Arrays and Strings

- [ ] 1.1 - Is Unique (p.192)
    - Implement an algorithm to determine if a string has all unique characters. What if you cannot use additional data structures?
    - Hints: #44, #117, #132
- [ ] 1.8 - Zero Matrix (p.204)
    - Write an algorithm such that if an element in an MxN matrix is 0, its entire row and column are set to 0.
    - Hints: #17, #74, #702
- [ ] 1.9 - String Rotation (p.206)
    - Assume you have a method `isSubstring` which checks if one word is a substring of another. Given two strings, `s1` and `s2`, write code to check if `s2` is a rotation of `s1` using only one call to `isSubstring` (e.g.,"waterbottle" is a rotation of "erbottlewat").
    - Hints: #34, #88, #104

Linked Lists

- [ ] 2.1 - Remove Dups (p.208)
    - Write code to remove duplicates from an unsorted linked list.
    - FOLLOW UP: How would you solve this problem if a temporary buffer is not allowed?
    - Hints: #9, #40

Stacks and Queues

- [ ] 3.2 - Stack Min (pg.232)
    - How would you design a stack which, in addition to `push` and `pop`, has a function `min` which returns the minimum element? `Push`, `pop` and `min` should all operate in `O(1)` time.
    - Hints:#27, #59, #78
- [ ] 3.3 - Stack of Plates (p.233)
    - Imagine a (literal) stack of plates. If the stack gets too high, it might topple. Therefore, in real life, we would likely start a new stack when the previous stack exceeds some threshold. Implement a data structure `SetOfStacks` that mimics this. `SetOfStacks` should be composed of several stacks and should create a new stack once the previous one exceeds capacity. `SetOfStacks.push()` and `SetOfStacks.pop()` should behave identically to a single stack (that is, `pop()` should return the same values as it would if there were just a single stack).
    - FOLLOW UP: Implement a function `popAt(int index)` which performs a pop operation on a specific sub-stack. Hints: #64, #81
- [ ] 3.4 - Queue via Stacks (p.236)
    - Implement a `MyQueue` class which implements a queue using two stacks.

- Hints: #98, #114

- [ ] 3.6 - Animal Shelter (p.239)
  - An animal shelter, which holds only dogs and cats, operates on a strictly "first in, first out" basis. People must adopt either the "oldest" (based on arrival time) of all animals at the shelter, or they can select whether they would prefer a dog or a cat (and will receive the oldest animal of that type). They cannot select which specific animal they would like. Create the data structures to maintain this system and implement operations such as `enqueue`, `dequeueAny`, `dequeueDog`, and `dequeueCat`. You may use the built-in `LinkedList` data structure.
  - Hints: #22, #56, #63

Trees and Graphs

- [ ] 4.4 - Check Balanced (p.244)
  - Implement a function to check if a binary tree is balanced. For the purposes of this question, a balanced tree is defined to be a tree such that the heights of the two subtrees of any node never differ by more than one.
  - Hints: #21, #33, #49, #105, #124

- [ ] 4.1 - Route Between Nodes (p.241)
  - Given a directed graph, design an algorithm to find out whether there is a route between two nodes.
  - Hints: #127

- [ ] 4.2 - Minimal Tree (p.242)
  - Given a sorted (increasing order) array with unique integer elements, write an algorithm to create a binary search tree with minimal height.
  - Hints: #19, #73, #116]

- [ ] 4.5 - Validate BST (p.246)
  - Implement a function to check if a binary tree is a binary search tree.
  - Hints: #35, #57, #86, #113, #128

- [ ] 4.6 - Successor (p.248)
  - Write an algorithm to find the "next" node (i.e., in-order successor) of a given node in a binary search tree. You may assume that each node has a link to its parent.
  - Hints: #79, #91

- [ ] 4.10 - Check Subtree (p.265)
  - `T1` and `T2` are two very large binary trees, with `T1` much bigger than `T2`. Create an algorithm to determine if `T2` is a subtree of `T1`.
  - A tree `T2` is a subtree of `T1` if there exists a node n in `T1` such that the subtree of n is identical to `T2`. That is, if you cut off the tree at node n, the two trees would be identical.
  - Hints: #4, #11, #18, #31, #37

Bit Manipulation

- [ ] 5.1 - Insertion (p.276)
  - You are given two 32-bit numbers, `N` and `M`, and two bit positions, `i` and `j`. Write a method to insert `M` into `N` such that `M` starts at bit `j` and ends at bit `i`. You can assume that the bits `j` through `i` have enough space to fit all of `M`. That is, if `M = 10011`, you can assume that there are at least 5 bits between `j` and `i`. You would not, for example, have `j = 3` and `i = 2`, because `M` could not fully fit between bit 3 and bit 2.
  - EXAMPLE:
    - `Input: N = 10000000000, M = 10011, i = 2, j = 6`
    - `Output: N = 10001001100`
  - Hints: #137, #169, #215

Object-Oriented Design

- [ ] 7.1 - Deck of Cards (p.305)
  - Design the data structures for a generic deck of cards. Explain how you would subclass the data structures to implement blackjack.
  - Hints: #153, #275
- [ ] 7.2 - Call Center (p.307)
  - Imagine you have a call center with three levels of employees: respondent, manager, and director. An incoming telephone call must be first allocated to a respondent who is free. If the respondent can't handle the call, he or she must escalate the call to a manager. If the manager is not free or not able to handle it, then the call should be escalated to a director. Design the classes and data structures for this problem. Implement a method `dispatchCall()` which assigns a call to the first available employee.
  - Hints: #363
- [ ] 7.5 - Online Book Reader (p.318)
  - Design the data structures for an online book reader system.
  - Hints: #344
- [ ] 7.7 - Chat Server (p.326)
  - Explain how you would design a chat server. In particular, provide details about the various backend components, classes, and methods. What would be the hardest problems to solve?
  - Hints: #213, #245, #271
- [ ] 7.11 - File System (p.337)
  - Explain the data structures and algorithms that you would use to design an in-memory file system. Illustrate with an example in code where possible.
  - Hints: #141, #216
- [ ] 7.12 - Hash Table (p.339)
  - Design and implement a hash table which uses chaining (linked lists) to handle collisions.
  - Hints:#287, #307

Recursion and Dynamic Programming

- [ ] 8.1 - Triple Step (p.342)
    - A child is running up a staircase with n steps and can hop either 1 step, 2 steps, or 3 steps at a time. Implement a method to count how many possible ways the child can run up the stairs.
    - Hints: #152, #178, #217, #237, #262, #359
- [ ] 8.2 - Robot in a Grid (p.344)
    - Imagine a robot sitting on the upper left corner of grid with r rows and c columns. The robot can only move in two directions, right and down, but certain cells are "off limits" such that the robot cannot step on them. Design an algorithm to find a path for the robot from the top left to the bottom right.
    - Hints: #331, #360, #388
- [ ] 8.3 - Magic Index (p.346)
    - A magic index in an array `A[0...n-1]` is defined to be an index such that `A[i] = i`. Given a sorted array of distinct integers, write a method to find a magic index, if one exists, in array `A`.
    - FOLLOW UP: What if the values are not distinct?
    - Hints: #170, #204, #240, #286, #340
- [ ] 8.4 - Power Set (p.348)
    - Write a method to return all subsets of a set.
    - Hints: #273, #290, #338, #354, #373
- [ ] 8.7 - Permutations without Dups (p.355)
    - Write a method to compute all permutations of a string of unique characters.
    - Hints: #150, #185, #200, #267, #278, #309, #335, #356
- [ ] 8.8 - Permutations with Dups (p.357)
    - Write a method to compute all permutations of a string whose charac ters are not necessarily unique. The list of permutations should not have duplicates.
    - Hints: #161, #190, #222, #255

System Design and Scalability

- [ ] 9.1 - Stock Data (p.372)
    - Imagine you are building some sort of service that will be called by up to 1,000 client applications to get simple end-of-day stock price information (open, close, high, low). You may assume that you already have the data, and you can store it in any format you wish. How would you design the client-facing service that provides the information to client applications? You are responsible for the development, rollout, and ongoing monitoring and maintenance of the feed. Describe the different methods you considered and why you would recommend your approach. Your service can use any technologies you wish, and can distribute the information to the client applications in any mechanism you choose.
    - Hints: #385, #396
- [ ] 9.2 - Social Network (p.374)

- How would you design the data structures for a very large social network like Facebook or LinkedIn? Describe how you would design an algorithm to show the shortest path between two people (e.g., Me -> Bob -> Susan -> Jason -> You).
- Hints: #270, #285, #304, #321

- [ ] 9.3 - Web Crawler (p.378)
  - If you were designing a web crawler, how would you avoid getting into infinite loops?
  - Hints: #334, #353, #365

- [ ] 9.4 - Duplicate URLs (p.380)
  - You Have 10 billion URLs. How do you detect the duplicate documents? In this case, assume "duplicate" means that the URLs are identical.
  - Hints: #326, #347

Sorting and Searching

- [ ] 10.3 - Search in Rotated Array (p.398)
  - Given a sorted array of n integers that has been rotated an unknown number of times, write code to find an element in the array. You may assume that the array was originally sorted in increasing order.
  - EXAMPLE:
    - `Input: find 5 in {15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14}`
    - `Output: 8 (the index of 5 in the array)`
  - Hints: #298, #310

- [ ] 10.9 - Sorted Matrix Search (p.407)
  - Given an M x N matrix in which each row and each column is sorted in ascending order, write a method to find an element.
  - Hints: #193, #211, #229, #251, #266, #279, #288, #291, #303, #317, #330

- [ ] 10.10 - Rank from Stream (p.412)
  - Imagine you are reading in a stream of integers. Periodically, you wish to be able to look up the rank of a number `x` (the number of values less than or equal to `x`). Implement the data structures and algorithms to support these operations. That is, implement the method `track(int x)`, which is called when each number is generated, and the method `getRankOfNumber(int x)`, which returns the number of values less than or equal to `x` (not including `x` itself).

Databases

- [ ] 14.5 - Denormalization (p.443)
  - What is denormalization? Explain the pros and cons.
  - Hints #444, #455

Threads and Locks

- [ ] 15.1 - Thread vs. Process (p.447)
  - What's the difference between a thread and a process?
  - Hints: #405

Moderate

- [ ] 16.4 - Tic Tac Win (p.466)
    - Design an algorithm to figure out if someone has won a game of tic-tac-toe.
    - Hints: #710, #732
- [ ] 16.17 - Contiguous Sequence (p.498)
    - You are given an array of integers (both positive and negative). Find the contiguous sequence with the largest sum. Return the sum.
    - EXAMPLE:
        - `Input: 2, -8, 3, -2, 4, -10`
        - `Output: 5 (i.e., {3, -2, 4})`
    - Hints: #537, #551, #567, #594, #614
- [ ] 16.24 - Pairs with Sum (p.520)
    - Design an algorithm to find all pairs of integers within an array which sum to a specified value.
    - Hints: #548, #597, #644, #673

Hard

- [ ] 17.22 - Word Transformer (p.602)
    - Given two words of equal length that are in a dictionary, write a method to transform one word into another word by changing only one letter at a time. The new word you get in each step must be in the dictionary.
    - EXAMPLE
        - `Input: DAMP, LIKE`
        - `Output: DAMP -> LAMP -> LIMP -> LIME -> LIKE`
- [ ] 17.23 - Max Black Square (p.608)
    - Imagine you have a square matrix, where each cell (pixel) is either black or white. Design an algorithm to find the maximum subsquare such that all four borders are filled with black pixels.
    - Hints: #684, #695, #705, #714, #721, #736