Leetcode Notes and Practice

Sebastian Pucher

September 8, 2025

Contents

1 Hashing **5**
- 1.1 Valid Anagram . . . 5
- 1.2 Twosum . . . 5
- 1.3 Group Anagram . . . 5
- 1.4 Single Number . . . 6
- 1.5 Encode / Decode . . . 6
- 1.6 Product of Array Except Itself . . . 6
- 1.7 Valid Sudoku . . . 7

2 Two Pointers **7**
- 2.1 Valid Palindrome . . . 7
- 2.2 Two Sum II (Two Integer Sum) . . . 7
- 2.3 3 Sum . . . 8
- 2.4 Longest Consecutive Sequence . . . 8
- 2.5 Valid Palindrome II . . . 9
- 2.6 Squares of a Sorted Array . . . 9

3 Sliding Window **10**
- 3.1 Best Time to Buy and Sell Stocks . . . 10
- 3.2 Longest Substring Without Repeating Characters . . . 10
- 3.3 Container With Most Water . . . 11
- 3.4 Longest Repeating Character Replacement . . . 11
- 3.5 Permutations in String . . . 11

4 Stack **12**
- 4.1 Valid Parentheses . . . 12
- 4.2 Min Stack . . . 13
- 4.3 Reverse Polish Notation . . . 13
- 4.4 Daily Temperatures . . . 14
- 4.5 Remove Adjacent Duplicates in String . . . 14

5 Binary Search **14**
5.1 Binary Search Recursive . . . 14
5.2 Binary Search Iterative . . . 15
5.3 Search 2D Matrix . . . 16
5.4 Koko eats Bananas . . . 16
5.5 Find Min in Rotated Sorted Array . . . 17
5.6 Search Rotated Sorted Array . . . 17
5.7 Search 2D Matrix II . . . 18

6 Linked Lists **18**
6.1 Reverse Linked List . . . 18
6.2 Cycle Detection Linked List . . . 19
6.3 Merge Two Sorted Lists . . . 19
6.4 Remove Nth Node From End of List . . . 20
6.5 Copy Linked List with Random Pointer . . . 20
6.6 Add Two Numbers In Linked List . . . 21
6.7 Find The Duplicate Number . . . 21
6.8 Sort Linked List . . . 22
6.9 Reorder List . . . 22

7 Trees **23**
7.1 Invert Binary Tree . . . 23
7.2 Max Depth of Binary Tree . . . 23
7.3 Max Diameter of Binary Tree . . . 24
7.4 Balanced Binary Tree . . . 24
7.5 Same Binary Tree . . . 25
7.6 Subtree of Another Tree . . . 25
7.7 Binary Tree Level Order Traversal . . . 26
7.8 Right side view of binary Tree . . . 26
7.9 Count Good Nodes in Binary Tree . . . 27
7.10 Validate Binary Search Tree . . . 27
7.11 Lowest Common Ancestor in Binary Tree . . . 28
7.12 Kth Smallest Element in BST . . . 28
7.13 Construct Tree from Preorder / Inorder Arrays . . . 29

8 Heap and Priority Queue **29**
8.1 Kth Largest Element in a Stream . . . 29
8.2 Last Stone Weight . . . 30
8.3 K Closest Points to Origin . . . 31
8.4 Kth Largest Element . . . 31
8.5 Task Scheduler . . . 32

9 Graphs **32**
9.1 Number of Islands . . . 32
9.2 Max Area of Islands . . . 33
9.3 Deep Clone a Graph . . . 33
9.4 Islands and Treasure . . . 34
9.5 Pacific and Atlantic Ocean . . . 34
9.6 Graph Valid Tree . . . 35
9.7 Number of Connected Components in an Undirected Graph . . . 35
9.8 Course Schedule . . . 36
9.9 Rotten Oranges . . . 36

10 Advanced Graphs **37**
10.1 Network Delay Time . . . 37

11 One-Dimensional Dynamic Programming **38**
11.1 Climbing Stairs . . . 38
11.2 Min Cost Climbing Stairs . . . 39
11.3 House Robber . . . 40
11.4 House Robber II . . . 40
11.5 Longest Palindromic Substring . . . 41
11.6 Palindromic Substrings . . . 41
11.7 Decode Ways . . . 42
11.8 Coin Change . . . 42

12 Two-Dimensional Dynamic Programming **43**
12.1 Unique Paths . . . 43
12.2 Longest Common Subsequence . . . 44

13 Backtracking **44**
13.1 Subsets . . . 44
13.2 Combination Sum . . . 45
13.3 Word Search . . . 45

14 Greedy **46**
14.1 Jump Game . . . 46
14.2 Gas Station . . . 47
14.3 Hand of Straights . . . 47
14.4 Maximum SubArray . . . 48

15 Math & Geometry **48**
15.1 Plus One . . . 48
15.2 Rotate Image . . . 49

16 Tries **49**
16.1 Implement Trie Prefix Tree . . . 49

17 Intervals **50**
17.1 Insert Intervals 50
17.2 Meeting Rooms 51
17.3 Meeting Rooms II 51
17.4 Merge Intervals 51
17.5 Non-overlapping Intervals 52

1 Hashing

1.1 Valid Anagram

Question: Given two strings s and t, return true if the two strings are anagrams of each other, otherwise return false

- Create two separate dictionaries
- Loop through one of the input strings, add key letter or letter freq.
- If dict are the same, return true

1.2 Twosum

Question: Given an array of integers nums and an integer target, return the indices i and j such that nums[i] + nums[j] == target and i != j

- Use a hashmap to store the index of each number in the array as the *value*
- On each iteration, check first to see if the difference between the target val and the current num is already stored in the hashmap
- If it is, then return the value at that key (the index), as well as the current index i
- If it's not, then add the current number and index to the hashmap
- **Key Idea:** Always check the existence between the target and the current number as a key in the hashmap first!

1.3 Group Anagram

Question: Given an array of strings strs, group all anagrams together into sublists. You may return the output in any order.

An anagram is a string that contains the exact same characters as another string, but the order of the characters can be different.

- The main idea of this question is to use a hashmap, and use arrays as keys. In this case, the array would have 26 elements, set to 0 to start.
- Each of the 26 elements represents a letter a-z. The idea is that you count each letter when traversing through the string.
- Once you have counted all the letters, use tupple() to change the mutable array into an immutable key that can be used to store the word.
- When adding it to the hash, first check to see if the key already exists. If so, add it to the array of other strs that have that mapping (add it to the values array at that key)
- If it's not a key, add it as a new key, and add the string as an element of an array to the value at that position.
- Return the values of the hashmap.

1.4 Single Number

Question: You are given a non-empty array of integers nums. Every integer appears twice except for one.

Return the integer that appears only once.

You must implement a solution with O(n) runtime complexity and use only O(1) extra space.

- Use a hash to keep track of numbers
- If a number is not in the hash, add it in, if it is already in, then remove it.
- If you repeat this process for every number, then the only value left when you've traversed through the whole array will be the single number: So return the only key left

1.5 Encode / Decode

Question: Design an algorithm to encode a list of strings to a single string. The encoded string is then decoded back to the original list of strings.

- The main idea here is to use two special characters that indicate the beginning and end of any given word
- Use the length of the string (an int) and some character like a $ sign, to indicate the start of the part of the string to begin decoding.
- Loop through the encoded string, checking for these conditions, then building back out the original string

1.6 Product of Array Except Itself

Question: Given an integer array nums, return an array output where output[i] is the product of all the elements of nums except nums[i]. Each product is guaranteed to fit in a 32-bit integer.

Follow-up: Could you solve it in O(n) time without using the division operation?

- **Main Idea:** Create two hashmaps one that keeps the current product of all numbers up until a current index and another that keeps a current product of all the numbers after a current index until the end of the input array
- This is like a memoization of all products before and after any given index. This will take 2 separate for loops
- After you have done that, loop a final time and multiply the prefix and suffix product for each index. Store this in the answer array

1.7 Valid Sudoku

Question: You are given a 9 x 9 Sudoku board board. A Sudoku board is valid if the following rules are followed:

Each row must contain the digits 1-9 without duplicates. Each column must contain the digits 1-9 without duplicates. Each of the nine 3 x 3 sub-boxes of the grid must contain the digits 1-9 without duplicates. Return true if the Sudoku board is valid, otherwise return false

- **Main Idea:** Create three separate hashmaps, one to track values in each row, each column and each segment of the Sudoku board. The segment hashmap relates to the 9 larger squared in the full board.
- Loop through each cell and check if the value already exists in each hashmap or not.
- The keys of the segment hashmap will be (i//3) and (j//3). Floor divide each index by three to get the region of the segment.
- Hard code all possible segment tuples into the starting hashmap before beginning the looping

2 Two Pointers

2.1 Valid Palindrome

Question: Given a string s, return true if it is a palindrome, otherwise return false.

A palindrome is a string that reads the same forward and backward. It is also case-insensitive and ignores all non-alphanumeric characters.

- First change the string to all lowercase with .lower() function, this ensures case sensitive args are taken care of
- Init left and right pointers to first and last characters
- Do a check using **.isalnum()**. If it is *not* alpha numeric, then increment or decrement the pointer and *continue* through the loop
- While left is less than or equal two right, compare the letters, if not the same, return false
- increment / decrement left and right at bottom of loop

2.2 Two Sum II (Two Integer Sum)

Question: Given an array of integers numbers that is sorted in non-decreasing order.

Return the indices (1-indexed) of two numbers, [index1, index2], such that they add up to a given target number target and index1 ¡ index2. Note that index1 and index2 cannot be equal, therefore you may not use the same element twice.

There will always be exactly one valid solution.

Your solution must use O(1) additional space.

- **Main idea:** Use two pointers, one at the beginning of the array, and one at the end
- If the sum number at the two pointers added together is greater than the target, then this means that the current sum is two big, and we must *decrement the right pointer*
- Following this same logic, if the current sum between the two pointers is less than the target, then we must *increment the left pointer*.
- Do this until sum == target, then return.

2.3 3 Sum

Question: Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] where nums[i] + nums[j] + nums[k] == 0, and the indices i, j and k are all distinct.

The output should not contain any duplicate triplets. You may return the output and the triplets in any order.

- **Main idea:** This question is very similar to Two Sum II: Main idea is to have two pointers, one at the beginning of the input array and one at the end of the input array. First step is to SORT the input array
- We need the sum of these to equal another value in the array multiplied by negative 1.
- Start by attempting to find what two other numbers can sum to equal the value at index 0.
- As before, we increment the left pointer if the sum value needs to be greater, and decrement the right pointer if it needs to be smaller. Repeat until something is found
- Repeat this process for every value in the original array: This takes $O(n^2)$ time.

2.4 Longest Consecutive Sequence

Question: Given an array of integers nums, return the length of the longest consecutive sequence of elements that can be formed.

A consecutive sequence is a sequence of elements in which each element is exactly 1 greater than the previous element. The elements do not have to be consecutive in the original array.

You must write an algorithm that runs in O(n) time.

- **Main idea:** Begin by sorting the Array: Start two pointers left and right at the beginning of the sorted array: left at 0, right at 1
- Keep a local max, and a global max: begin looping through the array, at any point, if the number in the right position is greater than the left by one, update the local max
- If the local max is ever greater than the global, set the greater to the local value.
- If the numbers at the left and right value equal each other, add one to both, and continue looping
- If the right pointer value is not greater than left by 1, reset local max, and set left = right
- After looping has completed, return the global max

2.5 Valid Palindrome II

Question: You are given a string s, return true if the s can be a palindrome after deleting at most one character from it.

A palindrome is a string that reads the same forward and backward.

Note: Alphanumeric characters consist of letters (A-Z, a-z) and numbers (0-9).

- **Main idea:** Begin looping through the question as usual. Make sure to make the overall string lowercase and check if each current character is alphanumeric. Once you hit a letter where the left and right pointers don't match eachother do the following:
- Create another isPal function, and check if starting at the left+1 OR right-1 values returns a valid pal
- If either of the two cases returns true, then return True, else return False

2.6 Squares of a Sorted Array

Question: You are given an integer array nums sorted in non-decreasing order, return an array of the squares of each number sorted in non-decreasing order.

- **Main idea:** Use two pointers, start at beginning and end of the array.
- If the squared value at the left is greater than the right, then add that value to the end of the output array, else end the right value to the end of the output array: repeat until output array is full
- NOTE: you can index from the end using -1, then continue decrementing that value

3 Sliding Window

3.1 Best Time to Buy and Sell Stocks

Question: You are given an integer array prices where prices[i] is the price of NeetCoin on the ith day.

You may choose a single day to buy one NeetCoin and choose a different day in the future to sell it.

Return the maximum profit you can achieve. You may choose to not make any transactions, in which case the profit would be 0.

- **Buy low... sell high**: To achieve this, init two pointers, left wanting to find the low value in the arr, and right wanting to find the high value. Init left to first index, and right to second index in array
- Begin looping through, so long as the right pointer is less than the length of the array
- If the right value is ever less than the left value, then you have found a new low to sell at. *So assign left equal to right, and increment right by one: continue the while loop*
- Upon each iteration check if the difference between right and left values is greater than global profit (which is init to 0). If so, update profit

3.2 Longest Substring Without Repeating Characters

Question: Given a string s, find the length of the longest substring without duplicate characters.

A substring is a contiguous sequence of characters within a string.

- **Main Idea:** This is a sliding window question. The idea is that you will have two pointers start at the beginning of the string, and increment the right pointer, so long as the next character hasn't already occurred in the substring
- More formally: get a local max-len, as well as a global max-len. Begin by incrementing the right pointer and adding the char to some tracked char array, so long if it isn't in the char array already.
- If it is, then we need to *increment the left pointer*, and remove whatever value the left pointer is at from our tracked chars arr or hashmap. When we remove a char, be sure to decrement our local max-len.
- Keep incrementing left pointer and removing from array until we've removed the character that the right pointer was pointing at. Once at this point, we can now continue to increment the right pointer as above.
- When incrementing right pointer, always update the local max, and check if it's greater than the global max

- Once the left pointer has reached the end of the string, stop and return the global max

3.3 Container With Most Water

Question: You are given an integer array heights where heights[i] represents the height of the ith bar.

You may choose any two bars to form a container. Return the maximum amount of water a container can store.

- **Main Idea:** Sliding window question, start a pointer at the beginning and at the end.
- Calculate the area using the smaller of the two. Update global max if needed.
- Move either the left or right pointer towards the center of the array: Always move the smaller of the two length pointers forward.
- If the lengths are the same, it doesn't matter which one you increment or decrement.
- After the pointer indexes equal each other, return the max area created.

3.4 Longest Repeating Character Replacement

Question: You are given a string s consisting of only uppercase english characters and an integer k. You can choose up to k characters of the string and replace them with any other uppercase English character.

After performing at most k replacements, return the length of the longest substring which contains only one distinct character.

- **Main Idea:** Sliding window question with a hashmap: Start two pointers at the beginning of the array. Begin incrementing the right pointer and updating the frequency of that character in the hashmap
- Core idea: Find the most frequent value in that window range: see if you can make all k replacements to the character: if you can set the result / max to the length of the range of the window.
- To find the max, simply take the max of all the values in the hashmap.
- If the you cannot make k replacements, update both the left and right pointer. Slide the full window over. Do this until the right pointer is less than the length of the array

3.5 Permutations in String

Question: You are given two strings s1 and s2.

Return true if s2 contains a permutation of s1, or false otherwise. That means if a permutation of s1 exists as a substring of s2, then return true.

Both strings only contain lowercase letters.

- **Main Idea:** Sliding window question that involves using a hashmap and re-assigning left and right pointers.
- Begin by storing all chars / frequencies of s1 into a hashmap
- Now init two pointers left and right at index zero of s1: the idea is that we well slide these pointers through the string and check if the window has the same hash count as the s1 hash
- Begin by checking if the current character at the right pointer is in the keys of the first hash, if it is, then add one to this hash, and increment the right pointer.
- If at any point while looping through, the hash maps equal each other, return true
- If the current right value is not in the hash map, then increment the left pointer once, and set the right pointer equal to the left pointer. Also reset the s2 hashmap
- If the count of the right pointer is greater than the count at that value in s1, then slide and remove values at the left pointer, until the frequencies are the same.
- If the right pointer reaches the end and there are no matches, return False

4 Stack

4.1 Valid Parentheses

Question: You are given a string s consisting of the following characters: '(', ')', '{', '}', '[' and ']'.

The input string s is valid if and only if: – Every open bracket is closed by the same type of close bracket. – Open brackets are closed in the correct order. – Every close bracket has a corresponding open bracket of the same type.

Return true if s is a valid string, and false otherwise.

- **Main idea:** Use a Stack to push open brackets, and pop closed brackets
- Before iterating through each char of the string, create an array of valid open brackets, and an array of valid close brackets (same positioning in each)
- For each char, if the char is neither in each bracket array, return false.
- If the char is an open bracket, push to stack
- If the char is an close bracket, *first check if the stack is non empty* then check to see if the last value on the stack is the corresponding open bracket to the current close bracket
- Continue looping if it is, return false if it isn't
- After the loop completes, ensure that the stack is empty, if it is, return true

4.2 Min Stack

Question: Design a stack class that supports the push, pop, top, and getMin operations.

– MinStack() initializes the stack object. – void push(int val) pushes the element val onto the stack. – void pop() removes the element on the top of the stack. – int top() gets the top element of the stack. – int getMin() retrieves the minimum element in the stack.

Each function should run in O(1) time.

- **Main idea:** This question is very easy except for returning the min in O(1) time. The idea here is to use another internal stack to keep track of all mins, as they are added.
- We can also set up another class var called current min. This is the value that we actually return when getMin() function is called.
- Using internal stack to keep track of min goes as follows: if a new added value is less than the current min, then update the new current min to this value, as well as add this value to the internal min stack
- If we pop the value, and it is the current min, then we need to pop the value from both the regular stack, as well as the min stack, and set the new current min to the top (last) value in the min stack. If the min stack is empty, just set the current min to NONE (current min will be init as NONE).

4.3 Reverse Polish Notation

Question: You are given an array of strings tokens that represents a valid arithmetic expression in Reverse Polish Notation.

Return the integer that represents the evaluation of the expression.

The operands may be integers or the results of other operations. The operators include '+', '-', '*', and '/'. Assume that division between integers always truncates toward zero.

- **Main idea:** This question implements a stack: Loop through the array and add the numbers to the stack: if once you hit an operation over a number, then pop the last two elements of the stack, and preform that operation on them
- Make sure you preform the operation in the correct order!
- Once you have preformed the operation, push the result back into the stack and continue
- Repeat until the stack is of length 1: the last element left in the stack is the answer

4.4 Daily Temperatures

Question: You are given an array of integers temperatures where temperatures[i] represents the daily temperatures on the ith day.

Return an array result where result[i] is the number of days after the ith day before a warmer temperature appears on a future day. If there is no day in the future where a warmer temperature will appear for the ith day, set result[i] to 0 instead.

- **Main idea:** This uses the idea of a monotonic stack: this means that the elements will be ordered in strictly increasing or decreasing order
- Keep a stack that stores both the temperature and index for all days in the array.
- While the length of the stack is greater than zero, and the last day is cooler than the current day, pop the last element out of the stack. Store the answer as the result of subtracting the index of the element just popped and the index i
- If the current element while looping through the array is smaller than the previous, add it to the stack
- Essentially: Loop through all items once: if the item is smaller than the last element on the stack, then just add it to the stack. If it's larger, then pop the last value, and the answer at the last value is the difference between the indices.

4.5 Remove Adjacent Duplicates in String

Question: You are given a string s consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them.

We repeatedly make duplicate removals on s until we no longer can.

Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.

- **Main idea:** Use a stack! Add to the stack so long as the last element in the stack doesn't equal the current element that you want to push in. If it does, pop out the last value
- After done iterating through the string, convert the stack (list) to a string, and return the string
-

5 Binary Search

5.1 Binary Search Recursive

Question: You are given an array of distinct integers nums, sorted in ascending order, and an integer target.

Implement a function to search for target within nums. If it exists, then return its index, otherwise, return -1.

Your solution must run in O(logn) time.

- **Main idea:** To solve this problem recursively, you must keep track of left and right indices upon each recursive call
- Initialize left to be index zero, and right to be last index (len(A) -1)
- *Base Case:* If left index equals right index, check if the value is in only position, if it is, return index, else return -1
- If left doesn't equal right, find the mid index, and compare target value to value at middle index
- If it equals the middle index, return
- If it's greater than the middle index, recurse, left = middle index + 1
- If it's less than the middle index, recurse, right = middle index - 1
- **If parameters don't use left and right, you'll have to add them initially as none as defaults, then set them before recursing**

5.2 Binary Search Iterative

Question: You are given an array of distinct integers nums, sorted in ascending order, and an integer target.

Implement a function to search for target within nums. If it exists, then return its index, otherwise, return -1.

Your solution must run in O(logn) time.

- **Main idea:** To solve this problem iteratively, you must keep track of left and right indices and use a while loop
- *While loop Condition:* Loop so long as right is greater than or equal to the left index
- *Base Case:* Within the while loop, calculate the mid index: **mid = left + (diff between right and left / 2)**
- If the target equals the mid index, return mid index
- If left target is greater, then left becomes mid index +1
- If it's less than the middle index right = middle index - 1
- return -1 if not found (outside while loop)

5.3 Search 2D Matrix

Question: You are given an m x n 2-D integer array matrix and an integer target.

Each row in matrix is sorted in non-decreasing order. The first integer of every row is greater than the last integer of the previous row. Return true if target exists within matrix or false otherwise.

Can you write a solution that runs in O(log(m * n)) time?

- **Main idea:** Need to preform binary search *twice*.
- First binary search is done on the first column, which approximates where the actual value is
- To preform this first search, set up *iterative* binary search as usual
- **Edge Case:** When checking if the value is greater than the first in the row, must also check if the value is less than or EQUAL TO the last value in that row. This means that the value is *in* that row.
- If this condition is true, then you have found the row, and you can break from the first binary search loop. Store this row as the second array you will search over
- Now preform Binary Search a second time on this row, and return true or false depending on if the value was found or not

5.4 Koko eats Bananas

Question: You are given an integer array piles where piles[i] is the number of bananas in the ith pile. You are also given an integer h, which represents the number of hours you have to eat all the bananas.

You may decide your bananas-per-hour eating rate of k. Each hour, you may choose a pile of bananas and eats k bananas from that pile. If the pile has less than k bananas, you may finish eating the pile but you can not eat from another pile in the same hour.

Return the minimum integer k such that you can eat all the bananas within h hours.

- **Main idea:** The hardest part of this question is knowing what to preform a binary search on, as we need to find the minimum integer k such that all bananas can be eaten within h hours. In this case, the max that k can be is the largest pile.
- *Simplified Solution:* Preform binary search from k = 1 to max pile amount in banana piles: In other words, preform binary search on [1, max(piles)].
- In this case, we start at the half way pile. For this k value, check to see if all the bananas can be completed with this rate.
- If it cannot, this k value is too small, and we need to retry with a new larger k. So preform binary search on the upper half.

- If all the values can be completed in this time, this is a valid k, but it isn't necessarily the *smallest* k value. So we need to preform binary search again anyway on the left side of the array.
- If the smaller value doesn't work, then return the current min. If it does, repeat until the current min is the smallest working one.

5.5 Find Min in Rotated Sorted Array

Question: Find Minimum in Rotated Sorted Array Solved You are given an array of length n which was originally sorted in ascending order. It has now been rotated between 1 and n times. For example, the array nums = [1,2,3,4,5,6] might become:

- [3,4,5,6,1,2] if it was rotated 4 times.
- [1,2,3,4,5,6] if it was rotated 6 times.

- **Main idea:** Begin by assuming that the min is just the first element in the array. Now set up binary search set up as usual. Star with the left and right pointers, left and index 0, and right at the end of the array
- Calculate the mid index as usual: here are the two cases:
- Case A: nums[mid-index] is less than current-smallest. That means we found a new smaller candidate. Since the array is rotated sorted, the actual minimum can only be at mid or to the left of mid (because the drop happens once).
- Case B: Other wise set the left pointer to one greater than the mid index

5.6 Search Rotated Sorted Array

Question: You are given an array of length n which was originally sorted in ascending order. It has now been rotated between 1 and n times. For example, the array nums = [1,2,3,4,5,6] might become:

- [3,4,5,6,1,2] if it was rotated 4 times.
- [1,2,3,4,5,6] if it was rotated 6 times.

Given the rotated sorted array nums and an integer target, return the index of target within nums, or -1 if it is not present.

You may assume all elements in the sorted rotated array nums are unique,

- **Main idea:** This question is mainly checking if a value is in the sorted left or sorted right portion of the input array, given that it can be rotated a certain amount.
- Begin setting up binary search as usual: First check if the mid index equals the value you're looking for, if so, return that index

- Next check to see if the value in the left pointer is less that or equal to the value in the mid index: if it is, the means WE ARE IN THE LEFT SORTED PORTION.
- If we are in the left sorted portion, next check to see if the target value is less than the left value, or greater than the mid value, this would mean we need to update the left pointer to one over the mid index
- If this isn't True, then we can safely assume that the target is in the left sorted portion, so move the right pointer
- If the value of the left pointer is not greater than or equal to the mid point, then this means we are in the right sorted portion
- We can preform many of the same steps, once we know we are in the right sorted portion: if the target is less than the mid point, or greater than the value at the right pointer, we know we need to decrement the right pointer, as the target is not in this portion of the array. If it is, then we need to update the left pointer, as we can assume that the value is indeed in this section of the array

5.7 Search 2D Matrix II

Question: Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

- **Main idea:** This question is not quite as fast and Search 2D Matrix 1: One solution is just to set up a binary search function, and implement binary search on every row: This would produce and m(log(n)) solution, which is pretty fast
- Another solution is to start at the bottom left corner of the array, and check if that value is greater than or less than the target: if its less than, increment the j value, if its greater than decrement the i value
- If i or j is ever outside of the bounds of the matrix, then return false: if decrementing i or incrementing j ever leads to the target value, return True

6 Linked Lists

6.1 Reverse Linked List

Question: Given the beginning of a singly linked list head, reverse the list, and return the new beginning of the list.

- **Main idea:** Set up a previous and current pointer node that will be used to reassign next pointer values
- First check to see if the head node exists, if so, init prev to None, and current to the head

- While loop condition is while current exists:
- Create a temp value that stores currents next node value
- Set current.next equal to prev
- Set prev equal to current
- Set current equal to temp
- Once this loop has finished, the current value will be None, and we need to return the new head, which is the last node that came before None
- So we return prev

6.2 Cycle Detection Linked List

Question: Given the beginning of a linked list head, return true if there is a cycle in the linked list. Otherwise, return false.

There is a cycle in a linked list if at least one node in the list can be visited again by following the next pointer.

- **Main idea:** Set up a two pointers, one fast pointer and one slow pointer
- The slow pointer will increment one each upon each iteration, the fast pointer will increment by two.
- The idea is that the gap between the fast and slow pointer will gradually grow smaller and smaller until they point to the same node. This can only happen if there is a loop in the linked list
- Set up both pointers to begin at the head node, if the head exists
- Looping condition: While first pointer exists: first = first.next
- Check if second.next and second.next.next exists, if so, second = second.next.next
- Check if first = second. *Note:* Must specifically check that first equals second and not their values. Checking if the node struct stored in memory is the same struct, not just the values!

6.3 Merge Two Sorted Lists

Question: You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

- **Main idea:** Create a new head node that you will add two each list to, depending on which list has the smaller node when comparing the two
- Set up a current node that will be assigned to the new head node
- While looping condition: If list1 and list2, continue looping and comparing

- At the beginning of each loop, create a new node, and assign current at list1 or list2 to the new node depending on which one is smaller (increment list1 = list1.next or list2 = list2.next, depending on which was smaller)
- Set the current.next to new node, and then increment current: current = current.next
- After while loop is finished, check if list1 or list2 still has any nodes, if either does, assign the remaining to the current.next
- Return new-head.next (because new head is just a dummy node, and doesn't have any value)

6.4 Remove Nth Node From End of List

Question: You are given the beginning of a linked list head, and an integer n.

Remove the nth node from the end of the list and return the beginning of the list.

- **Main idea:** Use a dummy node, and count all nodes in the list. After you have counted all the nodes, reset the current node to the beginning of the linked list (dummy node).
- Now, knowing how many nodes there are total, we can loop to the NODECOUNT - nth node. Once we have arrived at this node, we can delete the Nth node by setting the current to current.next.next

6.5 Copy Linked List with Random Pointer

Question: You are given the head of a linked list of length n. Unlike a singly linked list, each node contains an additional pointer random, which may point to any node in the list, or null.

Create a deep copy of the list.

The deep copy should consist of exactly n new nodes, each including:

- The original value val of the copied node
- A next pointer to the new node corresponding to the next pointer of the original node
- A random pointer to the new node corresponding to the random pointer of the original node
- Note: None of the pointers in the new list should point to nodes in the original list.

Return the head of the copied linked list.

- **Main idea:** Begin by creating a hashmap, where the key is the original node (pointer), and the value is a new node.

- Begin by looping through the original list, creating new nodes (and adding the value to each node)
- Now that you have all of the new nodes in the hashmap, loop through the original list again, and use the keys to determine where the random pointer should point to.
- Remember, the random pointer assignment should always point to a new copied node!
- Edge cases: There is a chance that the random pointer points to None, or the current.next is none. Before assigning a node, check to see if the value in the original linked list is None or not. If it's node, simply assign it to None, if not, assign it following the structure of the Algorithm.

6.6 Add Two Numbers In Linked List

Question: You are given two non-empty linked lists, l1 and l2, where each represents a non-negative integer.

The digits are stored in reverse order, e.g. the number 123 is represented as 3 -¿ 2 -¿ 1 -¿ in the linked list.

Each of the nodes contains a single digit. You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Return the sum of the two numbers as a linked list.

- **Main idea:** Use a carry bit: loop through both linked lists so long as current in each exist, and the carry is not 1
- In each iteration, if the first list exists, add its value to a current sum, if the second exists, add its value to a current sum, if the carry is 1, add the carry as well.
- If the current sum is over 9, subtract 10, and add 1 to the carry. Repeat until all nodes from original list have been added.
- Make sure to create nodes (and a dummy node to begin) as you go!

6.7 Find The Duplicate Number

Question: You are given an array of integers nums containing n + 1 integers. Each integer in nums is in the range [1, n] inclusive.

Every integer appears exactly once, except for one integer which appears two or more times. Return the integer that appears more than once.

Can you solve the problem without modifying the array nums and using O(1) extra space?

- **Main idea:** This is a cycle detection problem: the idea is that you use the values in the array as indicate into other parts of the array: if you arrive at the same spot using fast and slow pointers, then you've hit an interception!
- Once you've hit an interception, you must create a new pointer and assign it to the initial node.
- Now increment the initial pointer, and one of the pointers at an interception until they intercept once more.
- At this index, this is where the duplicate is!

6.8 Sort Linked List

Question: Given the head of a linked list, return the list after sorting it in ascending order.

- **Main idea:** Note, this is not the optimal solution, as the optimal solution uses constant space, but here is how you solved it first rip off the dome
- First Loop through all the nodes in the linked List, add them in an array.
- Next Heapify this array:
- Next for each value in the array, create a new node, and heappop / add the value to that node in a new list (you can create a dummy node to start).
- Return the dummy.next of the start node. This is a new list that was sorted in n(log(n)) time and is in ascending order

6.9 Reorder List

Question: You are given the head of a singly linked-list.

The positions of a linked list of length = 7 for example, can intially be represented as:

[0, 1, 2, 3, 4, 5, 6]

Reorder the nodes of the linked list to be in the following order:

[0, 6, 1, 5, 2, 4, 3]

- **Main idea:** Reverse the links of the second half of the linked list, then merge the two lists together
- Start by finding the mid point, this can be done with fast and slow pointers: start the first pointer at the first index, and the second index at the head.next
- Keep looping until the second pointer is None. At this point, the first pointer is at the halfway point, this means that the second linked list will start at first.next: Make sure to also mark END OF THE FIRST HALF AS NULL
- Now that you have the start of the second list, create a temp node, and reverse the linked list as you normally would

- Now that you have the second list reversed, its time to merge them together. Reset the first node to the front of the list, and the second node to the end of the second list
- The while condition is dependant on the SECOND POINTER: as the length of the second list can be shorter than the first, if the length is odd:
- Merge them together, first get the next values of the first and second pointer, then set the first pointer next to the second, and the second pointer next to the first-next. Then update first to first-next, and second to second-next
- This question is annoying and pretty hard!

7 Trees

7.1 Invert Binary Tree

Question: You are given the root of a binary tree root. Invert the binary tree and return its root.

- **Main idea:** Preform a DFS on the tree, before exploring left and right children, swap the left and right
- In explore function, first check if the current node is None, this is the base case: If it is, simply return
- If not None, create a temp node, and swap the left and right children. Then explore left node, and explore right node
- Explore function can be written as a part of the Solution Class
- In main function, call explore on root, then return root

7.2 Max Depth of Binary Tree

Question: Given the root of a binary tree, return its depth.

The depth of a binary tree is defined as the number of nodes along the longest path from the root node down to the farthest leaf node.

- **Main idea:** Preform a DFS on the tree, at each sub-problem, we want to pass up the max of the left and right at the current node
- In explore function, first check if the current node is None, this is the base case: If it is, simply return
- If not None, recurse on the left and right parts of the tree and store the depths, then return upward the max(left, right)
- Explore function can be written as a part of the Solution Class
- In main function, return the call to explore

7.3 Max Diameter of Binary Tree

Question: The diameter of a binary tree is defined as the length of the longest path between any two nodes within the tree. The path does not necessarily have to pass through the root.

The length of a path between two nodes in a binary tree is the number of edges between the nodes.

Given the root of a binary tree root, return the diameter of the tree.

- **Main idea:** Create a attribute of the Solution class (called max diameter) then preform a DFS on the tree
- The max diameter at each level is the longest number of edges spanning the left and right branches *added together*.
- The base case in this sense is if the current is NONE, it it is, RETURN -1, as there are no edges between the NONE node and the parent Node.
- At each sub-problem, recurse on the left and right sides (adding 1 to the return), and check if right added to left is greater than the member class var of max diameter.
- If it is, set max diameter.
- We want to pass up the longest span at any given node, which is the max between the left and right span
- Call explore function from max diameter, and return the int value that explore returns.

7.4 Balanced Binary Tree

Question: The depth of a binary tree is defined as the number of nodes along the longest path from the root node down to the farthest leaf node.

- **Main idea:** This question is a bit trickier, as you need to return a tuple, and not just a depth (like edge depth or node depth)
- To begin preform a DFS on the tree. Start be defining the base case. If the node is NONE, return (0, True), as a None node is balanced. Because we are defining balance in terms of node height, and not edge height, base case starts at 0 and not -1.
- Next, recurse on the left and right sub trees, and add one to the depth part of the tuple and store the True or False vale.
- Check if the distance between left and right is equal to 1 or 0, if it is, set a current balanced value equal to True.
- If this current balanced is True, and so are the left and right return T/F values are true, then return True upwards to the next subproblem.

- In terms of return depth upwards, we want to return the max depth between the left and right sub nodes.
- So, return (max(left, right), (is-bal and left-bal and right-bal)) upwards
- In main function, return the True or false value from the tuple as the answer

7.5 Same Binary Tree

Question: Given the roots of two binary trees p and q, return true if the trees are equivalent, otherwise return false.

Two binary trees are considered equivalent if they share the exact same structure and the nodes have the same values.

- **Main idea:** Use DFS to search through each tree, checking nodes along the way. If DFS is preformed in the same order for both trees, then they are the same
- Two ways of solving this question: First is less buggy, but also less efficient. Idea is to create one explore call, and a visited array, and add nodes to the array upon being visited. Call explore twice, with two separate visited arrays, one for each tree
- Check the arrays, if they are the same, return true, else return False.
- *NOTE*: Doing it this way means you need to express the NULL Node in the array: In the base case, make sure to add None to the array, when you hit a None node.
- **BETTER WAY:** A better way is to modify the explore function directly and have it accept two nodes.
- Base case in this example becomes if *both* nodes are None, return True upwards. If one node exists, and the other doesn't. Return False.
- Recurse on the left and right subtrees, comparing nodes as you go.

7.6 Subtree of Another Tree

Question: Given the roots of two binary trees root and subRoot, return true if there is a subtree of root with the same structure and node values of subRoot and false otherwise.

A subtree of a binary tree tree is a tree that consists of a node in tree and all of this node's descendants. The tree tree could also be considered as a subtree of itself.

- **Main idea:** Use DFS to search through the main root tree. At each node, check if the current node val equals the target node val. If they do, then we can begin an is same tree check with this node as our new root.
- Checking if a tree is the same can be done in the two ways described in the previous question. We can use a DFS approach, and explore / check as we go, or create arrays that track the order in which the tree was traversed, and compare the arrays

- *SERIALIZE THE TWO TREES*: After banging my head against the wall, the best move it to serialize the lists.
- Once this exploration and checking has been done, if the result is true, we must set a global variable to True, if it's not, we can *keep exploring* the main rooted tree.
- If the main rooted tree gets completely traversed, return false
- Can create the root serization in each stack frame: examine code for best practice in using serialization

7.7 Binary Tree Level Order Traversal

Question: Given a binary tree root, return the level order traversal of it as a nested list, where each sublist contains the values of nodes at a particular level in the tree, from left to right.

- **Main idea:** This question boils down to preforming breadth first search and assigning nodes a level number when preforming the traversal.
- To begin, set up a queue, because queues are used to preform BFS. Also set up a level number and assign level=0.
- Start by putting the root and level=0 in the queue as a tuple. Now preform BFS on the queue.
- Inside the explore function, if the current node has a left and right, add them to the queue as you normally would, but as a tuple with level+=1. More specifically add (current.left, level+=1) and (current.right, level+=1) to the queue.
- This will properly assign each node to a level. When returning back from BFS, simply group together all values that have the same level into an array.

7.8 Right side view of binary Tree

Question: You are given the root of a binary tree. Return only the values of the nodes that are visible from the right side of the tree, ordered from top to bottom.

- **Main idea:** Traverse the tree with Breadth first search, as you go, index values based on depth (pair value with it's current depth)
- After you have finished the traversal, loop through all elements. For any given element, if the next element in the array has a level thats one greater than the current, then you know that the current node is on right side view (add it to an array that tracks these nodes -¿ return array as answer)

7.9 Count Good Nodes in Binary Tree

Question: Within a binary tree, a node x is considered good if the path from the root of the tree to the node x contains no nodes with a value greater than the value of node x

Given the root of a binary tree root, return the number of good nodes within the tree.

- **Main idea:** Traverse the tree with DFS, add a global attribute count to the class
- Start traversing but pass along a max value node: start the max value at negative infinity
- As you traverse, if the current value is greater than or equal to the max value, add 1 to count, and pass the current value down up the recursion stack as the new max value: don't return anything recursively

7.10 Validate Binary Search Tree

Question: Given the root of a binary tree, return true if it is a valid binary search tree, otherwise return false.

A valid binary search tree satisfies the following constraints:

The left subtree of every node contains only nodes with keys less than the node's key. The right subtree of every node contains only nodes with keys greater than the node's key. Both the left and right subtrees are also binary search trees.

- **Main idea:** Recall that a valid binary search tree is where at each sub tree, the left node's value is always less than the parent's value, and the right node's value is always greater than the parents
- Recursively, we must check if all subtrees that came before are valid, and if the current left, right, and parent are valid.
- Let's start with the base case, if the current Node is None, return the minimum and maximum value at this subtree (which are both None), and also if it's a valid BTS or not: In this case, it is! so return (NONE, NONE, True)
- Recursively call of both left and right, and store left-min, left-max, and left-valid as values returned from the recursive call: do the same with the right values
- Check first to see if either left-valid and right-valid are False, if so, return false upward.
- If they are both true, then check if the current value is greater than max-left and less than the min-right, if so, then the current tree is valid
- *****Always pass upward the largest value on the left side of the tree, and the smallest value on the right side of the Tree. This ensures that parent trees remain valid****
-

7.11 Lowest Common Ancestor in Binary Tree

Question: Given a binary search tree (BST) where all node values are unique, and two nodes from the tree p and q, return the lowest common ancestor (LCA) of the two nodes.

The lowest common ancestor between two nodes p and q is the lowest node in a tree T such that both p and q as descendants. The ancestor is allowed to be a descendant of itself.

- **Main idea:** Leverage the fact that it's a binary tree: The main idea is that at any given time, if p and q are split on the left and right sides of the tree, then the current is the lowest common ancestor
- If at any point, the current is the left or right node, then it has to be the lowest common ancestor, this is a given. If the nodes split, we also return the current node
- If the values of p and q are BOTH greater than the current value, then recurse to the right sub node, and recursively call the function again
- If the values of p and q are BOTH less than the current value, then recurse to the left sub node, and recursively call the function again

7.12 Kth Smallest Element in BST

Question: Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) in the tree.

A binary search tree satisfies the following constraints:

- The left subtree of every node contains only nodes with keys less than the node's key.
- The right subtree of every node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees are also binary search trees

- **Main idea:** Preform a BFS on the tree: pass in an array that you will append nodes to as you go.
- Between the left and right recurse calls to traverse, add the current node to the ongoing array
- After the DFS has returned, the array will be in SORTED ORDER: this means that you can simply return the element at arr[k-1]

7.13 Construct Tree from Preorder / Inorder Arrays

Question: You are given two integer arrays preorder and inorder. preorder is the preorder traversal of a binary tree. inorder is the inorder traversal of the same tree. Both arrays are of the same size and consist of unique values.

Rebuild the binary tree from the preorder and inorder traversals and return its root.

- **Main idea:** You must solve this recursively: base case is if the preorder or inorder arrays are length 0: if they are, return None
- If they are not, then create a node called Root: Root will always be the first value in preorder array
- Next find the index of root in the preorder array: Call this index Mid:
- Root.left will equal : self.buildTree(preorder[1: mid + 1], inorder[:mid]) // Skip the first value for pre-order (as it is now a node: and go to the mid index inclusive): for inorder: go up until the mid index
- Root.right will equal : self.buildTree(preorder[mid + 1:], inorder[mid +1:]) // everything after the mid index, for both the preorder and inorder array
- You do not like this question

8 Heap and Priority Queue

8.1 Kth Largest Element in a Stream

Question: Design a class to find the kth largest integer in a stream of values, including duplicates. E.g. the 2nd largest from [1, 2, 3, 3] is 3. The stream is not necessarily sorted.

Implement the following methods:

- constructor(int k, int[] nums) Initializes the object given an integer k and the stream of integers nums.
- int add(int val) Adds the integer val to the stream and returns the kth largest integer in the stream.

- **Main idea:** Use a heap! By default, python has a built in heap library called heapq that uses min heaps
- In the constructor set the value k to self.k
- Next, heapify the array, this is basically a free operation, as it takes liner time : O(n). *NOTE:* Heapify happens in place (pass by reference), this means that we do not assign the array to a new variable.
- Next set the heapifyed array to another attribute variable (say streams).
- In the addition function, first heappush() the value into the array.

- **heappush() time complexity:** Heappush adds the new values to the last value of the tree, the bubbles up. Bubbling up involves swapping nodes upward towards the root. This swapping can occur a max number of times thats equal to the height of the tree. So worst case is O(log(n)), time.
- **heappop() time complexity:** Heappop is simular to heappush, as it swaps nodes. Only the root is swapped with a leaf node, the leaf node is removed and returned, and sink down swapping function is called on the root. This again can happen in O(log(n)) time.
- In our add function, after we've added the node, we can now heap pop so long as the length of the stream is greater than our kth value. In other words, keep popping until you've popped k times: return the kth element. Time complexity is O(klog(n))

8.2 Last Stone Weight

Question: You are given an array of integers stones where stones[i] represents the weight of the ith stone.

We want to run a simulation on the stones as follows:

At each step we choose the two heaviest stones, with weight x and y and smash them together

- If x == y, both stones are destroyed
- If x ¡ y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.
- Continue the simulation until there is no more than one stone remaining.

Return the weight of the last remaining stone or return 0 if none remain.

- **Main idea:** Use a heap! This is a max heap problem, so the hardest part about the solution is ensuring we're translating the question correctly while using our min heap. To begin, we can multiply every value by -1 in the array, then call heapify.
- In the simulation, we choose the two heaviest stones, which means the two most negative stones. So heappop twice, and set x and y equal to the return values. Loop and grab the two largest values so long as the length of the heap is greater than 1
- Now check to see if both of the values are the same, if they are, continue to the next iteration (continue inside while loop)
- If x ¡ y, then we must be careful in translating this to logic in the question, this condition will only happen if x is actually greater than y due to the values being negative. If this is the case, then we need to add y back in with weight y-x. In this case, we will add y back in with weight x - y. To see this math work out, I suggest writing out a couple of values to see how the negative works.

- Once the looping expression evals to false, check to see if there are 1 or None values left in the heap
- If there are none, then return 0, else, return -1 * heap[0] - ¿ remember to turn the negative back to positive!

8.3 K Closest Points to Origin

Question: You are given an 2-D array points where points[i] = [xi, yi] represents the coordinates of a point on an X-Y axis plane. You are also given an integer k.

Return the k closest points to the origin (0, 0).

The distance between two points is defined as the Euclidean distance ($sqrt((x1-x2)^2 + (y1-y2)^2)$).

You may return the answer in any order.

- **Main idea:** Use a heap to store the distance values! Start by looping through all the points, and calculating the distances of each point.
- Once you've calculated the distance, store that distance in an array that you'll heapify
- Also store the distance as the key in a hashmap: the values will be *an array of all the (x,y) coordinates that form this distance*
- Once you have heapifyed the distance array, begin heappopping and looking up a x,y values from the array of possible values.
- Add each (x,y) value as an array to an answer array. Do this k times, then return the answer array

8.4 Kth Largest Element

Question: Given an unsorted array of integers nums and an integer k, return the kth largest element in the array.

By kth largest element, we mean the kth largest element in the sorted order, not the kth distinct element.

Follow-up: Can you solve it without sorting?

- **Main idea:** Use a heap, heapify it, then loop k times and return the kth largest element
- This requires multiplying every element in the array by -1 to start, because we implement heapq which is a min heap
- This problem can also be solved traditionally with sorting

8.5 Task Scheduler

Question: You are given an array of CPU tasks tasks, where tasks[i] is an uppercase english character from A to Z. You are also given an integer n.

Each CPU cycle allows the completion of a single task, and tasks may be completed in any order.

The only constraint is that identical tasks must be separated by at least n CPU cycles, to cooldown the CPU.

Return the minimum number of CPU cycles required to complete all tasks.

- **Main idea:** This question involves using both a heap and a queue to schedule tasks. The intuition is to use a max heap that always pops the task with the most elements.
- Begin by creating a hashmap with all the tasks as the keys, and the value being their frequencies (the number of times they will need to get scheduled).
- Begin by popping the first task, and adding that task, along with its expiring time, to the queue.
- When adding a to the queue, always add the time to live, along side the CURRENT TIME.
- After each iteration, always check if the front of the queue has an expiring time, if so, add it back to the stack.
- When adding back to the stack, make sure to decrement the overall frequency.
- Repeat until the stack and queue are empty. Return the global counter.

9 Graphs

9.1 Number of Islands

Question: Given a 2D grid grid where '1' represents land and '0' represents water, count and return the number of islands.

An island is formed by connecting adjacent lands horizontally or vertically and is surrounded by water. You may assume water is surrounding the grid (i.e., all the edges are water).

- **Main idea:** Preform a DFS on the graph using a visited nodes list. Explore all neighbor nodes of a starting node, so long as the neighbor has not been visited.
- We will wrap the DFS call in a for loop that loops through every node. Every time the explore call returns, we will += our island counter by the return call of explore (which will return 1 or 0). This is the value that we will eventually return.
- Inside our explore function, we must first check to see if the node had a value 1 or 0, if its 0, simply 0

- If the value is 1, then we can explore all of the nodes neighbors.
- We can fist built our another function that returns all the neighbors of the given node. We can then loop through this list, for every non-visited neighbor.
- After we have finished the recursive calling, we can return 1 to the parent wrapper function.
- Now having finished exploring the graph, we can return the island counter.
- **Note:** I used a list of tuples (x,y) to store visited nodes, but there's probably a better way to do this.

9.2 Max Area of Islands

Question: You are given a matrix grid where grid[i] is either a 0 (representing water) or 1 (representing land).

An island is defined as a group of 1's connected horizontally or vertically. You may assume all four edges of the grid are surrounded by water.

The area of an island is defined as the number of cells within the island.

Return the maximum area of an island in grid. If no island exists, return 0.

- **Main idea:** Preform a DFS on the graph using a visited nodes list. Explore all neighbor nodes of a starting node, so long as the neighbor has not been visited.
- We will wrap the DFS call in a for loop that loops through every node. Inside the explore call, we need to have an internal counter that adds to the area as we traverse. We then return that area back to the parent function, and test to see if its greater than whatever our current max area is.
- As for the other inland question, we can fist built another function that returns all the neighbors of the given node. We can then loop through this list, for every non-visited neighbor.
- Once our main parent for loop has terminated, we should have counted all the island's areas, and found the largest.
- **Note:** Recursively internally counting can get tricky, make sure you consider what you're passing up to the parent. One way of doing this is to have a local area number set to 1, then return that area upward when you do the recursive call.

9.3 Deep Clone a Graph

Question: Given a node in a connected undirected graph, return a deep copy of the graph.

Each node in the graph contains an integer value and a list of its neighbors.

- **Main idea:** Start by creating a hashmap of new nodes, where the node val is the new, and a new initialized node is the value

- Next, preform a DFS or BFS on the original graph: this uses a stack or queue per usual, only now, make sure to set the value + neighbor nodes of the new nodes in your hashmap as you traverse.
- Make sure you add each neighbor to the new node * But make sure you don't have duplicate neighbors!

9.4 Islands and Treasure

Question: You are given a m×n 2D grid initialized with these three possible values:

- -1 - A water cell that can not be traversed.
- 0 - A treasure chest.
- INF - A land cell that can be traversed. We use the integer: 2147483647 to represent INF.

Fill each land cell with the distance to its nearest treasure chest. If a land cell cannot reach a treasure chest then the value should remain INF.

Assume the grid can only be traversed up, down, left, or right.

Modify the grid in-place.

- **Main idea:** This question is not bad: begin by setting up a bfs function: the main idea is that you'll loop through the grid, and anytime you hit a number, preform BFS on that value and see if you can find a treasure.
- Within BFS, make sure you are counting levels: if you ever hit a 0, then return the level count
- If you never hit a 0, return None from the BFS function, indicating that we do not need to change the value of this node
- Remember that you can create a nice helper function to get all neighbors!
- Whenever a -1 value is reached, simply do nothing, do not traverse or add to neighbor list
- use tuples (i,j) to store nodes in visited list

9.5 Pacific and Atlantic Ocean

Question: You are given a rectangular island heights where heights[r][c] represents the height above sea level of the cell at coordinate (r, c).

The islands borders the Pacific Ocean from the top and left sides, and borders the Atlantic Ocean from the bottom and right sides.

Water can flow in four directions (up, down, left, or right) from a cell to a neighboring cell with height equal or lower. Water can also flow into the ocean from cells adjacent to the ocean.

Find all cells where water can flow from that cell to both the Pacific and Atlantic oceans. Return it as a 2D list where each element is a list [r, c] representing the row and column of the cell. You may return the answer in any order.

- **Main idea:** Create two groups, nodes that are touching pacific water, and nodes that are touching atlantic water
- For each pacific water node, do a DFS to find all neighbors, where each neighbor is valid if its value is GREATER than the current node. Do this for all nodes touching pacific, and keep track of them
- Do the same for atlantic nodes, preform a DFS and check if each of these neighbors can reach the atlantic nodes.
- After you have preformed each DFS, check to find the nodes in both groups. Nodes in both groups are the answer.

9.6 Graph Valid Tree

Question: Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

- **Main idea:** A valid tree has the following components: No more than one connected component, Not Cyclical.
- In order to check these things, we can use DFS approach. Create a connected component counter. Loop through each node, and attempt to call DFS on the node (so long as it isn't already visited). Each time you finish the DFS, and the stack is empty, add 1 to the connected-component counter.
- This question gives you edges, begin by making these edges into a valid adjacency list
- Once you have finished all of the DFS, check the visited array: if the length of the visited array is not equal to n, then there is a cycle (nodes can reach the same node, from two separate nodes). This means that this is not a valid tree
- One edge case is if the node loops to itself, if this is the case, then it will return False as well
- If the connected components equal 1, and the length of visited equals n, return True

9.7 Number of Connected Components in an Undirected Graph

Question: There is an undirected graph with n nodes. There is also an edges array, where edges[i] = [a, b] means that there is an edge between node a and node b in the graph.

The nodes are numbered from 0 to n - 1.

Return the total number of connected components in that graph.

- **Main idea:** Begin by transforming the input edges into an adjacency list. Next create your stack and visited array: for each n, check first to see if n is tracked in the adjacency list. If it's not, its a separate component, and you can add one to a connected-component counter and continue.
- Make sure to attempt to call DFS on all the n values. After you have looped through all of them, return the connected component counter.

9.8 Course Schedule

Question: You are given an array prerequisites where prerequisites[i] = [a, b] indicates that you must take course b first if you want to take course a.

The pair [0, 1], indicates that must take course 1 before taking course 0.

There are a total of numCourses courses you are required to take, labeled from 0 to numCourses - 1.

Return true if it is possible to finish all courses, otherwise return false.

- **Main idea:** You need to keep track of two things, incoming edges for each node, as well as neighbors of each node (what classes can be taken next, after the current node is completed)
- To begin, set up an adjacency list, that keeps track of all classes that can be completed after the current node is done
- Now create an list that keeps track of all incoming edges. Because classes are indexed from 0 to numClasses, you can create an array: incoming-edges = [0]*numClasses
- Make sure to add all incoming edges to each class. Once done, add all classes with no incoming edges to a queue.
- While the queue isn't empty, pop a class from the queue, and check to see what classes can be processed after this class has been completed
- For each neighbor in this class's adjacency list, subtract one from that neighbors incoming edge list. If the incoming edge is now 0, then add that neighbor to the queue.
- Each time you pop from the queue, add 1 to a running completed course counter. After the queue is empty, check to see if the completed course amount equals the number of courses

9.9 Rotten Oranges

Question: You are given a 2-D matrix grid. Each cell can have one of three possible values:

- 0 representing an empty cell
- 1 representing a fresh fruit

- 2 representing a rotten fruit

Every minute, if a fresh fruit is horizontally or vertically adjacent to a rotten fruit, then the fresh fruit also becomes rotten.

Return the minimum number of minutes that must elapse until there are zero fresh fruits remaining. If this state is impossible within the grid, return -1.

- **Main idea:** This question implements multi source BFS, meaning that we have multiple sources where the BFS graph traversal occurs.
- To begin, loop through the grid and count all the fresh oranges, this will help us determine if all fresh are turned rotten at the end of the problem.
- When traversing, make sure to add at the rotten positions to the queue (there are our starting points)
- While the queue isn't empty, begin traversing: at the beginning, POP ALL THE ROTTEN from the queue: create a for loop that pops all rottens, then for that rotten, gather all neighbors that are fresh, and mark the fresh ones as rotten, then add those to the queue
- After the for loop has finished, this marks one unit of time passing, so increment time by 1.
- As you get the fresh fruit neighbors, count them.
- When the queue is empty, now check if the amount of fresh neighbors traversed is the same as the original counted. If this is the case, then return the time count var, if it isn't, return -1

10 Advanced Graphs

10.1 Network Delay Time

Question: You are given a network of n directed nodes, labeled from 1 to n. You are also given times, a list of directed edges where times[i] = (ui, vi, ti).

- ui is the source node (an integer from 1 to n)
- vi is the target node (an integer from 1 to n)
- ti is the time it takes for a signal to travel from the source to the target node (an integer greater than or equal to 0).

You are also given an integer k, representing the node that we will send a signal from.

Return the minimum time it takes for all of the n nodes to receive the signal. If it is impossible for all the nodes to receive the signal, return -1 instead.

- **Main idea:** This is a classic DIJKSTRAS ALGO approach. I'm going to walk through the best steps in order to implement DIJKSTRAS in python, because although you knew how to do this problem, implementing DIJKSTRAS correctly was actually really buggy and hard.
- To begin, change whatever internal data structure they give you into an adjacency list.
- The adjacency list is a dictionary, where the key is a node, and the value is a tuple with the neighbor node (v) and the weight between (u,v). **NOTE:** Not every node will be in the adjacency list! This is as expected
- Once you have created the adjacency list, we now need to create the distances dictionary that will keep track of the currently running distances to each node. To begin, set the start node k to distance 0. (Distance from k to itself is just 0). Next, for all the remaining nodes, set the distance to INF. (Will use math.inf, which requires importing math).
- Next, set up the min-heap that you will *heappush and heappop from.* Put the starting tupple (0, k) in the min heap
- Now start the main looping. While the length of the min heap isn't 0, pop the tuple with the smallest distance. Because this is a heap, just heappop.
- Now that you have the current smallest, check to see if the current stored distance to this node is smaller than the current distance popped from the heap. If it is smaller, then we can continue in the loop, as there is already a shorter distance found to this node.
- If it's not smaller, then for every neighbor in the adjacency list of this node, we must preform the following check: distances[v] ¿ curr-dist + w
- if this is true, then we must update distance[v] to curr-dist + w, and then heappush this neighbor and weight into the min-heap
- After the while loop has finished, and all nodes have been explored, we can check if any nodes were unreachable, in this case, if any nodes in the distances dictionary are math.inf, then return -1
- If all distances are in the dictionary, then we must return the max of all the distances in the dictionary, this equal the min over all nodes that needed to be communicated to. This problem was kinda annoying to solve because I had to implement DIJKSTRAS for the first time but moving forward hopefully it's not too bad

11 One-Dimensional Dynamic Programming

11.1 Climbing Stairs

Question: You are given an integer n representing the number of steps to reach the top of a staircase. You can climb with either 1 or 2 steps at a time.

Return the number of distinct ways to climb to the top of the staircase.

- **Main idea:** We can use **Memoization** to keep track of sub solutions to each part of the problem.
- **Subproblem:** At each step, the number of distinct ways that can be reached is the sum of the number of distinct ways that can be reached at steps n-1 and n-2. memo[n-1] + memo[n-2].
- We can set our current memo value, to the sum of the precious two memo values.
- To begin this problem, (or any DP problem), always start by populating the memo with the base cases of the problem.
- In this case, we populate our memo with memo = 1:1, 2:2
- At the beginning of our recursion, we will check to see if the current value step is in the memo, if it is, then we return that memo value.
- If not, we add a new entry to our memo that equals our sub problem: memo[n] = memo[n-1] + memo[n-2]. To do this, we first must check to make sure that memo[n-1] and memo[n-2] exist. If we're doing the top down approach, they won't, so we make a recursive call: memo[n] = self.memoClimb(n-1, memo) + self.memoClimb(n-2, memo).
- We then return the memo[n] upward
- **Note:** This problem can also be done iteratively, and honestly probably is easier to do that way, I put both my solutions in the python file.

11.2 Min Cost Climbing Stairs

Question: You are given an array of integers cost where cost[i] is the cost of taking a step from the ith floor of a staircase. After paying the cost, you can step to either the (i + 1)th floor or the (i + 2)th floor.

You may choose to start at the index 0 or the index 1 floor.

Return the minimum cost to reach the top of the staircase, i.e. just past the last index in cost.

- **Main idea:** We can use **Memoization** to keep track of sub solutions to each part of the problem.
- **Subproblem:** At each step, the minimal cost would be the cost to get to that current step, plus the min at steps n-1, n-2. : memo[step] = currentCost + min(self.memoStairs(step-1, cost, memo), self.memoStairs(step-2,cost, memo))
- To begin this problem, (or any DP problem), always start by populating the memo with the base cases of the problem.
- In this case, we populate our memo with the costs associated at steps 1 and 2
- At the beginning of our recursion, we will check to see if the current value step is in the memo, if it is, then we return that memo value.
- If not, we add a new entry to our memo that equals our sub problem.

- Because we need to get to the last stair, the stair that doesn't have a value, we need to create a current cost variable that tracks the current values current cost. If the last stair is met, then the current cost will just equal 0.
- After all of the recursion has been done, we will return memo[step]

11.3 House Robber

Question:

You are given an integer array nums where nums[i] represents the amount of money the ith house has. The houses are arranged in a straight line, i.e. the ith house is the neighbor of the (i-1)th and (i+1)th house.

You are planning to rob money from the houses, but you cannot rob two adjacent houses because the security system will automatically alert the police if two adjacent houses were both broken into.

Return the maximum amount of money you can rob without alerting the police.

- **Main idea:** We can use **Memoization** to keep track of sub solution: We can solve this one with bottom up approach.
- **Subproblem:** At each step, to maximize profit, we need to consider the following:
- SUB-PROBLEM max(house[i] + memo[i-2], memo[i-1])
- Start our memo with the max at house 1, which is just house 1, and the max at house 2, which is equal to max(house[0], house[1])
- After you have init the memo with the first two houses, preform the sub-problem check on values i=2 through n.
- The result last element of the memo

11.4 House Robber II

Question: You are given an integer array nums where nums[i] represents the amount of money the ith house has. The houses are arranged in a circle, i.e. the first house and the last house are neighbors.

You are planning to rob money from the houses, but you cannot rob two adjacent houses because the security system will automatically alert the police if two adjacent houses were both broken into.

Return the maximum amount of money you can rob without alerting the police.

- **Main idea:** We can use **Memoization** to keep track of sub solution: We can solve this one with bottom up approach.
- **Subproblem:** The sub-problem is the same as house robber I, but we need to run the memo on two instances, since the input array is "circular".
- SUB-PROBLEM max(house[i] + memo[i-2], memo[i-1])

- Run the bottom up iteration on the input array from index i=1, to i= len(input-arr): *IGNORE THE FIRST HOUSE
- Run the bottom up iteration on the input array from index i=0, to i=len(input-arr) -1: *IGNORE THE LAST HOUSE
- After you have solved both instances, the solution to this problem is the max of both of these.

11.5 Longest Palindromic Substring

Question: Given a string s, return the longest substring of s that is a palindrome.

A palindrome is a string that reads the same forward and backward.

If there are multiple palindromic substrings that have the same length, return any one of them.

- **Main idea:** The trick to this question is to check palindromes from the center character, having a left pointer expand leftwards from the center, and a right pointer expanding rightwards form the pointer
- The idea is that you'll check each potential palindrome starting at each character: to do this, have a for loop set up that loop through each character, for each character, create a left and right that starts at this current character. While these two characters are the same, spread the pointers away from the center.
- While doing this, check to see if the distances between the left and right pointers is every greater than an ongoing max variable (if it is, update the max, and keep track of the left and right position)
- NOTE: EDGE CASE: There is a huge edge case to this question: for each middle character, also create a for loop that checks with the pointers as the following
- Set up left = mid and right = mid +1 THIS IS CRITICAL: as this works on characters that are EVEN in LENGTH
- So for each mid character, you are actually running two while loop cases, with left, right starting at the middle, and for the second: left starting at the middle and right starting at mid +1 (make sure to check if left or right are ever out of bounds!)

11.6 Palindromic Substrings

Question: Given a string s, return the number of substrings within s that are palindromes.

A palindrome is a string that reads the same forward and backward.

- **Main idea:** This question is very similar to the Longest Pal substrings question: The idea is to check palindromes from the center character, and work your way outward with left and right pointers

- In this case, every time you increment the pointers and find a palindrome, add one to a count variable.
- NOTE: the edgecase still persists with even characters: make sure to run the inner while loop twice, starting the right index at one more then the left / mid index

11.7 Decode Ways

Question: A string consisting of uppercase english characters can be encoded to a number using the following mapping:

(a) 'A' → "1"

(b) 'B' → "2"

(c) ...

(d) 'Z' → "26"

To decode a message, digits must be grouped and then mapped back into letters using the reverse of the mapping above. There may be multiple ways to decode a message

- **Main idea:** Do Bottom Up DP from the END Of the string: Set it up such that the index in the memo are the keys, and the values are the amount of ways to count at any given point.
- Begin by initializing the memo with {len(s) : 1} This case is saying that given an empty string there's one way to decode it
- Next we will set up a loop that loops from len(s) -1, all the way to 0 (-1, -1, -1).
- Upon each loop here are the checks that you'll make
- if s[i] == "0" then then memo[i] will equal 0, as you cannot decode 0 in this case (if this case is true, continue looping to the next index)
- if s[i] != "0" then then memo[i] will equal whatever memo[i+1] is
- Additionally, check to see if so if the value at s[i] and s[i+1] equals a valid two digit number (10-26): if this is the case, ALSO ADD memo[i+2] to memo[i]
- After looping is complete, return memo[0] (the total sum of all the different ways)

11.8 Coin Change

Question: You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

- **Main idea:** COOL BUT TOUGH. We will use bottom up DP, begin by initializing a memo with each number from 0 to the target number (amount). This means you loop through 0 to amount+1
- Init all values in the memo to positve infinity, except for key 0, init that to 0.
- The idea here is that we want to find the minimum amount of coins of each value leading up to our target value amount.
- To do this, we will start a loop that will loop from 1 to amount (amount+1). At each amount, we will check all coin values in each array by subtracting the current amount (i) by the coin.
- If this amount is greater than or equal to zero, we need to update out memo.
- Our memo will be updated to the min between what's already in our memo : memo[i], as well as 1 + (i - coin[i]).
- Another way to formulate this is with: memo[i] = min(memo[i], memo[remainder] + 1).
- We're basically saying that the new memo equals whatever amount of coins it takes to create the remainder plus one for the current coin we're deducting. We also check to see if that is smaller than whatever is currently stored as the min.
- At the end of all the DP, we will check to see if memo[amount] == infinity: if it does, return -1, if it doesn't return memo[amount]

12 Two-Dimensional Dynamic Programming

12.1 Unique Paths

Question: There is an m x n grid where you are allowed to move either down or to the right at any point in time.

Given the two integers m and n, return the number of possible unique paths that can be taken from the top-left corner of the grid (grid[0][0]) to the bottom-right corner (grid[m - 1][n - 1]).

You may assume the output will fit in a 32-bit integer.

- **Main idea:** This question is not bad at all. Consider the following: At any point on the grid, you only have two options, move down, or move to the right. The total number of ways to reach the current square is the sum between the number of ways you could move at the square down, plus the number of ways from the square at the right.
- At each square, add the numbers of ways at grid[i+1][j] + grid[i][j+1]. This gives you the total number of ways at grid[i][j].
- Say you are just to the left of the final state, in this case, you cannot add whats below, because it doesn't exist. To solve this, expand the rows and columns by one, and fill in these as 0. So now when you add from this "out of bounds row", you simply add zero.

- Start looping the last row, upwards toward the starting position. Make sure to mark the final node grid[m-1][n-1] equal to 1!
- After you've looped from the bottom row upward to the top row, making sure to add the right and bottom cells as you go, the final total amount of ways is the numbers of ways at the starting element: grid[0][0]: this is the solution.

12.2 Longest Common Subsequence

Question: Given two strings text1 and text2, return the length of the longest common subsequence between the two strings if one exists, otherwise return 0.

A subsequence is a sequence that can be derived from the given sequence by deleting some or no elements without changing the relative order of the remaining characters.

For example, "cat" is a subsequence of "crabt". A common subsequence of two strings is a subsequence that exists in both strings.

- **Main idea:** Use a grid to keep track of the summation of sub-problems: begin by Initializing a grid with a length one greater than the width and height of the input text: text1 and text2. Fill the full 2d array with zeros
- Next, start at the postion [len(text1)-1][len(text2)-1] in the grid (bottom right corner, one row / col up)
- at each position in the grid, check the following:
- if text1[i] == text2[j] -¿ grid[i][j] = 1 + grid[i+1][j+1]
- else: grid[i][j] = max(grid[i+1][j], grid[i][j+1])
- Loop through the grid backwards by row and column, repeating this, until you have iterated through the full thing

i Once done, return the number (summation of all sub-problems) at grid[0][0] this is the answer

13 Backtracking

13.1 Subsets

Question: Given an array nums of unique integers, return all possible subsets of nums.

The solution set must not contain duplicate subsets. You may return the solution in any order.

- **Main idea:** This is a backtracking question. Begin be defining a backtracking function that takes in an index, and a current path
- The idea is that you'll either add the value at the current index to the current-path array, the recurse again: backtrack(index+1, current-path), or when that's done executing, pop the last value out, then recurse again using the same call

- BASE CASE, if the index is ever greater than or equal to the length of the input array, then add the current path to the result array, and return

13.2 Combination Sum

Question: You are given an array of distinct integers nums and a target integer target. Your task is to return a list of all unique combinations of nums where the chosen numbers sum to target.

The same number may be chosen from nums an unlimited number of times. Two combinations are the same if the frequency of each of the chosen numbers is the same, otherwise they are different.

You may return the combinations in any order and the order of the numbers in each combination can be in any order.

- **Main idea:** This is a backtracking question. The idea is to backtrack using two choices, to either continue adding the current value to an ongoing path (and adding that value to a running total), or skipping that value, and adding the next number instead.
- Set up a backtracking function that takes and index in, a path, and a total sum.
- BASE CASE: If the total is ever over the target amount, or the index is over the length of the input array, return
- If the total equals the target, then append the current-path to result array.
- Run the backtracking function on the current value (since there is not limitations on repeats): backtrack(start, current-path, total + nums[start]), then pop last value in the current-path, and add the next value: backtrack(start +1, current-path, total)

13.3 Word Search

Question: Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

- **Main idea:** This is a backtracking question. The general idea loop through the grid, and find the starting character, once found, we recursively try all neighboring characters.
- Begin by setting up a set to keep track of the current path of indices (i,j) that represent the word. When we loop through the target word, we will also have an indices to index into that word. Let's call this: index.
- Begin with a nested for loop that will loop: call the backtracing function of every value.

- To set up the backtracking function, create an internal function that takes indices i, j, as well as the index into the target word: pass i, j, 0 as the start for recursive call.
- Inside the backtracking function, lets consider when to stop the recursive calling, and return True: If the current index into the word equals the length of the target word, this means that we have reached the word successfully and we can return true.
- Next check to see if i or j are out of bounds (i¡0 or i¿= n...) If these conditions are True, return False from this path
- Next check to see if the current value in the grid at [i][j] is doesn't equal the index of the word: return False from path if its not:
- Next check to see if the tuple vale (i,j) is already in the path set: if it is, return False from this path (we cannot reuse letters)
- If all of these are True, then we can assume that the current letter is correct: add (i,j) to the path set.
- Next check if any of the directions are return True: so call: res = (dfs(i + 1, j, index + 1) or dfs(i - 1, j, index + 1) or dfs(i, j + 1, index + 1) or dfs(i, j - 1, index + 1))
- Before returning what any of these equal, pop the current (i,j) tuple from the path (use set remove function)
- Finally, return res: Back in the main nested for loop, if the backtrack function every returns true, then return true: if you finish looping through all the variables and nothing is true, return false

14 Greedy

14.1 Jump Game

Question: You are given an integer array nums where each element nums[i] indicates your maximum jump length at that position.

Return true if you can reach the last index starting from index 0, or false otherwise.

- **Main idea:** This question is greedy: the trick is to **start at the end of the array and work toward the beginning of the array**. If you work backwards, and you know that the previous element can reach the next, then you set the previous to the new end. Keep repeating this until your "end point" becomes the beginning of the array. If this is the case, then return true
- You can use two pointers, left and right. Start right pointer at the end of the array, and the left pointer at one less than right.
- Keep and ongoing variable that tracks the distance between the left and right nodes. (This value starts at 1)

- If the value at the left pointer is ever greater than the distance between the current left and right, then set the right to the current left, and reset the distance between back to 0.
- If not, then decrement the left variable, and add to the distance between.
- If left index becomes zero, and the current right index doesn't also equal zero, then the last element was not reachable from the first element: if it is zero, then return true

14.2 Gas Station

Question:

There are n gas stations along a circular route. You are given two integer arrays gas and cost where:

- gas[i] is the amount of gas at the ith station.
- cost[i] is the amount of gas needed to travel from the ith station to the (i + 1)th station. (The last station is connected to the first station)

You have a car that can store an unlimited amount of gas, but you begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index such that you can travel around the circuit once in the clockwise direction. If it's impossible, then return -1.

It's guaranteed that at most one solution exists.

- **Main idea:** Take the differences between the gas station positive amount, and the cost. Begin a tank counter variable that keeps track of the amount of gas in the tank. For each station, take the difference between the fill up amount, and the cost, and add it to the ongoing tank counter amount. Store this index that you started at as the first "potential solution index"
- If the tank counter amount is ever negative, reset the "potential solution index" to be the next index, and also set the tank amount back to zero
- Keep looping through the gas station array repeating this process: at the end of the loop, the index where the tank counter never went negative is the solution index.
- *Another thing to check first is if the summation of the gas array is greater than or equal to the summation of the cost array: if not, then return -1 as the solution is not possible*

14.3 Hand of Straights

Question: You are given an integer array hand where hand[i] is the value written on the ith card and an integer groupSize.

You want to rearrange the cards into groups so that each group is of size groupSize, and card values are consecutively increasing by 1.

Return true if it's possible to rearrange the cards in this way, otherwise, return false.

- **Main idea:** At any point in time, the smallest card will always have to be the start of group.
- Begin by storing all card values and frequencies into a hash map. Key is the card val, and the value is the frequency of the card. Once stored, start by popping the minimum key (smallest card val).
- Now that you have the smallest card, begin checking if the next subsequent card is in the hashmap: if it is, decrement that value, and continue incrementing up, until you have filled a group size.
- If at any point, you check to see if a value is in your hashmap to form a group, and it is not, return false.
- After you have formed a group, start the process over, and pop the minimum from the hash-map, then repeat the process.
- If the hashmap is empty, and all groups where made, then return true
- You can also do a preliminary check to see if the number of cards is divisible the group size. If it's not, go ahead and return false

14.4 Maximum SubArray

Question: Given an array of integers nums, find the subarray with the largest sum and return the sum.

A subarray is a contiguous non-empty sequence of elements within an array.

- **Main idea:** Begin with two values. A max-value that indicates the maximum subarray, and a current sum
- Begin looping through the array: While looping, first make a check to see if the current sum is negative:
- Whenever there is a negative sum (NEGATIVE PREFIX), reset the current sum back to 0. After this, set the max-value to the max of the current sum or whatever the max value is.
- Return the max value

15 Math & Geometry

15.1 Plus One

Question: You are given an integer array digits, where each digits[i] is the ith digit of a large integer. It is ordered from most significant to least significant digit, and it will not contain any leading zero.

Return the digits of the given integer after incrementing it by one.

- **Main idea:** Keep a carry bit that will be either 0 or 1 throughout the question.
- Begin by setting the carry bit to 1
- Loop through the input array backwards. If carry is ever 1, then add the current value at i and 1 together.
- If the sum is greater than 9, then make the current position 0, and set the carry to 1, continue looping through
- If the sum is less than or equal to 9, then simply make the result the current digit, and set the carry to 0.
- After looping through all the values, if the carry bit == 1, then add [1] to the left side of the array (beginning)

15.2 Rotate Image

Question: Given a square n x n matrix of integers matrix, rotate it by 90 degrees clockwise.

You must rotate the matrix in-place. Do not allocate another 2D matrix and do the rotation.

- **Main idea:** This question can broken into two pieces: First Transpose, then reverse along the middle y axis
- Begin with the transpose: You want to swap along the diagonal of the matrix: to do this, do nested for loop, with i looping through all values in length n: For j, we can leverage the fact that it's a square matrix: Start j at i+1. This will always start the j at the index of the diagonal in that particular row.
- At each value in the matrix[i][j], swap with matrix[j][i].
- Next, We need to reverse the array. To do this, set up a for loop that loops over all rows, but only up to column n//2.
- At each point, swap matrix[i][j] and matrix[i][n- j - 1]: this will swap at the corresponding column on the over side of the matrix

16 Tries

16.1 Implement Trie Prefix Tree

Question: A prefix tree (also known as a trie) is a tree data structure used to efficiently store and retrieve keys in a set of strings. Some applications of this data structure include auto-complete and spell checker systems.

- **Main idea:** First create a node class that represents a node in the tree: each node will have a hashset that can store children Nodes as values, and the current letter as a key. Each Node class also has member class variable called END, which indicates that this node is the end of particular word.
- For the Trie Prefix class, create a member variable called root, and create a Node for it
- For the insert function, the idea is that we'll loop through each level of the tree, and check if the current letter already exists in the tree as a child Node. Begin looping by setting a current var equal to root node.
- If at at any point, the letter does not exists as a child, add it to the children list. Once done looping through the word, make sure you set the last Node's end member equal to True
- For the search function, do the same by setting a current equal to the root to start, and begin traversing the tree. If at any point, the current letter in the word doesn't exist in the children set, return False. Check at the end if the last Node's end member var is equal to True, if so, return True
- startsWith is the same as search, but you don't need to check the end member var when completing the loop through

17 Intervals

17.1 Insert Intervals

Question: You are given an array of non-overlapping intervals intervals where intervals[i] = [start-i, end-i] represents the start and the end time of the ith interval. intervals is initially sorted in ascending order by start-i.

You are given another interval newInterval = [start, end].

Insert newInterval into intervals such that intervals is still sorted in ascending order by start-i and also intervals still does not have any overlapping intervals. You may merge the overlapping intervals if needed.

- **Main idea:** This question involves merging intervals, and tracking when it's safe to append either the newly inserted interval, or and interval from the array
- Begin by looping through all the intervals
- If the new interval's end is less than the current intervals start: then it is safe to append the New Interval to the output array: and we can then add all intervals from i onwards to the output array, and RETURN
- The second test (elif) is if the current intervals, end is less than the new intervals start. This means that it is safe to append the Current Interval
- If neither of these cases are true, then we are OVERLAPPING: and we need set out newInterval to the MIN of the new interval's start, and the current interval's

start (for the start point), and the MAX of the newInterval's end and the current interval's end for the end point:

- If we loop through the whole array, and haven't returned yet, then append the new Interval (because it hasn't been appended yet), then return the output array

17.2 Meeting Rooms

Question: Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

- **Main idea:** Sort the interval arrays by the starting time
- Loop through all intervals (starting at the second): and check of the current start time is any time before the previous end time: If it is, return false, if it isn't, return True

17.3 Meeting Rooms II

Question: Given an array of meeting time intervals intervals where intervals[i] = [start-i, end-i], return the minimum number of conference rooms required.

- **Main idea:** Add all start times to a start array and all end times to an end array: Sort both arrays
- Initialize two pointers, one at the beginning of the start array, and one at the beginning of the end array
- Begin looping through each array: If the value at the current index of the start array is less than the value at the current index of the end array, this indicates that there is overlap of meetings: Increment the conference room count AND increment the start array pointer
- If the end value is less than OR EQUAL TO the start value, this means that the meeting starts after, so we can decrement the conference room count, and increment the end pointer
- Continue looping so long as start and end pointers are in bounds of their arrays.
- While looping, keep track of the overall max value that the conference room val gets to.

17.4 Merge Intervals

Question: Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

- **Main idea:** Start by sorting the intervals, then adding the first to an output array

- Loop through the rest of the intervals starting from index 1: check to see if the last element of the output array overlaps with the current element (can use -1 as index)
- If it does, then update the last element to the min(last-start, current-start) and the max(last-end, current-end)
- After done looping, return the array

17.5 Non-overlapping Intervals

Question: Given an array of intervals intervals where intervals[i] = [start-i, end-i], return the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Note: Intervals are non-overlapping even if they have a common point. For example, [1, 3] and [2, 4] are overlapping, but [1, 2] and [2, 3] are non-overlapping.

- **Main idea:** Begin by sorting the intervals. After sorting create an output array, and put the first interval into that array
- Begin looping through the rest of the intervals, starting at index 1
- If there is overlap between the current interval, and the last interval in the array: replace the last interval in the array with whatever interval END EARLIER: GREEDY CHOICE
- At the end, compare the output array with the original array: the difference in length is the answer!