**Q1.** [**Contains Duplicate**](https://leetcode.com/problems/contains-duplicate/)**: Easy**

**Q2.** [**Valid Anagram**](https://leetcode.com/problems/valid-anagram/)**: Easy**

**Q3. 1** [**Two Sum**](https://leetcode.com/problems/two-sum/)**: Easy**

**Q4.** [**49. Group Anagrams**](https://leetcode.com/problems/group-anagrams)**: Medium**

Use a hashmap to solve this problem.

Trick: key of the hashmap will be a tuple of 26 integers(all letters in the strings are smallcase alphabets).

The value will be a list of all the strings that have the same alphabets in them.

**Q5.** [**347. Top K Frequent Elements**](https://leetcode.com/problems/top-k-frequent-elements)**: Medium**

3 solutions to this problem:

1. **Heap + hashmap:**

NOTE: came up with this solution. Tricky part was using python heaps

return heapq.nlargest(k, hashmap.keys(), key=hashmap.get) // key part still unclear

Complexity Analysis:

Time complexity : O(Nlogk) if k < Nk<N andO(N) in the particular case of N = kN=k. That ensures time complexity to be better than O(NlogN).

Space complexity :O(N+k) to store the hash map with not more N elements and a heap with k elements.

1. **Bucket sort using hashmaps:**

[Top K Frequent Elements - Bucket Sort - Leetcode 347 - Python](https://www.youtube.com/watch?v=YPTqKIgVk-k)

O(N) and O(N)

1. Using selection algorithm.:

Given in leetcode solution O(n) time. Will have to understand it.

**Q6.** [**238. Product of Array Except Self**](https://leetcode.com/problems/product-of-array-except-self)**: Medium**

For this problem, to have O(n) time and O(1) space complexity, we will run for loop 2x separately.

1. To calculate the product prefixes for a given number
2. To calculate the product postfixes of a given number

We will do this in the same output array to ensure constant memory usage.

NOTE: For this problem, using an output array is constant mem, according to problem def.

**Q7.** [**271. Encode and Decode Strings**](https://leetcode.com/problems/encode-and-decode-strings)**: Medium**

2 Approaches:

1. Take a chr whose ASCII value is more than 256.

While joining the single strings, add this chr in between. While decoding, split this string with this chr as a delimiter.

1. Use length of string + a special chr (#) as a delimiter:

[Encode and Decode Strings - Leetcode 271 - Python](https://www.youtube.com/watch?v=B1k_sxOSgv8)

**Q8.** [**125. Valid Palindrome**](https://leetcode.com/problems/valid-palindrome)**: Easy**

**Q9.** [**167. Two Sum II - Input Array Is Sorted**](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted)**: Medium**

Same as two sum 1 but constant space is possible only by using 2 pointers because hashing takes O(n) space.

Two pointers technique for a complexity of O(n) and space of O(1)

This is possible because the given array is sorted.

Keep shifting pointers L and R till their sum == target. Since, target is guaranteed to be present, this will work.

[TWO SUM II - Amazon Coding Interview Question - Leetcode 167 - Python](https://www.youtube.com/watch?v=cQ1Oz4ckceM)

**Q10.** [**15. 3Sum**](https://leetcode.com/problems/3sum) : **Medium**

Follow up on the Two Sum II problem. Solution would be of O(n^2).

Requires 2 loops, first loop variable is const. Second loop is the same as 2 sum,

L + R pointers approach.

Add all 3 numbers to check if sum is zero and keep appending the numbers to list.

Some intricacies should be clear by looking at the code.

[3Sum - Leetcode 15 - Python](https://www.youtube.com/watch?v=jzZsG8n2R9A)

**Q11 .** [**11. Container With Most Water**](https://leetcode.com/problems/container-with-most-water) **Medium**

[Container with Most Water - Leetcode 11 - Python](https://www.youtube.com/watch?v=UuiTKBwPgAo)

Inituition: The base of the container should be as wide as possible so start checking from farthest ends of the array. Hence, two pointer approach is ideal

At the farthest ends, take the min of the heights(values) in the and multiply with the base to give area.

Store this area if it is the max found till now.

Minimise R or Maximise L depending on the max of height between 2.

**Q12.** [**42. Trapping Rain Water**](https://leetcode.com/problems/trapping-rain-water) **Hard**

[Trapping Rain Water - Google Interview Question - Leetcode 42](https://www.youtube.com/watch?v=ZI2z5pq0TqA)

Tricky question. With two pointer approach, O(n) time and O(1) space solution.

For O(1) space solution, 2 pointers L + R. And 2 variables, maxLeft and maxRight.

L or R moves when whichever height L/R is the smaller one among the 2.

maxLeft/maxRight is updated respectively.

Res += min(maxLeft, maxRight) - height(i) ………..where i will be L or R depending on the if/else condition.

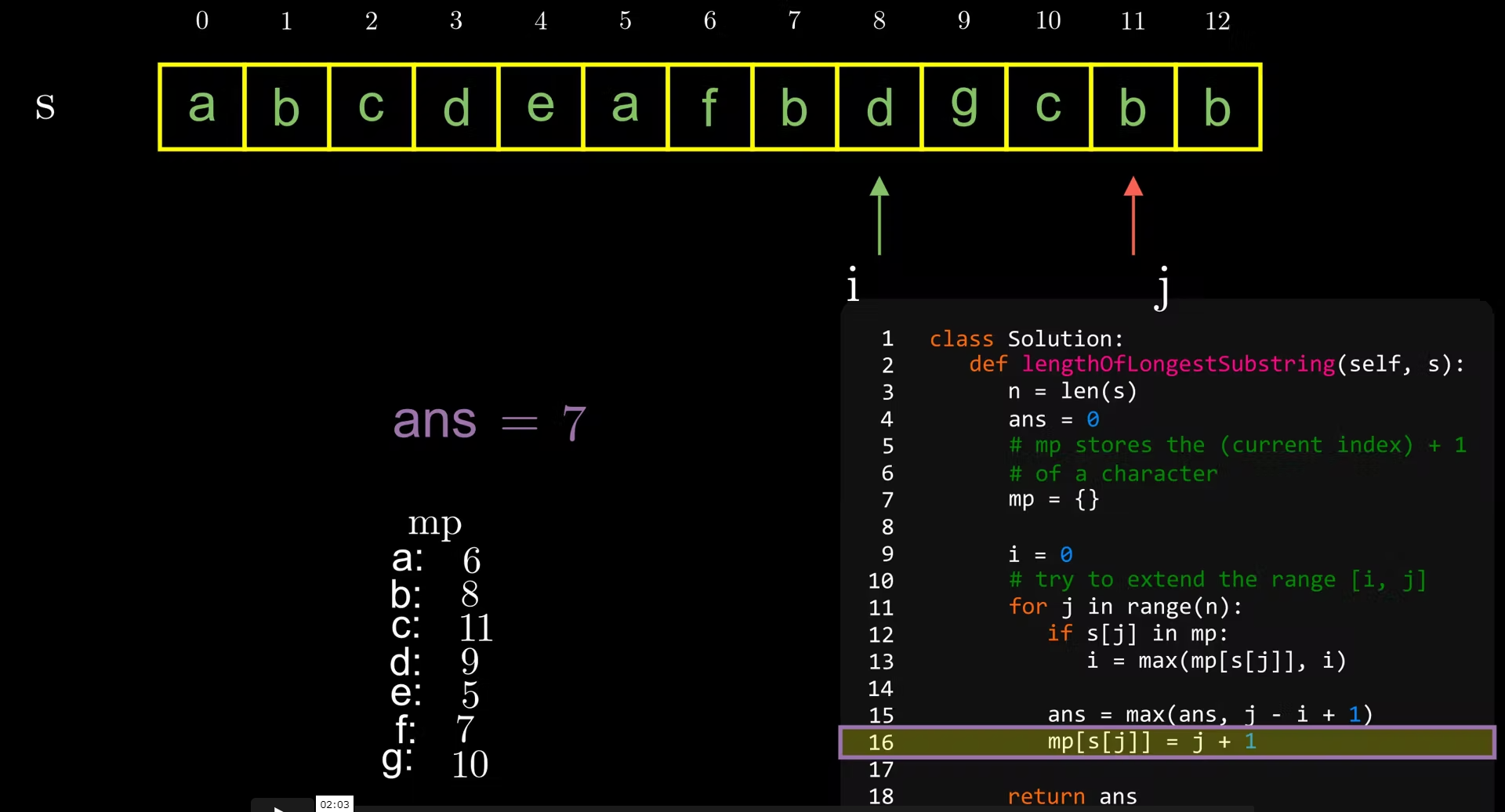
**Q13.** [**3. Longest Substring Without Repeating Characters**](https://leetcode.com/problems/longest-substring-without-repeating-characters)**: Medium**

NOTE: can use either hashmaps or sets to solve this problem. Choose a set because it is much easier to visualize.

Note: This is a tricky problem. Couldn't get the approach right. Understood, but not quite yet. Had to refer to the explanation. Video Explanation is great.

Given a string s, find the length of the longest substring without repeating characters.

Soln: Sliding window method using hashmaps:



1. Maintain 2 pointers, i and j for the left and right extremes of the window
2. J will iterate throughout the string and if the char at j is not in the hashmap (dictionary), we will add it to the dictionary as J +1, one value ahead of the repeated value (This is used to check the “active” index, i, of the substring) and update the returning output variable as j - i + 1
3. If the current char is present in the hashmap then the value is compared to the current value of the starting index, i.
4. If the value of i is greater then that means the value present in the hashmap is an old index for the current char.
5. If the value of the char in the hashmap is greater than the value of i then that means that this current char is already counted in the current substring. Hence we update the value of i to j, which means we move on to the next substring.

Note: By following all the above, we skip i to a position +1 from the repeating current char (j) which would have appeared in the substring anyway had we not skipped the initial index of the substring to position + 1. Position + 1 comes from the hashmap which stores the previous positions of the current char.

**Update: 3rd time solving this problem. Still couldn't get it. Solved it using the below solution.**

solution using sets: This is a better understanding solution.

[Longest Substring Without Repeating Characters - Leetcode 3 - Python](https://www.youtube.com/watch?v=wiGpQwVHdE0)

**Q14.** [**424. Longest Repeating Character Replacement**](https://leetcode.com/problems/longest-repeating-character-replacement)**: Medium**

A sliding window problem. Maintain 2 pointers, L + R, R will continuously move forward. L will only move forward if: **R - L + 1 - max(hashmap.values()) > k** (given in input).

As L moves forward, decrement the freq of hashmap[s[L-1]], since the window size has decreased, s[L-1] outside of the current window.

[Longest Repeating Character Replacement - Leetcode 424 - Python](https://www.youtube.com/watch?v=gqXU1UyA8pk)

O(26 \* N) time. Replacing max(hashmap.values()) with a single max\_freq variable, updated every cycle will make O(N) time.

**Q15.** [**76. Minimum Window Substring**](https://leetcode.com/problems/minimum-window-substring)**: Hard**

[Minimum Window Substring - Airbnb Interview Question - Leetcode 76](https://www.youtube.com/watch?v=jSto0O4AJbM)

Tricky question: refer above video to solve.

1. Maintain 2 hashmaps, for window and countT that will count the freq of chars. CountT will be const.
2. 2 variables, have and need. Need is len(countT), no. of unique requirements.

Have is to count the requirements in the current substrings.

1. 2 pointers, L + R. R moves with the for loop. Movement of L explained later.
2. Increment ‘have’ if window[current variable key] == countT[current variable key]
3. While have == need:

a. Update the result if the window size is smaller than the prev result.

b. Decrement count of window[current variable key] and check if it violates the window[current variable key] == countT[current variable key] condition. If yes, decrement have by 1.

c. increment L by 1, moving the window forward.

**Q16.** [**143. Reorder List**](https://leetcode.com/problems/reorder-list)**: Medium**

[Linkedin Interview Question - Reorder List - Leetcode 143 - Python](https://www.youtube.com/watch?v=S5bfdUTrKLM)

Combination of 3 questions:

1. Reversing a list.
2. Finding the middle element.
3. Merging 2 lists.
4. Find the middle element and split the list into first and second. Make the pointer on the last node of the first to point to None
5. Reverse the second list.
6. Merge the 2 lists alternately.

O(n) and O(1)

NOTE: Was able to solve this problem in 2nd attempt. Good Job

**Q17.** [**19. Remove Nth Node From End of List**](https://leetcode.com/problems/remove-nth-node-from-end-of-list)**: Medium**

2 pass approach:

Find out the length of the list. O(n)

Element that needs changing is length - n

Traverse using prehead till the above position and change the pointer.

NOTE: tricky part is to add a prehead to the given list which deals with certain edge cases.

1 pass approach:

Maintain 2 pointers, first and second, n+1 distance apart (n is given to be nth element from the end of the list)

Increment till first points at null. At this pointer second points at the element before the desired element.

[Remove Nth Node from End of List - Oracle Interview Question - Leetcode 19](https://www.youtube.com/watch?v=XVuQxVej6y8)

NOTE: Was able to solve this problem in 2nd attempt. Good Job

**Q18.** [**141. Linked List Cycle**](https://leetcode.com/problems/linked-list-cycle)**: Easy**

Floyd’s cycle finding algorithm:

2 pointers, slow and fast. Fast will start one step ahead of slow and moves in 2 steps. Slow moves in 1 step.

After a point both pointers will be equal if there is a cycle present. If not, fast will reach NULL.

NOTE: Was able to solve this problem in 2nd attempt. Will need to go through again

**Q19.** [**23. Merge k Sorted Lists**](https://leetcode.com/problems/merge-k-sorted-lists)**: Hard**

**Solved this using:** [Merge K Sorted Lists - Leetcode 23 - Python](https://www.youtube.com/watch?v=q5a5OiGbT6Q)

The above solution is O(Nlogk) but O(N) space which can be improved by leetcode approach 4 to O(1)

This youtube solution is less tricky and easy to follow.

Basic idea to solve this problem is to merge the lists that are adjacent to each other using a custom merge 2 lists function.

Keep doing this till you are left with one list. It will take logK time to merge K lists. And n times to traverse a list of size n

NOTE: Was able to solve this problem in 2nd attempt. Will need to go through again. Didnt try the most efficient solution of O(1) space.

**Q20** [**33. Search in Rotated Sorted Array**](https://leetcode.com/problems/search-in-rotated-sorted-array)**: Medium**

[Search in rotated sorted array - Leetcode 33 - Python](https://www.youtube.com/watch?v=U8XENwh8Oy8)

Take a mid pointer. Check in the first half of the list if 1st and last elements are ascending. Similarly in the other half.

In the half which is in proper order, search if target is out of bounds.

1. If yes, then check in the other half.
2. If within range, divide the current half further more and proceed to check

NOTE: include mid in the half that is sorted according to mid.

Understood the intuition. Could not write the code.

**Q21.** [**153. Find Minimum in Rotated Sorted Array**](https://leetcode.com/problems/find-minimum-in-rotated-sorted-array)**: Medium**

This is similar to the above problem.

NOTE: Leetcode given solution way simpler and easier to understand

After dividing the lists, compared the first element of the ordered half to the last element of the other half.

If the first element is smaller, return this element since this will be the minimum element. Otherwise check in the other half of the list.

[Find Minimum in Rotated Sorted Array - Binary Search - Leetcode 153 - Python](https://www.youtube.com/watch?v=nIVW4P8b1VA)

**Q22.** [**226. Invert Binary Tree**](https://leetcode.com/problems/invert-binary-tree)**: Easy**

Easy to solve recursively. Interchange the root.left and root.right recursively.

For the base case, return the root since the call wont be followed.

O(N) and space O(N) since recursion

Take a look at the iterative approach later. **Same iterative and recursive complexities.**

NOTE: Was able to solve this problem in 2nd attempt. Good Job. saw the iterative approach as well. Pretty easy to solve. Uses queues.

**Q23.** [**104. Maximum Depth of Binary Tree**](https://leetcode.com/problems/maximum-depth-of-binary-tree)**: Easy**

Easy recursive approach, basically counting variables and keeping a max variable.

O(N) and space O(N) since recursion

An iterative approach understood how to solve. The easy approach in general.

Using stack to imitate the stack calls and used in general to reduce the space complexity.

This above method has O(N), and space O(N) worst case (one-sided tree) or O(log N) avg case (balanced tree)

**Same iterative and recursive complexities.**

**Q24.** [**100. Same Tree**](https://leetcode.com/problems/same-tree)**: Easy**

Check recursively, all the nodes of both the trees. If there is even one node that is not matching return false else return true.

Base case: if p is None and q is None:

return True

**Same iterative and recursive complexities.**

O(N) time and O(logN) space (best case) or O(N) worst case.

**Q25.** [**572. Subtree of Another Tree**](https://leetcode.com/problems/subtree-of-another-tree)**: Easy**

Requires a helper function, isSameTree (same code as the question above).

Recursively check if the left or right subtree matches the given subtree.

[Time O(N \* M) | Space O(N + M)](https://leetcode.com/problems/subtree-of-another-tree/discuss/2254185/Python-Extension-of-Same-Tree-Problem-Time-O(N-*-M)-or-Space-O(N-%2B-M))

**Q26.** [**235. Lowest Common Ancestor of a Binary Search**](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree)**: Easy**

Take advantage of the BST property of the tree. The 2 nodes are guaranteed to be present.

According to the BST property, the left side of the current node is less than the current while the right side is greater.

We will travel the tree since initially, both the given values will lie on the same side of the tree till we find a split. (the values will lie on either side of the tree)

We have to find such a node that will cause this split and return it

O(N) and O(N) - recursive

O(N) and O(1) - iterative.

Understood in the 2nd attempt. The iterative method is the best for this question because the code is exactly the same as recursive.

**Q27.** [**102. Binary Tree Level Order Traversal**](https://leetcode.com/problems/binary-tree-level-order-traversal)**: Medium**

This is BFS -

To do this recursively (intuitive), first do DFS.

Create an empty list of lists, levels. Add an empty list for each level when first visiting the level. This can be known by checking len(levels) == level.

Level = 0 initially.

Level + 1 for each of the recursive left and right calls.

NOTE: For BFS with recursion, need a helper function to maintain levels.

**Same iterative and recursive complexities. But observed faster time when running iterative solution.**

[Binary Tree Level Order Traversal - BFS - Leetcode 102](https://www.youtube.com/watch?v=6ZnyEApgFYg)

Understood in the 2nd attempt. Wrote a working code

**Q28.** [**199. Binary Tree Right Side View**](https://leetcode.com/problems/binary-tree-right-side-view/)**: Medium**

Similar to the above problem. BFS iterative is intuitive here. Use a queue to traverse level-wise.

Take the last element of each level and add it to the op list.

O(N) and O(D) D is the diameter of the tree which can be N/2 worst case.

**Q29.** [**98. Validate Binary Search Tree**](https://leetcode.com/problems/validate-binary-search-tree)**: Medium**

Recursive solution:

Use a helper function, validate. With low as -inf and high as inf.

We have to check if node value falls between low and high.

Low and high will be adjusted as the recursion moves forward

O(N) and O(N).

**Same iterative and recursive complexities.**

**Q30.**[**230. Kth Smallest Element in a BST**](https://leetcode.com/problems/kth-smallest-element-in-a-bst)**: Medium**

Trick for this question is to use inorder traversal. Maintain an array and populate the array according to the inorder traversal.

Return the k-1 th element of the array.

This is a recursive solution. O(n) and O(n)

Iterative solution is better but not very intuitive to code. (Understood the solution).

2nd Attempt: Understood iterative approach better. Will need to review again

O(H + k), O(logN + k) balanced or O(N+k) unbalanced time, and O(logN) or O(N) space.

**Q31.** [**105. Construct Binary Tree from Preorder and Inorder Traversal**](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal)**: Medium**

Really tricky question. Followed neetcodes solution, easy but not that efficient:

[Construct Binary Tree from Inorder and Preorder Traversal - Leetcode 105 - Python](https://www.youtube.com/watch?v=ihj4IQGZ2zc)

The Leetcode solution is more efficient.

O(N) and O(N) - leetcode. For Neetcode it is probably N^2 time. He has used index function.

2nd Attempt: was able to recreate neetcodes solution. Understood really well leetcode’s solution.

**Q32.** [**124. Binary Tree Maximum Path Sum**](https://leetcode.com/problems/binary-tree-maximum-path-sum)**: Hard**

Tricky problem but understood the intuition.

Take a global variable to store the max sum. Recursively (DFS), solve the problem.

Compare the left + node + right sum with the global max. Update the global max accordingly.

Problem is hard because need to consider negative numbers. Before adding left + node + right, do max(left, 0) and max(right, 0). This was the fault in the previous attempt.

O(N) and O(N)/O(logN)

[Binary Tree Maximum Path Sum - DFS - Leetcode 124 - Python](https://www.youtube.com/watch?v=Hr5cWUld4vU)

2nd Attempt: Can tell it is a very elegant solution and difficult for me to come up with in one go. 4 conditions taken care of in one line condition. Understood it pretty well.

**Q33.** [**297. Serialize and Deserialize Binary Tree**](https://leetcode.com/problems/serialize-and-deserialize-binary-tree)**: Hard**

A very practical question. Serializing and deserializing a data structure (tree) to and from a string.

2 parts:

1. Serialization:

Using DFS preorder traversal for serialization since we know the value in the string is a node. Maintain a **list of strings (will have to convert the int to strings or join operation throws error)** initially to add these values to it. Replace the NULLs with a “None” string. In the end, using the join operation, return a string.

1. Deserialization:

Convert the string to a list using the split function. Now create a tree using the preorder method. After creating a node, pop off the 0th element. For the base case, if the value of the element is “None”, return None and pop off the element.

O(N) and O(N)

[Serialize and Deserialize Binary Tree - Preorder Traversal - Leetcode 297 - Python](https://www.youtube.com/watch?v=u4JAi2JJhI8)

**Q34.** [**200. Number of Islands**](https://leetcode.com/problems/number-of-islands)**: Medium**

Longest I have taken to understand any problem. Hopefully, it pays off.

[NUMBER OF ISLANDS - Leetcode 200 - Python](https://www.youtube.com/watch?v=pV2kpPD66nE)

NOTE: The leetcode solution is not ideal. It depends upon changing the input grid.

Neetcode guy solves the problem using BFS, O(M\*N), and O(M\*N) worst case (all the numbers in the grid are 1)

To solve, take a set to add the visited element indices.

Inside the BFS function, maintain a queue to add the next indices that are “1”and are not present in the set. Iteratively keep adding the indices within a loop for 4 directions of the current index till we run out of unvisited“1”.

2nd Attempt: I am better at this now. Wrote a DFS recursive solution myself.

Will need to look at it once more though

**Q35** [**133. Clone Graph**](https://leetcode.com/problems/clone-graph) **Medium**

Basically to understand graph traversal, structure and creation. Similar to a tree creation but trees are DAGs so you won't visit the same node again.

In graphs, you will probably visit the same node so need to keep an account. Will use hashmaps for this purpose.

Map the original node to the copy node.

2 approaches are DFS and BFS. BFS is much more convenient. Similar complexities,

O(V+E) and O(V). DFS space comp could be bigger.

BFS:

Add node to hashmap mapped to new node and queue.

While queue, node = popleft

For neighbor in node.neighbors

Check if neighbor present in hashmap, if not, add

hashmap[node].neighbor.append(hashmap[neighbor])

Return hashmap[node]

DFS:

Recursively keep on adding the nodes to each others neighbors list:

[Clone Graph - Depth First Search - Leetcode 133](https://www.youtube.com/watch?v=mQeF6bN8hMk)

**Q36.** [**417. Pacific Atlantic Water Flow**](https://leetcode.com/problems/pacific-atlantic-water-flow)**: Medium**

Good question. I can see myself improving. Missed a bit on the intuition but was able to code up the solution after looking at the solution though. But still was able to code it up by **thinking.**

Try out BFS solution. Feels easier. Solved using DFS:

[Pacific Atlantic Water Flow - Leetcode 417 - Python](https://www.youtube.com/watch?v=s-VkcjHqkGI)

Trick: the edges of the island are connected to 1 of the oceans. Reverse track (check if current height lower than previous to end the dfs in a direction).

Maintain 2 sets, pac and atl, and keep adding visited nodes if the nodes comply with the conditions.

Return a list of intersection of the 2 sets.

2nd attempt: understood very well but did not have time to understand to intuition by self.

O(M\*N), O(M\*N)

**Q37.** [**207. Course Schedule**](https://leetcode.com/problems/course-schedule)**: Medium**

NOTE: Also done by Topological sort in Leetcode. Should absolutely know that later.

[Course Schedule - Graph Adjacency List - Leetcode 207](https://www.youtube.com/watch?v=EgI5nU9etnU)

Trick: Have to detect if there is a cycle. If yes return False

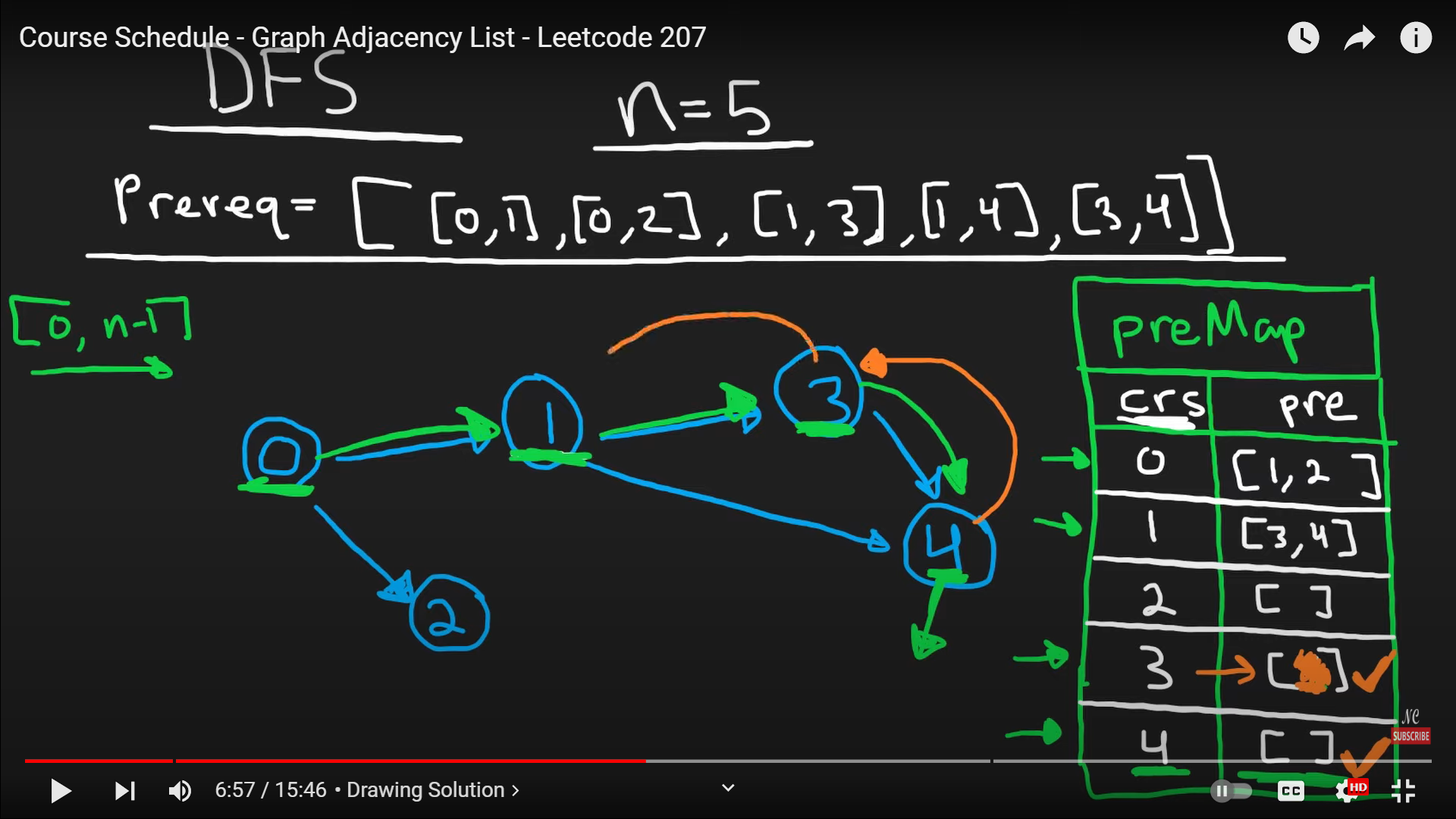
DFS with Adj list.

Create a hashmap from 0 to numCourses-1 as keys with empty lists as values.

Create a set, visited, to store if we have already visited the node in the same DFS trail (remove the node from the set once one path is complete so next path from the same node can start). Basically to detect cycle.

2nd Attempt: DFS approach – understood the approach but had to think more than the previous graph problems about how to write the code.

Topological sort:



DFS all the lists from 0 to numCourses-1. If a list is empty, it means no dependencies. Return True.

O(V+E) and O(4V+E)

**Q38.** [**323. Number of Connected Components in an Undirected Graph**](https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph)**:**

**Medium**

Straight forward question using DFS. Make sure to add edges in both directions to avoid errors (undirected graphs). Keep doing DFS and increment the variable during a new DFS.

**NOTE: This is a union-find problem. UPDATE: Solved using union-find**

[Number of Connected Components in an Undirected Graph - Union Find - Leetcode 323 - Python](https://www.youtube.com/watch?v=8f1XPm4WOUc)

O(V+E) and O(4V+E)

2nd Attempt: DFS – very easy

Union Find –

**Q39.** [**261. Graph Valid Tree**](https://leetcode.com/problems/graph-valid-tree)**: Medium**

Solved this question using Union-Find. Did it all by myself without needing to see the solution, which is progress.

Union-find:

2 functions

1. Find(node): Find returns the parent of the given node.
2. Union(node1, node2): returns joint of parents of node1 and node2 if both are from different sets. If they are from the same sets then you get a cycle by including the edge (node1,node2).

NOTE: (node1,node2) represents an edge that might cause 2 disjoint components to join.

2nd Attempt: could almost come up with a DFS solution but could not tell that this problem was of union find.

**Q40.** [**269. Alien Dictionary**](https://leetcode.com/problems/alien-dictionary)**: Hard**

Needs a lot to look into. Topological sort with post order DFS. For topological sort, can be represented with 3 values white(not visited), grey(visited), and black(cycle present).

Do not use visited as a set but use it as a map for the above 3 values.

Create edges according to the relations found by the given rules.

Because of post order, ans will be inverted. Reverse while joining.

Post order is used so that once a DFS is completed on a node and its children, marked as grey(False) or Black(True) we can add the word in the result.

[Alien Dictionary - Topological Sort - Leetcode 269 - Python](https://www.youtube.com/watch?v=6kTZYvNNyps)

**Q41.** [**198. House Robber**](https://leetcode.com/problems/house-robber)**: Medium**

This is supposed to be a classic DP problem. Understood and can implement the recursive(memoised) and the bottom up DP approaches.

O(N) and O(N)

But have some trouble wrapping my head around the optimized DP

O(N) and O(1)

Attempt 2: Understood the problem and the DP approach. Though had to go through the video. Still difficult to come up with a DP solution.

NOTE: NEED to come up with a recurrence relation for the DP approach otherwise very difficult

**Recurrence relation: max(nums[i] + subproblem(i+2), subproblem(i+1))**

[House Robber - Leetcode 198 - Python Dynamic Programming](https://www.youtube.com/watch?v=73r3KWiEvyk)

**Q42.** [**213. House Robber II**](https://leetcode.com/problems/house-robber-ii)**: Medium**

The same as the above problem, very simple solution couldnt figure out though.

Just write a sub-function as so you can run it twice on nums[1:] and nums[:-1].

Return the max of the above.

2nd Attempt: the solution still didn't strike me. It felt counter-intuitive and the sort of thing you would do as a last resort.

**Q43.** [**5. Longest Palindromic Substring**](https://leetcode.com/problems/longest-palindromic-substring)**: Medium**

Complexity Analysis

Time complexity : O(n^2)

Since expanding a palindrome around its center could take O(n)O(n) time, the overall complexity is O(n^2).

Space complexity : O(1)

[Longest Palindromic Substring - Python - Leetcode 5](https://www.youtube.com/watch?v=XYQecbcd6_c)

Trick: loop over every letter and expand the letter till it is a palindrome. If it is the longest, record it.

For odd and even palindromes, run it twice. With l,r = i,i and l,r = i, i+1 respectively.

2nd attempt: Didn’t remember the intuition. A glimpse at the code made it clear. The code is super easy after the intuition.

NOTE:INTUITION is everything for this one

**Q44.** [**647. Palindromic Substrings**](https://leetcode.com/problems/palindromic-substrings)**: Medium**

Exact same but an even simpler version of the above code.

Instead of recording the pointers while expanding, just set a counter to count the palindromic substrings. Return the final count.

**Q45.** [**91. Decode Ways**](https://leetcode.com/problems/decode-ways) **Medium**

Deceptively tricky question. Tricky because of the edge cases.

Came up with the recurrence relation but had difficulty changing it into the code. Had difficulty with memoization

Understood the O(n) space DP solution and implemented it. Could not implement constant space solution.

Trick is to add 1 to the result once you get to the end of the str. Or add zero if the starting char of the substring is 0.

[Decode Ways - Dynamic Programming - Leetcode 91 - Python](https://www.youtube.com/watch?v=6aEyTjOwlJU)

2nd Attempt: decision tree itself was wrong. Start with an empty op string and then choose till the string is valid.

Edge case 0 is the culprit. It becomes difficult to handle even recursive solution with this problem.

NOTE: This one is difficult for me. Cant write iterative solution without recursive.

**Q46.** [**322. Coin Change**](https://leetcode.com/problems/coin-change)**: Medium**

Very easily understandable iterative solution. But had a very difficult time coming up with a recursive solution even after understanding the recurrence relation.

[Coin Change - Dynamic Programming Bottom Up - Leetcode 322](https://www.youtube.com/watch?v=H9bfqozjoqs)

Side NOTE: It is a variation of an unbounded knapsack:

[16 Coin change problem: Minimum number of coins](https://www.youtube.com/watch?v=I-l6PBeERuc&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go&index=16)

O(S\*n) and O(S\*n) where S is the size of the amount and n are the number of coins.

**Q47.** [**152. Maximum Product Subarray**](https://leetcode.com/problems/maximum-product-subarray)**: Medium**

Similar to kadanes algorithm for calculating max sum subarray.

Trick: along with the max, maintain a min value as well. Keep updating the max and min values.

Return the max at the end.

[Maximum Product Subarray - Dynamic Programming - Leetcode 152](https://www.youtube.com/watch?v=lXVy6YWFcRM)

2nd attempt: was able to follow

**Q48.** [**139. Word Break**](https://leetcode.com/problems/word-break)**: Medium**

Trick: index on the main string. Keep for looping all the wordDict on the ith word in the main string. Stop when found true and skip i + len(w) positions. Here w is one of the words in wordDict.

[Word Break - Dynamic Programming - Leetcode 139 - Python](https://www.youtube.com/watch?v=Sx9NNgInc3A)

Was able to follow the DP solution but difficult to find the rec + memoi solution.

2nd attempt: was able to build a DP solution but need to take a look at the rec solution.

O(N\*M\*N) and O(N)

**Q49.** [**300. Longest Increasing Subsequence**](https://leetcode.com/problems/longest-increasing-subsequence)**: Medium**

**NOTE: Try the rec + memoi solution of the problem later.**

[Longest Increasing Subsequence - Dynamic Programming - Leetcode 300](https://www.youtube.com/watch?v=cjWnW0hdF1Y)

O(N^2) and O(N) —> using Dynamic programming

O(N \* logN) solution using Binary search —-> take a look later

Previous approach: Thought would be able to get an O(N) solution and was trying for it.

Keep a track of every subproblem on the DP array. (For loop inside a for loop).

**Q50.** [**62. Unique Paths**](https://leetcode.com/problems/unique-paths)**: Medium**

Pretty straightforward question. O(M\*N) and min(O(M),O(N)) complexities.

Will get the answer at the last block. Keep solving the subproblem.

Solution equation: Dp[i][j] = dp[i][j-1] + dp[i-1][j]

[Unique Paths - Dynamic Programming - Leetcode 62](https://www.youtube.com/watch?v=IlEsdxuD4lY)

NOTE: There is a 3rd approach - MATH which basically is a PnC solution with a very complicated but lower time complexity.

**Q51.** [**1143. Longest Common Subsequence**](https://leetcode.com/problems/longest-common-subsequence)**: Medium**

[Longest Common Subsequence - Dynamic Programming - Leetcode 1143](https://www.youtube.com/watch?v=Ua0GhsJSlWM)

Again, kind of a straightforward 2D DP question.

Will get the answer at the last block. Keep solving the subproblem.

Solution equation:

if text2[i-1] == text1[j-1]:

dp[i][j] = dp[i-1][j-1] + 1

else:

dp[i][j] = max(dp[i-1][j], dp[i][j-1])

O(M\*N) and O(M\*N) —----> space complexity could be improved to min(O(M),O(N)) by using a 1D array and a variable for previous value in one of the directions.

Detailed explanation given in solution section: Approach 4: Dynamic Programming with Space Optimization

**Q52.** [**53. Maximum Subarray**](https://leetcode.com/problems/maximum-subarray)**: Medium**

**Q53.** [**55. Jump Game**](https://leetcode.com/problems/jump-game)**: Medium**

[Jump Game - Greedy - Leetcode 55](https://www.youtube.com/watch?v=Yan0cv2cLy8)

Trick: think of it as if we are moving the goal post closer to first index after each iteration..

Basically, solving the sub-problem. If at the end, the goal post is index 0 then True else False.

**Q54.** [**208. Implement Trie (Prefix Tree)**](https://leetcode.com/problems/implement-trie-prefix-tree)**: Medium**

Pretty easy question. Was not able to do it because was something new.

Trick: TrieNode class is required and is not there by default in leetcode.

Trienode class will have fields: children (hashmap) and a boolean isEnd flag.

Rest of the 3 functions are basically the same with minor variations. Was able to figure that out by myself with the help of the video explanation:

[Implement Trie (Prefix Tree) - Leetcode 208](https://www.youtube.com/watch?v=oobqoCJlHA0)

Trie data structure complexities:

Insert: O(m) and O(m)

Search: O(m) and O(1)

startsWith: O(m) and O(1)

**Q55.** [**211. Design Add and Search Words Data Structure**](https://leetcode.com/problems/design-add-and-search-words-data-structure)**: Medium**

Almost had this question correct. Tricky part here is the use of “.”

Intuition was correct. Had problem with coding.

Recursion + iteration combined

[Design Add and Search Words Data Structure - Leetcode 211 - Python](https://www.youtube.com/watch?v=BTf05gs_8iU)

Search: O(M) and O(1) for words without dots

O(N\*26^M) and O(1) for words with all dots

M is the key length and N is the number of keys.

2nd Attempt: The recursive part was not very intuitive to me

**Q56.** [**39. Combination Sum**](https://leetcode.com/problems/combination-sum)**: Medium**

**NOTE: This is a backtracking question.**

NOTE: very similar to the subsets problem. Try this question using aditya verma’s approach later

I tried to solve it like the leetcode solution unknowingly. Not very intuitive at the place where i got stuck. (That part is essential to maintain unique output lists)

The solution by neetcode is the similar approach that Aditya Verma took for subsets question.

2 recursive calls - O(2^T) where T is the height of the decision tree. And O(2^T) space.

[Combination Sum - Backtracking - Leetcode 39 - Python](https://www.youtube.com/watch?v=GBKI9VSKdGg)

2nd attempt: will need to go through what neetcode did

**Q57.** [**79. Word Search**](https://leetcode.com/problems/word-search)**: Medium**

**NOTE: This is a backtracking question.**

[Word Search - Backtracking - Leetcode 79 - Python](https://www.youtube.com/watch?v=pfiQ_PS1g8E)

Solved it exactly like how neetcode would solve without any external help XD

Was stuck at a variable mismatch but thats okay! This is great progress! The question has just 39% acceptance. Keep moving forward!

It felt more like a graph question since had to do DFS in 4 directions.

Trick: straightforward, scan through the grid with each word as starting point.

If the word matches, do DFS else return False. Use a set to mark visited nodes.

This is a very brute force solution:

O(M\*N\*4^word) and O(L)

2nd Attempt: since solved it in first attempt, still remember it.

**Q58.** [**212. Word Search II**](https://leetcode.com/problems/word-search-ii)**: Hard**

2 things involved to solve this question:

**Backtracking + Tries**

Very tricky to solve considering the last test case in python which needs pruning of the prefix tree for the word added to the result if the word is isolated.

Backtracking i figured out for word search I problem. Trie is not that difficult to implement.

Trick: making the trie of the given list of words and not the grid itself.

Continuing from word search, the structure will be the same except some changes here and there.

1. Check if the current word tile is in the children hashmap of the current node.
2. DFS will not return bool but returns None. res is a set so when a word from the trie reaches its end and we have traced it on the grid, add the word to the res set.
3. Final return: convert res set to list

[Word Search II - Backtracking Trie - Leetcode 212 - Python](https://www.youtube.com/watch?v=asbcE9mZz_U)

**Time: O(M\*N\*4^word) and space: O(N) where N is total no. of letters in Trie.**

**Q59.** [**57. Insert Interval**](https://leetcode.com/problems/insert-interval)**: Medium**

This is a greedy problem.

[Insert Interval - Leetcode 57 - Python](https://www.youtube.com/watch?v=A8NUOmlwOlM)

This question is relatively straightforward. Create a new output list. Add the non overlapping intervals to the output list. If overlapping case, then merge the intervals, taking the intervals from the intervals list into consideration one at a time. Take min and max of lower and upper bounds.

O(n) and O(n)

**Q60.** [**56. Merge Intervals**](https://leetcode.com/problems/merge-intervals)**: Medium**

Mistake: tried to maintain a merge variable to add later like in the above question.

Just reference the last entry of the result variable and compare current entries of the input list to it. Much easier.

[Merge Intervals - Sorting - Leetcode 56](https://www.youtube.com/watch?v=44H3cEC2fFM)

O(N) because of sorting and O(N)

**Q61.** [**435. Non-overlapping Intervals**](https://leetcode.com/problems/non-overlapping-intervals)**: Medium**

Definitely not an easy question. Requires to go through. Good job on figuring out the intuition and drawing the diagram just like neetcode helped. Getting there slowly. Missed one thing that is one case of variation about updating the prevEnd but rest was spot on.

Requires initial sorting of start as well as end intervals.

O(NlogN) and O(1)

[Non-Overlapping Intervals - Leetcode 435 - Python](https://www.youtube.com/watch?v=nONCGxWoUfM)

**Q62.** [**252. Meeting Rooms**](https://leetcode.com/problems/meeting-rooms)**: Easy**

Very easy question if done the above 3 questions. Just have to detect overlap.

O(NlogN) and O(1)

**Q63.** [**253. Meeting Rooms II**](https://leetcode.com/problems/meeting-rooms-ii)**: Medium**

Slightly trickier than the rest of the interval problems. Did come very close to the min heap solution. Note to self: Should use more heaps.

Can come up with the heap approach.

[Meeting Rooms II - Leetcode 253 - Python](https://www.youtube.com/watch?v=FdzJmTCVyJU)

Approach 2: chronological ordering

The problem looks for max overlap at a given time. So maintain a variable to check the max overlaps throughout.

Separate out the starts and ends and sort both lists. We don`t care which meeting begins and which ends as long as we can get a room empty for the next meeting.

Maintain 2 pointers for both the lists. Increment count var and move by 1, start pointer, if start<end. If not, decrement count and increment end pointer.

O(NlogN) and O(N)

**Q64.** [**48. Rotate Image**](https://leetcode.com/problems/rotate-image)**: Medium**

Trick: First Transpose then reflect.

**Q65.** [**54. Spiral Matrix**](https://leetcode.com/problems/spiral-matrix)**: Medium**

Looks difficult or tricky but is straightforward. 4 pointers and 4 for loops inside a while loop for each pointer.

Trick: after 1st 2 for loops, check if pointers overlap or out of bounds.

[Spiral Matrix - Microsoft Interview Question - Leetcode 54](https://www.youtube.com/watch?v=BJnMZNwUk1M)

O(M\*N) and O(1)

**Q66.** [**73. Set Matrix Zeroes**](https://leetcode.com/problems/set-matrix-zeroes)**: Medium**

[Set Matrix Zeroes - In-place - Leetcode 73](https://www.youtube.com/watch?v=T41rL0L3Pnw)

Kind of a tricky one to do in O(1) space.

Maintain the first row first col as the arrays to store which row or col will be zero.

Maintain another variable for either row or col since the 2 arrays will overlap for one value.

O(M\*N) and O(1)

2nd Attempt: Understood both the space and O(1) solution.

**Q67.** [**191. Number of 1 Bits**](https://leetcode.com/problems/number-of-1-bits)**: Easy**

n= n & n-1. Keep doing till n = 0. Increment count for each operation.

O(logN) and O(1)

**Q68.** [**338. Counting Bits**](https://leetcode.com/problems/counting-bits)**: Easy**

Maintain a DP of len n+1. Each value of DP[i] will be DP[i & i-1] + 1

O(N) and O(N).

**Q69.** [**190. Reverse Bits**](https://leetcode.com/problems/reverse-bits)**: Easy**

O(1) and O(1)/O(n) since we know we have to reverse a 32 bit integer

Most efficient solution:

Switching the bits first 16,8,4,2, and 1

# n = (n >> 16) | (n << 16)

# n = ((n & 0xff00ff00) >> 8 | (n & 0x00ff00ff) << 8)

# n = ((n & 0xf0f0f0f0) >> 4 | (n & 0x0f0f0f0f) << 4)

# n = ((n & 0xcccccccc) >> 2 | (n & 0x33333333) << 2)

# n = ((n & 0xaaaaaaaa) >> 1 | (n & 0x55555555) << 1)

# return n

**Q70.** [**268. Missing Number**](https://leetcode.com/problems/missing-number)**: Easy**

Approach 1: Exp\_sum = n \* (n+1)/2 ….. Ans = Exp\_sum - sum

Approach 2: XOR

XOR index i and nums[i] for all values. XOR for 1 additional value len(nums). This will give the ans.

O(N) and O(1)

**Q71.** [**128. Longest Consecutive Sequence**](https://leetcode.com/problems/longest-consecutive-sequence)**: Medium**

Copy nums into a set.

Check for only if nums[i]-1 not in set. Then reset outputTemp.

In this condition itself, keep checking in a while loop and keep incrementing outputTemp.

[Leetcode 128 - LONGEST CONSECUTIVE SEQUENCE](https://www.youtube.com/watch?v=P6RZZMu_maU)

**Q72.** [**121. Best Time to Buy and Sell Stock**](https://leetcode.com/problems/best-time-to-buy-and-sell-stock)**: Easy**

Was able to come up with a solution in the last review

**Q73.** [**206. Reverse Linked List**](https://leetcode.com/problems/reverse-linked-list)**: Easy**

Was able to come up with an iterative solution in the last review.

Had some difficulty with the recursive solution.

**Q74.** [**21. Merge Two Sorted Lists**](https://leetcode.com/problems/merge-two-sorted-lists)**: Easy**

Was able to come up with an iterative solution in the last review.

Had a lot of difficulty to visualise the recursive solution.

**Q75.** [**70. Climbing Stairs**](https://leetcode.com/problems/climbing-stairs)**: Easy**

2nd Attempt: understood this problem better than last time. My DP is terrible but I can see progress. Will take longer time than other topics.

NOTE: NEED to come up with a recurrence relation for the DP approach otherwise very difficult

**Q76.** [**371. Sum of Two Integers**](https://leetcode.com/problems/sum-of-two-integers): **Easy**

This problem is a bit tricky because of python representation of numbers.

There is no 32 bit limit here for numbers and negative numbers are the real problem here.

This is because negatives are not represented the same way as that in JAVA.

Basic logic of the problem in Java:

[Sum of Two Integers - Leetcode 371 - Java](https://www.youtube.com/watch?v=gVUrDV4tZfY)

O(1) and O(1)

NOTE: Need to look in the bit representation of negative numbers in python.