Problem	Difficulty	Repeat		Notes
Two Sum	E	✓		O(n) soln. store remainders in dictionary, pass thru again and check if compliment is in dict
Reverse Linked L	E	✓		prev, next, curr
Climbing Stairs	E	✓		Memo dictionary, fibonacci recursion
Maximum Depth	E	✓		if root 1+left+right
Kth smallest elm	E	✓		in order traversal list return kth index
Valid Parenthese	E	✓		iterate w stack, only put open paren on stack, check if closed paren has corresponding on top of stack
LCA of BST	E	✓		check if ur p or q, then if not go right or left depending
Binary Tree Inord	E	✓		left, node, right
Valid Palindrome	E	✓		easy
Shuffle Playlist	E/M			if i want O(1) access to end of list, I can find my random index, swap it with the last index, and pop in O(1) time
Permutations	M	✓		check append condition, that perm_list = len(list) - recurse on for loop, terrible time comlpexity O(n!)
Unique Path	M	✓		unique paths = unique paths from right + unique paths from top
Longest Increasi	M	✓		DP, keep track of longest inc. subsequence at each element, choose to continue the longest one which you are greater than
Merge Sort	M			Merge with two pointers, sort by breaking up into two and swapping at base case of length 2, Merge(Sort) - merge your sort calls
House Robbers I	M	✓		DP, either choose the house and add its value with your call or go to the next one
House Robbers I	M			DP, call house robbers excluding first index and exlcuding last index
Maximum subarr	E	✓		Kadene's Algorithm, iterate thru list, keep track of sum and max_so_far, if sum < 0, set it to 0, you don't want to start from there, else add it and if sum > max_ then max_ = sum
Invert Tree	E			flip right and left using tmp, call on right and left
Remove Nth Nod	M	✓		count the nodes in linked list, then iterate back through decrementing counter
Longest Substrin	M	✓		sliding window, if you encounter a repeat then slide left side, if not slide right slide
Top K Frequent E	M	✓		use heap, heapq nlargest, O(nlogk) heap of size k operations are O(logk) to make a heap its O(n)
Clone Graph	M	✓		using BFS queue, copy a node if you never encountered it before, if you have take it from the copy dictionary, add the neighbors clone or create neighbors clone then continue on
Valid Anagaram	E	✓		count dict., easy
Binary Tree ZigZ	M			Same as level order traversal, except keep a flag, if true then add in nodes like a queue (L to R) add in at back then append to level list, if false then add nodes in like a stack (R to L) add in at back then pop off the back becaue the rightmost nodes will be queued up last so add those first (can be done by inserting nodes to front of level list)
Contingous Array	M			Clever trick, keep the sum as you go, +1 for 1s and -1s for 0s, if you encounter sum of 0 then everything up to that point is equal # of 0s and 1s, if you encounter the same sum twice, then everything in between is equal 1s and 0s, so keep track of first place you encounter a sum and compare index-first index with max_ when you encounter it again
Kth Largest Elem	M	✓		Sort and return k-1th index
Same Tree	E			Run through the tree and if you encounter a node that is the same value as subroot then run a helper that checks if the trees are the same
Longest Commor	M	✓		DP, two variables for subproblem, check if index u are on for A and B strings are the same character if so add 1 and decrement if not take max of decrement A or B
Coin Change	M	✓		DP, two varaibles for subproblem, one for the cost and one for the index of coins array, subtract the coin from the cost and call again + the same cost with index decremented to next coin
Number of Island	M	✓		DFS, go in all directions, count connected components
Binary Tree Leve	M	✓		Use a queue to keep track of nodes you have to go through, keep a counter that is the length of the queue at every new level,decrement as you pop off, reset when it hits 0
Course Schedule	M			Construct a graph with an edge from a prereq to its class, detect a cycle with DFS with a seen set, reset seen at the end of recursion, keep track of nodes that have no cycles
First Missing Pos	H	✓		Check if there is a 1, change negative numbers to 1, set index of number you encounter to 1, first missing number will be between 1 and n+1 where n is length of array