Passed

Not passed

Not completed yet

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Difficulty | Question Title | Solving Strategy | Note | Runtime | Memory |
| Better than this % of online submissions. Default to using Java and comparing against all Java submissions unless otherwise specified. | |
| 1 | Easy | Two Sum | Use TreeMap to store value to list of Indexes (reverse Key order) | Not an evident Map question.  Correctly identified corner case (3 + 3 = 6 where we have two 3 and one 3). | 44.26% | 16.26% |
| 7 | Medium | Reverse Integer | Same strategy to Q8M | Did not pay attention on how the program should handle Integer overflow case (it needs to return 0 rather than Integer.MIN\_VALUE or Integer.MAX\_VALUE) | 99.91% | 5.25% |
| 8 | Medium | String to Integer (atoi) | Use Queue (FIFO usage) to store digits, and then form number. | Address Integer overflow without using long is the most tricky one and I’ve struggled on. Although I can manipulate the inequality equation by dividing Integer.MAX\_VALUE, I did not come up with a correct solution until viewing [this video](https://www.bilibili.com/video/BV1kV411i7kM).  Also, did not understand correctly (it says that except the leading space and -/+ as the first non-space character in the input String, every number after a non-number should be ignored) | 80.61% | 5.15% |
| 9 | Easy | Palindrome Number | Use Deque (double-ended Queue) to compare num.removeFirst() is equalt to num.removeLast() |  | 68.16% | 22.22% |
| 11 | Medium | Container With Most Water | Use two pointers to narrow the pool’s width, moving the pointer (that has a relatively shorter height) inwards. | Previously also used a more brute-force style of correct solution (iterating left pointer from 0 to height.length – 1 and iterate right pointer from height.length – 1 to left + 1), with pruning. Discovered optimization while watching [this video](https://www.bilibili.com/video/BV1Ef4y1z75r). | 12.93% | 5.03% |
| Area math calculation property here: if we move a shorter side inwards, there’s no way to attain a larger area regardless how tall that bars inwards (of the current longer side) has.  Test cases that makes me believe this thought [1,9,6,2,5,4,8,3,7] and [1,9,6,2,5,4,8,**11**,7] | |
| 15 | Medium | 3Sum | Use two pointers on a sorted array (after locking a variable called a) | The efficient algorithm still need O(n^2) time, can’t improve further. As a baseline benchmark (CSE 417 is useful here), a brute-force solution needs O(n^3) time.  Previously used an approach similar to Q1E by using HashMap, but the code is too long and still no progress. Hence watch this solution video that uses 2-pointer approach. | 57.12% | 27.72% |
| Reason that 2-Pointer approach on a sorted array would work: when I intentionally sort the original array, if three number in different indices (with respect to the processed array) add up to zero, it will fulfill the i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0 requirement (with respect to original array). | |
| 20 | Easy | Valid Parentheses | Use Stack to store left parentheses |  | 36.28% | 26.99% |
| 21 | Easy | Merge Two Sorted Lists | Iteration.  Use two pointers to track which elements we pick next from original list. | Will greatly speed up operation if we run out of one LinkedList by directly link result’s next empty node to the unprocessed original node. | ~~100.00%~~ | 70.56 % |
| 24 | Medium | Swap Nodes in Pairs | Iteration.  Manipulate pointers to achieve goal. |  | ~~100.00%~~ | 72.21 % |
| 26 | Easy | Remove Duplicates from Sorted Array | Implement the remove() method on Java’s ArrayList (that uses array as its underlying data storage).  Use a size variable to describe the size of resultant array. | Some language does not have array and only has List ADT (and programmers use List ADT like an array), but I take this challenge. | 10.10% | 10.60% |
| 33 | Medium | Search in Rotated Sorted Array | Used similar approach from Q153M (binary search) | Did realize that I need to use 4 conditions to test whether go to left half and right half based on the lowVal and halfVal, but [this video hints](https://www.bilibili.com/video/BV1154y187P5) that I also need highVal to facilitate which half should go (it still has 4 conditions though) | ~~100%~~ | 15.28% |
| 48 | Medium | Rotate Image | Use Queue to add the new number we’re replacing (Last-In-First-Out data flow). Use recursion to continuously rotate inner and inner squares until we hit the square with length of 0 or 1 (in this base case, no change is needed). | The input matrix is guaranteed to be a square makes this problem a bit easier, as I can assume that we always start at (x,x) position from top-left.  It is also possible to use recursion because the side’s length of inner square is always 2 less than the outer square. | 8.52% | 13.76% |
| 53 | Easy | Maximum Subarray | Dynamic programming that stores max sum so far (by taking [this approach](https://www.bilibili.com/video/BV1Ez4y1S7WP))  Can’t use brute force (exceed time limit) | Note: there’s no way in Java to easily convert Array to List, you need to manually add all of them. | 34.33% | 99.74% |
| 54 | Medium | Spiral Matrix | Maintain a HashSet<Coord> that stores visited vertex.  Perform DFS search on the entire matrix via a pre-defined way. | There are a lot of edge cases (8 cases) that I need to consider in my solution. A more simpler solution with fewer edge cases does exist, but I choose to continue my own approach if it passes.  If I use HashSet of a custom data structure, I need to override both equals and hashCode , otherwise the Set.contains() wo’t work | ~~100%~~ | 18.45% |
| 55 | Medium | Jump Game | A typical DP problem, and state transition is obvious.  Use a boolean[] to describe whether one can jump to this location. |  | 11.38% | 6.10% |
| 56 | Medium | Merge Intervals | Same strategy as Q57M |  | 42.96% | 5.36% |
| 57 | Medium | Insert Interval | First insert the new interval into the correct location in the original list based on its start time, and then perform interval merging steps. | You CAN use int[] as the type in Java’s List, Queue, Map ADT.  primitives does not have a.compareTo(b) method. public int compareTo(int[] a, int[] b) {  return **a[0] - b[0]**;  }  will sort in ascending order. | 8.56% | 5.27% |
| 62 | Medium | Unique Paths | Use same strategy as Q63 | Much easier than Q63 (I solved Q63 before solving Q62) | ~~100%~~ | 15.82% |
| 63 | Medium | Unique Paths 2 | Use similar strategy of CSE 417 Baby Yoda assignment’s solution strategy. | Previously used DFS from start node and it will fail in edge cases.  [This solution video](https://www.bilibili.com/video/BV1tp4y1S7FW) reminds me the CSE 417 Baby Yoda and proceed from there. | ~~100%~~ | 46.08% |
| 70 | Easy | Climbing Stairs | Use int[i]=j that stores number of ways to climb stairs up to the i-th position. | Initially think how to break down n to ones and twos, but according to [this video](https://www.bilibili.com/video/BV1SJ411n7bx), I should start from 1 all the way up to n. | ~~100%~~ | 18.14% |
| 71 | Medium | Simplify Path | Use Stack to store folder hierarchy | We're always removing the most inner directory if we encounter .. , no other ADT operation, so use Stack | 56.41% | 5.71% |
| 73 | Medium | Set Matrix Zeros | Use two HashSet to store rows and cols that all elements within needs to be set to 0. Iterate the matrix board to add rows and cols, and iterate the board again to set elements to 0. | My solution uses O(2\*mn) time and O(m+n) space, there’s a better solution do exist that uses O(1) space.  Using something Hash is a very useful hint from related topics. | 39.15% | 6.21% |
| 79 | Medium | Word Search | Use same approach of Q200M (perform DFS if we found a matching letter)  My original solution uses the right approach, except I can’t correctly un-choose a word by setting visited[r][c] back to false if there’s no match after traversing to 3 unvisited neighbors. [This video](https://www.bilibili.com/video/BV1d4411W7ZR) re-organized base cases in dfs function to correct the un-choose behavior (we need to un-choose on the recursion backtracking stage). | Q200M is a bit easier than Q79M, because Q200M don’t need to un-choose a land whereas Q79M need to un-choose a letter if word is not found. | 79.13% | 32.43% |
| 85 | Medium | Remove Duplicates from Sorted Array II | Use same approach as Q26E. | Need to use debugger tracing tool to correctly find my mistake in original solution. Otherwise I am not able to implement this program correctly. | 17.57% | 7.07% |
| 91 | Medium | Decode Ways | The DP solution is adapted [from this video](https://www.bilibili.com/video/BV1w7411i7PH), using dp[i]=j to describe there are j number of ways to decode the string from 0 to i-1 . | Previously uses similar approach of Q79M (with a little reference from Q91M), it is correct exceeds time limit when s has more digits than around 40 digits.  It’s like CSE 417’s B1rthday Sauerkraut where the state transition function is not obvious/evident.  A problem that can be solved by recursion may able to be solved by dynamic programming. | 95.27% | 6.98% |
| 93 | Medium | Restore IP Addresses | Use knapsack’s choose-explore-unchoose recursion method. | Previously tried the choose-explore-unchoose recursion by using int[4] to store each address block.  Did not remember that I don’t need to un-choose if I use substring until [I saw this solution video](https://www.bilibili.com/video/BV1354y1U7dr), as String will always passed as copy in function instances. | 61.05% | 78.10% |
| 94 | Easy | Binary Tree Inorder Traversal | Continuing to push left node to Stack until we reached to the left-most node of the current tree/sub-tree. Then, because the pointer will not change while popping out Stack, we will print out the left, root, and right child (if we do have a right child). | Recursion solution is very easy, so I use iterative approach. | 35.69% | 5.06% |
| 96 | Medium | Unique Binary Search Trees |  | Did realize that I need to use bottom-up DP to solve this problem (it’s not a typical BST problem), like CSE 417’s B1irthday Sauerkraut problem’s bottom-up approach, but I do not know how to creatively use the property of BST to find how to transition states. |  |  |
| 98 | Medium | Validate Binary Search Tree | Use a valid BST has sorted in-order traversal sequence, test whether such sequence is un-sorted (borrow the monotonic stack concept) | Note: can also use the non-recursive approach of Binary Tree in-order traversal and add logics to check values being pushed to / popped out of stack to check BST validity. | 13.91% | 8.93% |
| 99 | Medium | Recovery Binary Search Tree | Use the property of in-order traversal on a valid BST (in-order traversal will result in a sorted increasing array) | As soon as I recalled the property of in-order traversal of a valid BST by [watching this video](https://www.bilibili.com/video/BV1rE411B7XB), I come up my code afterwards. | 18.46% | 90.22% |
| 100 | Easy | Same Tree | Base case: check whether tree 1 and tree 2’s node is null.  Recursive case: iterate call left children and right children. Use back tracking to determine whether two trees are same. | Comparing the Node object itself is not comparing the Node’s value | ~~100%~~ | 90.60% |
| 102 | Medium | Binary Tree Level Order Traversal | Use 2-Queue method to perform layered BFS traversal on a tree. |  | 80.48% | 11.03% |
| 103 | Medium | Binary Tree Zigzag Level Order Traversal | Use 2-Queue method to perform layered BFS traversal on a tree. |  | 86.16% | 20.55% |
| 104 | Easy | Maximum Depth of Binary Tree | Use 2-Queue implementation of layered BFS traversal. | Clue: Tree is a simplified case of Graph. | 27.68% | 24.41% |
| 113 | Medium | Path Sum II | Use LinkedList to store values, only copy the new list if the remaining sum is 0 and we’re at the leaf node. | There may be a simpler solution that do not need a lot of base cases. | 32.52% | 56.52% |
| 114 | Medium | Flatten Binary Tree to Linked List | Use Queue to store Pre-order traversal, and repeatedly poll Queue to flatten tree. | Should immediately return if the root is empty. | 32.46% | 57.29% |
| 120 | Medium | Triangle | Use the same data structure List<List<Integer>> to store the min sum so far. |  | 25.15% | 76.77% |
| 121 | Easy | Best Time to Buy and Sell Stock | Use similar DP approach to Q122M, except I need to modify it so I will always buy from $0 profit ([video reference](https://www.bilibili.com/video/BV1GZ4y1c7RR)) | Greedy Algorithm would also work, which is why this question is marked as Easy. | 5.27% | 40.32% |
| 122 | Medium | Best Time to Buy and Sell Stock II | Use DP to store max profit so far if I hold (or not hold) this stock on the i-th day | Felt it was easier than Q121E | 5.52% | 6.58% |
| 123 | Hard | Best Time to Buy and Sell Stock III | Use similar DP approach to Q122M, except I need to use a third dimension to store transaction number ([video reference](https://www.bilibili.com/video/BV1GZ4y1c7RR)).  This is similar to CSE 417 Baby Yoda problem that need to use 3-rd dimension to store number of forces used. | Need to consider several edge cases that this state can’t exist (such as 2 transactions on the first day (i=0) | 13.46% | 43.88% |
| 133 | Medium | Clone Graph | Use BFS to copy all Node from the original Graph, and then use BFS to connect all Node from the cloned Graph together. |  | 51.57% | 7.23% |
| 138 | Medium | Copy List with Random Pointer | Use Two-Map method to store the 1-1 relationship (cue) between relative index and the Node object itself | Tricky to manipulate pointers and store the object in Two-Map when deep copying Nodes (before fill in their random pointer) | 31.90% | 45.73% |
| 141 | Easy | Linked List Cycle | Use HashSet to determine whether we’ve encountered a previous node (this means we have a cycle) |  | 8.54% | 5.12% |
| 143 | Medium | Reorder List | Use Dequeue to store nodes (hint: repeatedly use first and last of the “remaining” elements), and pop Stacks to return reversed Linked List.  Mistake: should not use Stack nor a regular Queue. | Forgot that size, isEmpty of Stack and Queue is a function call not a property. | 25.10% | 27.14% |
| 144 | Easy | Binary Tree Preorder Traversal | Add root to stack, and while stack is not empty, pop root, add right if not null, and add left if not null. | Recursion solution is very easy, so I use iterative approach. | 37.24% | 10.28% |
| 145 | Easy | Binary Tree Postorder Traversal | [The solution](https://blog.csdn.net/sgbfblog/article/details/7773103) that I will "never" forget, traverse the order using modified pre-order: root, right, left; then reverse the traverse order (which yields left, right, root). | Recursion solution is very easy, so I use iterative approach. | 38.24% | 13.90% |
| 146 | Medium | LRU Cache | Use HashMap and doubly linked list to achieve O(1) case. Nodes flows like a Queue if I just need insertion.  [This video](https://www.bilibili.com/video/BV1ZJ411g7ay) hints me to write dummy root and end nodes, in which root and end pointer points to, reducing the need to write many edge cases. | This is a manual implementation of Java’s LinkedHashSet ADT.  Previously did use HashMap and doubly linked-list, but did not assign a dummy node for root and end pointers (which forces me to write plenty of edge cases if root/end is empty, we only have 1 or 2 element(s) stored in the cache, capacity is 1 or 2, etc.).  It’s easier for me to understand solution by re-factoring commonly-used lines of codes (which adds semantic meaning on code). | 83.27% | 74.66% |
| 152 | Medium | Maximum Product Subarray | Used similar DP approach to Q53E including state changer function ([this video shows](https://www.bilibili.com/video/BV1kp4y1b74r) I need to maintain a min value so far and flip min and max when we encounter negative number, which I think is very clever). | Previously used 2-D array DP approach and exceeds memory limit.  The challenging part is recognizing that multiplying 2 negative number yields a positive number, so we can’t copy Q53E DP approach. | 95.92% | 17.25% |
| 153 | Medium | Find Minimum in Rotated Sorted Array | Question asks to implement algorithm that runs in O(log•n) time, repeatedly divide in halves.  Given array is sorted in custom order, use a variant of binary search. | Trust my instinct when program the binary search function with modification. | ~~100%~~ | 9.37% |
| 188 | Hard | Best Time to Buy and Sell Stock IV | Use identical approach to Q123H |  | 96.96% | 78.39% |
| 198 | Medium | House Robber | An evident and simple DP problem.  int[x][y] = amount of money I can earn so far on whether I robbed x house (y=1) or not robbed this house (y=0) |  | ~~100.00%~~ | 46.71% |
| 199 | Medium | Binary Tree Right Side View | Use two Queue method to find each layer in a tree.  Store the right-most side value while we proceed to the next layer. | I previously use a Map to store the right-most value with a layer number, but it’s inefficient. | 75.66% | 42.14% |
| 200 | Medium | Number of Islands | Same as Q695 | This question uses char[ ][ ] while Q695 uses int[ ][ ], so if I need to compare char[ ][ ] against a value (not the ASCII code), need to use char[r][c] == ‘1’ (note single quote mark) | 24.54% | 12.67% |
| 202 | Easy | Happy Number | Use HashMap-like mindset to store the square between 0 and 9, and use HashSet to determine whether it’s looping infinitely. | [This video](https://www.bilibili.com/video/BV1CA41187LQ) hints me on using HashSet to determine whether we’re looping. | 37.96% | 38.48% |
| 203 | Easy | Remove Linked List Elements | Use pointer fields to skip deleted value in original LinkedList.  Program will skip everything if the List only contains the value to be removed. | Need to use .next.next (jump one further) to achieve correct result.  Low memory usage compared to other Java implementations. | 74.26% | 98.24% |
| 205 | Easy | Isomorphic Strings | Use forward and reverse HashMap<Character, Character> . |  | 47.19% | 87.24% |
| 206 | Easy | Reverse Linked List | Use Stack to store nodes (hint: reverse the order), and pop Stacks to return reversed Linked List. | This solution does not create new ListNode, just through manipulating nodes of existing pointers. | 7.35% | 86.68% |
| 207 | Medium | Course Schedule | Same as Q210M | Succeed in Q210M before transferring the Topo Sort knowledge to Q207.  Can not skip a for-loop that creates all node in graph (I’m not adding neighbors in this step) even for node that has no neighbor. | 47.81% | 10.57% |
| 208 | Medium | Implement Trie (Prefix Tree) | Use this [video’s trie tree design](https://www.bilibili.com/video/BV1uh411o7Si) (it’s so clever that it adapts LinkedList traversal concept, which makes implementation easier). Coded independently. | Previously, I design another form of Tenerary Search tree (all letters less than current goes left, all letters greater than current goes right, go down means select this word), and this implementation is more challenging. | 60.25% | 15.42% |
| 210 | Medium | Course Schedule II | Created a Directed Graph using Adjacency List and conduct topological sort on it to output a course schedule. | Solution is initially correct but non optimal (it passes the 1000 courses test but time out on the 2000 courses test) due to inefficient topo sort implementation (consistently find node with no incoming edges).  Used [this video](https://www.bilibili.com/video/BV1Ut411a74a) to create a more efficient Topological Sort algorithm using DFS and recursion. | 36.62% | 5.20% |
| 211 | Medium | Design Add and Search Words Data Structure | Use same approach as Q208M except I need to use recursion for search (as I need to traverse all non-null children when I encounter .) | Further optimization to increase in-practice runtime for this particular problem: we can call the non-. case first and then the . case (since the problem guarantees up to 3 . in all test cases). However, I failed to make such optimization. | 5.00% | 6.71% |
| 213 | Medium | House Robber II | Same approach as Q198, except I need to consider 3 cases as hinted by [this video](https://www.bilibili.com/video/BV1sK411u7ZY). | Adding a second for-loop does not change runtime complexity, as it will increase by constant factor not input size. | ~~100%~~ | 46.11% |
| 215 | Medium | Kth Largest Element in an Array | Use same approach as Q703E | Did not immediately realize that I should use regular PriorityQueue instead of PriorityQueue in reverse order. | 73.51% | 25.92% |
| 217 | Easy | Contains Duplicate | Find duplicate → test uniqueness→use Set |  | 67.21% | 35.60% |
| 221 | Medium | Maximum Square | Use CSE 417 Homework 5 Q1 approach. | Forgot the definition of return number in CSE 417 Q1 | 60.42% | 10.70% |
| 224 | Hard | Basic Calculator | RPN calculator | I do remember Reverse Polish Operation beforehand. |  |  |
| 225 | Easy | Implement Stack using Queues | Rotate Queue when remove or peek the “Stack” |  | ~~100%~~ | 6.08% |
| 226 | Easy | Invert Binary Tree | Perform DFS traversal on the original tree while creating a new tree using values of existing tree. | Originally did not find any ideas, then I saw this is a DFS/BFS problem as this question tag, and immediately come up with a DFS solution. | ~~100%~~ | 6.02% |
| 230 | Medium | Kth Smallest Element in a BST | Perform in-order traversal and only store the k-th element. | If a binary search tree is valid, its in-order traversal result is a monotonic increasing Stack. | ~~100%~~ | 63.92% |
| 231 | Easy | Power of Two | Convert to binary string, check whether it only has one leading 1 using regular expression. | Did not realize that all negative values should be false.  Need to import regex package that’s not part of standard LeetCode testing environment.  import java.util.regex.\*; | 11.70% | 5.05% |
| 232 | Easy | Implement Queue using Stacks | Use two stacks to rotate content back to original order. |  | ~~100%~~ | 5.16% |
| 234 | Easy | Palindrome Linked List | Use Dequeue (hint: repeatedly compare first and last elements) to store nodes, and repeatedly pull nodes off from both ends to see whether it’s palindrome. | O(N) runtime complexity O(N) space complexity. There’s an O(1) space complexity solution. | 34.72% | 29.47% |
| 235 | Easy | Lowest Common Ancestor of a Binary Search Tree | Perform DFS.  Use the BST property: values in left sub tree are guaranteed to be less than root, values in the right sub tree are guaranteed to be greater than root. |  | 66.37% | 17.73% |
| 238 | Medium | Product of Array Except Self | Use two HashMap to store products of all numbers before index and all numbers after index. |  | 5.73% | 5.21% |
| 242 | Easy | Valid Anagram | Use two char[] to store all letters in s and t , sort these two char[] and test array equality. | arr1 == arr2 tests whether two pointers references to the same array. Arrays.equals(arr1, arr2) tests whether two arrays has identical content in same order. | 50.36% | 8.25% |
| 252 | Easy | Meeting Rooms | Used Q57M’s PriorityQueue idea that sorts interval by start time, then by end time, with different behaviors while removing elements in PriorityQueue | The intervals.length = 0 edge case looks odd. | 65.41% | 11.39% |
| 253 | Medium | Meeding Rooms II | Use similar approach to Q252E, except we need to store the end time of all meeting rooms instead of 1 meeting room for Q252E. | Within the while loop, I can take advantage on greedy algorithm in several places. | 42.12% | 13.79% |
| 255 | Medium | Verify Preorder Sequence in Binary Search Tree | Use Stack and compare whether the right subtree is smaller than the parent.  Use [this approach](https://www.youtube.com/watch?v=Psce8aMuX8s), but independently write code | Should not use recursion as it won’t have access to the parent node. | 74.57% | 97.22% |
| 256 | Medium | Paint House | Use DP approach (similar to Q746E) |  | 67.97% | 50.00% |
| 258 | Easy | Add Digits | Use traditional while loop and stop if sum < 10 |  | 16.88% | 36.39% |
| 261 | Medium | Graph Valid Tree | Used Graph building code from Q323M  Use the tree property (valid tree has node-1 number of edges)  And use BFS to validate whether we have only 1 connected component. | Previously focus too much on validating whether the graph has only 1 connected component and no cycle (hence borrowed from Q323M and Q210M).  This video hints me to use the property of a valid tree data structure and starts from there. No need to write a separate cycle detection logic. | 60.25% | 62.88% |
| 265 | Medium | Paint House II | Use same approach to Q256M | Note: there’s no built-in function in Java that translates Array to a List ADT.  The CSE 417 Baby Yoda programming question’s approach provides a hint (my code will work for an arbitrarily number of available forces) | 96.30% | 86.83% |
| 268 | Easy | Missing Number | Use Set to store numbers, and iterate from 0 to nums.length to find missing number. | My approach uses O(n) time and space, a more efficient algorithm using O(n) time and O(1) space does exist (subtract the sum with and without the missing number)  Another approach (I can think of) is O(nlogn) time (sort the array) and O(1) space. | 26.20% | 65.08% |
| 274 | Medium | H-Index | Reverse sort the array, and then iterate through this array. | Perhaps the most challenging part is to translate H-Index’s definition into the DP-like definition (if the arr[i] is larger than i + 1, then the author’s H-index is guaranteed to be at least i + 1)  Hence I won’t be able to think this quick if I’m not familiar with DP. | 12.36% | 6.60% |
| 290 | Easy | Word Pattern | Use 1-1 relationship (Two Map) strategy. |  | 45.73% | 33.26% |
| 300 | Medium | Longest Increasing Subsequence | Based on [this approach](https://www.bilibili.com/video/BV19b4y1R7K3) without looking at code, my code writeup is my own work. | CSE 417 problem | 34.49% | 57.78% |
| 309 | Medium | Best Time to Buy and Sell Stock with Cooldown | Same approach of Q122M, be careful how to handle 1-day cooldown and buy stock at day 2 (i=1) | Cue: cooldown period is fixed not varied, which makes problem easier. | 32.77% | 36.82% |
| 317 | Hard | Shortest Distance from All Buildings | Used DP bottom-up approach, conduct layered-BFS for all houses. For each empty land, store the steps to get to all houses, and sum it up to get total travel distance. | Cue: BFS guarantees shortest distance since it explores 1 level at a time.  Need to conduct 2-D layered BFS (using 2 Queue method), so I need to create an auxiliary data structure. (currently, this step is necessary for C++ and Java, but not Python as Python has built-in 2-D tuple) | 28.57% | 33.01% |
| 323 | Medium | Number of Connected Components in an Undirected Graph | Use adjacency list to build a graph (cue: we can’t repeatedly traverse entire int[][] edges to find neighbors), and then conduct BFS on all component. | Need to handle a case where a component only has 1 node. | 15.60% | 42.41% |
| 324 | Medium | Wiggle Sort II |  | [This video](https://www.bilibili.com/video/BV1Bz4y117Fr) hints me that this question tests how well you understand Quick Sort, so I better need to master it first. |  |  |
| 332 | Medium | Coin Change | Use DP (how many amount can we make change by using n number of coins). Use an approach similar to 2 Queues for layered BFS search to efficiently retrieve amount that I can make change using n-1 coins. | CSE 417 Long Form written problem | 10.04% | 5.11% |
| 334 | Medium | Increasing Triplet Subsequence | Borrowed the DP-like approach (longest increasing subsequence and maximum subarray sum) from Q456M, storing smallest value and largest value so far. Then test whether num[i] > small[i] and num[i] < large[i] (between 1 and length-2, inclusive) which tests nums[i] **< nums[j] <** nums[k] | Felt too confused on programming the second for-loop that stores max value so far (confused on the start and end boundary), but ultimately solved it independently after 5 tries.  I felt that it uses a simple case of Monotonic Stack. | 41.18% | 61.28% |
| 337 | Medium | House Robber III | Use Q198 approach, and referenced from [this video](https://www.bilibili.com/video/BV1sK411u7ZY) regarding merging results from 2 trees | Use Q198 approach, faced challenges when combining results from left and right sub-tree.  Need to combine results when back-tracking, rather than before initiate recursion. | ~~100%~~ | 45.72% |
| 339 | Medium | Nested List Weight Sum | Same as Q364 | More straightforward than Q364. | ~~100.00%~~ | 74.89% |
| 347 | Medium | Top K Frequent Elements | Use Two Map to store Integer → Frequency and Frequency → Integer relationships. | By overriding TreeMap’s comparator method, we can iterate the Frequency → Integer Map in reverse order (of all Keys), so we can achieve top-K result. | 52.10% | 49.88% |
| 364 | Medium | Nested List Weight Sum II | First use recursion to find the depth, then use recursion to calculated weighted sum. |  | 11.29% | 22.58% |
| 366 | Medium | Find Leaves of Binary Tree | Perform DFS, increase layer only when we’re backtracking. | DFS operations can be improved by remove some auxiliary data structures. | 8.17% | 21.01% |
| 417 | Medium | Pacific Atlantic Water Flow | New after this [video](https://www.youtube.com/watch?v=krL3r7MY7Dc):  Still use bottom-up DP to determine whether we can go to the ocean, but use DFS in 4 directions to find inlands at which water can flow to ocean.  Original approach that does not pass all test cases:  Use DP concept (bottom-up) to fill out whether we can go to Pacific from 1 cell or go to Atlantic from 1 cell.  Cue to consider DP solution: if water can flow to Pacific at this cell, if the adjacent cell is higher/equal, then the adjacent cell can go to this case. | More edge cases to consider while scanning the entire 2-D array  Note: Inefficient DFS solution do exist (perform DFS on any cell; determine whether it can go to Pacific or Atlantic) | 82.55% | 65.31% |
| 451 | Medium | Sort Characters By Frequency | Use two Map (Character to Frequency) and (Frequency to list of Character) to construct result String. | A better solution using Heap do exist, although the two Map approach is my original approach. | 10.32% | 47.43% |
| 456 | Medium | 132 Pattern | Use DP-like concept to store minimum value so far before nums[i], use monotonic decreasing stack to find the max value and the largest value after this max (and smaller than max) (whether nums[j] > nums[k]). | Did not realize that I need to store the minimum number so far before nums[i] , which facilitates testing whether nums[k] > nums[i] , until I watch [this video](https://www.bilibili.com/video/BV1J64y1m7Uv?p=1). | 19.70% | 74.72% |
| 468 | Medium | Validate IP Address | Use Regex to test IPv4 and IPv6 pattern, use whether String to Integer to String to test for leading zero in IPv4 | This problem has more edge case than other problems prior to complete this.  The argument for str.split() is a regex not string literal.  . (period) in regex has special meaning, use this to represent .  .split("\\."); | 21.80% | 42.22% |
| 472 | Hard | Concatenated Words | Use recursion (DFS, [similar to this approach](https://www.bilibili.com/video/BV1QJ411W7Bn)) to repeatedly add words to the temp string.  Use pruning to stop recursing impossible solutions.  Then I found that I need to manipulate the concatenated string itself rather than brute force using words to form a string. | Pruning was the most difficult part (need to know you don’t need to dive further as no solution would found).  Constraint is 0 ≤ words[i].length ≤ 1000  Hence we need to check empty string (str.length == 0) case and skip it.  A [dynamic programming solution](https://www.bilibili.com/video/BV1gf4y1p7PF) is more optimal but I my original (inefficient) DFS is my own idea. |  |  |
| 473 | Medium | Matchsticks to Square | Use [this approach](https://www.bilibili.com/video/BV1D5411j7UE), with my modified code | Did not initially realize that pre-calculate the length of square side (total matchstick length ÷ 4) is the starting point to solve this problem. Then this problem is a regular choose-test-unchoose recursion problem. | 42.16% | 96.11% |
| 476 | Easy | Number Complement | Flip the binary representation of original string (no leading 0) and then zero out all 1 that’s precede the complement. | Initially know that I need to zero out all 1 that precede the complement (I don’t know how to use bit operation to do this trick) but [this video](https://www.bilibili.com/video/BV1PW411y7mv) provides a bit operation approach. | 5.13% | 5.09% |
| 496 | Easy | Next Greater Element I | Create a Monotonic increasing stack to store increasing number from original array, along with a HashMap storing all element in nums2 and its next greater element. | Did know to use Monotonic increasing Stack to solve this problem (because we always quest for a greater element than self), but did not know how to add values there nor how to query Stack’s value to determine the answer. | 99.02% | 10.76% |
| 503 | Medium | Next Greater Element II | Used a very similar approach as Q496E. | Pay too little attention on the question’s prompt that asking finding an element strictly greater than current element (hence a=a should return -1).  Otherwise, this question is easier than I thought, if I thoroughly understand how to deal with Q496E | 16.38% | 8.69% |
| 509 | Easy | Fibonacci Number | Bottom-up approach, as it only needs two number before the n for each computation. | CSE 417 problem | ~~100%~~ | 82.92% |
| 513 | Medium | Find Bottom Left Tree Value | Use two Queue method to find layers, and store the first value in the layer right above the inner while loop. |  | 10.48% | 6.76% |
| 515 | Medium | Find Largest Value in Each Tree Row | Double Queue method from lecture. |  | 10.84% | 13.31% |
| 518 | Medium | Coin Change 2 | Used the same DP approach of 332, but the state transfer function is referred from [this video link](https://www.bilibili.com/video/BV1kX4y1P7M3). |  | 71.43% | 59.21% |
| 572 | Easy | Subtree of Another Tree | Use the exact same tree method from Q100E, then recursively compare whether the original tree’s root, root.left, and root.right child is identical to the subtree. | My approach and recursion process is correct except how I should return the Boolean value from DFS, and I correct the syntax [using this video](https://www.bilibili.com/video/BV1HW411h7r8). | 96.81% | 11.08% |
| 673 | Medium | Number of Longest Increasing Subsequence | Similar to Q300, and I’ve wrote the CSE 417 code to extract the sequence itself | Partially correct, it does not pass all test cases.  Did not realize that the number of unique increasing subsequence itself should also be stored as DP state. |  |  |
| 674 | Easy | Longest Continuous Increasing Subsequence | Use similar approach to Q53 | Need to assign return value to 1 at first instead of Integer.MIN\_VALUE, otherwise it will fail in case when array contains 1 number only. | 99.06% | 65.46% |
| 678 | Medium | Valid Parenthesis String | Use Q20E stack approach to count ( and ) pairs | [Did not consider](https://www.bilibili.com/video/BV1h34y1Q7oG?) when Stack only has ( and \*, cases such as (\*\*( should return false even though number of \* and ( matches.  Also, did not consider prioritize remove the right-most ( before consider removing \* from stack when we encounter ) | 45.60% | 53.88% |
| 690 | Medium | Employee Importance | Use two HashMap to store every employee’s importance and subordinates, where two Map’s Key is employee’s ID.  Cue for using HashMap: employee ID are guaranteed to be unique but values are not unique. | Forgot that Map remove value by providing a Key, as I’m thinking whether there’s other Map’s remove methods. | 8.39% | 6.92% |
| 695 | Medium | Max Area of Island | Use regular 2-D array iteration from top left to bottom right (row by row). When we encounter 1, recursively scan top down right left (order matters!) until we run out of recursion. | Two edge cases:  Need to use visited[][] to track whether array has been visited.  Need to account when grid has no island or the largest island is ⅃ shaped | 31.36% | 72.81% |
| 703 | Easy | Kth Largest Element in a Stream | Use PriorityQueue that stores only number that’s at or larger than k-th largest elements.  Did not even think about TreeSet since we allow duplicate. | Did realized to use PriorityQueue (but it does not have get(i) method that returns the i-th element in PriorityQueue), but did not realize that I only need to store numbers that’re at or larger than k-th largest element until I watch [this video](https://www.bilibili.com/video/BV1q4411y7tu). | 99.88% | 55.65% |
| 714 | Medium | Best Time to Buy and Sell Stock with Transaction Fee | Same approach to Q122M with only a little modification at selling stock. |  | 19.66% | 13.98% |
| 739 | Medium | Daily Temperatures | Use monotonic stack to store increasing temperatures, record the number of days you have to wait while popping out the stack to maintain monotonic property. | Attempted to use Q496E approach and add count variable that records how many times we pop the stack at first, it fails at edge case. Afterwards, [this video](https://www.bilibili.com/video/BV1xf4y1Y7Co) explains that the number of days you have to wait can be calculated during the process of popping out the stack. | 17.72% | 6.17% |
| 746 | Easy | Min Cost Climbing Stairs | Use DP approach to fill out whether I spend money or not to climb i-th stair. | Since it’s an Easy question, a more intuitive solution might exist if I don’t know DP beforehand. | 86.76% | 79.56% |
| 760 | Easy | Find Anagram Mappings | Create a value to index HashMap for nums2. |  | 91.80% | 15.12% |
| 856 | Medium | Score of Parenthesis | Use DFS approach to combine return value of each sub-problem, which can easily fulfill A+B and 2\*A case.  This is not a regular parenthesis problem that employs Stack and no recursion, I immediately realized the difference by looking at how we count score of layered parenthesis pairs (A+B and 2\*A) | Initial solution causes inner layer continue to take care of parenthesis of outer layer on the right; hence need to follow [this DFS solution](https://leetcode.com/problems/score-of-parentheses/submissions/) that employs left and right boundary. | ~~100%~~ | 55.24% |
| 993 | Easy | Cousins in Binary Tree | Use 2-Queue method layered BFS traversal to find whether children with value of x and y has a same parent value. | Note: take advantage of this constraint (1 <= Node.val <= 100), and use int[2] instead of Map ADT if the number of element is fixed, which saves time and memory.  Array re-assignment by pre-filling value needs to be written like:  arr = new int[] {-1, -2, -3, -4}; | ~~100%~~ | 71.09% |
| 1021 | Easy | Remove Outermost Parentheses | Use same method of Q20E, except using an int rather than Stack to store number of left parenthesis so far |  | 5.32% | 5.49% |
| 1046 | Easy | Last Stone Weight | Use reverse Priority Queue | When you need to override the Comparator of this PriorityQueue and this PriorityQueue stores primitive-typed values, you need to use the wrapper class in the Comparator.  new Comparator<Integer>() {  public int compare(**Integer** a, **Integer** b) {  return -a.compareTo(b);  }  }); | 97.53% | 5.28% |
| 1108 | Easy | Defanging an IP Address | Use str.replaceAll() one-liner. | The first argument (find) for str.replaceAll() is a regex not string literal, but the second argument (replacement) is a string literal.  . (period) in regex has special meaning, use this to represent .  .split("\\."); | 25.09% | 63.97% |
| 1583 | Medium | Count Unhappy Friends |  | Similar to CSE 417 Homework 1 (Stable Matching problem using Gale Shapley algorithm) |  |  |
| 1945 | Easy | Sum of Digits of String After Convert | (JavaScript)  Use Array.reduce and Array.map to efficiently iterate the entire array. | JavaScript’s array.foreach and similar functions (containing an anonymous function) is noticeably simplier than Java | 70.99% (JavaScript) | 93.13% (JavaScript) |