**LeetCode Grind75 Questions**

**1. Two Sum (1st in LeetCode)**

1.Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

Example 3:

Input: nums = [3,3], target = 6

Output: [0,1]

Constraints:

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* Only one valid answer exists.

Follow-up: Can you come up with an algorithm that is less than O(n2) time complexity?

**2. Valid Parantheses (20 in LC)**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

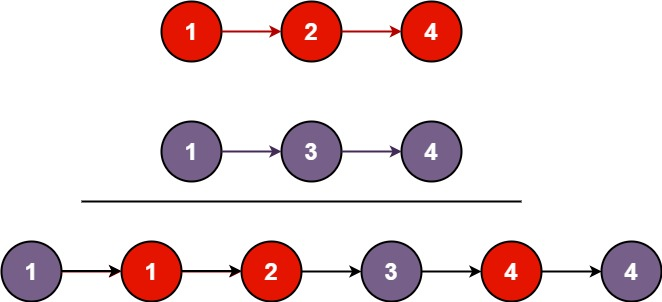
**3. Merge Two Sorted List (21 in LC)**

**You are given the heads of two sorted linked lists list1 and list2.**

Merge the two lists in a one sorted list. The list should be made by splicing together the nodes of the first two lists.

Return the head of the merged linked list.

Example 1:



Input: list1 = [1,2,4], list2 = [1,3,4]

Output:[1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []

Output: []

Example 3:

Input: list1 = [], list2 = [0]

Output: [0]

Constraints:

* The number of nodes in both lists is in the range [0, 50].
* -100 <= Node.val <= 100
* Both list1 and list2 are sorted in non-decreasing order.

**4. Best Time to Buy and Sell Stock (121 in LC)**

**You are given an array prices where prices[i] is the price of a given stock on the ith day.**

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [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.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

Constraints:

* 1 <= prices.length <= 105
* 0 <= prices[i] <= 104

**5. Valid Palindrome (125 in LC)**

A phrase is a palindromeif, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a palindrome, or false otherwise.

Example 1:

Input: s = "A man, a plan, a canal: Panama"

Output: true

Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: s = "race a car"

Output: false

Explanation: "raceacar" is not a palindrome.

Example 3:

Input: s = " "

Output: true

Explanation: s is an empty string "" after removing non-alphanumeric characters.

Since an empty string reads the same forward and backward, it is a palindrome.

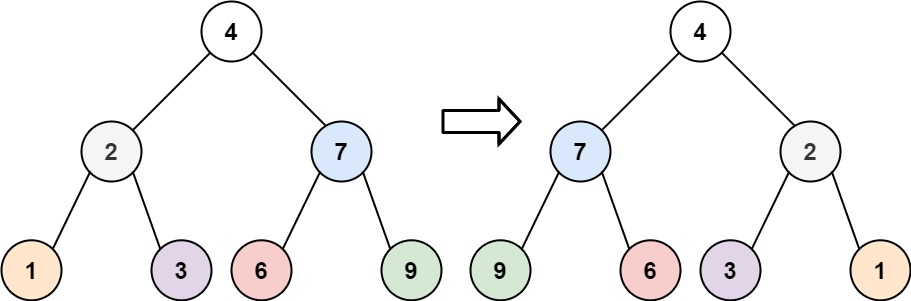
Constraints:

* 1 <= s.length <= 2 \* 105
* s consists only of printable ASCII characters.

**6. Invert Binary Tree (226 in LC)**

Given the root of a binary tree, invert the tree, and return its root.

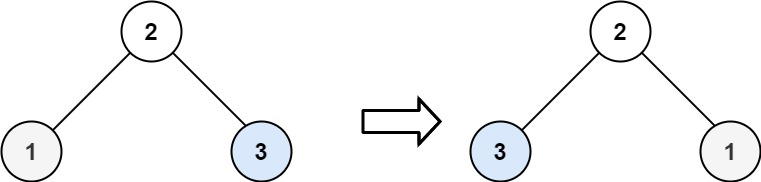
Example 1:

****

Input: root = [4,2,7,1,3,6,9]

Output: [4,7,2,9,6,3,1]

Example 2:

****

Input: root = [2,1,3]

Output: [2,3,1]

Example 3:

Input: root = []

Output: []

Constraints:

* The number of nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**7. Valid Anagram (242 in LC)**

**Given two strings s and t, return true if t is an anagram of s, and false otherwise.**

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = "rat", t = "car"

Output: false

Constraints:

* 1 <= s.length, t.length <= 5 \* 104
* s and t consist of lowercase English letters.

Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

**8. Binary Search (704 in LC)**

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Explanation: 9 exists in nums and its index is 4

Example 2:

Input: nums = [-1,0,3,5,9,12], target = 2

Output: -1

Explanation: 2 does not exist in nums so return -1

Constraints:

* 1 <= nums.length <= 104
* -104 < nums[i], target < 104
* All the integers in nums are unique.
* nums is sorted in ascending order.

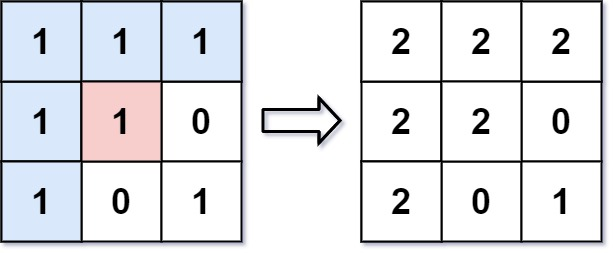
**9. Flood Fill (733 in LC)**

**An image is represented by an m x n integer grid image where image[i][j] represents the pixel value of the image.**

You are also given three integers sr, sc, and color. You should perform a flood fill on the image starting from the pixel image[sr][sc].

To perform a flood fill, consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color), and so on. Replace the color of all of the aforementioned pixels with color.

Return the modified image after performing the flood fill.

Example 1:

Input: image = [[1,1,1],[1,1,0],[1,0,1]], sr = 1, sc = 1, color = 2

Output: [[2,2,2],[2,2,0],[2,0,1]]

Explanation: From the center of the image with position (sr, sc) = (1, 1) (i.e., the red pixel), all pixels connected by a path of the same color as the starting pixel (i.e., the blue pixels) are colored with the new color.

Note the bottom corner is not colored 2, because it is not 4-directionally connected to the starting pixel.

Example 2:

Input: image = [[0,0,0],[0,0,0]], sr = 0, sc = 0, color = 0

Output: [[0,0,0],[0,0,0]]

Explanation: The starting pixel is already colored 0, so no changes are made to the image.

Constraints:

* m == image.length
* n == image[i].length
* 1 <= m, n <= 50
* 0 <= image[i][j], color < 216
* 0 <= sr < m
* 0 <= sc < n

**10. Lowest Common Ancestor of a Binary Search Tree (235 in LC)**

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the [definition of LCA on Wikipedia](https://en.wikipedia.org/wiki/Lowest_common_ancestor): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

Example 1:

****

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

Output: 6

Explanation: The LCA of nodes 2 and 8 is 6.

Example 2:

****

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

Output: 2

Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

Example 3:

Input: root = [2,1], p = 2, q = 1

Output: 2

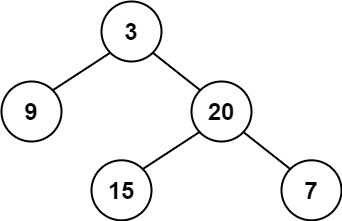
Constraints:

* The number of nodes in the tree is in the range [2, 105].
* -109 <= Node.val <= 109
* All Node.val are unique.
* p != q
* p and q will exist in the BST.

**11. Balanced Binary Tree (110 in LC)**

Given a binary tree, determine if it is height-balanced. A height-balanced binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

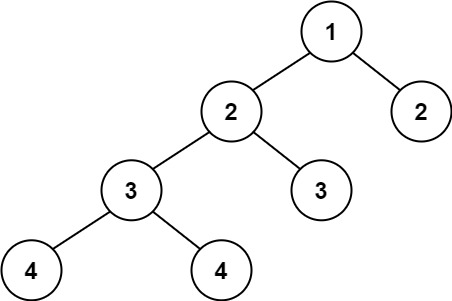
Example 1:

****

Input: root = [3,9,20,null,null,15,7]

Output: true

Example 2:

****

Input: root = [1,2,2,3,3,null,null,4,4]

Output: false

Example 3:

Input: root = []

Output: true

Constraints:

* The number of nodes in the tree is in the range [0, 5000].
* -104 <= Node.val <= 104

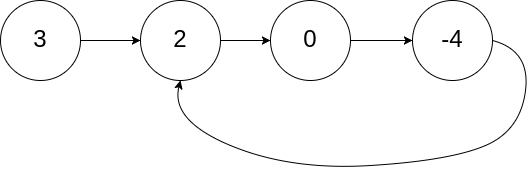
**12. Linked List Cycle (141 in LC)**

**Given head, the head of a linked list, determine if the linked list has a cycle in it.**

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to. Note that pos is not passed as a parameter.

Return true if there is a cycle in the linked list. Otherwise, return false.

Example 1:

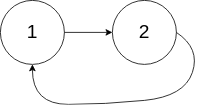
****

Input:head = [3,2,0,-4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:

****

Input: head = [1,2], pos = 0

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Example 3:

****

Input: head = [1], pos = -1

Output: false

Explanation: There is no cycle in the linked list.

Constraints:

* The number of the nodes in the list is in the range [0, 104].
* -105 <= Node.val <= 105
* pos is -1 or a valid index in the linked-list.

Follow up: Can you solve it using O(1) (i.e. constant) memory?

**13. Implement Queue using Stacks**

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

Implement the MyQueue class:

* void push(int x) Pushes element x to the back of the queue.
* int pop() Removes the element from the front of the queue and returns it.
* int peek() Returns the element at the front of the queue.
* boolean empty() Returns true if the queue is empty, false otherwise.

Notes:

* You must use only standard operations of a stack, which means only push to top, peek/pop from top, size, and is empty operations are valid.
* Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Example 1:

Input

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output

[null, null, null, 1, 1, false]

Explanation

MyQueue myQueue = new MyQueue();

myQueue.push(1); // queue is: [1]

myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)

myQueue.peek(); // return 1

myQueue.pop(); // return 1, queue is [2]

myQueue.empty(); // return false

Constraints:

* 1 <= x <= 9
* At most 100 calls will be made to push, pop, peek, and empty.
* All the calls to pop and peek are valid.

Follow-up: Can you implement the queue such that each operation is [**amortized**](https://en.wikipedia.org/wiki/Amortized_analysis) O(1) time complexity? In other words, performing n operations will take overall O(n) time even if one of those operations may take longer.