

Video Solution	Category	Name	Link	Notes					
<a href="https://youtu.be/KLUXCFGStnA">https://youtu.be/KLUXCFGStnA</a>	Arrays	Two Sum	<a href="https://leetcode">https://leetcode</a>	use hash map to instantly check for difference value, map will add index of last occurrence of a num, don't use same element twice;					
<a href="https://youtu.be/1pkOpXD63yU">https://youtu.be/1pkOpXD63yU</a>	Arrays	Best Time to Buy and Sell Stock	<a href="https://leetcode">https://leetcode</a>	find local min and search for local max, sliding window;					
<a href="https://youtu.be/3oamzN90kPg">https://youtu.be/3oamzN90kPg</a>	Arrays	Contains Duplicate	<a href="https://leetcode">https://leetcode</a>	hashset to get unique values in array, to check for duplicates easily					
<a href="https://youtu.be/bNvIQI2wAlk">https://youtu.be/bNvIQI2wAlk</a>	Arrays	Product of Array Except Self	<a href="https://leetcode">https://leetcode</a>	make two passes, first in-order, second in-reverse, to compute products					
<a href="https://youtu.be/sWz3MMTOeEg">https://youtu.be/sWz3MMTOeEg</a>	Arrays	Maximum Subarray	<a href="https://leetcode">https://leetcode</a>	pattern: prev subarray cant be negative, dynamic programming: compute max sum for each prefix					
<a href="https://youtu.be/IXv6YWFcRm">https://youtu.be/IXv6YWFcRm</a>	Arrays	Maximum Product Subarray	<a href="https://leetcode">https://leetcode</a>	dp: compute max and max-abs-val for each prefix subarr;					
<a href="https://youtu.be/nlVW4f8b1VA">https://youtu.be/nlVW4f8b1VA</a>	Arrays	Find Minimum in Rotated Sorted Array	<a href="https://leetcode">https://leetcode</a>	check if half of array is sorted in order to find pivot, arr is guaranteed to be in at most two sorted subarrays					
<a href="https://youtu.be/U8XENwh8Oy8">https://youtu.be/U8XENwh8Oy8</a>	Arrays	Search in Rotated Sorted Array	<a href="https://leetcode">https://leetcode</a>	at most two sorted halves, mid will be apart of left sorted or right sorted, if target is in range of sorted portion then search it, otherwise search other half					
<a href="https://youtu.be/jzZsG8nZR9A">https://youtu.be/jzZsG8nZR9A</a>	Arrays	3Sum	<a href="https://leetcode">https://leetcode</a>	sort input, for each first element, find next two where -a = b+c, if a=prevA, skip a, if b=prevB skip b to elim duplicates; to find b,c use two pointers, left/right on remaining list;					
<a href="https://youtu.be/UuItKBwPeAo">https://youtu.be/UuItKBwPeAo</a>	Arrays	Container With Most Water	<a href="https://leetcode">https://leetcode</a>	shrinking window, left/right initially at endpoints, shift the pointer with min height;					
<a href="https://youtu.be/gVuUDv4ZfY">https://youtu.be/gVuUDv4ZfY</a>	Binary	Sum of Two Integers	<a href="https://leetcode">https://leetcode</a>	add bit by bit, be mindful of carry, after adding, if carry is still 1, then add it as well;					
<a href="https://youtu.be/5Km3utixwZs">https://youtu.be/5Km3utixwZs</a>	Binary	Number of 1 Bits	<a href="https://leetcode">https://leetcode</a>	modulo, and dividing n; mod and div are expensive, to divide use bit shift, instead of mod to get 1's place use bitwise & 1;					
<a href="https://youtu.be/RyBM56RUwRm">https://youtu.be/RyBM56RUwRm</a>	Binary	Counting Bits	<a href="https://leetcode">https://leetcode</a>	write out result for num=16 to figure out pattern; res[i] = res[i - offset], where offset is the biggest power of 2 <= i;					
<a href="https://youtu.be/WnPLSRLSANE">https://youtu.be/WnPLSRLSANE</a>	Binary	Missing Number	<a href="https://leetcode">https://leetcode</a>	compute expected sum - real sum; xor n with each index and value;					
<a href="https://youtu.be/UcoN6UJAIG4">https://youtu.be/UcoN6UJAIG4</a>	Binary	Reverse Bits	<a href="https://leetcode">https://leetcode</a>	reverse each of 32 bits;					
<a href="https://youtu.be/Y0IT9F7qI">https://youtu.be/Y0IT9F7qI</a>	Dynamic Programm	Climbing Stairs	<a href="https://leetcode">https://leetcode</a>	subproblem find (n-1) and (n-2), sum = n;					
<a href="https://youtu.be/H9bfgaozogs">https://youtu.be/H9bfgaozogs</a>	Dynamic Programm	Coin Change	<a href="https://leetcode">https://leetcode</a>	top-down: recursive dfs, for amount, branch for each coin, cache to store prev coin_count for each amount; bottom-up: compute coins for amount = 1, up until n, using for each coin (amount - coin), cache prev values					
<a href="https://youtu.be/cjWnW0hdEYI">https://youtu.be/cjWnW0hdEYI</a>	Dynamic Programm	Longest Increasing Subsequence	<a href="https://leetcode">https://leetcode</a>	recursive: for each num, get subseq with num and without num, only include num if prev was less, cache solution of each; dp=subseq length which must end with each num, curr num must be after a prev dp or by itself;					
<a href="https://youtu.be/UaOghsJSIWM">https://youtu.be/UaOghsJSIWM</a>	Dynamic Programm	Longest Common Subsequence	<a href="https://leetcode">https://leetcode</a>	recursive: if first chars are equal find lcs of remaining of each, else max of: lcs of first and remain of 2nd and lcs of 2nd remain of first, cache result; nested forloop to compute the cache without recursion;					
<a href="https://youtu.be/5x9NNgInc3A">https://youtu.be/5x9NNgInc3A</a>	Dynamic Programm	Word Break Problem	<a href="https://leetcode">https://leetcode</a>	for each prefix, if prefix is in dict and wordbreak(remaining str)=True, then return True, cache result of wordbreak;					
<a href="https://youtu.be/GBKl9V5kdGg">https://youtu.be/GBKl9V5kdGg</a>	Dynamic Programm	Combination Sum	<a href="https://leetcode">https://leetcode</a>	visualize the decision tree, base case is curSum = or > target, each candidate can have children of itself or elements to right of it inorder to elim duplicate solutions;					
<a href="https://youtu.be/73z3KWiEvYk">https://youtu.be/73z3KWiEvYk</a>	Dynamic Programm	House Robber	<a href="https://leetcode">https://leetcode</a>	for each num, get max of prev subarr, or num + prev subarr not including last element, store results of prev, and prev not including last element					
<a href="https://youtu.be/rWAJCFYYOvM">https://youtu.be/rWAJCFYYOvM</a>	Dynamic Programm	House Robber II	<a href="https://leetcode">https://leetcode</a>	subarr = arr without first & last, get max of subarr; then pick which of first/last should be added to it					
<a href="https://youtu.be/6aEyTJOWdJU">https://youtu.be/6aEyTJOWdJU</a>	Dynamic Programm	Decode Ways	<a href="https://leetcode">https://leetcode</a>	can cur char be decoded in one or two ways? Recursion -> cache -> iterative dp solution, a lot of edge cases to determine, 52, 31, 29, 10, 20 only decoded one way, 11, 26 decoded two ways					
<a href="https://youtu.be/lIEsdxuD4IY">https://youtu.be/lIEsdxuD4IY</a>	Dynamic Programm	Unique Paths	<a href="https://leetcode">https://leetcode</a>	work backwards from solution, store paths for each position in grid, to further optimize, we don't store whole grid, only need to store prev row;					
<a href="https://youtu.be/Yan0cv2cIy8">https://youtu.be/Yan0cv2cIy8</a>	Dynamic Programm	Jump Game	<a href="https://leetcode">https://leetcode</a>	visualize the recursive tree, cache solution for O(n) time/mem complexity, iterative is O(1) mem, just iterate backwards to see if element can reach goal node, if yes, then set it equal to goal node, continue;					
<a href="https://youtu.be/mQeF6bN8hMk">https://youtu.be/mQeF6bN8hMk</a>	Graph	Clone Graph	<a href="https://leetcode">https://leetcode</a>	recursive dfs, hashmap for visited nodes					
<a href="https://youtu.be/Egl5nU9etnU">https://youtu.be/Egl5nU9etnU</a>	Graph	Course Schedule	<a href="https://leetcode">https://leetcode</a>	build adjacency, list with edges, run dfs on each V, if while dfs on V we see V again, then loop exists, otherwise V isnt in a loop, 3 states= not visited, visited, still visiting					
<a href="https://youtu.be/s-ViciHkGJI">https://youtu.be/s-ViciHkGJI</a>	Graph	Pacific Atlantic Water Flow	<a href="https://leetcode">https://leetcode</a>	dfs each cell, keep track of visited, and track which reach pac, atl; dfs on cells adjacent to pac, atl, find overlap of cells that are visited by both pac and atl cells;					
<a href="https://youtu.be/pZKpPD66nE">https://youtu.be/pZKpPD66nE</a>	Graph	Number of Islands	<a href="https://leetcode">https://leetcode</a>	foreach cell, if cell is 1 and unvisited run dfs, increment count and marking each contiguous 1 as visited					
<a href="https://youtu.be/P6RZZMu_maU">https://youtu.be/P6RZZMu_maU</a>	Graph	Longest Consecutive Sequence	<a href="https://leetcode">https://leetcode</a>	use brute force and try to optimize, consider the max subseq containing each num; add each num to hashset, for each num if num-1 doesn't exist, count the consecutive nums after num, ie num+1; there is also a union-find solution;					
<a href="https://youtu.be/6kTZvNnVnpS">https://youtu.be/6kTZvNnVnpS</a>	Graph	Alien Dictionary (Leetcode Premium)	<a href="https://leetcode">https://leetcode</a>	chars of a word not in order, the words are in order, find adjacency list of each unique char by iterating through adjacent words and finding first chars that are different, run topsort on graph and do loop detection;					
<a href="https://youtu.be/bx5UuowmnoQ">https://youtu.be/bx5UuowmnoQ</a>	Graph	Graph Valid Tree (Leetcode Premium)	<a href="https://leetcode">https://leetcode</a>	union find, if union return false, loop exists, at end size must equal n, or its not connected; dfs to get size and check for loop, since each edge is double, before dfs on neighbor of N, remove N from neighbor list of neighbor;					
<a href="https://youtu.be/8f1XPm4WOUc">https://youtu.be/8f1XPm4WOUc</a>	Graph	Number of Connected Components in	<a href="https://leetcode">https://leetcode</a>	dfs on each node that hasn't been visited, increment component count, adjacency list; bfs and union find are possible;					
<a href="https://youtu.be/A8NUOmIwOIM">https://youtu.be/A8NUOmIwOIM</a>	Interval	Insert Interval	<a href="https://leetcode">https://leetcode</a>	insert new interval in order, then merge intervals; newinterval could only merge with one interval that comes before it, then add remaining intervals;					
<a href="https://youtu.be/44H3cEC2fMg">https://youtu.be/44H3cEC2fMg</a>	Interval	Merge Intervals	<a href="https://leetcode">https://leetcode</a>	sort each interval, overlapping intervals should be adjacent, iterate and build solution; also graph method, more complicated					
<a href="https://youtu.be/nONCGxWoUfM">https://youtu.be/nONCGxWoUfM</a>	Interval	Non-overlapping Intervals	<a href="https://leetcode">https://leetcode</a>	instead of removing, count how max num of intervals you can include, sort intervals, dp to compute max intervals up until the i-th interval;					
<a href="https://youtu.be/PaIqZVPbHg">https://youtu.be/PaIqZVPbHg</a>	Interval	Meeting Rooms (Leetcode Premium)	<a href="https://leetcode">https://leetcode</a>	sort intervals by start time, if second interval doesn't overlap with first, then third def wont overlap with first;					
<a href="https://youtu.be/FdmJmTCVYjU">https://youtu.be/FdmJmTCVYjU</a>	Interval	Meeting Rooms II (Leetcode Premium)	<a href="https://leetcode">https://leetcode</a>	we care about the points in time where we are starting/ending a meeting, we already are given those, just separate start/end and traverse counting num of meetings going at these points in time; for each meeting check if a prev meeting has finished before curr started, using min heap;					
<a href="https://youtu.be/G0_I-ZF0S38">https://youtu.be/G0_I-ZF0S38</a>	Linked List	Reverse a Linked List	<a href="https://leetcode">https://leetcode</a>	iterate through maintaining cur and prev; recursively reverse, return new head of list					
<a href="https://youtu.be/g8Te7iFR3vc">https://youtu.be/g8Te7iFR3vc</a>	Linked List	Detect Cycle in a Linked List	<a href="https://leetcode">https://leetcode</a>	dict to remember visited nodes; two pointers at different speeds, if they meet there is loop					
<a href="https://youtu.be/XldIkg956u0">https://youtu.be/XldIkg956u0</a>	Linked List	Merge Two Sorted Lists	<a href="https://leetcode">https://leetcode</a>	insert each node from one list into the other					
<a href="https://youtu.be/g5a5OIgbTgQ">https://youtu.be/g5a5OIgbTgQ</a>	Linked List	Merge K Sorted Lists	<a href="https://leetcode">https://leetcode</a>	divide and conquer, merge lists, N total nodes, k-lists, O(N*logk). For each list, find min val, insert it into list, use priorityQ to optimize finding min O(N*logk)					
<a href="https://youtu.be/XVuQxVej6y8">https://youtu.be/XVuQxVej6y8</a>	Linked List	Remove Nth Node From End Of List	<a href="https://leetcode">https://leetcode</a>	use dummy node at head of list, compute len of list; two pointers, second has offset of n from first;					
<a href="https://youtu.be/55bdfUtrKLM">https://youtu.be/55bdfUtrKLM</a>	Linked List	Reorder List	<a href="https://leetcode">https://leetcode</a>	reverse second half of list, then easily reorder it; non-optimal way is to store list in array;					
<a href="https://youtu.be/T41r1OL3Pnw">https://youtu.be/T41r1OL3Pnw</a>	Matrix	Set Matrix Zeroes	<a href="https://leetcode">https://leetcode</a>	use sets to keep track of all rows, cols to zero out, after, for each num if it is in a zero row or col then change it to 0; flag first cell in row, and col to mark row/col that needs to be zeroed;					
<a href="https://youtu.be/BjnMZNwUk1M">https://youtu.be/BjnMZNwUk1M</a>	Matrix	Spiral Matrix	<a href="https://leetcode">https://leetcode</a>	keep track of visited cells; keep track of boundaries, layer-by-layer;					
<a href="https://youtu.be/FMSJS57eO1w">https://youtu.be/FMSJS57eO1w</a>	Matrix	Rotate Image	<a href="https://leetcode">https://leetcode</a>	rotate layer-by-layer, use that it's a square as advantage, rotate positions in reverse order, store a in temp, a = b, b = c, c = d, d = temp;					
<a href="https://youtu.be/pfIq_PS1g8E">https://youtu.be/pfIq_PS1g8E</a>	Matrix	Word Search	<a href="https://leetcode">https://leetcode</a>	dfs on each cell, for each search remember visited cells, and remove cur visited cell right before you return from dfs;					
<a href="https://youtu.be/wlGQwVHdEO">https://youtu.be/wlGQwVHdEO</a>	String	Longest Substring Without Repeating	<a href="https://leetcode">https://leetcode</a>	sliding window, if we see same char twice within curr window, shift start position;					
<a href="https://youtu.be/gqXU1UyA8pk">https://youtu.be/gqXU1UyA8pk</a>	String	Longest Repeating Character Replace	<a href="https://leetcode">https://leetcode</a>	PAY ATTENTION: limited to chars A-Z; for each capital char, check if it could create the longest repeating substr, use sliding window to optimize; check if windowlen=1 works, if yes, increment len, if not, shift window right;					
<a href="https://youtu.be/Sto04AJbM">https://youtu.be/Sto04AJbM</a>	String	Minimum Window Substring	<a href="https://leetcode">https://leetcode</a>	need n num of unique char in T, HAVE is num of char we have valid count for, sliding window, move right until valid, if valid, increment left until invalid, to check validity keep track if the count of each unique char is satisfied;					
<a href="https://youtu.be/9UlitnBqnCGA">https://youtu.be/9UlitnBqnCGA</a>	String	Valid Anagram	<a href="https://leetcode">https://leetcode</a>	hashmap to count each char in str1, decrement for str2;					
<a href="https://youtu.be/vzdNOK2oB2E">https://youtu.be/vzdNOK2oB2E</a>	String	Group Anagrams	<a href="https://leetcode">https://leetcode</a>	for each of 26 chars, use count of each char in each word as tuple for key in dict, value is the list of anagrams;					
<a href="https://youtu.be/WT7iTs7kDFMg">https://youtu.be/WT7iTs7kDFMg</a>	String	Valid Parentheses	<a href="https://leetcode">https://leetcode</a>	push opening brace on stack, pop if matching close brace, at end if stack empty, return true;					
<a href="https://youtu.be/jXU16kPFWg">https://youtu.be/jXU16kPFWg</a>	String	Valid Palindrome	<a href="https://leetcode">https://leetcode</a>	left, right pointers, update left and right until each points at alphanum, compare left and right, continue until left >= right, don't distinguish between upper/lowercase;					
<a href="https://youtu.be/XYQecbdc6_c">https://youtu.be/XYQecbdc6_c</a>	String	Longest Palindromic Substring	<a href="https://leetcode">https://leetcode</a>	foreach char in str, consider if were the middle, consider if pali was odd or even;					
<a href="https://youtu.be/4RACAs5-du8">https://youtu.be/4RACAs5-du8</a>	String	Palindromic Substrings	<a href="https://leetcode">https://leetcode</a>	same as longest palindromic string, each char in str as middle and expand outwards, do same for pali of even len; maybe read up on manachers alg					
<a href="https://youtu.be/81k_sxO5gv8">https://youtu.be/81k_sxO5gv8</a>	String	Encode and Decode Strings (Leetcode)	<a href="https://leetcode">https://leetcode</a>	store length of str before each string and delimiter like '#';					
<a href="https://youtu.be/hTM3phVl6YQ">https://youtu.be/hTM3phVl6YQ</a>	Tree	Maximum Depth of Binary Tree	<a href="https://leetcode">https://leetcode</a>	recursive dfs to find max-depth of subtrees; iterative bfs to count number of levels in tree					
<a href="https://youtu.be/v8bbckKxCwO">https://youtu.be/v8bbckKxCwO</a>	Tree	Same Tree	<a href="https://leetcode">https://leetcode</a>	recursive dfs on both trees at the same time; iterative bfs compare each level of both trees					
<a href="https://youtu.be/OnSn2XEQ4MY">https://youtu.be/OnSn2XEQ4MY</a>	Tree	Invert/Flip Binary Tree	<a href="https://leetcode">https://leetcode</a>	recursive dfs to invert subtrees; bfs to invert levels, use collections.deque; iterative dfs is easy with stack if doing pre-order traversal					
<a href="https://youtu.be/Hr5cWUld4vU">https://youtu.be/Hr5cWUld4vU</a>	Tree	Binary Tree Maximum Path Sum	<a href="https://leetcode">https://leetcode</a>	helper returns maxpathsum without splitting branches, inside helper we also update maxSum by computing maxpathsum WITH a split;					
<a href="https://youtu.be/6ZnyEAppFYg">https://youtu.be/6ZnyEAppFYg</a>	Tree	Binary Tree Level Order Traversal	<a href="https://leetcode">https://leetcode</a>	iterative bfs, add prev level which doesn't have any nulls to the result;					
<a href="https://youtu.be/u4IAI2JJIH8">https://youtu.be/u4IAI2JJIH8</a>	Tree	Serialize and Deserialize Binary Tree	<a href="https://leetcode">https://leetcode</a>	bfs every single non-null node is added to string, and it's children are added too, even if they're null, deserialize by adding each non-null node to queue, deque node, it's children are next two nodes in string;					
<a href="https://youtu.be/E36O5SWp-LE">https://youtu.be/E36O5SWp-LE</a>	Tree	Subtree of Another Tree	<a href="https://leetcode">https://leetcode</a>	traverse s to check if any subtree in s equals t; merkle hashing?					
<a href="https://youtu.be/lhJlQGZ2zc">https://youtu.be/lhJlQGZ2zc</a>	Tree	Construct Binary Tree from Preorder	<a href="https://leetcode">https://leetcode</a>	first element in pre-order is root, elements left of root in in-order are left subtree, right of root are right subtree, recursively build subtrees;					
<a href="https://youtu.be/s6ATEkipzow">https://youtu.be/s6ATEkipzow</a>	Tree	Validate Binary Search Tree	<a href="https://leetcode">https://leetcode</a>	trick is use built in python min/max values float("inf"), "-inf", as parameters; iterative in-order traversal, check each val is greater than prev;					

<a href="https://youtu.be/5LUXSvjmGCw">https://youtu.be/5LUXSvjmGCw</a>	Tree	Kth Smallest Element in a BST	<a href="https://leetcode">https://leetcode</a>	non-optimal store tree in sorted array; iterative dfs in-order and return the kth element processed, go left until null, pop, go right once;					
<a href="https://youtu.be/gs2LMfuOR9k">https://youtu.be/gs2LMfuOR9k</a>	Tree	Lowest Common Ancestor of BST	<a href="https://leetcode">https://leetcode</a>	compare p, q values to curr node, base case: one is in left, other in right subtree, then curr is lca;					
<a href="https://youtu.be/oobqoCJIHAQ">https://youtu.be/oobqoCJIHAQ</a>	Tree	Implement Trie (Prefix Tree)	<a href="https://leetcode">https://leetcode</a>	node has children characters, and bool if its an ending character, node DOESN'T have or need char, since root node doesn't have a char, only children;					
<a href="https://youtu.be/BTT0Sgs_8IU">https://youtu.be/BTT0Sgs_8IU</a>	Tree	Add and Search Word	<a href="https://leetcode">https://leetcode</a>	if char = ".", run search for remaining portion of word on all of curr nodes children;					
<a href="https://youtu.be/asbcE9mZr_U">https://youtu.be/asbcE9mZr_U</a>	Tree	Word Search II	<a href="https://leetcode">https://leetcode</a>	trick: I though use trie to store the grid, reverse thinking, instead store dictionary words, dfs on each cell, check if cell's char exists as child of root node in trie, if it does, update currNode, and check neighbors, a word could exist multiple times in grid, so don't add duplicates;					
<a href="https://youtu.be/q5a5OIGbT6Q">https://youtu.be/q5a5OIGbT6Q</a>	Heap	Merge K Sorted Lists	<a href="https://leetcode">https://leetcode</a>	we always want the min of the current frontier, we can store frontier in heap of size k for efficient pop/push; divide and conquer merging lists;					
<a href="https://youtu.be/YPTqKigVk-k">https://youtu.be/YPTqKigVk-k</a>	Heap	Top K Frequent Elements	<a href="https://leetcode">https://leetcode</a>	minheap that's kept at size k, if its bigger than k pop the min, by the end it should be left with k largest;					
<a href="https://youtu.be/itmhHWaHupI">https://youtu.be/itmhHWaHupI</a>	Heap	Find Median from Data Stream	<a href="https://leetcode">https://leetcode</a>	maintain curr median, and all num greater than med in a minHeap, and all num less than med in a maxHeap, after every insertion update median depending on odd/even num of elements;					