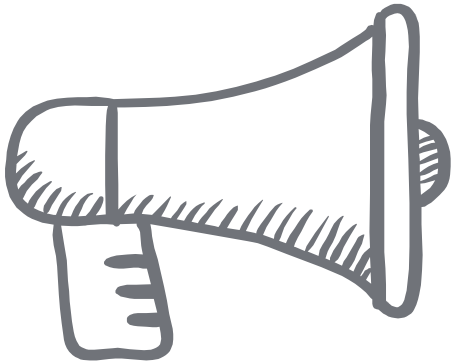
Core Data Structures

Session 1

Announcements

1. HackerRank FAQ link in course portal
2. Solutions will be posted



Goal of this week:

Be comfortable using and recognizing when to use heaps, stacks, queues, and hash tables.

Overview

1. Practice Tips
2. Communication
3. Getting Unstuck
4. Interview Process
5. Heaps (quick review)
6. In class exercises #1
7. Hash Tables, Stacks, Queues (Quick Review)
8. Choosing the right data structure
9. In class exercises #2

Practice tips

Practice tips

Start with easy problems first

Be consistent! Do 1-2 problems a day

- Occassionally 3 - 4 to prep for full time interviews

Emulate a real life setting.

- Timebox your practice, use repl.it

Find friends (pod mates?) to practice with

Tackling a Practice Problem

1. Struggle with the problem, try to match to a technique
    a. Max 30 minutes for medium, 40 minutes for a hard
2. Run code through **at least 5 test cases** you created
3. Run code through leetcode
    a. If it fails, debug your code
4. After the time limit is up, look at the solution
    a. Understand the solution
    b. Compare your solution (time/ space complexity, code quality)
    c. Is the solution using a library that I don't know about?

If you couldn’t solve the problem...

Try again!

1. Write down the problem
2. In the next 1-2 days:
    a. Repeat the steps 1-3 from the last slide

Communication

Think of the interview as a collaborative problem solving

Communication Tips

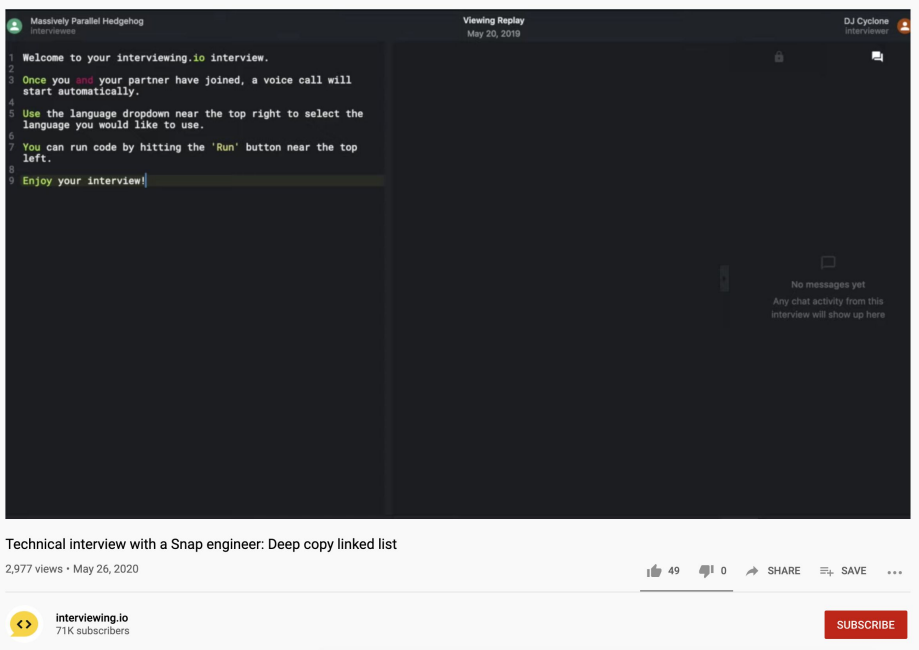
1. Be respectful of the interviewer
2. Understand the question: don’t rush into problem solving
3. Slow down
4. Validate your approach with the interviewer
5. Keep talking

How to get unstuck in an interview

1. Walk through a few examples, try to get a sense of if there is a pattern
2. Is this problem similar to one I've done before?
3. Think out loud
4. Try to come up with an approach even if it's inefficient
    a. Optimize later on
5. Don't give up! Keep a good attitude!

Recorded Phone Screens

[Interviewing.io on Youtube]



Interview Process

Interview process: Full Time

Step 1: Recruiter Screen

Runthrough of resume

Example: Facebook's Engineering Interview Post

Interview process: Full Time

Step 1: Recruiter Screen

Runthrough of resume

Step 2: Coding Challenge

Varies in time limit

Example: [Facebook's Engineering Interview Post]

Interview process: Full Time

Step 1: Recruiter Screen

Runthrough of resume

Step 2: Coding Challenge

Varies in time limit

Step 3: Phone Screen

45 - 60 minutes

Example: [Facebook's Engineering Interview Post]

Interview process: Full Time

Step 1: Recruiter Screen

Runthrough of resume

Step 2: Coding Challenge

Varies in time limit

Step 3: Phone Screen

45 - 60 minutes

Step 4: Onsite

4-7 interviews, ~45 min each

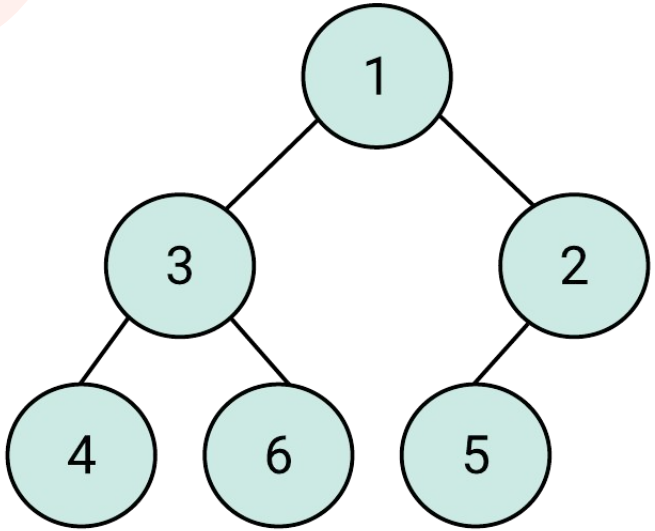
Example: [Facebook's Engineering Interview Post]

Behind the scenes

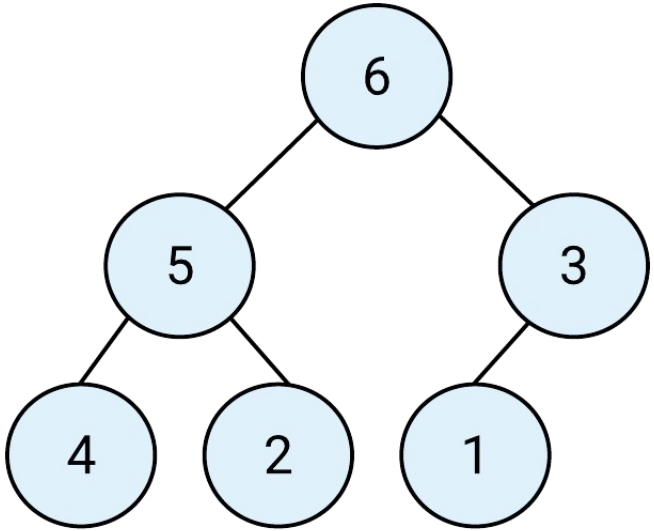
1. Pre Interview: Interviewer gets copy of your resume
2. Post interview: Interviewer evaluation
    a. Notes & Overall Rating
3. Post Onsite Calibration Meeting
    a. Every interviewer comes together for a discussion

Heaps

Heaps: Recap



Min heap



Max Heap

Heaps: Recap

- Reading largest or smallest element: O(1)
- **Insertion**: O(log n)
- **Deletion**: O(log n)
- **Creating a heap from a list**: O(n)

Heaps Implementations

Python: heapq

- `heapq.heapify(x)`
- `heapq.heappush(heap, item)`
- `heapq.heappop(heap)`
- `...and more`

Java: PriorityQueue

- `add(E e)`
- `peek()`
- `poll()`
- `...and more`

Heaps: Key Takeaways

- Useful when for getting the largest or smallest elements, and you don't care about fast lookup, delete, or search.
- Common data structure to use for "top K elements" questions
    - Examples: Top K frequent words, Find k pairs with smallest sums
- Building a heap from a list only takes O(n) time
    - Can optimize solutions by building a heap from a list instead of running insertion n times to create the heap.
- Be comfortable with the heap libraries for the language you code in
- Guide: https://guides.codepath.com/compsci/Heaps

In Class Exercise #1

Exercise: List Representation of a Min Heap

1. Spend ~20 minutes to work through the problem in exercise set #1
2. Collaborate with the group to come up with a solution
    a. Come up with example input/ output
    b. Come up with edge cases
    c. Talk about approach before coding (space & runtime analysis too!!)
    d. Code up the solution
    e. Run test cases through it
3. **GOAL:** have a solution coded up by the end of the session

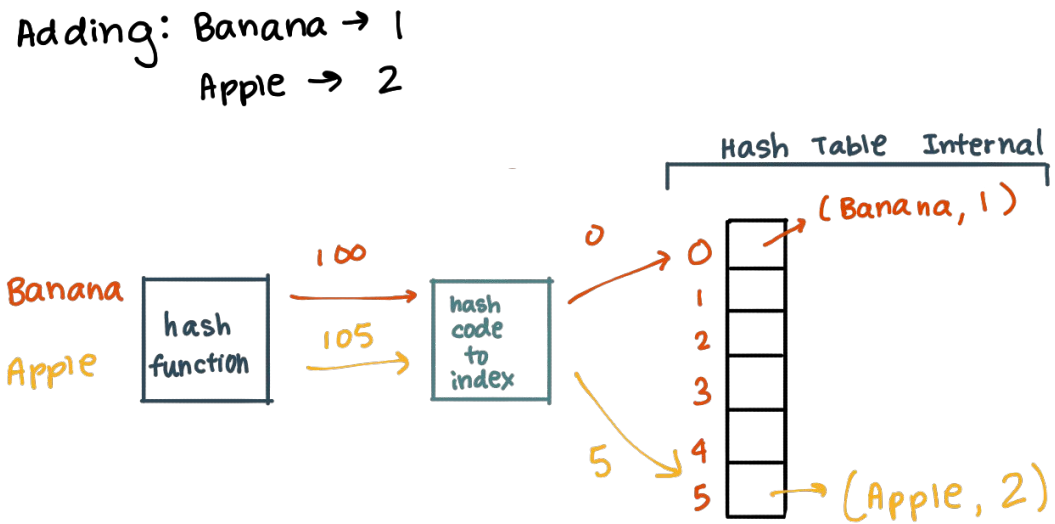
Recap: List Representation of a Min Heap

1. Example input/ outputs
    a. Inserting 2, 3, 1, 4 → [1, 3, 2, 4]
    b. Completely out of order insertion
        i. Inserting 4, 3, 2, 1 → [1, 2, 3, 4]
2. Edge cases
    a. Inserting into an empty heap
3. Runtime analysis
    a. **Time**: O(log n)
    b. **Space**: O(n)

Hash Tables

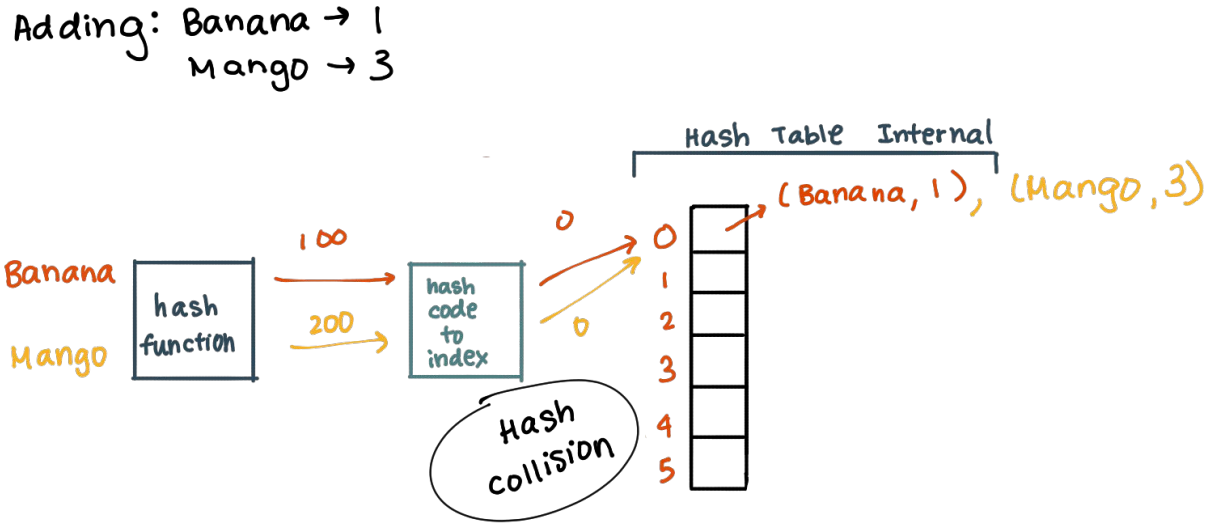
Hash Tables

- Used for storing mappings
- Fast access times: on average, O(1) for insertion, lookup, deletion



Hash Tables: Recap

- Worst case, all operations (insert, lookup, delete) can be O(n)
- Different ways of dealing with collisions: chaining, open addressing
  - Review review how these two methods work



Hash Table- Takeaways

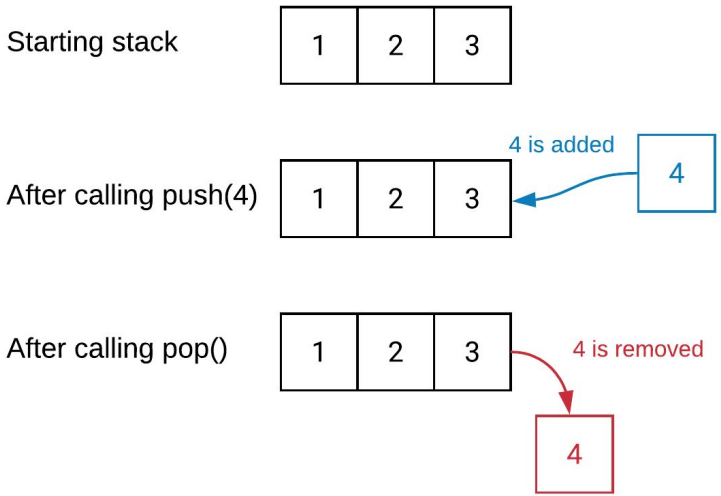
- **Extremely useful** data structure
  - keep them in mind as a tool for most interview questions
- Best performance for lookups, inserts, and deletions
- Should understand how hash tables work internally and how collisions are dealt with
- Guide: https://guides.codepath.com/compsci/Hash-Tables

Stacks & Queues

Stacks

Stores objects in a last in, first out (LIFO)

- Think of it in terms of a stack of plates



Stack Libraries

Python: `build in list type`

- `stack.pop()`
- `stack[-1]`
- `stack.append(e)`
- `...and more`

Java: `Stack`

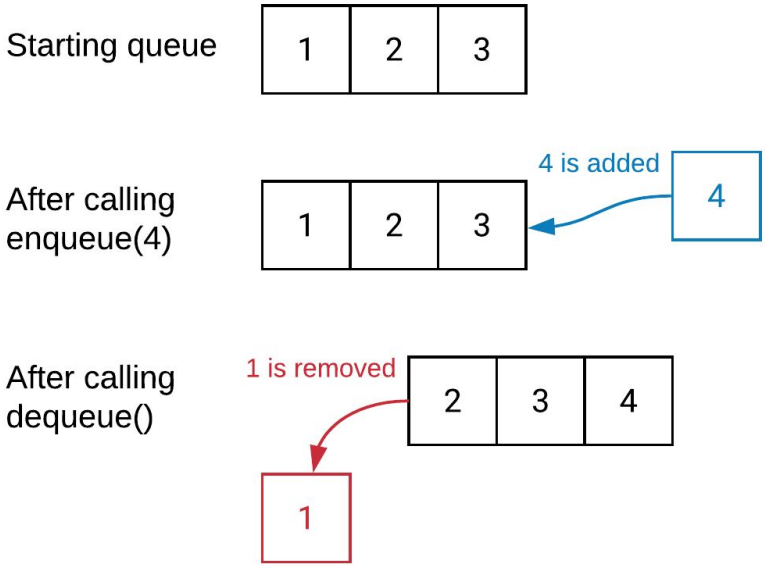
- `peek()`
- `pop()`
- `push(E item)`
- `...and more`

Stacks- Takeaways

- Stacks can be used to implement recursive solutions iteratively
  - If you're comfortable with implementing questions recursively, try implementing them iteratively as well
- Common stack problems
  - Implement your own stack class
  - Parsing
  - Backtracking (more on this later)

Queues

- Stores objects in a first in, first out (FIFO)



Queue Libraries

Python: deque

- `q.append(e)`
- `q[0]`
- `q.popleft()`
- `...and more`

Java: `LinkedList`

- `addLast(E item)`
- `removeFirst()`
- `getFirst()`
- `...and more`

Queues- Takeaways

- Useful when the ordering of the data matters as it preserves that ordering.
- Common queue problems
    - Implement your own queue class

Tips on what data structure to use

- Learning what data structure to use takes practice
- When reviewing a data structure...
  - How does this compare to other data structures I know?
  - What does this data structure do well?
  - What does this data structure do poorly at?
- When you read a solution to a coding problem, spend some time analyzing why a data structure was optimal for the problem

In Class Exercise #2

UMPIRE

1. Understand
2. Match
3. **Plan / Pseudocode (very important!)**
4. Implement
5. Reflect and verify
6. Evaluate performance

In class exercises #2

1. Go through exercises (link)
    a. Approximate time per question beside each question
    b. Test each problem thoroughly before moving on
2. It is **NOT** expected that you finish all the problems
3. Use the UMPIRE method
4. Designate one person as a leader per question and rotate
    a. Leader leads discussion
    b. Implements the code on the repl.it

Recap: In Class Exercises #2

1. **Hash Table Word Count** & **Multimaps**
    a. String Regex & hash table practice
2. **Design a Minimum Stack**
    a. Great practice for building out a data structure
    b. **Tip**! Test each of the methods along to way to validate that each one works as expected
3. **K closest points to the origin**
    a. Good heap practice

Recap: Data Structures

- Heaps
  - Best for: getting or keeping track of a biggest or smallest element
  - Not good for: searching for a specific element
- Hash tables
  - Best for: frequent lookups
  - Not good for: ordering objects in any form
- Stacks
  - Best for: removing most recent element
  - Not good for: searching for a specific element
- Queues
  - Best for: need an ordering of elements in a FIFO ordering
  - Not good for: searching for a specific element
- **Trade-off between ordering of elements and searching for elements**

Reminders

1. Review some of the practice problems from today, go over guides
2. Review solutions for warm up questions & session 1
    a. Solutions will be posted soon after lecture
3. Review Post Session Problems