear or see in the question!

2. Ask Questions

Make sure you fully understand what's the input, what you are expected to do, and what's the output.

Clarify any ambiguity and validate your assumptions.

Ask about how they want edge cases to be handled.

2. Ask Questions

You are asked to design an algorithm to sort a list. What do you ask?

- What sort of list? An array? A linked list?
- What does the list hold? Numbers, characters, strings?
- If numbers, are they integers?
- Where did the values come from? Are they IDs? Values of something? Any special characteristics about the items?
- How many items are there?

2. Ask Questions

General questions that you can ask:

- How big is the size of the input?
- How big is the range of values?
- What kind of values are they? Are there negative numbers? Floating points?
- Will there be empty inputs?
- Are there duplicates within the input?
- What are some extreme cases of the input?
- How is the input stored? If you are given a dictionary of words, is it a list of strings or a trie?

3. Draw an Example

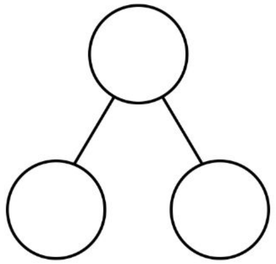
Once you're clear about the question, it's good to come up with one or two examples to run through and make sure you understood everything correctly + potentially find patterns that you didn't see before.

You can even use these examples for testing later!

You have to be smart about the example(s). Don't come up with ones that are too large, such that it takes too much time to run through, nor ones that are too small, which will probably not reveal any patterns.

3. Draw an Example

Is this a good example for a binary search tree question?



A bit too small. Cannot observe any patterns from it.

Not specific. A BST should contain values. What if the numbers contained within affect how the problem should be solved?

This is actually a special case. It's not only balanced, but also a perfect tree, where each node has two children. Special cases can be deceiving.

Proactive Communication

Summary:

1. Listen Carefully
2. Ask Questions
3. Draw an Example

Designing Your Algorithm

You shouldn't be coding anything until you've designed and run through your algorithm with your interviewer!

Your first step after the previous step (Proactive Communication) should be to state a brute force solution.

Some candidates don't state it because they feel it's obvious and terrible. But here's the thing: even if it's obvious for you, it's not necessarily obvious for all candidates.

You don't want your interviewer to think that you're struggling to even see the easy solution.

Sometimes, the brute force solution may be the best possible solution, but most of the times, it's not. Explain what the space and time complexities are, then start improving it.

Consider these two cases:

Case 1

Quickly understands the problem and recognises the less-efficient solution, but doesn't state it as it's “obvious”.

Spends 10 minutes trying to come up with an optimal solution, but isn't very clear in communication. Interviewer struggles to follow.

Starts writing code but does not arrive at a working solution. No time to catch any bugs since there's no time to test!

Runs out of time to explain time and space complexities and tradeoffs between different approaches.

Case 2

Does not have a strong grasp of the problem; they have never seen a problem like this before.

Asks the interviewer a few questions, then spends 10 minutes sketching some diagrams to arrive at the brute force solution.

Starts writing code for the brute force solution. Makes a few errors, but catches them without hints when testing.

Correctly explains time and space complexities, and discusses potential areas for improvements.

I have a brute force solution. Now what?

Our next step is to optimise our solution! Note that in the process of doing this, make sure to discuss complexities every step along the way!

Steps that you can do:

- Apply general tips
- Look for BUD
- DIY
- Simplify and Generalise
- Base Case and Build
- Data Structure Brainstorm

1. General TIPS

1. Is there any unused information? e.g. list is sorted
2. Use a fresh example. Sometimes you may see a new pattern.
3. Solve it "incorrectly". Sometimes, having an incorrect solution helps you to figure out what's needed for it to be correct.
4. Trade space for time, or vice versa.
5. Precompute information, so that subsequent solving is easier.
6. Use a hash table. They are super common and important.

2. Look for BUD

Look out for these three things:

Bottlenecks: Even if you optimise your O(n) step to O(1), if you have a O(n log n) step, the overall runtime will still be that.

Unnecessary Work: Are you doing extra steps? Can you potentially terminate early?

Duplicated Work: Are you doing the same things in multiple places?

3. DIY

Try to imagine you're solving the problem in real life. What might you do?

For example, when you pass a dictionary to someone who doesn't know binary search, it's likely that they will still do binary search to find a word in real life.

4. Simplify and Generalise

Tweak some constraint of the problem to make it easier, and try solving that first. Once you've solved that, then try to solve the original problem.

For example, if you're trying to see if you can form a paragraph from a given array of words, you can try to simplify it to forming a word with a given array of characters.

One solution would be to use an array to track the frequency of each character. We can then "upsize" this solution and use a frequency hash table for paragraphs and words.

5. Base Case and Build

Solve for n = 1, n = 2 and so on.

Then, figure out how to derive n = 3 from the previous two solutions.

This method generally leads to more recursive solutions. You can try to convert it to an iterative one, if possible.

6. Data Structure Brainstorm

Some of you may already use this technique for CS2040.

Simply run through all the data structures you know of, and try to apply each one to the problem.

This is most applicable when the problem is highly general, and seems like many solutions are possible.

Designing Your Algorithm

Summary:

1. Come up with a brute force solution
2. Optimise from there
    a. Apply general tips
    b. Look for BUD
    c. DIY
    d. Simplify and Generalise
    e. Base Case and Build
    f. Data Structure Brainstorm
3. Discuss complexities and tradeoffs at every step!

Designing Your Algorithm

Once your interviewer says they’re good with your algorithm, you now have the **green light** to start coding!

Next Week: Techniques to apply WHILE coding.

Demo with Kevin!

Using TIPS!

Outline for Tonight

- What are Technical Interviews?
- Preparation before the Interview
- Breaking down the Technical Interview
- Pre-Coding Techniques
- Rules for Mock Interviews

Rules for Our Mock Interviews

1. During a mock interview, the two participants will take turns to "interview" each other.
2. Each participant will ask one question. You will need to prepare a question before the mock interview to ask the other participant.
    a. It is best to check with the other participant what difficulty they wish to attempt prior to the interview.
    b. You may wish to attempt an easier question during mock interviews, so that you can practice your communication skills, instead of spending time getting stuck on hard questions.
3. As an interviewer, try to take down notes via the TIPS app about how the other participant is doing. These will be shared with the other person at the end.
    a. For example, are they asking good questions? Is their approach very original and interesting?
    b. Or perhaps what they didn't do so well, e.g. were there parts where you were unable to follow their thought process? Can their code be improved?

How to use the TIPS app for mock interviews

Pairing for Mock Interviews

1. We will randomly assign you with another member in this programme via a Coursemology announcement.
2. You can contact the other person via their Coursemology email.
   a. Arrange what time the mock interview will be.
   b. Can choose to continue communication via Telegram, etc.
3. **NOTE:** You will not be paired if you failed to meet the requirements for the past two weeks.
   a. Our priority is to ensure that everyone's mock interview experience will be pleasant. As such, we will unfortunately have to remove those who may not be able to commit to our programme.

Pairing for Mock Interviews

4. Please establish contact by **end of Friday!**
    a. If your partner does not respond by the end of Friday, please contact us and let us know ASAP. We will arrange for an alternative partner for you.
    b. If you cannot participate in the interview for any reason, please let us know ASAP.
5. Note that our system will not recognise interviews that are too short. But rest assured that if you do your mock interview properly, you will exceed that minimum threshold.

Resume Review Instructions

1. We will send out an email to you sometime this week containing two resumes for you to review.
2. A homework assignment will be opened on Coursemology with two questions for you to fill in your reviews for the respective resumes.
3. To be completed by **Sunday, 19 June, 2359hrs**.

QnA