

Product of Array Except Self

Given an integer array `nums`, return *an array answer such that `answer[i]` is equal to the product of all the elements of `nums` except `nums[i]`*.

The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

You must write an algorithm that runs in $O(n)$ time and without using the division operation.

Example 1:

Input: `nums = [1,2,3,4]`

Output: `[24,12,8,6]`

Example 2:

Input: `nums = [-1,1,0,-3,3]`

Output: `[0,0,9,0,0]`

Constraints:

- $2 \leq \text{nums.length} \leq 10^5$
- $-30 \leq \text{nums}[i] \leq 30$
- The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

Follow up: Can you solve the problem in $O(1)$ extra space complexity? (The output array **does not** count as extra space for space complexity analysis.)

Maximum Subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

Example 1:

Input: `nums = [-2,1,-3,4,-1,2,1,-5,4]`

Output: 6

Explanation: `[4,-1,2,1]` has the largest sum = 6.

Example 2:

Input: `nums = [1]`

Output: 1

Example 3:

Input: `nums = [5,4,-1,7,8]`

Output: 23

Constraints:

- $1 \leq \text{nums.length} \leq 3 \times 10^4$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

Follow up: If you have figured out the $O(n)$ solution, try coding another solution using the **divide and conquer** approach, which is more subtle.

Maximum Product Subarray

Given an integer array `nums`, find a contiguous non-empty subarray within the array that has the largest product, and return *the product*.

It is **guaranteed** that the answer will fit in a **32-bit** integer.

A **subarray** is a contiguous subsequence of the array.

Example 1:

Input: `nums = [2,3,-2,4]`

Output: 6

Explanation: `[2,3]` has the largest product 6.

Example 2:

Input: `nums = [-2,0,-1]`

Output: 0

Explanation: The result cannot be 2, because `[-2,-1]` is not a subarray.

Constraints:

- $1 \leq \text{nums.length} \leq 2 * 10^4$
- $-10 \leq \text{nums}[i] \leq 10$
- The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

Find Minimum in Rotated Sorted Array

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array `nums = [0, 1, 2, 4, 5, 6, 7]` might become:

- `[4, 5, 6, 7, 0, 1, 2]` if it was rotated 4 times.
- `[0, 1, 2, 4, 5, 6, 7]` if it was rotated 7 times.

Notice that **rotating** an array `[a[0], a[1], a[2], ..., a[n-1]]` 1 time results in the array `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]`.

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array*.

You must write an algorithm that runs in $O(\log n)$ time.

Example 1:

Input: `nums = [3,4,5,1,2]`

Output: 1

Explanation: The original array was `[1,2,3,4,5]` rotated 3 times.

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`

Output: 0

Explanation: The original array was `[0,1,2,4,5,6,7]` and it was rotated 4 times.

Example 3:

Input: `nums = [11,13,15,17]`

Output: 11

Explanation: The original array was `[11,13,15,17]` and it was rotated 4 times.

Constraints:

- `n == nums.length`
- `1 <= n <= 5000`
- `-5000 <= nums[i] <= 5000`
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between 1 and `n` times.

Search in Rotated Sorted Array

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **rotated** at an unknown pivot index `k` ($0 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index 3 and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the rotation and an integer `target`, return *the index of target if it is in nums, or -1 if it is not in nums*.

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

Example 1:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 0`

Output: 4

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 3`

Output: -1

Example 3:

Input: `nums = [1]`, `target = 0`

Output: -1

Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- All values of `nums` are **unique**.

- `nums` is guaranteed to be rotated at some pivot.
- $-10^4 \leq \text{target} \leq 10^4$

3Sum

Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that $i \neq j$, $i \neq k$, and $j \neq k$, and $nums[i] + nums[j] + nums[k] == 0$.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: `nums = [-1,0,1,2,-1,-4]`

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

Example 2:

Input: `nums = []`

Output: `[]`

Example 3:

Input: `nums = [0]`

Output: `[]`

Constraints:

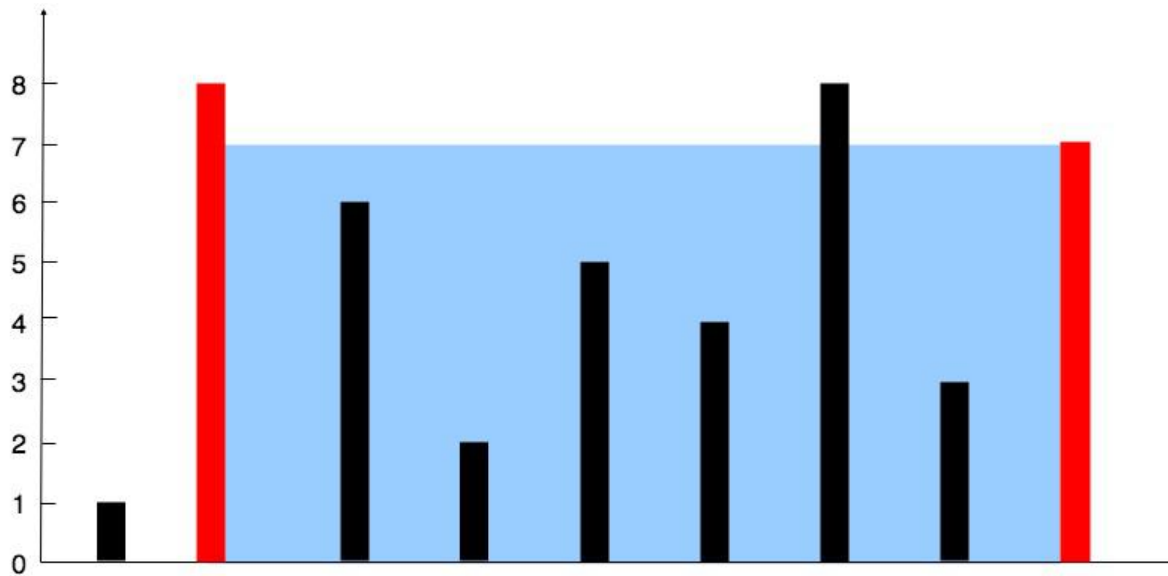
- $0 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

Container With Most Water

Given n non-negative integers a_1, a_2, \dots, a_n , where each represents a point at coordinate (i, a_i) . n vertical lines are drawn such that the two endpoints of the line i is at (i, a_i) and $(i, 0)$. Find two lines, which, together with the x-axis forms a container, such that the container contains the most water.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]
Output: 1

Example 3:

Input: height = [4,3,2,1,4]
Output: 16

Example 4:

Input: height = [1,2,1]
Output: 2

Constraints:

- $n == \text{height.length}$
- $2 \leq n \leq 10^5$
- $0 \leq \text{height}[i] \leq 10^4$

Binary

Sum of Two Integers

Given two integers a and b, return *the sum of the two integers without using the operators + and -*.

Example 1:

Input: a = 1, b = 2

Output: 3

Example 2:

Input: a = 2, b = 3

Output: 5

Constraints:

- $-1000 \leq a, b \leq 1000$

Number of 1 Bits

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the [Hamming weight](#)).

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation 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**, the input represents the signed integer. - 3.

Example 1:

Input: n = 000000000000000000000000000000001011

Output: 3

[illegible]

Example 2:

[illegible]

Output: 1

Explanation: The input binary string 000000000000000000000000010000000 has a total of one '1' bit.

Example 3:

Input: n = 111111111111111111111111111111101

Output: 31

[illegible]

Constraints:

- The input must be a **binary string** of length 32.

Follow up: If this function is called many times, how would you optimize it?

Counting Bits

Given an integer n , return an array `ans` of length $n + 1$ such that for each i ($0 \leq i \leq n$), `ans[i]` is the **number of 1's** in the binary representation of i .

Example 1:

Input: $n = 2$

Output: `[0,1,1]`

Explanation:

0 --> 0

1 --> 1

2 --> 10

Example 2:

Input: $n = 5$

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

Explanation:

0 --> 0

1 --> 1

2 --> 10

3 --> 11

4 --> 100

5 --> 101

Constraints:

- $0 \leq n \leq 10^5$

Follow up:

- It is very easy to come up with a solution with a runtime of $O(n \log n)$. Can you do it in linear time $O(n)$ and possibly in a single pass?
- Can you do it without using any built-in function (i.e., like `__builtin_popcount` in C++)?

Missing Number

Given an array `nums` containing n distinct numbers in the range $[0, n]$, return *the only number in the range that is missing from the array*.

Follow up: Could you implement a solution using only $O(1)$ extra space complexity and $O(n)$ runtime complexity?

Example 1:

Input: `nums = [3,0,1]`

Output: 2

Explanation: $n = 3$ since there are 3 numbers, so all numbers are in the range $[0,3]$. 2 is the missing number in the range since it does not appear in `nums`.

Example 2:

Input: `nums = [0,1]`

Output: 2

Explanation: $n = 2$ since there are 2 numbers, so all numbers are in the range $[0,2]$. 2 is the missing number in the range since it does not appear in `nums`.

Example 3:

Input: `nums = [9,6,4,2,3,5,7,0,1]`

Output: 8

Explanation: $n = 9$ since there are 9 numbers, so all numbers are in the range $[0,9]$. 8 is the missing number in the range since it does not appear in `nums`.

Example 4:

Input: `nums = [0]`

Output: 1

Explanation: $n = 1$ since there is 1 number, so all numbers are in the range $[0,1]$. 1 is the missing number in the range since it does not appear in `nums`.

Constraints:

- `n == nums.length`
- `1 <= n <= 104`
- `0 <= nums[i] <= n`
- All the numbers of `nums` are **unique**.

Reverse Bits

Reverse bits of a given 32 bits unsigned integer.

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 a signed integer type. They should not affect your implementation, as the integer's internal binary representation 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?

Example 1:

Input: n = 00000010100101000001111010011100

Output: 964176192 (00111001011110000010100101000000)

Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101

Output: 3221225471 (10111111111111111111111111111111)

Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is 10111111111111111111111111111111.

Constraints:

- The input must be a **binary string** of length 32

Dynamic Programming

Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: $n = 2$

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step
2. 2 steps

Example 2:

Input: $n = 3$

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

Constraints:

- $1 \leq n \leq 45$

Coin Change

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return `-1`.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input: `coins = [1,2,5]`, `amount = 11`

Output: `3`

Explanation: $11 = 5 + 5 + 1$

Example 2:

Input: `coins = [2]`, `amount = 3`

Output: `-1`

Example 3:

Input: `coins = [1]`, `amount = 0`

Output: `0`

Example 4:

Input: `coins = [1]`, `amount = 1`

Output: `1`

Example 5:

Input: coins = [1], amount = 2
Output: 2

Constraints:

- $1 \leq \text{coins.length} \leq 12$
- $1 \leq \text{coins}[i] \leq 2^{31} - 1$
- $0 \leq \text{amount} \leq 10^4$

Longest Increasing Subsequence

Given an integer array `nums`, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, `[3, 6, 2, 7]` is a subsequence of the array `[0, 3, 1, 6, 2, 2, 7]`.

Example 1:

Input: `nums = [10,9,2,5,3,7,101,18]`

Output: 4

Explanation: The longest increasing subsequence is `[2,3,7,101]`, therefore the length is 4.

Example 2:

Input: `nums = [0,1,0,3,2,3]`

Output: 4

Example 3:

Input: `nums = [7,7,7,7,7,7,7,7]`

Output: 1

Constraints:

- `1 <= nums.length <= 2500`
- `-104 <= nums[i] <= 104`

Follow up: Can you come up with an algorithm that runs in $O(n \log(n))$ time complexity?

Longest Common Subsequence

Given two strings `text1` and `text2`, return *the length of their longest **common subsequence***. If there is no **common subsequence**, return `0`.

A **subsequence** of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

- For example, "ace" is a subsequence of "abcde".

A **common subsequence** of two strings is a subsequence that is common to both strings.

Example 1:

Input: `text1 = "abcde"`, `text2 = "ace"`

Output: 3

Explanation: The longest common subsequence is "ace" and its length is 3.

Example 2:

Input: `text1 = "abc"`, `text2 = "abc"`

Output: 3

Explanation: The longest common subsequence is "abc" and its length is 3.

Example 3:

Input: `text1 = "abc"`, `text2 = "def"`

Output: 0

Explanation: There is no such common subsequence, so the result is 0.

Constraints:

- `1 <= text1.length, text2.length <= 1000`
- `text1` and `text2` consist of only lowercase English characters.

Word Break

Given a string `s` and a dictionary of strings `wordDict`, return `true` if `s` can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

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`

Constraints:

- `1 <= s.length <= 300`
- `1 <= wordDict.length <= 1000`
- `1 <= wordDict[i].length <= 20`
- `s` and `wordDict[i]` consist of only lowercase English letters.
- All the strings of `wordDict` are **unique**.

Combination Sum IV

Given an array of **distinct** integers `nums` and a target integer `target`, return *the number of possible combinations that add up to target*.

The answer is **guaranteed** to fit in a **32-bit** integer.

Example 1:

Input: `nums = [1,2,3]`, `target = 4`

Output: 7

Explanation:

The possible combination ways are:

(1, 1, 1, 1)

(1, 1, 2)

(1, 2, 1)

(1, 3)

(2, 1, 1)

(2, 2)

(3, 1)

Note that different sequences are counted as different combinations.

Example 2:

Input: `nums = [9]`, `target = 3`

Output: 0

Constraints:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 1000`
- All the elements of `nums` are **unique**.

- $1 \leq \text{target} \leq 1000$

Follow up: What if negative numbers are allowed in the given array? How does it change the problem? What limitation we need to add to the question to allow negative numbers?

House Robber

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 systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight without alerting the police*.

Example 1:

Input: `nums = [1,2,3,1]`

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = $1 + 3 = 4$.

Example 2:

Input: `nums = [2,7,9,3,1]`

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = $2 + 9 + 1 = 12$.

Constraints:

- $1 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 400$

House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle**. That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system connected, and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight without alerting the police*.

Example 1:

Input: `nums = [2,3,2]`

Output: 3

Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.

Example 2:

Input: `nums = [1,2,3,1]`

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

Example 3:

Input: `nums = [1,2,3]`

Output: 3

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 1000`

Decode Ways

A message containing letters from A-Z can be **encoded** into numbers using the following mapping:

'A' -> "1"

'B' -> "2"

⋮
'Z' -> "26"

To **decode** an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:

- "AAJF" with the grouping (1 1 10 6)
- "KJF" with the grouping (11 10 6)

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string *s* containing only digits, return *the number of ways to decode it*.

The answer is guaranteed to fit in a **32-bit** integer.

Example 1:

Input: *s* = "12"

Output: 2

Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: *s* = "226"

Output: 3

Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input: s = "0"

Output: 0

Explanation: There is no character that is mapped to a number starting with 0.

The only valid mappings with 0 are 'J' -> "10" and 'T' -> "20", neither of which start with 0.

Hence, there are no valid ways to decode this since all digits need to be mapped.

Example 4:

Input: s = "06"

Output: 0

Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").

Constraints:

- $1 \leq s.length \leq 100$
- s contains only digits and may contain leading zero(s).

Unique Paths

A robot is located at the top-left corner of a $m \times n$ grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Example 1:



Input: $m = 3, n = 7$

Output: 28

Example 2:

Input: $m = 3, n = 2$

Output: 3

Explanation:

From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Down -> Down

2. Down -> Down -> Right
3. Down -> Right -> Down

Example 3:

Input: $m = 7, n = 3$
Output: 28

Example 4:

Input: $m = 3, n = 3$
Output: 6

Constraints:

- $1 \leq m, n \leq 100$
- It's guaranteed that the answer will be less than or equal to $2 * 10^9$.

Jump Game

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` *if you can reach the last index, or false otherwise.*

Example 1:

Input: `nums = [2,3,1,1,4]`

Output: `true`

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: `nums = [3,2,1,0,4]`

Output: `false`

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

- `1 <= nums.length <= 104`
- `0 <= nums[i] <= 105`

Graph

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 value (`int`) and a list (`List[Node]`) of its neighbors.

```
class Node {
    public int val;
    public List<Node> neighbors;
}
```

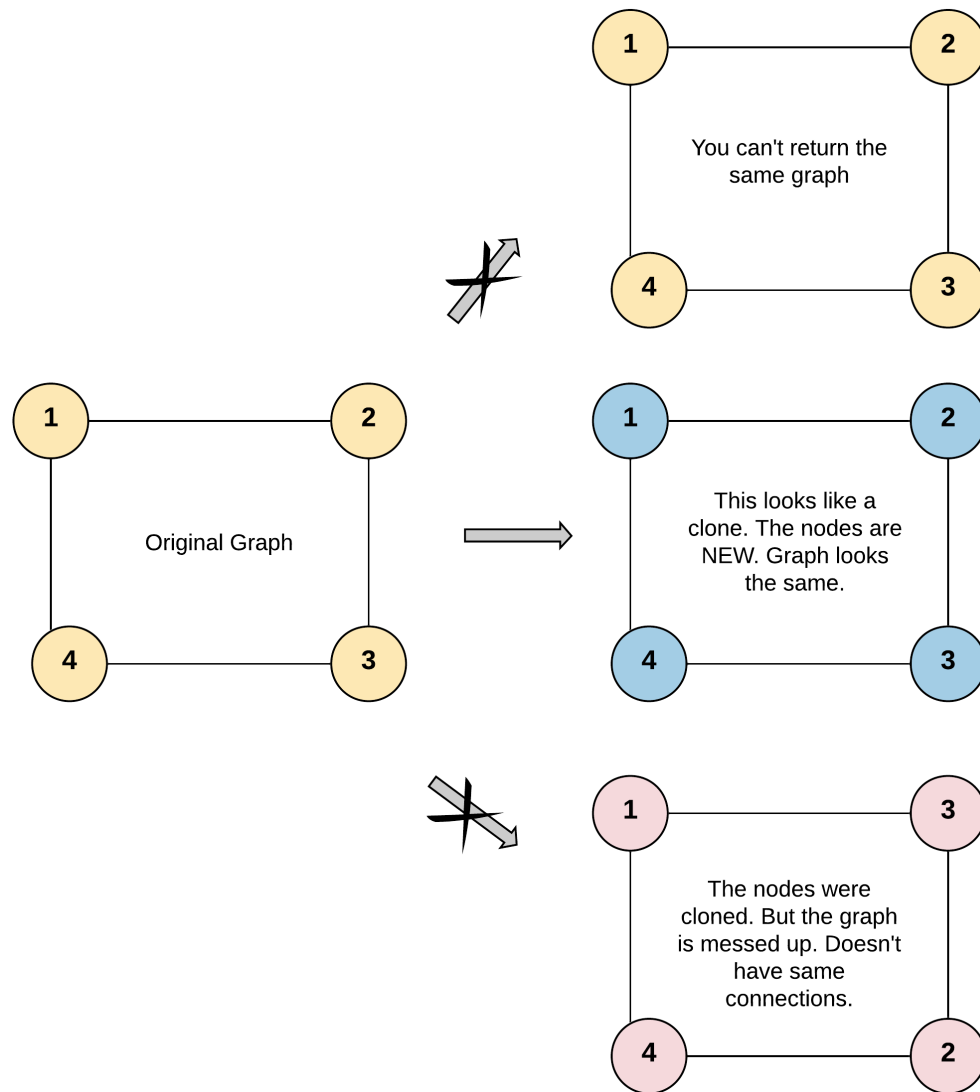
Test case format:

For simplicity, each node's value is the same as the node's index (1-indexed). For example, the first node with `val == 1`, the second node with `val == 2`, and so on. The graph is represented in the test case using an adjacency list.

An adjacency list is a collection of unordered **lists** used to represent a finite graph. Each list describes the set of neighbors of a node in the graph.

The given node will always be the first node with `val = 1`. You must return the **copy of the given node** as a reference to the cloned graph.

Example 1:



Input: adjList = [[2,4],[1,3],[2,4],[1,3]]

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

Explanation: There are 4 nodes in the graph.

1st node (val = 1)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).

2nd node (val = 2)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).

3rd node (val = 3)'s neighbors are 2nd node (val = 2) and 4th node (val = 4).
4th node (val = 4)'s neighbors are 1st node (val = 1) and 3rd node (val = 3).

Example 2:



Input: adjList = [[]]

Output: [[]]

Explanation: Note that the input contains one empty list. The graph consists of only one node with val = 1 and it does not have any neighbors.

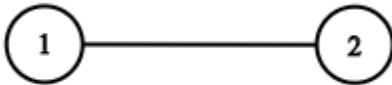
Example 3:

Input: adjList = []

Output: []

Explanation: This an empty graph, it does not have any nodes.

Example 4:



Input: `adjList = [[2],[1]]`

Output: `[[2],[1]]`

Constraints:

- The number of nodes in the graph is in the range $[0, 100]$.
- $1 \leq \text{Node.val} \leq 100$
- `Node.val` is unique for each node.
- There are no repeated edges and no self-loops in the graph.
- The Graph is connected and all nodes can be visited starting from the given node.

Course Schedule

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Example 1:

Input: `numCourses = 2, prerequisites = [[1,0]]`

Output: `true`

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: `numCourses = 2, prerequisites = [[1,0],[0,1]]`

Output: `false`

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Constraints:

- `1 <= numCourses <= 105`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= ai, bi < numCourses`
- All the pairs `prerequisites[i]` are **unique**.

Pacific Atlantic Water Flow

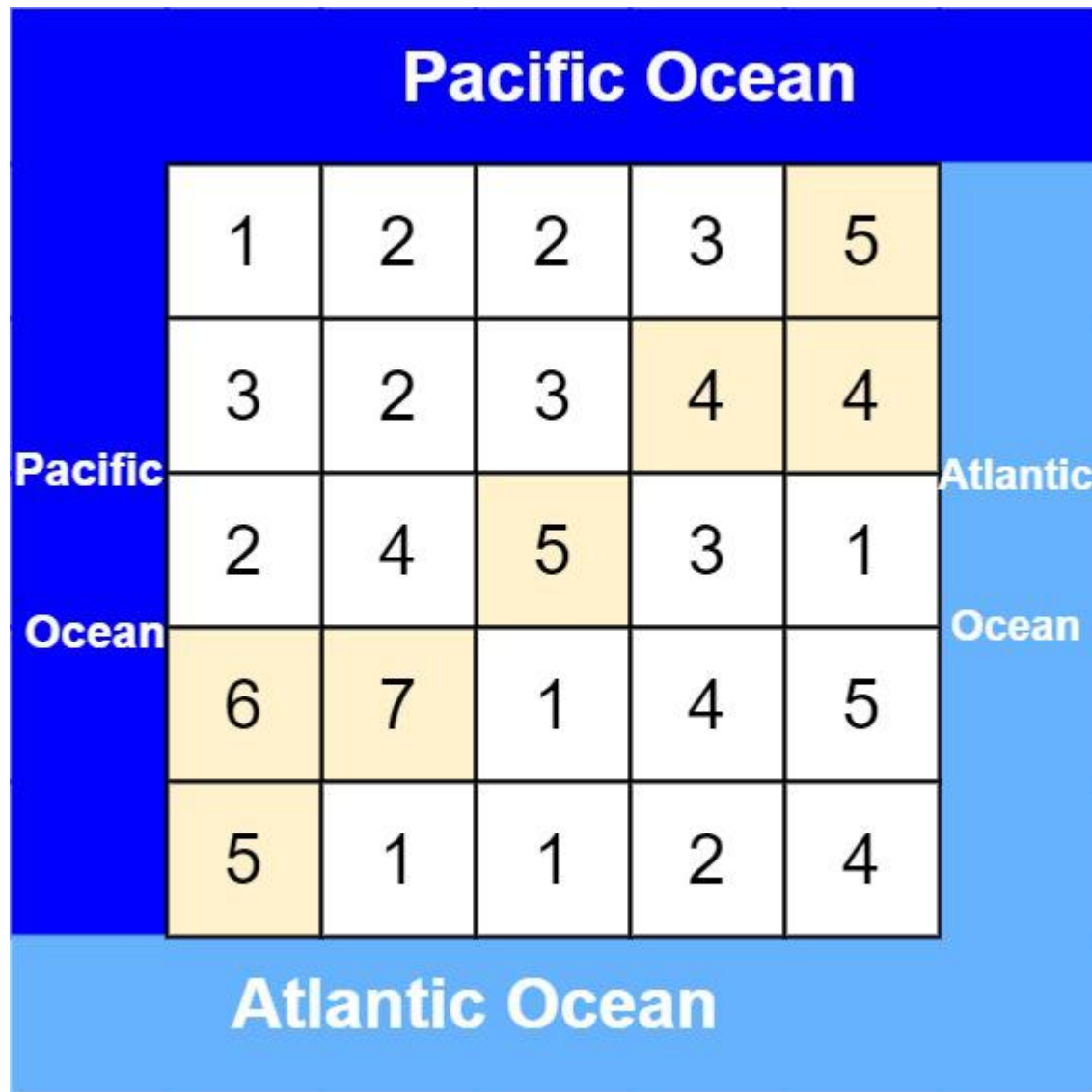
There is an $m \times n$ rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

The island is partitioned into a grid of square cells. You are given an $m \times n$ integer matrix `heights` where `heights[r][c]` represents the **height above sea level** of the cell at coordinate (r, c) .

The island receives a lot of rain, and the rain water can flow to neighboring cells directly north, south, east, and west if the neighboring cell's height is **less than or equal to** the current cell's height. Water can flow from any cell adjacent to an ocean into the ocean.

Return a **2D list** of grid coordinates `result` where `result[i] = [ri, ci]` denotes that rain water can flow from cell (r_i, c_i) to **both** the Pacific and Atlantic oceans.

Example 1:



Input: heights = [[1,2,2,3,5],[3,2,3,4,4],[2,4,5,3,1],[6,7,1,4,5],[5,1,1,2,4]]
 Output: [[0,4],[1,3],[1,4],[2,2],[3,0],[3,1],[4,0]]

Example 2:

Input: heights = [[2,1],[1,2]]
Output: [[0,0],[0,1],[1,0],[1,1]]

Constraints:

- $m == \text{heights.length}$
- $n == \text{heights}[r].\text{length}$
- $1 \leq m, n \leq 200$
- $0 \leq \text{heights}[r][c] \leq 10^5$

Number of Islands

Given an $m \times n$ 2D binary grid `grid` which represents a map of '1's (land) and '0's (water), return *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: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
```

Output: 1

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
```

Output: 3

Constraints:

- $m == \text{grid.length}$
- $n == \text{grid}[i].\text{length}$
- $1 \leq m, n \leq 300$

- `grid[i][j]` is '0' or '1'.

Longest Consecutive Sequence

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence*.

You must write an algorithm that runs in $O(n)$ time.

Example 1:

Input: `nums = [100,4,200,1,3,2]`

Output: 4

Explanation: The longest consecutive elements sequence is `[1, 2, 3, 4]`. Therefore its length is 4.

Example 2:

Input: `nums = [0,3,7,2,5,8,4,6,0,1]`

Output: 9

Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Alien Dictionary

Graph Valid Tree

Number of Connected Components in an Undirected Graph

Interval

Insert Interval

You are given an array of non-overlapping intervals `intervals` where `intervals[i] = [starti, endi]` represent the start and the end of the i^{th} interval and `intervals` is sorted in ascending order by `starti`. You are also given an interval `newInterval = [start, end]` that represents the start and end of another interval.

Insert `newInterval` into `intervals` such that `intervals` is still sorted in ascending order by `starti` and `intervals` still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return `intervals` *after the insertion*.

Example 1:

Input: `intervals = [[1,3],[6,9]]`, `newInterval = [2,5]`

Output: `[[1,5],[6,9]]`

Example 2:

Input: `intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]]`, `newInterval = [4,8]`

Output: `[[1,2],[3,10],[12,16]]`

Explanation: Because the new interval `[4,8]` overlaps with `[3,5]`, `[6,7]`, `[8,10]`.

Example 3:

Input: `intervals = []`, `newInterval = [5,7]`

Output: `[[5,7]]`

Example 4:

Input: `intervals = [[1,5]]`, `newInterval = [2,3]`

Output: `[[1,5]]`

Example 5:

Input: intervals = [[1,5]], newInterval = [2,7]
Output: [[1,7]]

Constraints:

- $0 \leq \text{intervals.length} \leq 10^4$
- $\text{intervals}[i].\text{length} == 2$
- $0 \leq \text{start}_i \leq \text{end}_i \leq 10^5$
- intervals is sorted by start_i in **ascending** order.
- $\text{newInterval.length} == 2$
- $0 \leq \text{start} \leq \text{end} \leq 10^5$

Merge Intervals

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Example 1:

Input: `intervals = [[1,3],[2,6],[8,10],[15,18]]`

Output: `[[1,6],[8,10],[15,18]]`

Explanation: Since intervals `[1,3]` and `[2,6]` overlaps, merge them into `[1,6]`.

Example 2:

Input: `intervals = [[1,4],[4,5]]`

Output: `[[1,5]]`

Explanation: Intervals `[1,4]` and `[4,5]` are considered overlapping.

Constraints:

- $1 \leq \text{intervals.length} \leq 10^4$
- $\text{intervals}[i].\text{length} == 2$
- $0 \leq \text{start}_i \leq \text{end}_i \leq 10^4$

Non-overlapping Intervals

Given an array of intervals `intervals` where `intervals[i] = [starti, endi]`, return *the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.*

Example 1:

Input: `intervals = [[1,2],[2,3],[3,4],[1,3]]`

Output: 1

Explanation: `[1,3]` can be removed and the rest of the intervals are non-overlapping.

Example 2:

Input: `intervals = [[1,2],[1,2],[1,2]]`

Output: 2

Explanation: You need to remove two `[1,2]` to make the rest of the intervals non-overlapping.

Example 3:

Input: `intervals = [[1,2],[2,3]]`

Output: 0

Explanation: You don't need to remove any of the intervals since they're already non-overlapping.

Constraints:

- $1 \leq \text{intervals.length} \leq 10^5$
- $\text{intervals}[i].\text{length} == 2$
- $-5 * 10^4 \leq \text{start}_i < \text{end}_i \leq 5 * 10^4$

Meeting Rooms

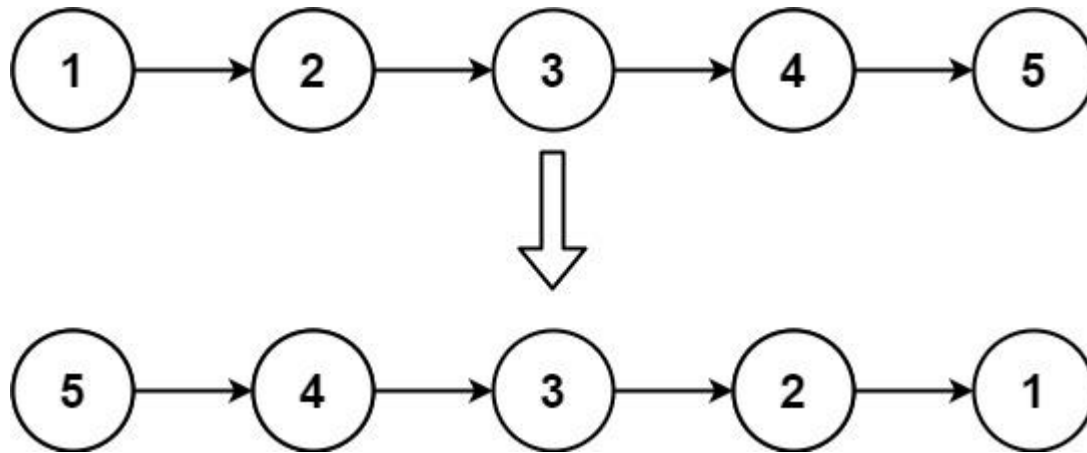
Meeting Rooms II

Linked List

Reverse Linked List

Given the head of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:



Input: head = [1,2,3,4,5]

Output: [5,4,3,2,1]

Example 2:

Input: head = [1,2]

Output: [2,1]

Example 3:

Input: head = []

Output: []

Constraints:

- The number of nodes in the list is the range $[0, 5000]$.
- $-5000 \leq \text{Node.val} \leq 5000$

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

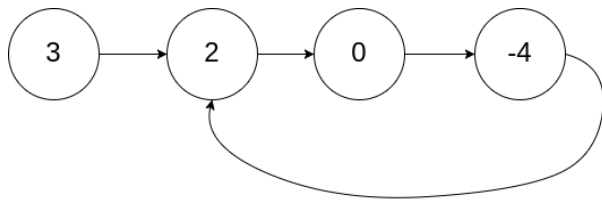
Linked List Cycle

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:

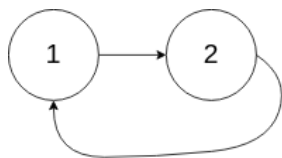


Input: `head = [3,2,0,-4]`, `pos = 1`

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:

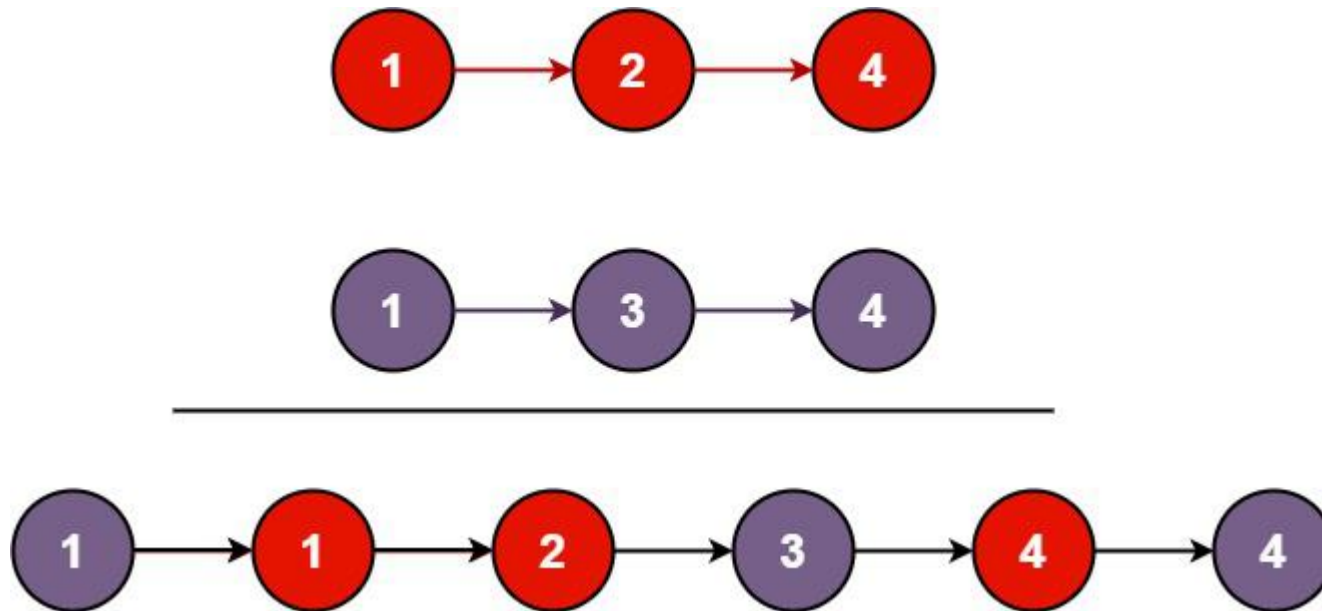
- The number of the nodes in the list is in the range $[0, 10^4]$.
- $-10^5 \leq \text{Node.val} \leq 10^5$
- pos is -1 or a **valid index** in the linked-list.

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

Merge Two Sorted Lists

Merge two sorted linked lists and return it as a **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Example 1:



Input: l1 = [1,2,4], l2 = [1,3,4]

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

Example 2:

Input: l1 = [], l2 = []

Output: []

Example 3:

Input: $l1 = []$, $l2 = [0]$
Output: $[0]$

Constraints:

- The number of nodes in both lists is in the range $[0, 50]$.
- $-100 \leq \text{Node.val} \leq 100$
- Both $l1$ and $l2$ are sorted in **non-decreasing** order.

Merge k Sorted Lists

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: `lists = [[1,4,5],[1,3,4],[2,6]]`

Output: `[1,1,2,3,4,4,5,6]`

Explanation: The linked-lists are:

```
[
  1->4->5,
  1->3->4,
  2->6
]
```

merging them into one sorted list:

`1->1->2->3->4->4->5->6`

Example 2:

Input: `lists = []`

Output: `[]`

Example 3:

Input: `lists = [[]]`

Output: `[]`

Constraints:

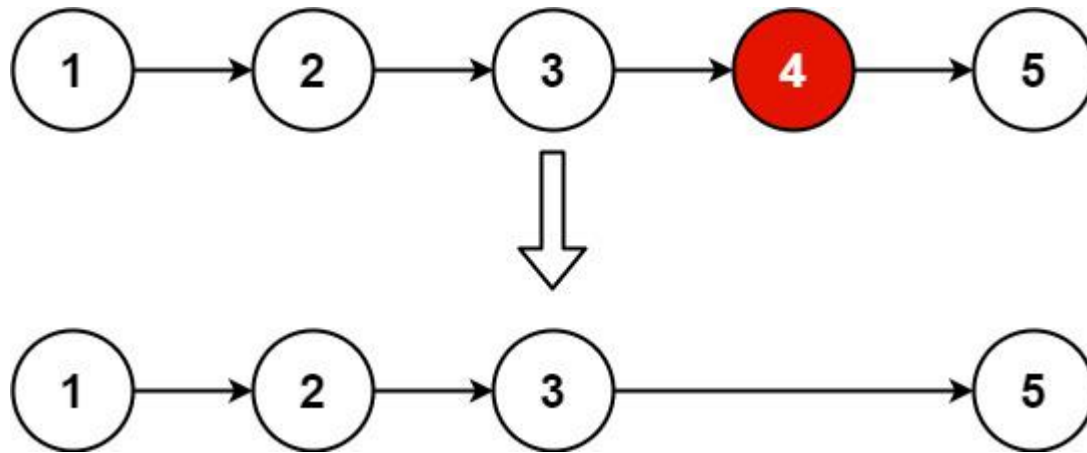
- `k == lists.length`

- $0 \leq k \leq 10^4$
- $0 \leq \text{lists}[i].\text{length} \leq 500$
- $-10^4 \leq \text{lists}[i][j] \leq 10^4$
- $\text{lists}[i]$ is sorted in **ascending order**.
- The sum of $\text{lists}[i].\text{length}$ won't exceed 10^4 .

Remove Nth Node From End of List

Given the head of a linked list, remove the n^{th} node from the end of the list and return its head.

Example 1:



Input: head = [1,2,3,4,5], n = 2

Output: [1,2,3,5]

Example 2:

Input: head = [1], n = 1

Output: []

Example 3:

Input: head = [1,2], n = 1

Output: [1]

Constraints:

- The number of nodes in the list is `sz`.
- `1 <= sz <= 30`
- `0 <= Node.val <= 100`
- `1 <= n <= sz`

Follow up: Could you do this in one pass?

Reorder List

You are given the head of a singly linked-list. The list can be represented as:

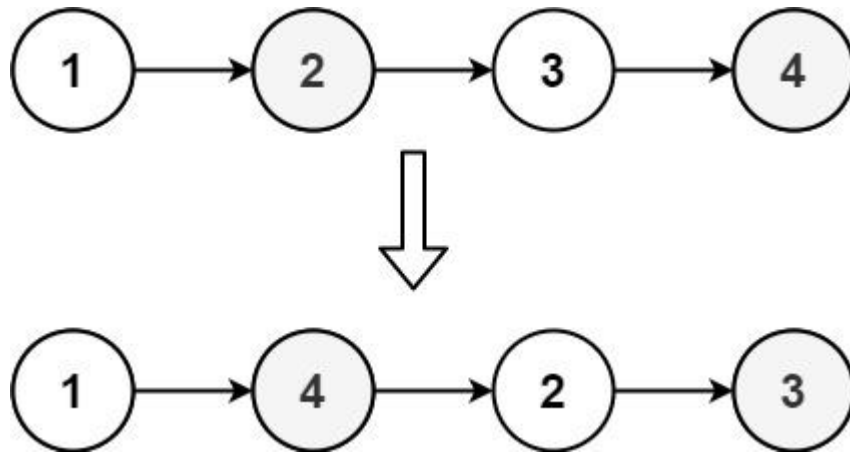
$$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$$

Reorder the list to be on the following form:

$$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \dots$$

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

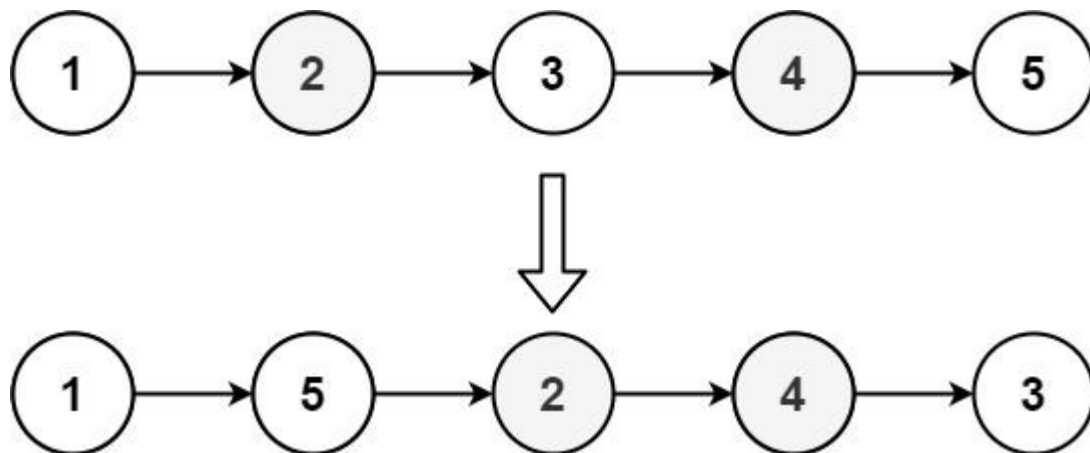
Example 1:



Input: head = [1,2,3,4]

Output: [1,4,2,3]

Example 2:



Input: head = [1,2,3,4,5]

Output: [1,5,2,4,3]

Constraints:

- The number of nodes in the list is in the range $[1, 5 * 10^4]$.
- $1 \leq \text{Node.val} \leq 1000$

Matrix

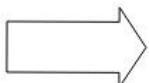
Set Matrix Zeroes

Given an $m \times n$ integer matrix `matrix`, if an element is 0, set its entire row and column to 0's, and return *the matrix*.

You must do it [in place](#).

Example 1:

1	1	1
1	0	1
1	1	1



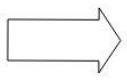
1	0	1
0	0	0
1	0	1

Input: `matrix = [[1,1,1],[1,0,1],[1,1,1]]`

Output: `[[1,0,1],[0,0,0],[1,0,1]]`

Example 2:

0	1	2	0
3	4	5	2
1	3	1	5



0	0	0	0
0	4	5	0
0	3	1	0

Input: `matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]`

Output: `[[0,0,0,0],[0,4,5,0],[0,3,1,0]]`

Constraints:

- `m == matrix.length`
- `n == matrix[0].length`
- `1 <= m, n <= 200`
- `-231 <= matrix[i][j] <= 231 - 1`

Follow up:

- A straightforward solution using $O(mn)$ space is probably a bad idea.
- A simple improvement uses $O(m + n)$ space, but still not the best solution.
- Could you devise a constant space solution?

Spiral Matrix

Given an $m \times n$ matrix, return *all elements of the matrix in spiral order*.

Example 1:

1	→	2	→	3
4	→	5		↓ 6
↑ 7	←	8	←	↓ 9

Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]

Example 2:

1 →	2 →	3 →	4
5 →	6 →	7	8
9 ←	10 ←	11 ←	12

Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

Output: [1,2,3,4,8,12,11,10,9,5,6,7]

Constraints:

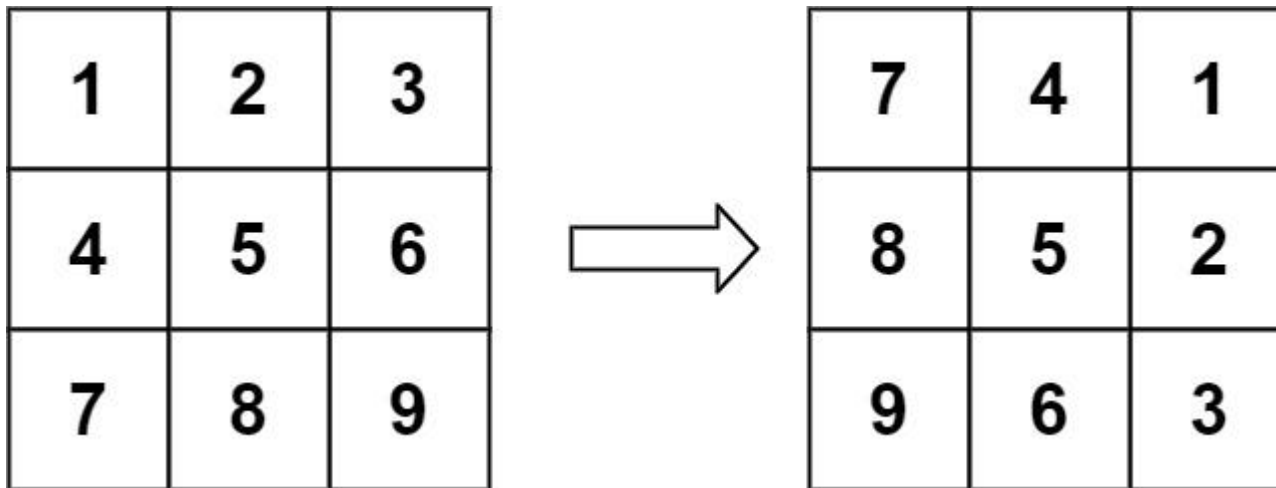
- m == matrix.length
- n == matrix[i].length
- 1 <= m, n <= 10
- -100 <= matrix[i][j] <= 100

Rotate Image

You are given an $n \times n$ 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

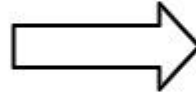


Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [[7,4,1],[8,5,2],[9,6,3]]

Example 2:

5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16



15	13	2	5
14	3	4	1
12	6	8	9
16	7	10	11

Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

Example 3:

Input: matrix = [[1]]

Output: [[1]]

Example 4:

Input: matrix = [[1,2],[3,4]]

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

Constraints:

- matrix.length == n

- `matrix[i].length == n`
- `1 <= n <= 20`
- `-1000 <= matrix[i][j] <= 1000`

Word Search

Given an $m \times n$ grid of characters `board` and a string `word`, return `true` *if word exists in the grid*.

The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:

A	B	C	E
S	F	C	S
A	D	E	E

Input: `board = [["A","B","C","E"], ["S","F","C","S"], ["A","D","E","E"]]`, `word = "ABCCED"`

Output: `true`

Example 2:

A	B	C	E
S	F	C	S
A	D	E	E

Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"
Output: true

Example 3:

A	B	C	E
S	F	C	S
A	D	E	E

Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"
Output: false

Constraints:

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `1 <= word.length <= 15`
- board and word consists of only lowercase and uppercase English letters.

Follow up: Could you use search pruning to make your solution faster with a larger board?

String

Longest Substring Without Repeating Characters

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

Example 1:

Input: `s = "abcabcbb"`

Output: 3

Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: `s = "bbbbbb"`

Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: `s = "pwwkew"`

Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Example 4:

Input: `s = ""`

Output: 0

Constraints:

- $0 \leq s.length \leq 5 * 10^4$
- `s` consists of English letters, digits, symbols and spaces.

Longest Repeating Character Replacement

You are given a string s and an integer k . You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most k times.

Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

Example 1:

Input: $s = \text{"ABAB"}, k = 2$

Output: 4

Explanation: Replace the two 'A's with two 'B's or vice versa.

Example 2:

Input: $s = \text{"AABABBA"}, k = 1$

Output: 4

Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.

Constraints:

- $1 \leq s.length \leq 10^5$
- s consists of only uppercase English letters.
- $0 \leq k \leq s.length$

Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return *the **minimum window substring** of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string ""*.

The testcases will be generated such that the answer is **unique**.

A **substring** is a contiguous sequence of characters within the string.

Example 1:

Input: $s = \text{"ADOBECODEBANC"}, t = \text{"ABC"}$

Output: `"BANC"`

Explanation: The minimum window substring `"BANC"` includes 'A', 'B', and 'C' from string t .

Example 2:

Input: $s = \text{"a"}, t = \text{"a"}$

Output: `"a"`

Explanation: The entire string s is the minimum window.

Example 3:

Input: $s = \text{"a"}, t = \text{"aa"}$

Output: `""`

Explanation: Both 'a's from t must be included in the window.

Since the largest window of s only has one 'a', return empty string.

Constraints:

- $m == s.length$
- $n == t.length$

- $1 \leq m, n \leq 10^5$
- s and t consist of uppercase and lowercase English letters.

Follow up: Could you find an algorithm that runs in $O(m + n)$ time?

Valid Anagram

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

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?

Group Anagrams

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

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: `strs = ["eat","tea","tan","ate","nat","bat"]`
Output: `[["bat"],["nat","tan"],["ate","eat","tea"]]`

Example 2:

Input: `strs = [""]`
Output: `[[""]]`

Example 3:

Input: `strs = ["a"]`
Output: `[["a"]]`

Constraints:

- `1 <= strs.length <= 104`
- `0 <= strs[i].length <= 100`
- `strs[i]` consists of lowercase English letters.

Valid Parentheses

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.

Example 1:

Input: *s* = "()"

Output: true

Example 2:

Input: *s* = "()[]{}"

Output: true

Example 3:

Input: *s* = "("

Output: false

Example 4:

Input: *s* = "([)]"

Output: false

Example 5:

Input: *s* = "{[]}"

Output: true

Constraints:

- $1 \leq s.length \leq 10^4$
- s consists of parentheses only '()[]{}'.

Valid Palindrome

Given a string S , determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Example 1:

Input: $s = \text{"A man, a plan, a canal: Panama"}$

Output: true

Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:

Input: $s = \text{"race a car"}$

Output: false

Explanation: "raceacar" is not a palindrome.

Constraints:

- $1 \leq s.length \leq 2 * 10^5$
- s consists only of printable ASCII characters.

Longest Palindromic Substring

Given a string S , return *the longest palindromic substring* in S .

Example 1:

Input: $s = \text{"babad"}$

Output: "bab"

Note: "aba" is also a valid answer.

Example 2:

Input: $s = \text{"cbbdd"}$

Output: "bb"

Example 3:

Input: $s = \text{"a"}$

Output: "a"

Example 4:

Input: $s = \text{"ac"}$

Output: "a"

Constraints:

- $1 \leq s.length \leq 1000$
- s consist of only digits and English letters.

Palindromic Substrings

Given a string S , return *the number of **palindromic substrings** in it*.

A string is a **palindrome** when it reads the same backward as forward.

A **substring** is a contiguous sequence of characters within the string.

Example 1:

Input: $s = \text{"abc"}$

Output: 3

Explanation: Three palindromic strings: "a", "b", "c".

Example 2:

Input: $s = \text{"aaa"}$

Output: 6

Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".

Constraints:

- $1 \leq s.length \leq 1000$
- s consists of lowercase English letters.

Encode and Decode Strings

Tree

Maximum Depth of Binary Tree

Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:

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

Example 2:

Input: root = [1,null,2]
Output: 2

Example 3:

Input: root = []
Output: 0

Example 4:

Input: root = [0]
Output: 1

Constraints:

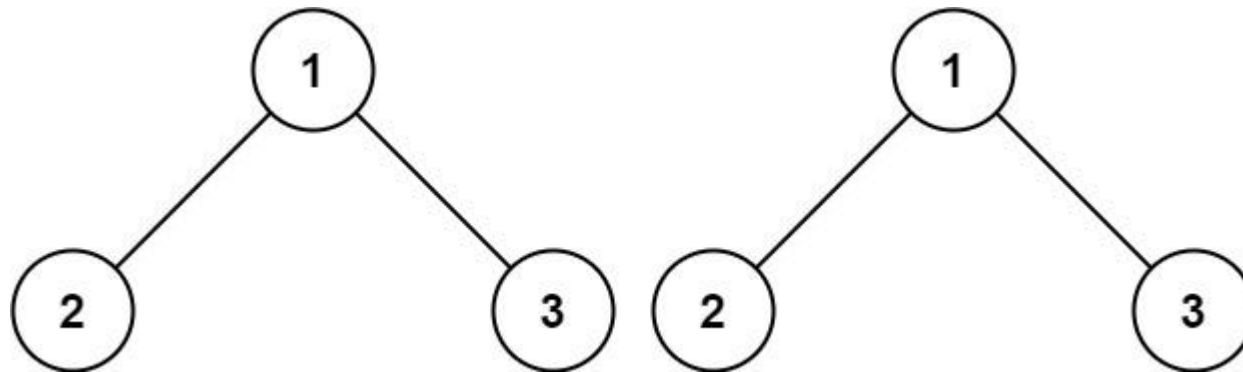
- The number of nodes in the tree is in the range $[0, 10^4]$.
- $-100 \leq \text{Node.val} \leq 100$

Same Tree

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

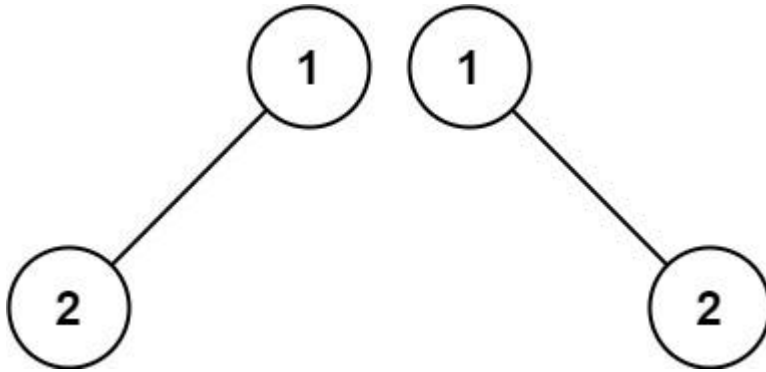
Example 1:



Input: p = [1,2,3], q = [1,2,3]

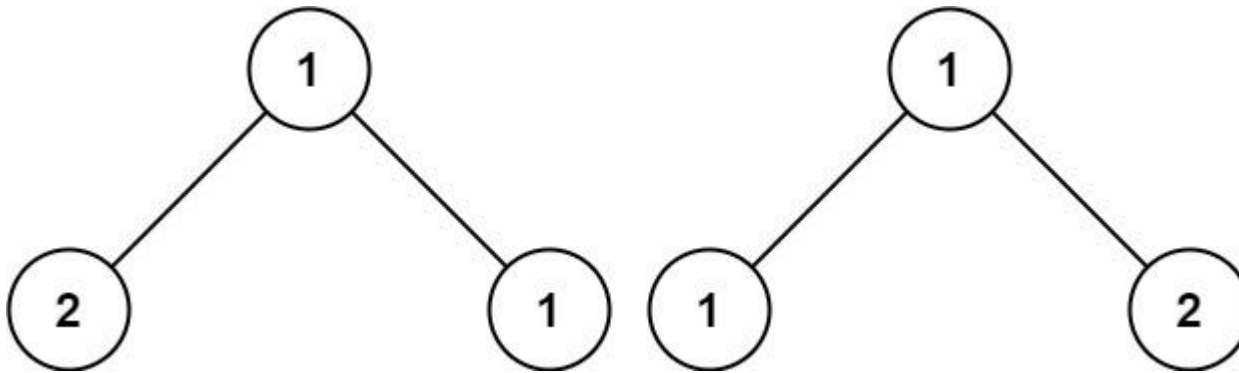
Output: true

Example 2:



Input: $p = [1, 2]$, $q = [1, \text{null}, 2]$
Output: false

Example 3:



Input: $p = [1, 2, 1]$, $q = [1, 1, 2]$
Output: false

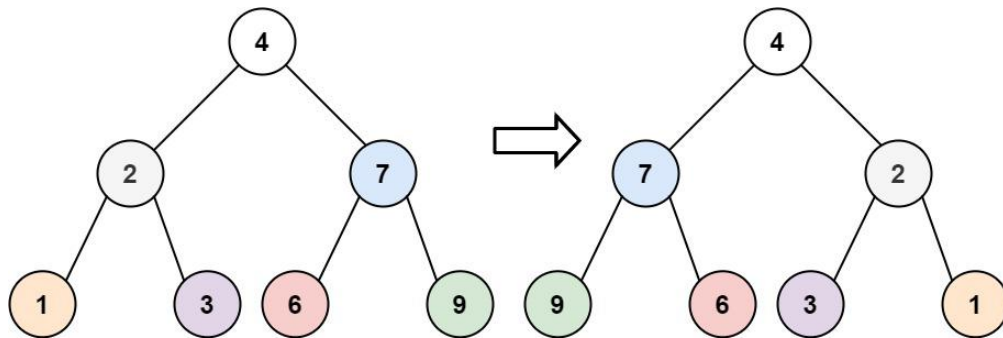
Constraints:

- The number of nodes in both trees is in the range $[0, 100]$.
- $-10^4 \leq \text{Node.val} \leq 10^4$

Invert Binary Tree

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

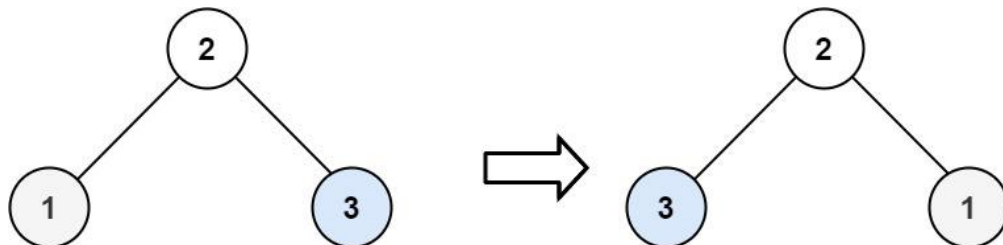
Example 1:



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 $[0, 100]$.
- $-100 \leq \text{Node.val} \leq 100$

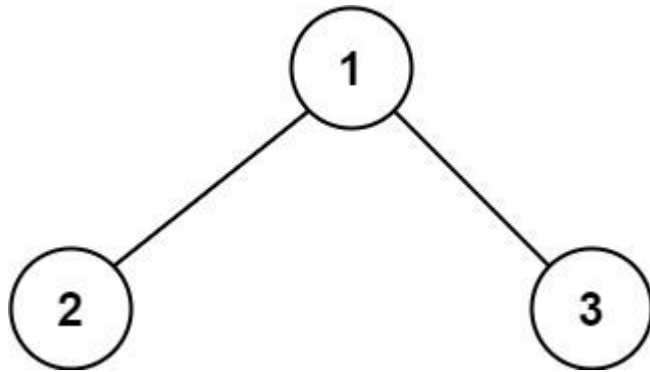
Binary Tree Maximum Path Sum

A **path** in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence **at most once**. Note that the path does not need to pass through the root.

The **path sum** of a path is the sum of the node's values in the path.

Given the root of a binary tree, return *the maximum path sum of any path*.

Example 1:

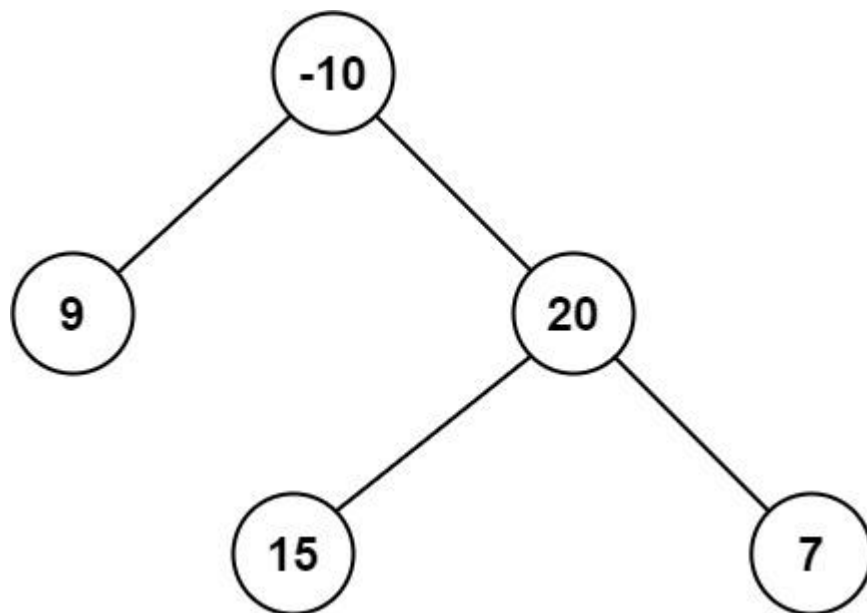


Input: root = [1,2,3]

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of $2 + 1 + 3 = 6$.

Example 2:



Input: root = [-10,9,20,null,null,15,7]

Output: 42

Explanation: The optimal path is 15 -> 20 -> 7 with a path sum of $15 + 20 + 7 = 42$.

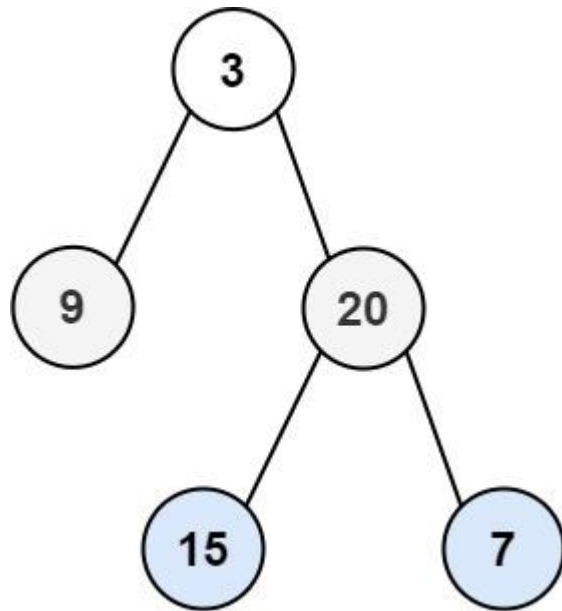
Constraints:

- The number of nodes in the tree is in the range $[1, 3 * 10^4]$.
- $-1000 \leq \text{Node.val} \leq 1000$

Binary Tree Level Order Traversal

Given the root of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:



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

Output: [[3],[9,20],[15,7]]

Example 2:

Input: root = [1]

Output: [[1]]

Example 3:

Input: root = []
Output: []

Constraints:

- The number of nodes in the tree is in the range $[0, 2000]$.
- $-1000 \leq \text{Node.val} \leq 1000$

Serialize and Deserialize Binary Tree

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Clarification: The input/output format is the same as [how LeetCode serializes a binary tree](#). You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Example 1:

Input: root = [1,2,3,null,null,4,5]
Output: [1,2,3,null,null,4,5]

Example 2:

Input: root = []
Output: []

Example 3:

Input: root = [1]
Output: [1]

Example 4:

Input: root = [1,2]
Output: [1,2]

Constraints:

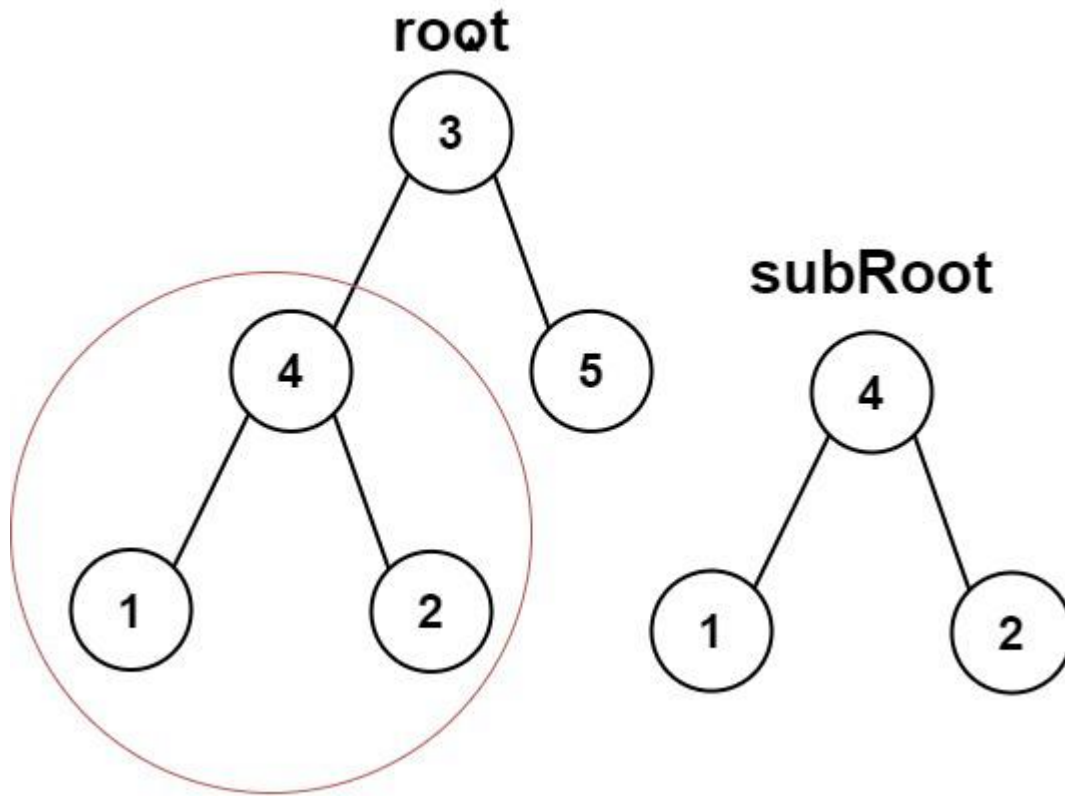
- The number of nodes in the tree is in the range $[0, 10^4]$.
- $-1000 \leq \text{Node.val} \leq 1000$

Subtree of Another Tree

Given the roots of two binary trees `root` and `subRoot`, return `true` if there is a subtree of `root` with the same structure and node values of `subRoot` and `false` otherwise.

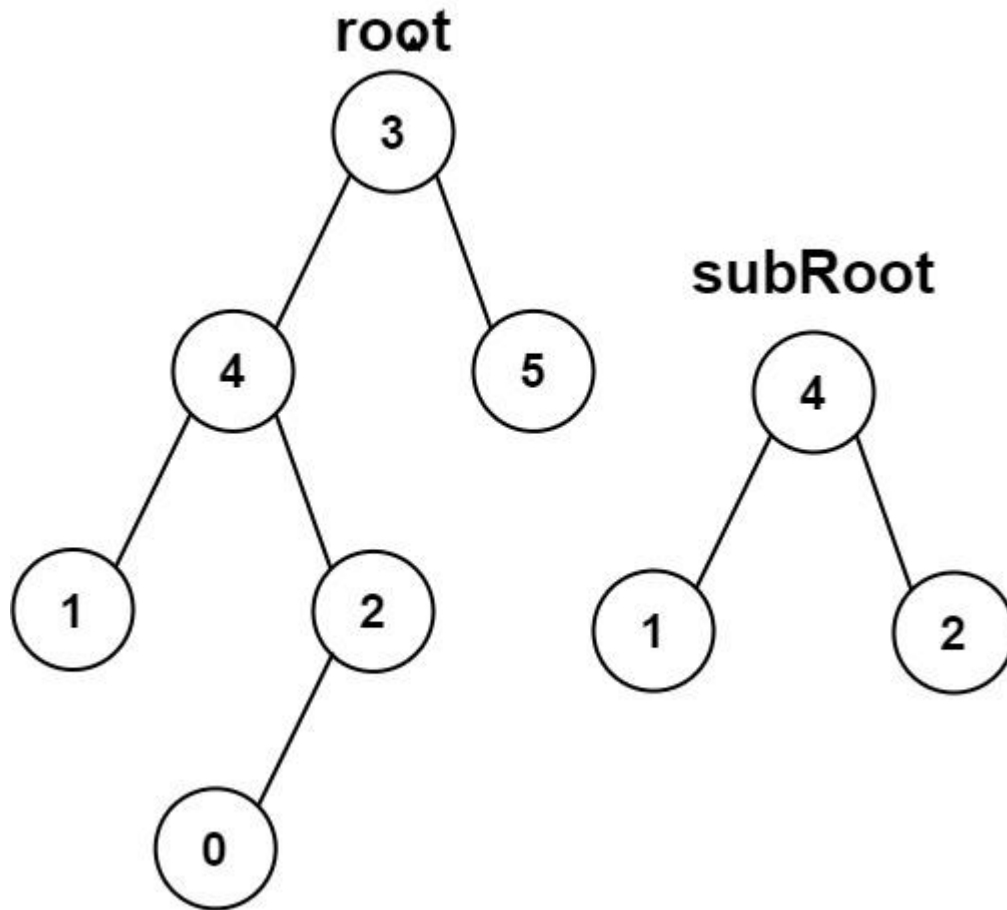
A subtree of a binary tree `tree` is a tree that consists of a node in `tree` and all of this node's descendants. The tree `tree` could also be considered as a subtree of itself.

Example 1:



Input: root = [3,4,5,1,2], subRoot = [4,1,2]
Output: true

Example 2:



Input: root = [3,4,5,1,2,null,null,null,null,0], subRoot = [4,1,2]
Output: false

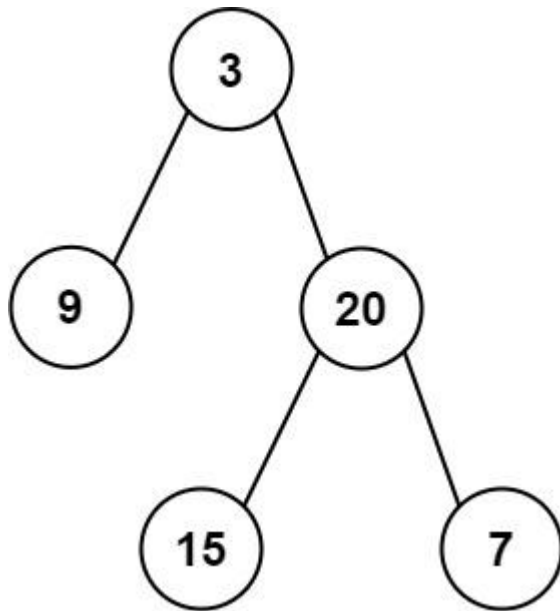
Constraints:

- The number of nodes in the `root` tree is in the range $[1, 2000]$.
- The number of nodes in the `subRoot` tree is in the range $[1, 1000]$.
- $-10^4 \leq \text{root.val} \leq 10^4$
- $-10^4 \leq \text{subRoot.val} \leq 10^4$

Construct Binary Tree from Preorder and Inorder Traversal

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.

Example 1:



Input: `preorder = [3,9,20,15,7]`, `inorder = [9,3,15,20,7]`

Output: `[3,9,20,null,null,15,7]`

Example 2:

Input: `preorder = [-1]`, `inorder = [-1]`

Output: `[-1]`

Constraints:

- `1 <= preorder.length <= 3000`
- `inorder.length == preorder.length`
- `-3000 <= preorder[i], inorder[i] <= 3000`
- preorder and inorder consist of **unique** values.
- Each value of inorder also appears in preorder.
- preorder is **guaranteed** to be the preorder traversal of the tree.
- inorder is **guaranteed** to be the inorder traversal of the tree.

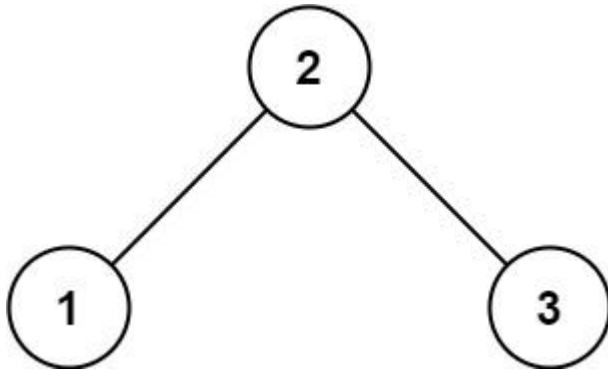
Validate Binary Search Tree

Given the root of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

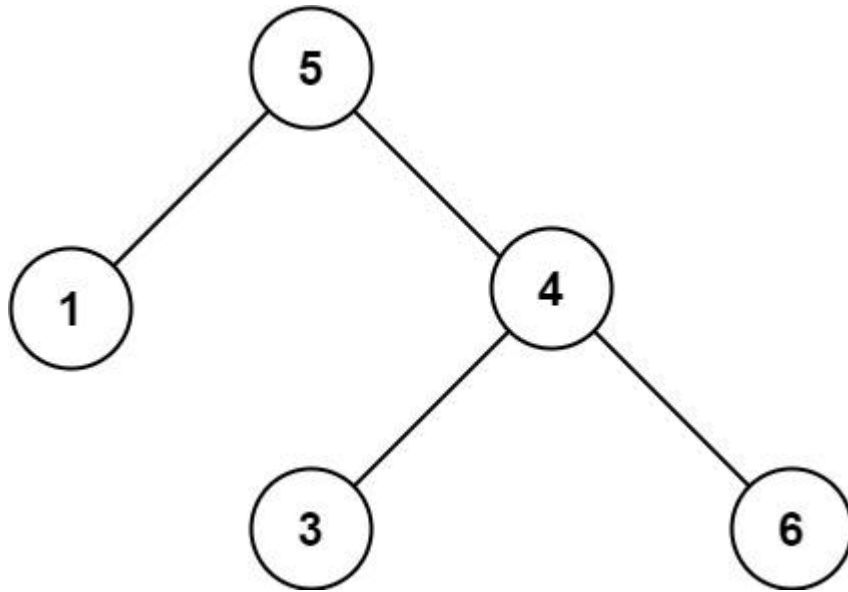
Example 1:



Input: root = [2,1,3]

Output: true

Example 2:



Input: root = [5,1,4,null,null,3,6]

Output: false

Explanation: The root node's value is 5 but its right child's value is 4.

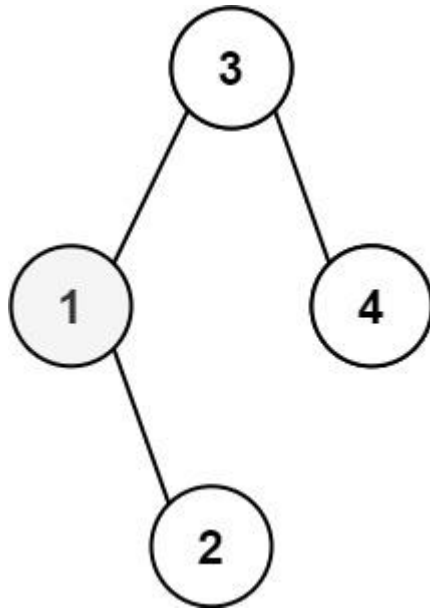
Constraints:

- The number of nodes in the tree is in the range $[1, 10^4]$.
- $-2^{31} \leq \text{Node.val} \leq 2^{31} - 1$

Kth Smallest Element in a BST

Given the root of a binary search tree, and an integer k , return *the* k^{th} (**1-indexed**) *smallest element in the tree*.

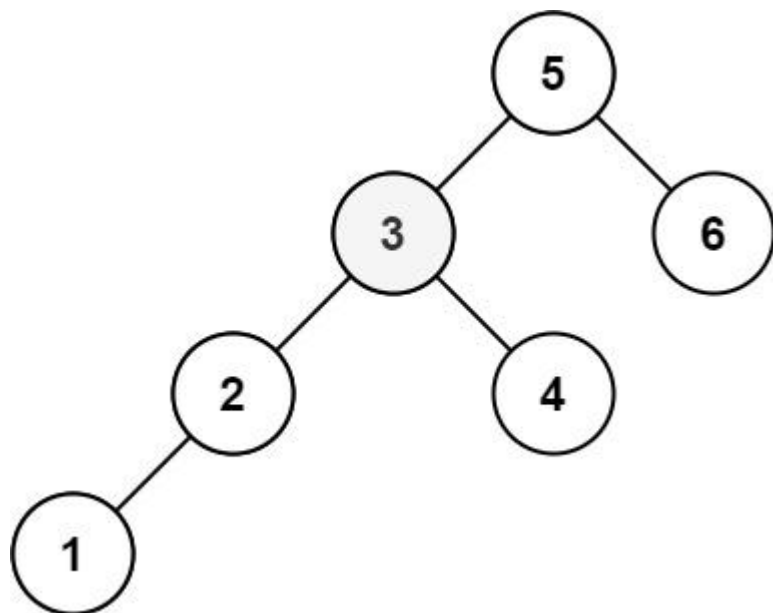
Example 1:



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

Output: 1

Example 2:



Input: root = [5,3,6,2,4,null,null,1], k = 3

Output: 3

Constraints:

- The number of nodes in the tree is n.
- $1 \leq k \leq n \leq 10^4$
- $0 \leq \text{Node.val} \leq 10^4$

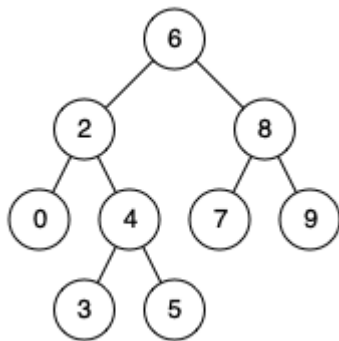
Follow up: If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the kth smallest frequently, how would you optimize?

Lowest Common Ancestor of a Binary Search Tree

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

According to the [definition of LCA on Wikipedia](#): “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:

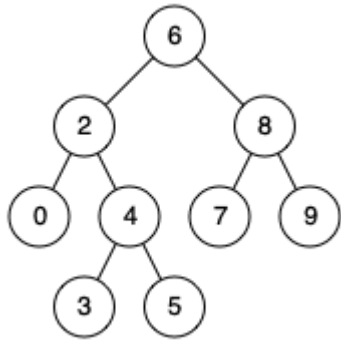


Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

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 $[2, 10^5]$.
- $-10^9 \leq \text{Node.val} \leq 10^9$
- All `Node.val` are **unique**.
- $p \neq q$
- p and q will exist in the BST.

Implement Trie (Prefix Tree)

A [trie](#) (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string word into the trie.
- `boolean search(String word)` Returns true if the string word is in the trie (i.e., was inserted before), and false otherwise.
- `boolean startsWith(String prefix)` Returns true if there is a previously inserted string word that has the prefix prefix, and false otherwise.

Example 1:

Input

```
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
```

```
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
```

Output

```
[null, null, true, false, true, null, true]
```

Explanation

```
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");    // return True
trie.search("app");      // return False
trie.startsWith("app");  // return True
trie.insert("app");
trie.search("app");      // return True
```

Constraints:

- $1 \leq \text{word.length}, \text{prefix.length} \leq 2000$
- word and prefix consist only of lowercase English letters.
- At most $3 \cdot 10^4$ calls **in total** will be made to insert, search, and startsWith.

Word Search II

Given an $m \times n$ board of characters and a list of strings `words`, return *all words on the board*.

Each word must be constructed from letters of sequentially adjacent cells, where **adjacent cells** are horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Example 1:

Input: board = `[["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]]`,
words = `["oath","pea","eat","rain"]`
Output: `["eat","oath"]`

Example 2:

a	b
c	d

Input: board = `[["a","b"],["c","d"]]`, words = `["abcb"]`
Output: `[]`

Constraints:

- $m == \text{board.length}$
- $n == \text{board}[i].\text{length}$

- $1 \leq m, n \leq 12$
- `board[i][j]` is a lowercase English letter.
- $1 \leq \text{words.length} \leq 3 * 10^4$
- $1 \leq \text{words}[i].\text{length} \leq 10$
- `words[i]` consists of lowercase English letters.
- All the strings of `words` are unique.

Heap

Merge k Sorted Lists

You are given an array of k linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: `lists = [[1,4,5],[1,3,4],[2,6]]`

Output: `[1,1,2,3,4,4,5,6]`

Explanation: The linked-lists are:

```
[  
  1->4->5,  
  1->3->4,  
  2->6  
]
```

merging them into one sorted list:

`1->1->2->3->4->4->5->6`

Example 2:

Input: `lists = []`

Output: `[]`

Example 3:

Input: `lists = [[]]`

Output: `[]`

Constraints:

- `k == lists.length`

- $0 \leq k \leq 10^4$
- $0 \leq \text{lists}[i].\text{length} \leq 500$
- $-10^4 \leq \text{lists}[i][j] \leq 10^4$
- $\text{lists}[i]$ is sorted in **ascending order**.
- The sum of $\text{lists}[i].\text{length}$ won't exceed 10^4 .

Top K Frequent Elements

Given an integer array `nums` and an integer `k`, return *the k most frequent elements*. You may return the answer in **any order**.

Example 1:

Input: `nums = [1,1,1,2,2,3]`, `k = 2`

Output: `[1,2]`

Example 2:

Input: `nums = [1]`, `k = 1`

Output: `[1]`

Constraints:

- `1 <= nums.length <= 105`
- `k` is in the range `[1, the number of unique elements in the array]`.
- It is **guaranteed** that the answer is **unique**.

Follow up: Your algorithm's time complexity must be better than $O(n \log n)$, where `n` is the array's size.

Find Median from Data Stream

The **median** is the middle value in an ordered integer list. If the size of the list is even, there is no middle value and the median is the mean of the two middle values.

- For example, for `arr = [2, 3, 4]`, the median is 3.
- For example, for `arr = [2, 3]`, the median is $(2 + 3) / 2 = 2.5$.

Implement the `MedianFinder` class:

- `MedianFinder()` initializes the `MedianFinder` object.
- `void addNum(int num)` adds the integer `num` from the data stream to the data structure.
- `double findMedian()` returns the median of all elements so far. Answers within 10^{-5} of the actual answer will be accepted.

Example 1:

Input

```
["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]
```

```
[[], [1], [2], [], [3], []]
```

Output

```
[null, null, null, 1.5, null, 2.0]
```

Explanation

```
MedianFinder medianFinder = new MedianFinder();
medianFinder.addNum(1);      // arr = [1]
medianFinder.addNum(2);      // arr = [1, 2]
medianFinder.findMedian();    // return 1.5 (i.e., (1 + 2) / 2)
medianFinder.addNum(3);      // arr[1, 2, 3]
medianFinder.findMedian();    // return 2.0
```

Constraints:

- $-10^5 \leq \text{num} \leq 10^5$
- There will be at least one element in the data structure before calling `findMedian`.
- At most $5 * 10^4$ calls will be made to `addNum` and `findMedian`.

Follow up:

- If all integer numbers from the stream are in the range $[0, 100]$, how would you optimize your solution?
- If 99% of all integer numbers from the stream are in the range $[0, 100]$, how would you optimize your solution?

