S.No.	Name of the Program	Program Description

101.	**Symmetric Tree**	Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).
		For example, this binary tree [1,2,2,3,4,4,3] is symmetric:
		    1
		   / \\
		  2 2
		 / \\ / \\
		3 4 4 3
		 But the following [1,2,2,null,3,null,3] is not:
		    1
		   / \\
		  2 2
		   \\ \\
		   3 3
		Note:
		Bonus points if you could solve it both recursively and iteratively.

102.	**Binary Tree Level Order Traversal**	Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).
		For example:
		Given binary tree [3,9,20,null,null,15,7],
		    3
		   / \\
		  9 20
		    / \\
		   15 7
		return its level order traversal as:
		[
		  [3],
		  [9,20],
		  [15,7]
		]

103.	**Binary Tree Zigzag Level Order Traversal**	Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).
		For example:
		Given binary tree [3,9,20,null,null,15,7],
		    3
		   / \\
		  9 20
		    / \\
		   15 7
		return its zigzag level order traversal as:
		[
		  [3],
		  [20,9],
		  [15,7]
		]

104.	**Maximum Depth of Binary Tree**	Given a binary tree, find its maximum depth.
		The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.
		Note: A leaf is a node with no children.
		Example:
		Given binary tree [3,9,20,null,null,15,7],
		    3
		   / \\
		  9 20
		    / \\
		   15 7
		return its depth = 3.

105.	**Construct Binary Tree from Preorder and Inorder Traversal**	Given preorder and inorder traversal of a tree, construct the binary tree.
		Note:
		You may assume that duplicates do not exist in the tree.
		For example, given
		preorder = [3,9,20,15,7]
		inorder = [9,3,15,20,7]
		Return the following binary tree:
		    3
		   / \
		  9 20
		    / \
		   15 7

106.	**Construct Binary Tree from Inorder and Postorder Traversal**	Given inorder and postorder traversal of a tree, construct the binary tree.
		Note:
		You may assume that duplicates do not exist in the tree.
		For example, given
		inorder = [9,3,15,20,7]
		postorder = [9,15,7,20,3]
		Return the following binary tree:
		    3
		   / \
		  9 20
		    / \
		   15 7

107.	**Binary Tree Level Order Traversal II**	Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).
		For example:
		Given binary tree [3,9,20,null,null,15,7],
		    3
		   / \
		  9 20
		    / \
		   15 7
		return its bottom-up level order traversal as:
		[
		  [15,7],
		  [9,20],
		  [3]
		]

108.	**Convert Sorted Array to Binary Search Tree**	Given an array where elements are sorted in ascending order, convert it to a height balanced BST. For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1. **Example:** Given the sorted array: [-10,-3,0,5,9], One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST: 0 / \ -3 9 / / -10 5
109.	**Convert Sorted List to Binary Search Tree**	Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST. For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1. **Example:** Given the sorted linked list: [-10,-3,0,5,9], One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST: 0

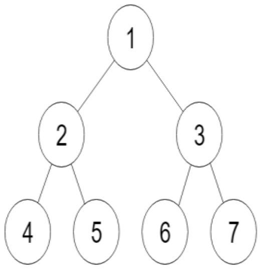
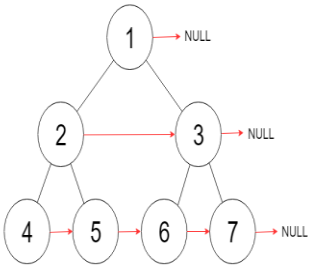
		/ \ -3 9 / / -10 5
110.	**Balanced Binary Tree**	Given a binary tree, determine if it is height-balanced. For this problem, a height-balanced binary tree is defined as: a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1. **Example 1:** Given the following tree [3,9,20,null,null,15,7]: 3 / \ 9 20 / \ 15 7 Return true. **Example 2:** Given the following tree [1,2,2,3,3,null,null,4,4]: 1 / \ 2 2 / \ 3 3 / \ 4 4 Return false.
111.	**Minimum Depth of Binary Tree**	Given a binary tree, find its minimum depth. The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node. **Note:** A leaf is a node with no children. **Example:** Given binary tree [3,9,20,null,null,15,7], 3 / \ 9 20 / \ 15 7 return its minimum depth = 2.
112.	**Path Sum**	Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. **Note:** A leaf is a node with no children. **Example:** Given the below binary tree and sum = 22, **5** **/** \ **4** 8 **/** / \ **11** 13 4 / \ \ 7 **2** 1 return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.
113.	**Path Sum II**	Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum. **Note:** A leaf is a node with no children. **Example:** Given the below binary tree and sum = 22, **5** **/ ** **4** **8** **/** / \ **11** 13 **4**

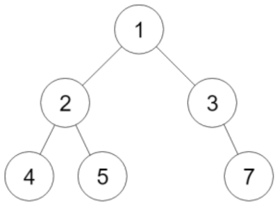
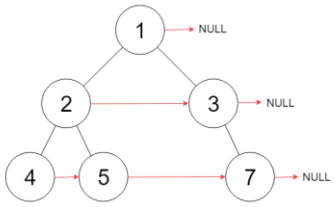
		/ \ / \ 7 **2** **5** 1 Return: [[5,4,11,2], [5,8,4,5]]

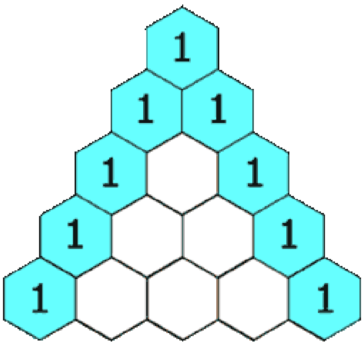
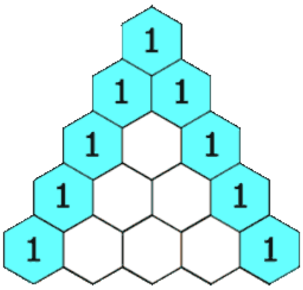
114.	**Flatten Binary Tree to Linked List**	Given a binary tree, flatten it to a linked list in-place.
		For example, given the following tree:
		    1
		   / \
		  2 5
		 / \ \
		3 4 6
		The flattened tree should look like:
		1
		 \
		  2
		   \
		    3
		     \
		      4
		       \
		        5
		         \
		          6

115.	**Distinct Subsequences**	Given a string **S** and a string **T**, count the number of distinct subsequences of **S** which equals **T**. A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not). **Example 1:** **Input:** S = "rabbbit", T = "rabbit" **Output:** 3 **Explanation:** As shown below, there are 3 ways you can generate "rabbit" from S. (The caret symbol ^ means the chosen letters) rabbbit ^^^^ ^^ rabbbit ^^ ^^^^ rabbbit ^^^ ^^^ **Example 2:** **Input:** S = "babgbag", T = "bag" **Output:** 5 **Explanation:** As shown below, there are 5 ways you can generate "bag" from S. (The caret symbol ^ means the chosen letters) babgbag ^^ ^ babgbag ^^ ^ babgbag ^ ^^ babgbag ^ ^^ babgbag

		^^^

116.	**Populating Next Right Pointers in Each Node**	You are given a perfect binary tree where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:
		struct Node {
		  int val;
		  Node *left;
		  Node *right;
		  Node *next;
		}
		Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.
		Initially, all next pointers are set to NULL.
		Example:
		
		Figure A
		
		Figure B
		Input:
		{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,"right":null,"val":4},"next":null,"right":{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"right":{"$id":"5","left":{"$id":"6","left":null,"next":null,"right":null,"val":6},"next":null,"right":{"$id":"7","left":null,"next":null,"right":null,"val":7},"val":3},"val":1}
		Output:
		{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":{"$id":"6","left":null,"next":null,"right":null,"val":7},"right":null,"val":6},"right":null,"val":5},"right":null,"val":4},"next":{"$id":"7","left":{"$ref":"5"},"next":null,"right":{"$ref":"6"},"val":3},"right":{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"7"},"val":1}
		Explanation: Given the above perfect binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B.
		Note:
		You may only use constant extra space.
		Recursive approach is fine, implicit stack space does not count as extra space for this problem.

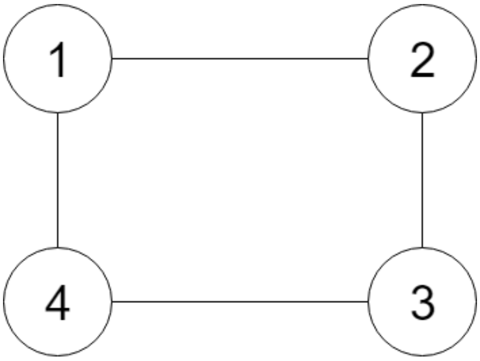
117.	**Populating Next Right Pointers in Each Node II**	Given a binary tree
		struct Node {
		  int val;
		  Node *left;
		  Node *right;
		  Node *next;
		}
		Populate each next pointer to point to its next right node. If there is no next right node, the next pointer should be set to NULL.
		Initially, all next pointers are set to NULL.
		
		Example:
		
		Figure A
		
		Figure B
		Input:
		{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":null,"right":null,"val":4},"next":null,"right":{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"right":{"$id":"5","left":null,"n

		ext":null,"right":{"$id":"6","left":null,"next":null,"right":null,"val":7},"val":3},"val":1} Output: {"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":null,"right":null,"val":7},"right":null,"val":5},"right":null,"val":4},"next":{"$id":"6","left":null,"next":null,"right":{"$ref":"5"},"val":3},"right":{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"6"},"val":1} Explanation: Given the above binary tree (Figure A), your function should populate each next pointer to point to its next right node, just like in Figure B. Note: You may only use constant extra space. Recursive approach is fine, implicit stack space does not count as extra space for this problem.
118.	**Pascal's Triangle**	Given a non-negative integer *numRows*, generate the first *numRows* of Pascal's triangle.  In Pascal's triangle, each number is the sum of the two numbers directly above it. **Example:** **Input:** 5 **Output:** [[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1]]
119.	**Pascal's Triangle II**	Given a non-negative index *k* where *k* ≤ 33, return the *k*th index row of the Pascal's triangle. Note that the row index starts from 0.  In Pascal's triangle, each number is the sum of the two numbers directly above it. **Example:** **Input:** 3 **Output:** [1,3,3,1] **Follow up:** Could you optimize your algorithm to use only *O*(*k*) extra space?
120.	**Triangle**	Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below. For example, given the following triangle [[**2**], [**3**,4], [6,**5**,7], [4,**1**,8,3]]

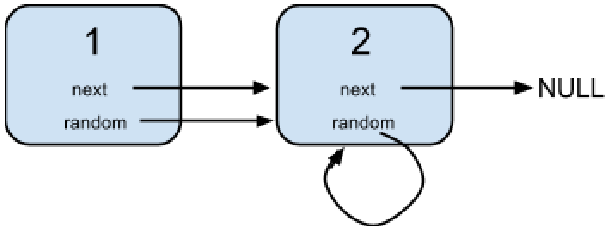
		The minimum path sum from top to bottom is 11 (i.e., **2** + **3** + **5** + **1** = 11). **Note:** Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.
121.	**Best Time to Buy and Sell Stock**	Say you have an array for which the *i*th element is the price of a given stock on day *i*. If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit. Note that you cannot sell a stock before you buy one. **Example 1:** **Input:** [7,1,5,3,6,4] **Output:** 5 **Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5. Not 7-1 = 6, as selling price needs to be larger than buying price. **Example 2:** **Input:** [7,6,4,3,1] **Output:** 0 **Explanation:** In this case, no transaction is done, i.e. max profit = 0.
122.	**Best Time to Buy and Sell Stock II**	Say you have an array for which the *i*th element is the price of a given stock on day *i*. Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times). **Note:** You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again). **Example 1:** **Input:** [7,1,5,3,6,4] **Output:** 7 **Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4. Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3. **Example 2:** **Input:** [1,2,3,4,5] **Output:** 4 **Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are engaging multiple transactions at the same time. You must sell before buying again. **Example 3:** **Input:** [7,6,4,3,1] **Output:** 0 **Explanation:** In this case, no transaction is done, i.e. max profit = 0.
123.	**Best Time to Buy and Sell Stock III**	Say you have an array for which the *i*th element is the price of a given stock on day *i*. Design an algorithm to find the maximum profit. You may complete at most *two* transactions. **Note:** You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again). **Example 1:** **Input:** [3,3,5,0,0,3,1,4] **Output:** 6 **Explanation:** Buy on day 4 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3. Then buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3. **Example 2:** **Input:** [1,2,3,4,5] **Output:** 4 **Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4. Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are engaging multiple transactions at the same time. You must sell before buying again. **Example 3:** **Input:** [7,6,4,3,1] **Output:** 0 **Explanation:** In this case, no transaction is done, i.e. max profit = 0.
124.	**Binary Tree Maximum Path Sum**	Given a **non-empty** binary tree, find the maximum path sum. For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go

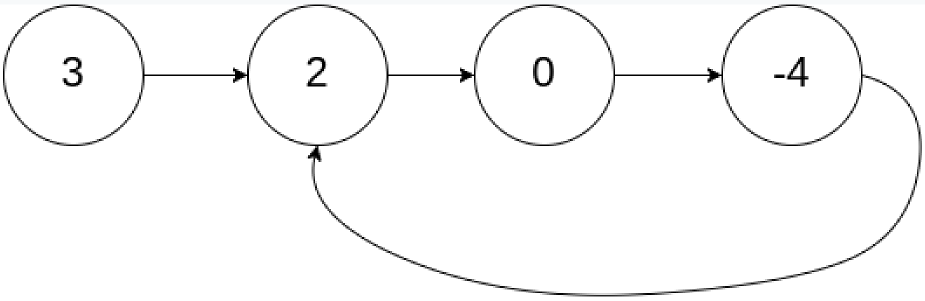
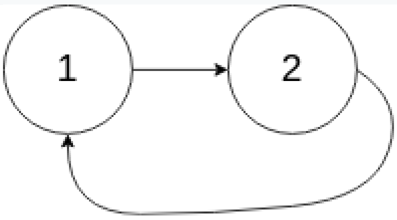
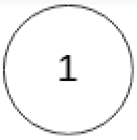
		through the root. **Example 1:** **Input:** [1,2,3] **1** **/ ** **2 3** **Output:** 6 **Example 2:** **Input:** [-10,9,20,null,null,15,7] -10 / \ 9 **20** **/ ** **15 7** **Output:** 42
125.	**Valid Palindrome**	Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases. **Note:** For the purpose of this problem, we define empty string as valid palindrome. **Example 1:** **Input:** "A man, a plan, a canal: Panama" **Output:** true **Example 2:** **Input:** "race a car" **Output:** false
126.	**Word Ladder II**	Given two words (beginWord and endWord), and a dictionary's word list, find all shortest transformation sequence(s) from beginWord to endWord, such that: Only one letter can be changed at a time Each transformed word must exist in the word list. Note that beginWord is not a transformed word. Note: Return an empty list if there is no such transformation sequence. All words have the same length. All words contain only lowercase alphabetic characters. You may assume no duplicates in the word list. You may assume beginWord and endWord are non-empty and are not the same. Example 1: Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"] Output: [["hit","hot","dot","dog","cog"], ["hit","hot","lot","log","cog"]] Example 2: Input: beginWord = "hit" endWord = "cog" wordList = ["hot","dot","dog","lot","log"] Output: [] Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.
127.	**Word Ladder**	Given two words (beginWord and endWord), and a dictionary's word list, find the length of shortest transformation sequence from beginWord to endWord, such that: Only one letter can be changed at a time.

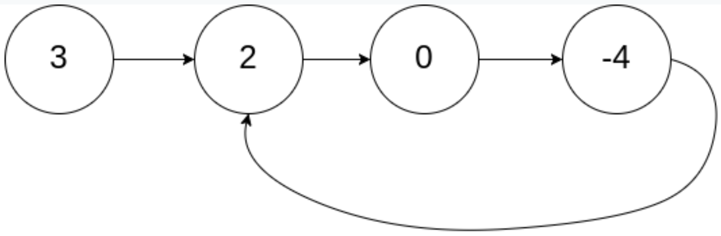
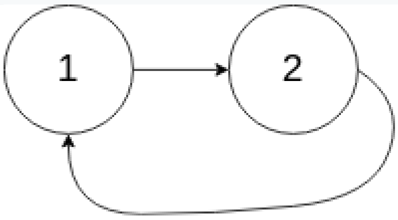
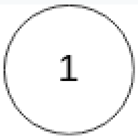
		Each transformed word must exist in the word list. Note that beginWord is not a transformed word. Note: Return 0 if there is no such transformation sequence. All words have the same length. All words contain only lowercase alphabetic characters. You may assume no duplicates in the word list. You may assume beginWord and endWord are non-empty and are not the same. Example 1: Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"] Output: 5 Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5. Example 2: Input: beginWord = "hit" endWord = "cog" wordList = ["hot","dot","dog","lot","log"] Output: 0 Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.
128.	**Longest Consecutive Sequence**	Given an unsorted array of integers, find the length of the longest consecutive elements sequence. Your algorithm should run in O(*n*) complexity. **Example:** **Input:** [100, 4, 200, 1, 3, 2] **Output:** 4 **Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.
129.	**Sum Root to Leaf Numbers**	Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number. An example is the root-to-leaf path 1->2->3 which represents the number 123. Find the total sum of all root-to-leaf numbers. **Note:** A leaf is a node with no children. **Example:** **Input:** [1,2,3] 1 / \\ 2 3 **Output:** 25 **Explanation:** The root-to-leaf path 1->2 represents the number 12. The root-to-leaf path 1->3 represents the number 13. Therefore, sum = 12 + 13 = 25. **Example 2:** **Input:** [4,9,0,5,1] 4 / \\ 9 0 / \\ 5 1 **Output:** 1026 **Explanation:** The root-to-leaf path 4->9->5 represents the number 495. The root-to-leaf path 4->9->1 represents the number 491. The root-to-leaf path 4->0 represents the number 40. Therefore, sum = 495 + 491 + 40 = 1026.
130.	**Surrounded Regions**	Given a 2D board containing 'X' and 'O' (**the letter O**), capture all regions surrounded by 'X'. A region is captured by flipping all 'O's into 'X's in that surrounded region.

		Example: X X X X X O O X X X O X X O X X After running your function, the board should be: X X X X X X X X X X X X X O X X **Explanation:** Surrounded regions shouldn't be on the border, which means that any 'O' on the border of the board are not flipped to 'X'. Any 'O' that is not on the border and it is not connected to an 'O' on the border will be flipped to 'X'. Two cells are connected if they are adjacent cells connected horizontally or vertically.
131.	**Palindrome Partitioning**	Given a string *s*, partition *s* such that every substring of the partition is a palindrome. Return all possible palindrome partitioning of *s*. **Example:** **Input:** "aab" **Output:** [["aa","b"], ["a","a","b"]]
132.	**Palindrome Partitioning II**	Given a string *s*, partition *s* such that every substring of the partition is a palindrome. Return the minimum cuts needed for a palindrome partitioning of *s*. **Example:** **Input:** "aab" **Output:** 1 **Explanation:** The palindrome partitioning ["aa","b"] could be produced using 1 cut.
133.	**Clone Graph**	Given a reference of a node in a connected undirected graph, return a **deep copy** (clone) of the graph. Each node in the graph contains a val (int) and a list (List[Node]) of its neighbors. **Example:**  **Input:** {"$id":"1","neighbors":[{"$id":"2","neighbors":[{"$ref":"1"},{"$id":"3","neighbors":[{"$ref":"2"},{"$id":"4","neighbors":[{"$ref":"3"},{"$ref":"1"}],"val":4}],"val":3}],"val":2},{"$ref":"4"}],"val":1} **Explanation:** Node 1's value is 1, and it has two neighbors: Node 2 and 4. Node 2's value is 2, and it has two neighbors: Node 1 and 3. Node 3's value is 3, and it has two neighbors: Node 2 and 4. Node 4's value is 4, and it has two neighbors: Node 1 and 3. **Note:** 1. The number of nodes will be between 1 and 100. 2. The undirected graph is a simple graph, which means no repeated edges and no self-loops in the graph. 3. Since the graph is undirected, if node *p* has node *q* as neighbor, then node *q* must have node *p* as neighbor too. 4. You must return the **copy of the given node** as a reference to the cloned graph.

134.	**Gas Station**	There are N gas stations along a circular route, where the amount of gas at station i is gas[i]. You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations. Return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1. Note: If there exists a solution, it is guaranteed to be unique. Both input arrays are non-empty and have the same length. Each element in the input arrays is a non-negative integer. Example 1: Input: gas = [1,2,3,4,5] cost = [3,4,5,1,2] Output: 3 Explanation: Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank = 0 + 4 = 4 Travel to station 4. Your tank = 4 - 1 + 5 = 8 Travel to station 0. Your tank = 8 - 2 + 1 = 7 Travel to station 1. Your tank = 7 - 3 + 2 = 6 Travel to station 2. Your tank = 6 - 4 + 3 = 5 Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3. Therefore, return 3 as the starting index. Example 2: Input: gas = [2,3,4] cost = [3,4,3] Output: -1 Explanation: You can't start at station 0 or 1, as there is not enough gas to travel to the next station. Let's start at station 2 and fill up with 4 unit of gas. Your tank = 0 + 4 = 4 Travel to station 0. Your tank = 4 - 3 + 2 = 3 Travel to station 1. Your tank = 3 - 3 + 3 = 3 You cannot travel back to station 2, as it requires 4 unit of gas but you only have 3. Therefore, you can't travel around the circuit once no matter where you start.
135.	**Candy**	There are N children standing in a line. Each child is assigned a rating value. You are giving candies to these children subjected to the following requirements: Each child must have at least one candy. Children with a higher rating get more candies than their neighbors. What is the minimum candies you must give? Example 1: Input: [1,0,2] Output: 5 Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively. Example 2: Input: [1,2,2] Output: 4 Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively. The third child gets 1 candy because it satisfies the above two conditions.
136.	**Single Number**	Given a **non-empty** array of integers, every element appears *twice* except for one. Find that single one. **Note:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory? **Example 1:** **Input:** [2,2,1] **Output:** 1 **Example 2:** **Input:** [4,1,2,1,2] **Output:** 4

137.	**Single Number II**	Given a **non-empty** array of integers, every element appears *three* times except for one, which appears exactly once. Find that single one. **Note:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory? **Example 1:** **Input:** [2,2,3,2] **Output:** 3 **Example 2:** **Input:** [0,1,0,1,0,1,99] **Output:** 99
138.	**Copy List with Random Pointer**	A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null. Return a deep copy of the list. Example 1:  Input: {"$id":"1","next":{"$id":"2","next":null,"random":{"$ref":"2"},"val":2},"random":{"$ref":"2"},"val":1} Explanation: Node 1's value is 1, both of its next and random pointer points to Node 2. Node 2's value is 2, its next pointer points to null and its random pointer points to itself. Note: You must return the copy of the given head as a reference to the cloned list.
139.	**Word Break**	Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words. Note: The same word in the dictionary may be reused multiple times in the segmentation. You may assume the dictionary does not contain duplicate words. Example 1: Input: s = "leetcode", wordDict = ["leet", "code"] Output: true Explanation: Return true because "leetcode" can be segmented as "leet code". Example 2: Input: s = "applepenapple", wordDict = ["apple", "pen"] Output: true Explanation: Return true because "applepenapple" can be segmented as "apple pen apple". Note that you are allowed to reuse a dictionary word. Example 3: Input: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"] Output: false
140.	**Word Break II**	Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, add spaces in s to construct a sentence where each word is a valid dictionary word. Return all such possible sentences. Note: The same word in the dictionary may be reused multiple times in the segmentation. You may assume the dictionary does not contain duplicate words. Example 1: Input: s = "catsanddog" wordDict = ["cat", "cats", "and", "sand", "dog"] Output: ["cats and dog",

		"cat sand dog"] Example 2: Input: s = "pineapplepenapple" wordDict = ["apple", "pen", "applepen", "pine", "pineapple"] Output: ["pine apple pen apple", "pineapple pen apple", "pine applepen apple"] Explanation: Note that you are allowed to reuse a dictionary word. Example 3: Input: s = "catsandog" wordDict = ["cats", "dog", "sand", "and", "cat"] Output: []
141.	**Linked List Cycle**	Given a linked list, determine if it has a cycle in it. To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list. **Example 1:** **Input:** head = [3,2,0,-4], pos = 1 **Output:** true **Explanation:** There is a cycle in the linked list, where tail connects to the second node.  **Example 2:** **Input:** head = [1,2], pos = 0 **Output:** true **Explanation:** There is a cycle in the linked list, where tail connects to the first node.  **Example 3:** **Input:** head = [1], pos = -1 **Output:** false **Explanation:** There is no cycle in the linked list.  **Follow up:** Can you solve it using *O(1)* (i.e. constant) memory?
142.	**Linked List Cycle II**	Given a linked list, return the node where the cycle begins. If there is no cycle, return null. To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

		Note: Do not modify the linked list. **Example 1:** **Input:** head = [3,2,0,-4], pos = 1 **Output:** tail connects to node index 1 **Explanation:** There is a cycle in the linked list, where tail connects to the second node.  **Example 2:** **Input:** head = [1,2], pos = 0 **Output:** tail connects to node index 0 **Explanation:** There is a cycle in the linked list, where tail connects to the first node.  **Example 3:** **Input:** head = [1], pos = -1 **Output:** no cycle **Explanation:** There is no cycle in the linked list.  Follow-up: Can you solve it without using extra space?
143.	**Reorder List**	Given a singly linked list *L*: *L*0→*L*1→…→*Ln*-1→*L*n, reorder it to: *L*0→*Ln*→*L*1→*Ln*-1→*L*2→*Ln*-2→… You may **not** modify the values in the list's nodes, only nodes itself may be changed. **Example 1:** Given 1->2->3->4, reorder it to 1->4->2->3. **Example 2:** Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
144.	**Binary Tree Preorder Traversal**	Given a binary tree, return the *preorder* traversal of its nodes' values. **Example:** **Input:** [1,null,2,3] 1 \\ 2 / 3 **Output:** [1,2,3] **Follow up:** Recursive solution is trivial, could you do it iteratively?
145.	**Binary Tree Postorder Traversal**	Given a binary tree, return the *postorder* traversal of its nodes' values. **Example:** **Input:** [1,null,2,3] 1 \\ 2 / 3 **Output:** [3,2,1]

		Follow up: Recursive solution is trivial, could you do it iteratively?

146.	**LRU Cache**	Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and put.
		get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
		put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.
		The cache is initialized with a **positive** capacity.
		Follow up:
		Could you do both operations in O(1) time complexity?
		Example:
		LRUCache cache = new LRUCache(2 /* capacity */);
		cache.put(1, 1);
		cache.put(2, 2);
		cache.get(1); // returns 1
		cache.put(3, 3); // evicts key 2
		cache.get(2); // returns -1 (not found)
		cache.put(4, 4); // evicts key 1
		cache.get(1); // returns -1 (not found)
		cache.get(3); // returns 3
		cache.get(4); // returns 4

147.	**Insertion Sort List**	Sort a linked list using insertion sort. A graphical example of insertion sort. The partial sorted list (black) initially contains only the first element in the list. With each iteration one element (red) is removed from the input data and inserted in-place into the sorted list Algorithm of Insertion Sort: Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. Example 1: Input: 4->2->1->3 Output: 1->2->3->4 Example 2: Input: -1->5->3->4->0 Output: -1->0->3->4->5
148.	**Sort List**	Sort a linked list in $O(n \log n)$ time using constant space complexity. **Example 1:** **Input:** 4->2->1->3 **Output:** 1->2->3->4 **Example 2:** **Input:** -1->5->3->4->0 **Output:** -1->0->3->4->5
149.	**Max Points on a Line**	Given *n* points on a 2D plane, find the maximum number of points that lie on the same straight line. **Example 1:** **Input:** [[1,1],[2,2],[3,3]] **Output:** 3 **Explanation:** ^ | | o

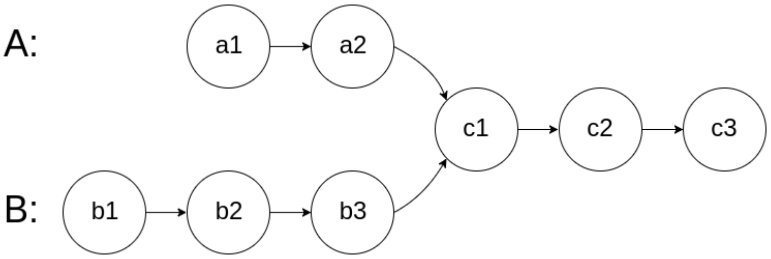
		| o | o +-------------> 0 1 2 3 4 **Example 2:** **Input:** [[1,1],[3,2],[5,3],[4,1],[2,3],[1,4]] **Output:** 4 **Explanation:** ^ | | o | o o | o | o o +-------------------> 0 1 2 3 4 5 6 **NOTE:** input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.
150.	**Evaluate Reverse Polish Notation**	Evaluate the value of an arithmetic expression in Reverse Polish Notation. Valid operators are +, -, *, /. Each operand may be an integer or another expression. **Note:** • Division between two integers should truncate toward zero. • The given RPN expression is always valid. That means the expression would always evaluate to a result and there won't be any divide by zero operation. **Example 1:** **Input:** ["2", "1", "+", "3", "*"] **Output:** 9 **Explanation:** ((2 + 1) * 3) = 9 **Example 2:** **Input:** ["4", "13", "5", "/", "+"] **Output:** 6 **Explanation:** (4 + (13 / 5)) = 6 **Example 3:** **Input:** ["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"] **Output:** 22 **Explanation:** ((10 * (6 / ((9 + 3) * -11))) + 17) + 5 = ((10 * (6 / (12 * -11))) + 17) + 5 = ((10 * (6 / -132)) + 17) + 5 = ((10 * 0) + 17) + 5 = (0 + 17) + 5 = 17 + 5 = 22
151.	**Reverse Words in a String**	Given an input string, reverse the string word by word. Example 1: Input: "the sky is blue" Output: "blue is sky the" Example 2: Input: " hello world! " Output: "world! hello" Explanation: Your reversed string should not contain leading or trailing spaces. Example 3: Input: "a good example" Output: "example good a" Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string. Note: A word is defined as a sequence of non-space characters.

		Input string may contain leading or trailing spaces. However, your reversed string should not contain leading or trailing spaces. You need to reduce multiple spaces between two words to a single space in the reversed string.
152.	**Maximum Product Subarray**	Given an integer array nums, find the contiguous subarray within an array (containing at least one number) which has the largest product. **Example 1:** **Input:** [2,3,-2,4] **Output:** 6 **Explanation:** [2,3] has the largest product 6. **Example 2:** **Input:** [-2,0,-1] **Output:** 0 **Explanation:** The result cannot be 2, because [-2,-1] is not a subarray.
153.	**Find Minimum in Rotated Sorted Array**	Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]). Find the minimum element. You may assume no duplicate exists in the array. **Example 1:** **Input:** [3,4,5,1,2] **Output:** 1 **Example 2:** **Input:** [4,5,6,7,0,1,2] **Output:** 0
154.	**Find Minimum in Rotated Sorted Array II**	Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand. (i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]). Find the minimum element. The array may contain duplicates. **Example 1:** **Input:** [1,3,5] **Output:** 1 **Example 2:** **Input:** [2,2,2,0,1] **Output:** 0 **Note:** • This is a follow up problem to Find Minimum in Rotated Sorted Array. • Would allow duplicates affect the run-time complexity? How and why?

155.	**Min Stack**	Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.
		push(x) -- Push element x onto stack.
		pop() -- Removes the element on top of the stack.
		top() -- Get the top element.
		getMin() -- Retrieve the minimum element in the stack.
		
		Example:
		MinStack minStack = new MinStack();
		minStack.push(-2);
		minStack.push(0);
		minStack.push(-3);
		minStack.getMin(); --> Returns -3.
		minStack.pop();
		minStack.top(); --> Returns 0.
		minStack.getMin(); --> Returns -2.

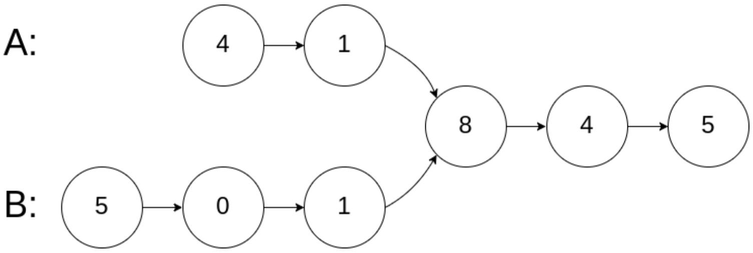
156.	Binary Tree upside down	
157.	Read N characters Given Read4	
158.	Read N characters	

<table>
<tr><td></td><td>Given Read4- multiple times</td><td></td></tr>
<tr><td>159.</td><td>Longest substring with at most two distinct characters</td><td></td></tr>
<tr><td>160.</td><td><b>Intersection of Two Linked Lists</b></td><td>
Write a program to find the node at which the intersection of two singly linked lists begins.
For example, the following two linked lists:



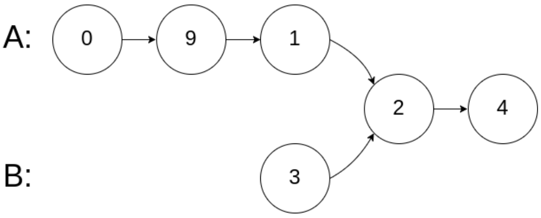
begin to intersect at node c1.

Example 1:



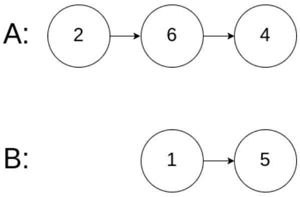
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3
Output: Reference of the node with value = 8
Input Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,0,1,8,4,5]. There are 2 nodes before the intersected node in A; There are 3 nodes before the intersected node in B.

Example 2:



Input: intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
Output: Reference of the node with value = 2
Input Explanation: The intersected node's value is 2 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [0,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1 node before the intersected node in B.

Example 3:

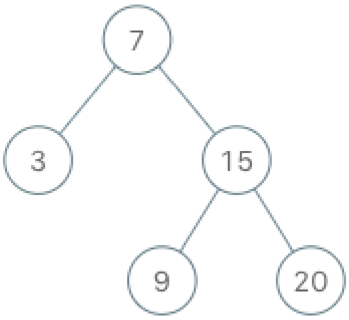


Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
Output: null
Input Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0, while skipA and skipB can be arbitrary values.
Explanation: The two lists do not intersect, so return null.
</td></tr>
</table>

		Notes: If the two linked lists have no intersection at all, return null. The linked lists must retain their original structure after the function returns. You may assume there are no cycles anywhere in the entire linked structure. Your code should preferably run in O(n) time and use only O(1) memory.
161.	One Edit Distance	
162.	**Find Peak Element**	A peak element is an element that is greater than its neighbors. Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index. The array may contain multiple peaks, in that case return the index to any one of the peaks is fine. You may imagine that nums[-1] = nums[n] = -∞. **Example 1:** **Input: nums** = [1,2,3,1] **Output:** 2 **Explanation:** 3 is a peak element and your function should return the index number 2. **Example 2:** **Input: nums** = [1,2,1,3,5,6,4] **Output:** 1 or 5 **Explanation:** Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6. **Note:** Your solution should be in logarithmic complexity.
163.	Missing Ranges	
164.	**Maximum Gap**	Given an unsorted array, find the maximum difference between the successive elements in its sorted form. Return 0 if the array contains less than 2 elements. Example 1: Input: [3,6,9,1] Output: 3 Explanation: The sorted form of the array is [1,3,6,9], either (3,6) or (6,9) has the maximum difference 3. Example 2: Input: [10] Output: 0 Explanation: The array contains less than 2 elements, therefore return 0. Note: You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range. Try to solve it in linear time/space.
165.	**Compare Version Numbers**	Compare two version numbers version1 and version2. If version1 > version2 return 1; if version1 < version2 return -1;otherwise return 0. You may assume that the version strings are non-empty and contain only digits and the . character. The . character does not represent a decimal point and is used to separate number sequences. For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision. You may assume the default revision number for each level of a version number to be 0. For example, version number 3.4 has a revision number of 3 and 4 for its first and second level revision number. Its third and fourth level revision number are both 0. Example 1: Input: version1 = "0.1", version2 = "1.1" Output: -1 Example 2: Input: version1 = "1.0.1", version2 = "1" Output: 1 Example 3: Input: version1 = "7.5.2.4", version2 = "7.5.3" Output: -1 Example 4:

		Input: version1 = "1.01", version2 = "1.001" Output: 0 Explanation: Ignoring leading zeroes, both “01” and “001" represent the same number “1” Example 5: Input: version1 = "1.0", version2 = "1.0.0" Output: 0 Explanation: The first version number does not have a third level revision number, which means its third level revision number is default to "0" Note: Version strings are composed of numeric strings separated by dots . and this numeric strings may have leading zeroes. Version strings do not start or end with dots, and they will not be two consecutive dots.
166.	**Fraction to Recurring Decimal**	Given two integers representing the numerator and denominator of a fraction, return the fraction in string format. If the fractional part is repeating, enclose the repeating part in parentheses. **Example 1:** **Input:** numerator = 1, denominator = 2 **Output:** "0.5" **Example 2:** **Input:** numerator = 2, denominator = 1 **Output:** "2" **Example 3:** **Input:** numerator = 2, denominator = 3 **Output:** "0.(6)"
167.	**Two Sum II - Input array is sorted**	Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number. The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Note: Your returned answers (both index1 and index2) are not zero-based. You may assume that each input would have exactly one solution and you may not use the same element twice. Example: Input: numbers = [2,7,11,15], target = 9 Output: [1,2] Explanation: The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.
168.	**Excel Sheet Column Title**	Given a positive integer, return its corresponding column title as appear in an Excel sheet. For example: 1 -> A 2 -> B 3 -> C ... 26 -> Z 27 -> AA 28 -> AB ... **Example 1:** **Input:** 1 **Output:** "A" **Example 2:** **Input:** 28 **Output:** "AB" **Example 3:** **Input:** 701 **Output:** "ZY"
169.	**Majority Element**	Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times. You may assume that the array is non-empty and the majority element always exist in the array.

		Example 1: **Input:** [3,2,3] **Output:** 3 **Example 2:** **Input:** [2,2,1,1,1,2,2] **Output:** 2
170.	Two Sum III- Data Structure Design	
171.	**Excel Sheet Column Number**	Given a column title as appear in an Excel sheet, return its corresponding column number. For example: A -> 1 B -> 2 C -> 3 ... Z -> 26 AA -> 27 AB -> 28 ... **Example 1:** **Input:** "A" **Output:** 1 **Example 2:** **Input:** "AB" **Output:** 28 **Example 3:** **Input:** "ZY" **Output:** 701
172.	**Factorial Trailing Zeroes**	Given an integer n, return the number of trailing zeroes in n!. **Example 1:** **Input:** 3 **Output:** 0 **Explanation:** 3! = 6, no trailing zero. **Example 2:** **Input:** 5 **Output:** 1 **Explanation:** 5! = 120, one trailing zero. Note: Your solution should be in logarithmic time complexity.

173.	**Binary Search Tree Iterator**	Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.
		Calling next() will return the next smallest number in the BST.
		
		Example:
		
		BSTIterator iterator = new BSTIterator(root);
		iterator.next(); // return 3
		iterator.next(); // return 7
		iterator.hasNext(); // return true

		iterator.next(); // return 9
		iterator.hasNext(); // return true
		iterator.next(); // return 15
		iterator.hasNext(); // return true
		iterator.next(); // return 20
		iterator.hasNext(); // return false
		
		Note:
		next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.
		You may assume that next() call will always be valid, that is, there will be at least a next smallest number in the BST when next() is called.

174.	**Dungeon Game**	The demons had captured the princess (**P**) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (**K**) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess. The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately. Some of the rooms are guarded by demons, so the knight loses health (*negative* integers) upon entering these rooms; other rooms are either empty (*0's*) or contain magic orbs that increase the knight's health (*positive* integers). In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step. **Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.** For example, given the dungeon below, the initial health of the knight must be at least **7** if he follows the optimal path RIGHT-> RIGHT -> DOWN -> DOWN. -2 (K)-33-5-1011030-5 (P) **Note:** • The knight's health has no upper bound. • Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.
175.	**Combine Two Tables**	SQL Schema Table: Person +-------------+---------+ | Column Name | Type | +-------------+---------+ | PersonId | int | | FirstName | varchar | | LastName | varchar | +-------------+---------+ PersonId is the primary key column for this table. Table: Address +-------------+---------+ | Column Name | Type | +-------------+---------+ | AddressId | int | | PersonId | int | | City | varchar | | State | varchar |

```
+-------------+---------+
```

AddressId is the primary key column for this table.

Write a SQL query for a report that provides the following information for each person in the Person table, regardless if there is an address for each of those people:
FirstName, LastName, City, State

176. **Second Highest Salary**

Write a SQL query to get the second highest salary from the Employee table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the query should return 200 as the second highest salary. If there is no second highest salary, then the query should return null.

```
+---------------------+
| SecondHighestSalary |
+---------------------+
| 200                 |
+---------------------+
```

177. **Nth Highest Salary**

Write a SQL query to get the *n*th highest salary from the Employee table.

```
+----+--------+
| Id | Salary |
+----+--------+
| 1  | 100    |
| 2  | 200    |
| 3  | 300    |
+----+--------+
```

For example, given the above Employee table, the *n*th highest salary where *n* = 2 is 200. If there is no *n*th highest salary, then the query should return null.

```
+------------------------+
| getNthHighestSalary(2) |
+------------------------+
| 200                    |
+------------------------+
```

178. **Rank Scores**

Write a SQL query to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value. In other words, there should be no "holes" between ranks.

```
+----+-------+
| Id | Score |
+----+-------+
| 1  | 3.50  |
| 2  | 3.65  |
| 3  | 4.00  |
| 4  | 3.85  |
| 5  | 4.00  |
| 6  | 3.65  |
+----+-------+
```

For example, given the above Scores table, your query should generate the following report (order by highest score):

```
+-------+------+
| Score | Rank |
+-------+------+
| 4.00  | 1    |
| 4.00  | 1    |
| 3.85  | 2    |
| 3.65  | 3    |
| 3.65  | 3    |
```

<table>
<tr><td></td><td></td><td>| 3.50 | 4 |
+-------+------+</td></tr>
<tr><td>179.</td><td><b>Largest Number</b></td><td>Given a list of non negative integers, arrange them such that they form the largest number.
<b>Example 1:</b>
<b>Input:</b> [10,2]
<b>Output:</b> "210"
<b>Example 2:</b>
<b>Input:</b> [3,30,34,5,9]
<b>Output:</b> "9534330"
<b>Note:</b> The result may be very large, so you need to return a string instead of an integer.</td></tr>
<tr><td>180.</td><td><b>Consecutive Numbers</b></td><td>Write a SQL query to find all numbers that appear at least three times consecutively.
+----+-----+
| Id | Num |
+----+-----+
| 1 | 1 |
2	1
3	1
4	2
5	1
6	2
7	2 | +----+-----+ For example, given the above Logs table, 1 is the only number that appears consecutively for at least three times. +-----------------+ | ConsecutiveNums | +-----------------+ | 1 | +-----------------+</td></tr>
<tr><td>181.</td><td><b>Employees Earning More Than Their Managers</b></td><td>The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id. +----+-------+--------+-----------+ | Id	Name	Salary	ManagerId | +----+-------+--------+-----------+ | 1	Joe	70000	3
2	Henry	80000	4
3	Sam	60000	NULL
4	Max	90000	NULL	 +----+-------+--------+-----------+ Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager. +----------+ 	Employee	 +----------+ 	Joe	 +----------+</td></tr>
<tr><td>182.</td><td><b>Duplicate Emails</b></td><td>Write a SQL query to find all duplicate emails in a table named Person. +----+---------+ 	Id	Email	 +----+---------+ 	1	a@b.com
2	c@d.com
3	a@b.com | +----+---------+ For example, your query should return the following for the above table: +---------+ | Email | +---------+ | a@b.com |</td></tr>
</table>

<table>
<tr><td></td><td></td><td>+---------+
<b>Note</b>: All emails are in lowercase.</td></tr>
<tr><td>183.</td><td><b>Customers Who Never Order</b></td><td>Suppose that a website contains two tables, the Customers table and the Orders table. Write a SQL query to find all customers who never order anything.
Table: Customers.
+----+-------+
| Id | Name |
+----+-------+
| 1 | Joe |
2	Henry
3	Sam
4	Max	 +----+-------+ Table: Orders. +----+------------+ 	Id	CustomerId	 +----+------------+ 	1	3
2	1 | +----+------------+ Using the above tables as example, return the following: +-----------+ | Customers | +-----------+ | Henry
Max	 +-----------+</td></tr>
<tr><td>184.</td><td><b>Department Highest Salary</b></td><td>The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id. +----+-------+--------+--------------+ | Id	Name	Salary	DepartmentId | +----+-------+--------+--------------+ | 1	Joe	70000	1
2	Jim	90000	1
3	Henry	80000	2
4	Sam	60000	2
5	Max	90000	1 | +----+-------+--------+--------------+ The Department table holds all departments of the company. +----+----------+ | Id	Name | +----+----------+ | 1	IT
2	Sales | +----+----------+ Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter). +------------+----------+--------+ | Department	Employee	Salary | +------------+----------+--------+ | IT	Max	90000
IT	Jim	90000
Sales	Henry	80000	 +------------+----------+--------+ <b>Explanation:</b> Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.</td></tr>
<tr><td>185.</td><td><b>Department Top Three</b></td><td>The Employee table holds all employees. Every employee has an Id, and there is also a column for the department Id.</td></tr>
</table>

<table>
<tr><td></td><td><b>Salaries</b></td><td>+----+-------+--------+--------------+
| Id | Name | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1 | Joe | 85000 | 1 |
2	Henry	80000	2
3	Sam	60000	2
4	Max	90000	1
5	Janet	69000	1
6	Randy	85000	1
7	Will	70000	1 | +----+-------+--------+--------------+ The Department table holds all departments of the company. +----+----------+ | Id	Name | +----+----------+ | 1	IT
2	Sales | +----+----------+ Write a SQL query to find employees who earn the top three salaries in each of the department. For the above tables, your SQL query should return the following rows (order of rows does not matter). +------------+----------+--------+ | Department	Employee	Salary | +------------+----------+--------+ | IT	Max	90000
IT	Randy	85000
IT	Joe	85000
IT	Will	70000
Sales	Henry	80000
Sales	Sam	60000	 +------------+----------+--------+ <b>Explanation:</b> In IT department, Max earns the highest salary, both Randy and Joe earn the second highest salary, and Will earns the third highest salary. There are only two employees in the Sales department, Henry earns the highest salary while Sam earns the second highest salary.</td></tr>
<tr><td>186.</td><td><b>Reverse words in a given string</b></td><td></td></tr>
<tr><td>187.</td><td><b>Repeated DNA Sequences</b></td><td>All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.
Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.
<b>Example:</b>
<b>Input:</b> s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

<b>Output:</b> ["AAAAACCCCC", "CCCCCAAAAA"]</td></tr>
<tr><td>188.</td><td><b>Best Time to Buy and Sell Stock IV</b></td><td>Say you have an array for which the i-th element is the price of a given stock on day i.
Design an algorithm to find the maximum profit. You may complete at most k transactions.
Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
<b>Example 1:</b>
<b>Input:</b> [2,4,1], k = 2
<b>Output:</b> 2
<b>Explanation:</b> Buy on day 1 (price = 2) and sell on day 2 (price = 4), profit = 4-2 = 2.
<b>Example 2:</b>
<b>Input:</b> [3,2,6,5,0,3], k = 2
<b>Output:</b> 7
<b>Explanation:</b> Buy on day 2 (price = 2) and sell on day 3 (price = 6), profit = 6-2 = 4.</td></tr>
</table>

		Then buy on day 5 (price = 0) and sell on day 6 (price = 3), profit = 3-0 = 3.
189.	**Rotate Array**	Given an array, rotate the array to the right by k steps, where k is non-negative. Example 1: Input: [1,2,3,4,5,6,7] and k = 3 Output: [5,6,7,1,2,3,4] Explanation: rotate 1 steps to the right: [7,1,2,3,4,5,6] rotate 2 steps to the right: [6,7,1,2,3,4,5] rotate 3 steps to the right: [5,6,7,1,2,3,4] Example 2: Input: [-1,-100,3,99] and k = 2 Output: [3,99,-1,-100] Explanation: rotate 1 steps to the right: [99,-1,-100,3] rotate 2 steps to the right: [3,99,-1,-100] Note: Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem. Could you do it in-place with O(1) extra space?
190.	**Reverse Bits**	Reverse bits of a given 32 bits unsigned integer. **Example 1:** **Input:** 00000010100101000001111010011100 **Output:** 00111001011110000010100101000000 **Explanation:** The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000. **Example 2:** **Input:** 11111111111111111111111111111101 **Output:** 10111111111111111111111111111111 **Explanation:** The input binary string **11111111111111111111111111111101** represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is **10101111110010110010011101101001**. **Note:** • Note that in some languages such as Java, there is no unsigned integer type. In this case, both input and output will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned. • In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above the input represents the signed integer -3 and the output represents the signed integer -1073741825. Follow up: If this function is called many times, how would you optimize it?
191.	**Number of 1 Bits**	Write a function that takes an unsigned integer and return the number of '1' bits it has (also known as the Hamming weight). **Example 1:** **Input:** 00000000000000000000000000001011 **Output:** 3 **Explanation:** The input binary string **00000000000000000000000000001011** has a total of three '1' bits. **Example 2:** **Input:** 00000000000000000000000010000000 **Output:** 1 **Explanation:** The input binary string **00000000000000000000000010000000** has a total of one '1' bit. **Example 3:** **Input:** 11111111111111111111111111111101 **Output:** 31 **Explanation:** The input binary string **11111111111111111111111111111101** has a total of thirty one '1' bits. **Note:** • Note that in some languages such as Java, there is no unsigned integer type. In this case, the input

		will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned. • In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 3** above the input represents the signed integer -3. Follow up: If this function is called many times, how would you optimize it?
192.	**Word Frequency**	Write a bash script to calculate the frequency of each word in a text file words.txt. For simplicity sake, you may assume: • words.txt contains only lowercase characters and space ' ' characters. • Each word must consist of lowercase characters only. • Words are separated by one or more whitespace characters. **Example:** Assume that words.txt has the following content: the day is sunny the the the sunny is is Your script should output the following, sorted by descending frequency: the 4 is 3 sunny 2 day 1 Note: • Don't worry about handling ties, it is guaranteed that each word's frequency count is unique. • Could you write it in one-line using Unix pipes?
193.	**Valid Phone Numbers**	Given a text file file.txt that contains list of phone numbers (one per line), write a one liner bash script to print all valid phone numbers. You may assume that a valid phone number must appear in one of the following two formats: (xxx) xxx-xxxx or xxx-xxx-xxxx. (x means a digit) You may also assume each line in the text file must not contain leading or trailing white spaces. **Example:** Assume that file.txt has the following content: 987-123-4567 123 456 7890 (123) 456-7890 Your script should output the following valid phone numbers: 987-123-4567 (123) 456-7890
194.	**Transpose File**	Given a text file file.txt, transpose its content. You may assume that each row has the same number of columns and each field is separated by the ' ' character. **Example:** If file.txt has the following content: name age alice 21 ryan 30 Output the following: name alice ryan age 21 30
195.	**Tenth Line**	Given a text file file.txt, print just the 10th line of the file. Example: Assume that file.txt has the following content: Line 1 Line 2 Line 3 Line 4 Line 5 Line 6 Line 7 Line 8

<table>
<tr><td></td><td></td><td>Line 9
Line 10
Your script should output the tenth line, which is:
Line 10
Note:
1. If the file contains less than 10 lines, what should you output?
2. There's at least three different solutions. Try to explore all possibilities.</td></tr>
<tr><td>196.</td><td><b>Delete Duplicate Emails</b></td><td>Write a SQL query to <b>delete</b> all duplicate email entries in a table named Person, keeping only unique emails based on its smallest Id.
+----+------------------+
| Id | Email |
+----+------------------+
| 1 | john@example.com |
| 2 | bob@example.com |
| 3 | john@example.com |
+----+------------------+
Id is the primary key column for this table.
For example, after running your query, the above Person table should have the following rows:
+----+------------------+
| Id | Email |
+----+------------------+
| 1 | john@example.com |
| 2 | bob@example.com |
+----+------------------+
<b>Note:</b>
Your output is the whole Person table after executing your sql. Use delete statement.</td></tr>
<tr><td>197.</td><td><b>Rising Temperature</b></td><td>Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.
+---------+------------------+------------------+
| Id(INT) | RecordDate(DATE) | Temperature(INT) |
+---------+------------------+------------------+
| 1 | 2015-01-01 | 10 |
2	2015-01-02	25
3	2015-01-03	20
4	2015-01-04	30 | +---------+------------------+------------------+ For example, return the following Ids for the above Weather table: +----+ | Id | +----+ | 2
4	 +----+</td></tr>
<tr><td>198.</td><td><b>House Robber</b></td><td>You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.
Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
<b>Example 1:</b>
<b>Input:</b> [1,2,3,1]
<b>Output:</b> 4
<b>Explanation:</b> Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
<b>Example 2:</b>
<b>Input:</b> [2,7,9,3,1]
<b>Output:</b> 12
<b>Explanation:</b> Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
             Total amount you can rob = 2 + 9 + 1 = 12.</td></tr>
</table>

199.	**Binary Tree Right Side View**	Given a binary tree, imagine yourself standing on the *right* side of it, return the values of the nodes you can see ordered from top to bottom. **Example:** **Input:** [1,2,3,null,5,null,4] **Output:** [1, 3, 4] **Explanation:** 1 <--- / \\ 2 3 <--- \\ \\ 5 4 <---
200.	**Number of Islands**	Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water. Example 1: **Input:** 11110 11010 11000 00000 **Output:** 1 Example 2: **Input:** 11000 11000 00100 00011 **Output:** 3