
13. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

| Symbol | Value |
|--------|-------|
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1000 |

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

Input: s = "III"
Output: 3
Explanation: III = 3.

Example 2:

Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.

Example 3:

Input: s = "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints:

- 1 <= s.length <= 15
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is **guaranteed** that s is a valid roman numeral in the range [1, 3999].

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example 1:

Input: strs = ["flower","flow","flight"]
Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"]
Output: ""
Explanation: There is no common prefix among the input strings.

Constraints:

- 1 <= strs.length <= 200
 - 0 <= strs[i].length <= 200
 - strs[i] consists of only lower-case English letters.
-

20. 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`

Constraints:

- `1 <= s.length <= 104`
- `s` consists of parentheses only `'()[]{}'`.

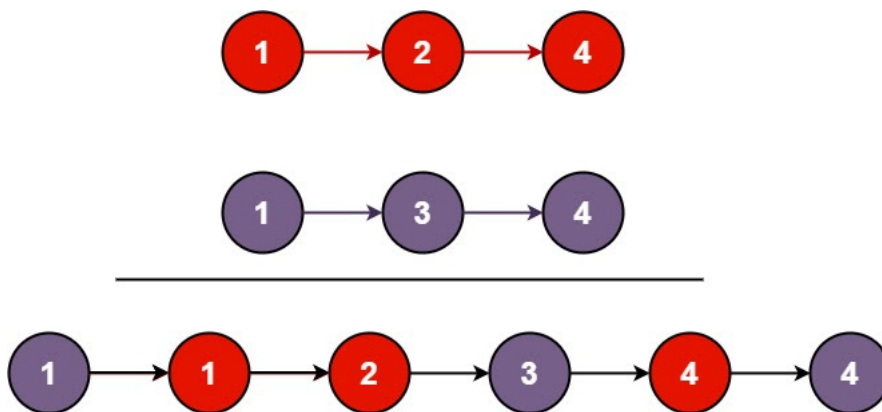
21. Merge Two Sorted Lists

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.

26. Remove Duplicates from Sorted Array

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first `k` slots of `nums`*.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with $O(1)$ extra memory.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

Input: `nums = [1,1,2]`
Output: `2`, `nums = [1,2,_]`
Explanation: Your function should return `k = 2`, with the first two elements of `nums` being 1 and 2 respectively. It does not matter what you leave beyond the returned `k` (hence they are underscores).

Example 2:

Input: `nums = [0,0,1,1,1,2,2,3,3,4]`
Output: `5`, `nums = [0,1,2,3,4,_,_,_,_,_]`
Explanation: Your function should return `k = 5`, with the first five elements of `nums` being 0, 1, 2, 3, and 4 respectively. It does not matter what you leave beyond the returned `k` (hence they are underscores).

Constraints:

- `1 <= nums.length <= 3 * 104`
- `-100 <= nums[i] <= 100`
- `nums` is sorted in **non-decreasing order**.

28. Implement strStr()

Implement [strStr\(\)](#).

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`.

Clarification:

What should we return when `needle` is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when `needle` is an empty string. This is consistent to C's [strstr\(\)](#) and Java's [indexOf\(\)](#).

Example 1:

Input: `haystack = "hello"`, `needle = "ll"`
Output: `2`

Example 2:

Input: `haystack = "aaaaa"`, `needle = "bba"`
Output: `-1`

Constraints:

- `1 <= haystack.length, needle.length <= 104`

- haystack and needle consist of only lowercase English characters.

35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order. You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: nums = [1,3,5,6], target = 5
Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2
Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7
Output: 4

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- nums contains **distinct** values sorted in **ascending** order.
- $-10^4 \leq \text{target} \leq 10^4$

53. 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:

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

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.

66. Plus One

You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the i^{th} digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives $123 + 1 = 124$.
Thus, the result should be [1,2,4].

Example 2:

Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives $4321 + 1 = 4322$.
Thus, the result should be [4,3,2,2].

Example 3:

Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives $9 + 1 = 10$.
Thus, the result should be [1,0].

Constraints:

- $1 \leq \text{digits.length} \leq 100$
- $0 \leq \text{digits}[i] \leq 9$
- digits does not contain any leading 0's.

67. Add Binary

Given two binary strings a and b, return *their sum as a binary string*.

Example 1:

Input: a = "11", b = "1"
Output: "100"

Example 2:

Input: a = "1010", b = "1011"
Output: "10101"

Constraints:

- $1 \leq \text{a.length}, \text{b.length} \leq 10^4$
- a and b consist only of '0' or '1' characters.
- Each string does not contain leading zeros except for the zero itself.

69. Sqrt(x)

Given a non-negative integer x, compute and return *the square root of x*.

Since the return type is an integer, the decimal digits are **truncated**, and only **the integer part** of the result is returned.

Note: You are not allowed to use any built-in exponent function or operator, such as `pow(x, 0.5)` or `x ** 0.5`.

Example 1:

Input: x = 4
Output: 2

Example 2:

Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

Constraints:

- $0 \leq x \leq 2^{31} - 1$

70. 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:

- $1 \leq n \leq 45$

88. Merge Sorted Array

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers m and n , representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of $m + n$, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. `nums2` has a length of n .

Example 1:

Input: `nums1 = [1,2,3,0,0,0]`, $m = 3$, `nums2 = [2,5,6]`, $n = 3$

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

Explanation: The arrays we are merging are `[1,2,3]` and `[2,5,6]`.

The result of the merge is `[1,2,2,3,5,6]` with the underlined elements coming from `nums1`.

Example 2:

Input: `nums1 = [1]`, $m = 1$, `nums2 = []`, $n = 0$

Output: `[1]`

Explanation: The arrays we are merging are `[1]` and `[]`.

The result of the merge is `[1]`.

Example 3:

Input: `nums1 = [0]`, $m = 0$, `nums2 = [1]`, $n = 1$

Output: `[1]`

Explanation: The arrays we are merging are `[]` and `[1]`.

The result of the merge is `[1]`.

Note that because $m = 0$, there are no elements in `nums1`. The 0 is only there to ensure the merge result can fit in `nums1`.

Constraints:

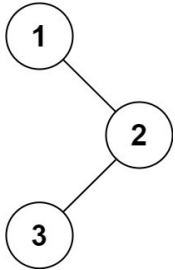
- `nums1.length == m + n`
- `nums2.length == n`
- $0 \leq m, n \leq 200$
- $1 \leq m + n \leq 200$
- $-10^9 \leq \text{nums1}[i], \text{nums2}[j] \leq 10^9$

Follow up: Can you come up with an algorithm that runs in $O(m + n)$ time?

94. Binary Tree Inorder Traversal

Given the root of a binary tree, return *the inorder traversal of its nodes' values*.

Example 1:



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

Example 2:

Input: root = []
Output: []

Example 3:

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

Constraints:

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

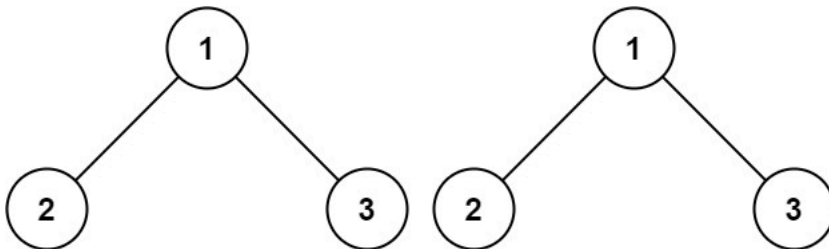
Follow up: Recursive solution is trivial, could you do it iteratively?

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.

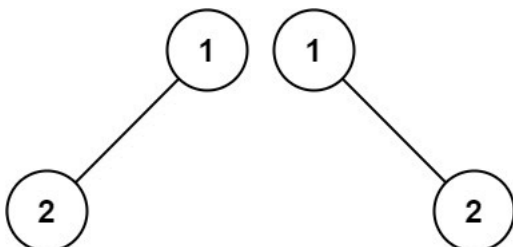
Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:



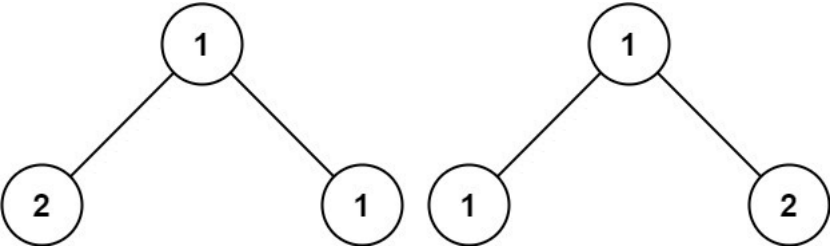
Input: p = [1,2,3], q = [1,2,3]
Output: true

Example 2:



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

Example 3:



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

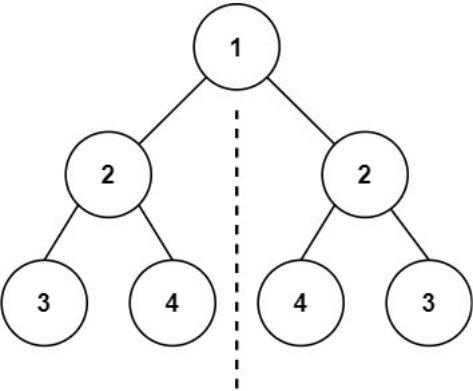
Constraints:

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

101. Symmetric Tree

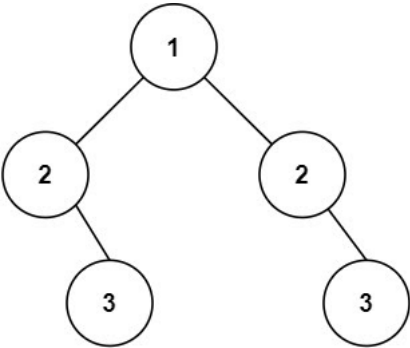
Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Example 1:



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

Example 2:



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

Constraints:

- The number of nodes in the tree is in the range [1, 1000].
- $-100 \leq \text{Node.val} \leq 100$

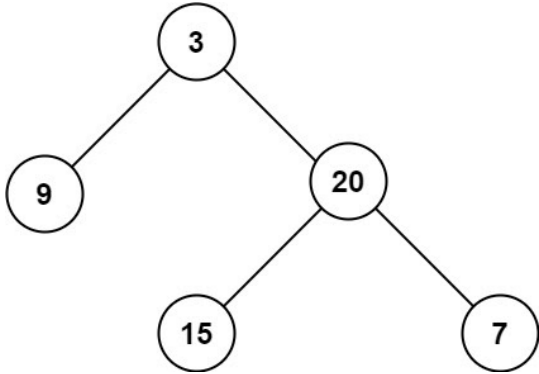
Follow up: Could you solve it both recursively and iteratively?

104. 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

Constraints:

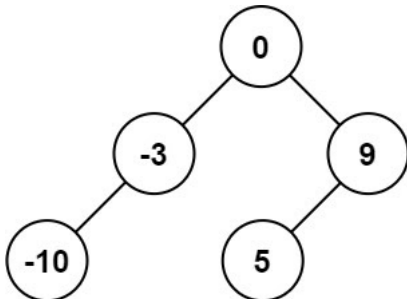
- The number of nodes in the tree is in the range $[0, 10^4]$.
- $-100 \leq \text{Node.val} \leq 100$

108. Convert Sorted Array to Binary Search Tree

Given an integer array **nums** where the elements are sorted in **ascending order**, convert *it to a **height-balanced** binary search tree*.

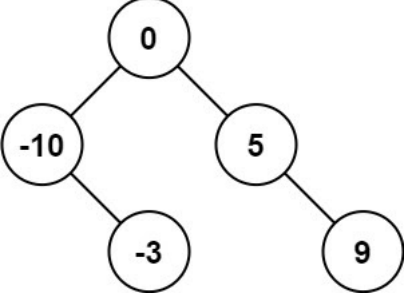
A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Example 1:

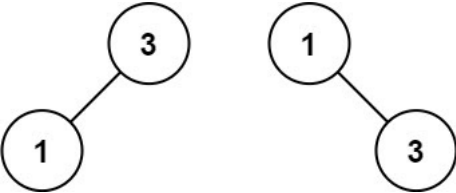


Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:



Example 2:



Input: nums = [1,3]
Output: [3,1]
Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Constraints:

- 1 <= nums.length <= 10⁴
- -10⁴ <= nums[i] <= 10⁴
- nums is sorted in a **strictly increasing** order.

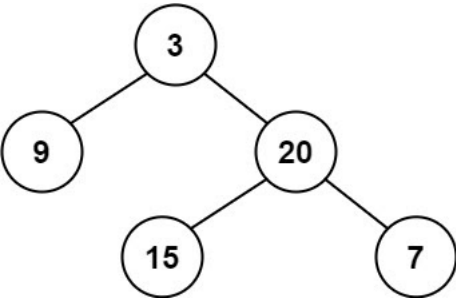
110. Balanced Binary Tree

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

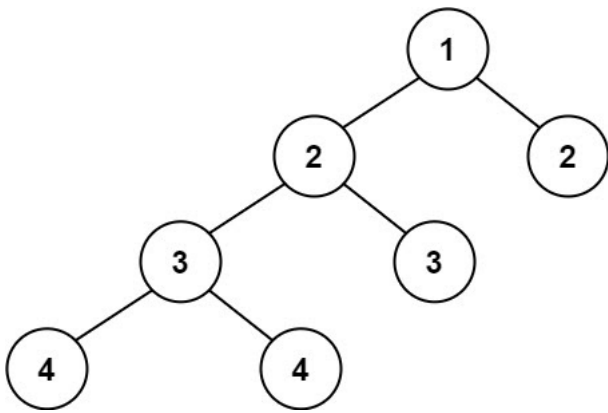
a binary tree in which the left and right subtrees of *every* node differ in height by no more than 1.

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:

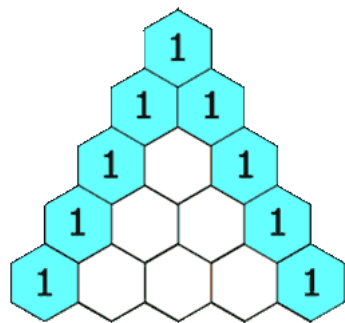
Input: root = []
Output: true

Constraints:

- The number of nodes in the tree is in the range [0, 5000].
- $-10^4 \leq \text{Node.val} \leq 10^4$

118. Pascal's Triangle

Given an integer numRows, return the first numRows of **Pascal's triangle**.
In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Example 2:

Input: numRows = 1
Output: [[1]]

Constraints:

- $1 \leq \text{numRows} \leq 30$

121. Best Time to Buy and Sell Stock

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:

- $1 \leq \text{prices.length} \leq 10^5$
- $0 \leq \text{prices}[i] \leq 10^4$

125. Valid Palindrome

A phrase is a **palindrome** if, 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:

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

136. Single Number

Given a **non-empty** array of integers *nums*, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input: nums = [2,2,1]

Output: 1

Example 2:

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

Output: 4

Example 3:

Input: nums = [1]

Output: 1

Constraints:

- $1 \leq \text{nums.length} \leq 3 \times 10^4$
- $-3 \times 10^4 \leq \text{nums}[i] \leq 3 \times 10^4$
- Each element in the array appears twice except for one element which appears only once.

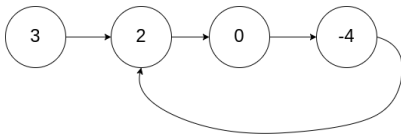
141. Linked List Cycle

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:

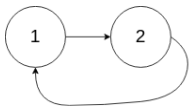


Input: head = [3,2,0,-4], pos = 1

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?

155. Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

- MinStack() initializes the stack object.
- void push(int val) pushes the element val onto the stack.
- void pop() removes the element on the top of the stack.
- int top() gets the top element of the stack.
- int getMin() retrieves the minimum element in the stack.

Example 1:

Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[[]]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top(); // return 0
minStack.getMin(); // return -2

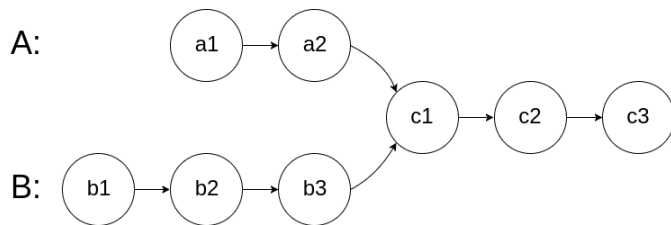
Constraints:

- $-2^{31} \leq \text{val} \leq 2^{31} - 1$
- Methods pop, top and getMin operations will always be called on **non-empty** stacks.
- At most $3 * 10^4$ calls will be made to push, pop, top, and getMin.

160. Intersection of Two Linked Lists

Given the heads of two singly linked-lists headA and headB, return *the node at which the two lists intersect*. If the two linked lists have no intersection at all, return null.

For example, the following two linked lists begin to intersect at node c1:



The test cases are generated such that there are no cycles anywhere in the entire linked structure.

Note that the linked lists must **retain their original structure** after the function returns.

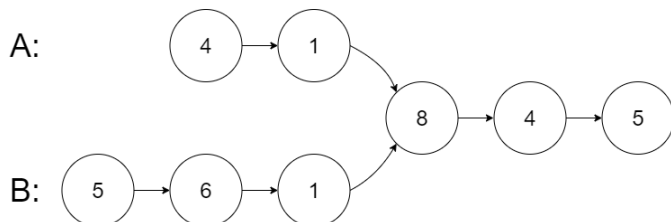
Custom Judge:

The inputs to the **judge** are given as follows (your program is **not** given these inputs):

- intersectVal - The value of the node where the intersection occurs. This is 0 if there is no intersected node.
- listA - The first linked list.
- listB - The second linked list.
- skipA - The number of nodes to skip ahead in listA (starting from the head) to get to the intersected node.
- skipB - The number of nodes to skip ahead in listB (starting from the head) to get to the intersected node.

The judge will then create the linked structure based on these inputs and pass the two heads, headA and headB to your program. If you correctly return the intersected node, then your solution will be **accepted**.

Example 1:

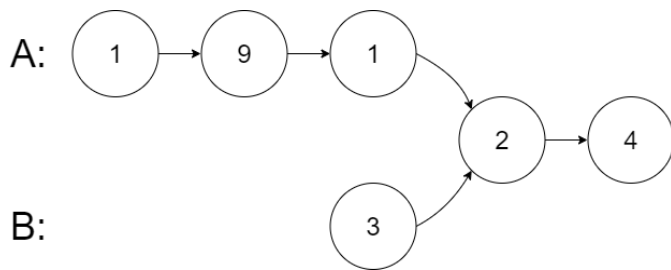


Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,6,1,8,4,5], skipA = 2, skipB = 3

Output: Intersected at '8'

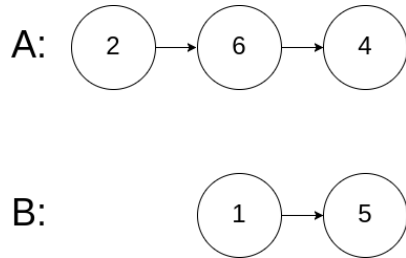
Explanation: The intersected node's value is 8 (note that this must not be 0 if the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of B, it reads as [5,6,1,8,4,5]. There are 2 nodes before the intersected node

Example 2:



Input: intersectVal = 2, listA = [1,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
Output: Intersected at '2'
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 [1,9,1,2,4]. From the head of B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A;

Example 3:



Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
Output: No intersection
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, and skipA and skipB can be arbitrary values.
 Explanation: The two lists do not intersect, so return null.

Constraints:

- The number of nodes of listA is in the m.
- The number of nodes of listB is in the n.
- $1 \leq m, n \leq 3 \times 10^4$
- $1 \leq \text{Node.val} \leq 10^5$
- $0 \leq \text{skipA} < m$
- $0 \leq \text{skipB} < n$
- intersectVal is 0 if listA and listB do not intersect.
- $\text{intersectVal} == \text{listA}[\text{skipA}] == \text{listB}[\text{skipB}]$ if listA and listB intersect.

Follow up: Could you write a solution that runs in $O(m + n)$ time and use only $O(1)$ memory?

 169. Majority Element

Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than $\lfloor n / 2 \rfloor$ times. You may assume that the majority element always exists in the array.

Example 1:

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

Example 2:

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

Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5 \times 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$

Follow-up: Could you solve the problem in linear time and in $O(1)$ space?

171. Excel Sheet Column Number

Given a string `columnTitle` that represents the 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: `columnTitle = "A"`
Output: 1

Example 2:

Input: `columnTitle = "AB"`
Output: 28

Example 3:

Input: `columnTitle = "ZY"`
Output: 701

Constraints:

- `1 <= columnTitle.length <= 7`
- `columnTitle` consists only of uppercase English letters.
- `columnTitle` is in the range ["A", "FXSHRXW"].

190. 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`.

Example 1:

Input: `n = 00000010100101000001111010011100`
Output: 964176192 (00111001011110000010100101000000)
Explanation: The input binary string `00000010100101000001111010011100` represents the unsigned integer 43261596, so return 964176192 which

Example 2:

Input: `n = 11111111111111111111111111111101`
Output: 3221225471 (10111111111111111111111111111111)
Explanation: The input binary string `11111111111111111111111111111101` represents the unsigned integer 4294967293, so return 3221225471 whi

Constraints:

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

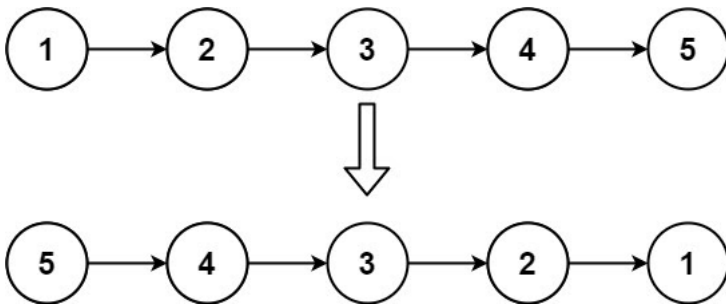
Follow up: If this function is called many times, how would you optimize it?

191. 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](#)).

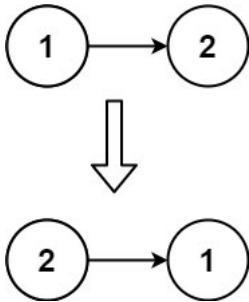
Note:

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 <= Node.val <= 5000

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

 217. Contains Duplicate

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

Input: nums = [1,2,3,1]
Output: true

Example 2:

Input: nums = [1,2,3,4]
Output: false

Example 3:

Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

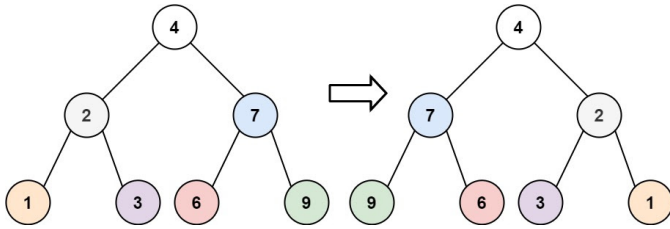
Constraints:

- 1 <= nums.length <= 10⁵
- -10⁹ <= nums[i] <= 10⁹

 226. Invert Binary Tree

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

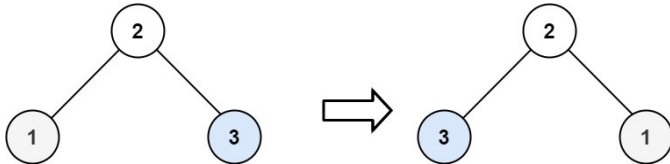
Example 1:



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

232. 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:

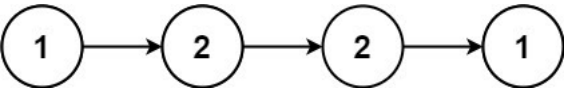
- $1 \leq x \leq 9$
- At most 100 calls will be made to `push`, `pop`, `peek`, and `empty`.
- All the calls to `pop` and `peek` are valid.

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

234. Palindrome Linked List

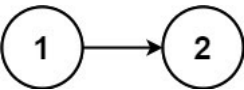
Given the head of a singly linked list, return `true` if it is a palindrome.

Example 1:



Input: head = [1,2,2,1]
Output: true

Example 2:



Input: head = [1,2]
Output: false

Constraints:

- The number of nodes in the list is in the range $[1, 10^5]$.
- $0 \leq \text{Node.val} \leq 9$

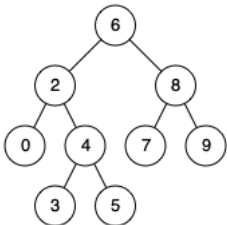
Follow up: Could you do it in $O(n)$ time and $O(1)$ space?

235. 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.

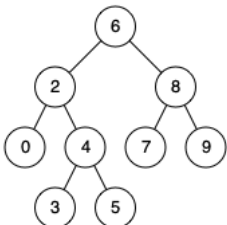
According to the [definition of LCA on Wikipedia](#): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

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:

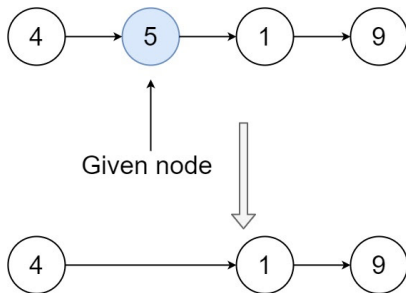
- The number of nodes in the tree is in the range $[2, 10^5]$.
- $-10^9 \leq \text{Node.val} \leq 10^9$
- All `Node.val` are **unique**.
- $p \neq q$
- p and q will exist in the BST.

237. Delete Node in a Linked List

Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list, instead you will be given access to **the node to be deleted** directly.

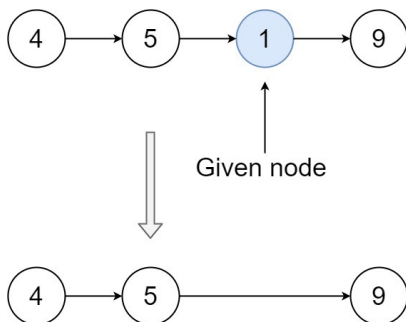
It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

Example 1:



Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become 4 → 1 → 9 after calling your function.

Example 2:



Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should become 4 → 5 → 9 after calling your function.

Constraints:

- The number of the nodes in the given list is in the range $[2, 1000]$.
- $-1000 \leq \text{Node.val} \leq 1000$
- The value of each node in the list is **unique**.
- The node to be deleted is **in the list** and is **not a tail node**

242. Valid Anagram

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:

- $1 \leq s.length, t.length \leq 5 \times 10^4$
- 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?

268. 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*.

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 the array.

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 the array.

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 the array.

Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 10^4$
- $0 \leq \text{nums}[i] \leq n$
- All the numbers of nums are **unique**.

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

278. First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions $[1, 2, \dots, n]$ and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

Input: n = 5, bad = 4
Output: 4
Explanation:
call `isBadVersion(3)` -> false
call `isBadVersion(5)` -> true

call `isBadVersion(4)` -> `true`
Then 4 is the first bad version.

Example 2:

Input: `n = 1, bad = 1`
Output: `1`

Constraints:

- $1 \leq \text{bad} \leq n \leq 2^{31} - 1$

283. Move Zeroes

Given an integer array `nums`, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

Input: `nums = [0,1,0,3,12]`
Output: `[1,3,12,0,0]`

Example 2:

Input: `nums = [0]`
Output: `[0]`

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

Follow up: Could you minimize the total number of operations done?

326. Power of Three

Given an integer `n`, return *true* if it is a power of three. Otherwise, return *false*.

An integer `n` is a power of three, if there exists an integer `x` such that $n == 3^x$.

Example 1:

Input: `n = 27`
Output: `true`

Example 2:

Input: `n = 0`
Output: `false`

Example 3:

Input: `n = 9`
Output: `true`

Constraints:

- $-2^{31} \leq n \leq 2^{31} - 1$

Follow up: Could you solve it without loops/recursion?

338. Counting Bits

Given an integer n , return *an array* *ans* of length $n + 1$ such that for each i ($0 \leq i \leq n$), $\text{ans}[i]$ 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++)?

344. Reverse String

Write a function that reverses a string. The input string is given as an array of characters s .

You must do this by modifying the input array [in-place](#) with $O(1)$ extra memory.

Example 1:

Input: $s = ["h", "e", "l", "l", "o"]$
Output: $["o", "l", "l", "e", "h"]$

Example 2:

Input: $s = ["H", "a", "n", "n", "a", "h"]$
Output: $["h", "a", "n", "n", "a", "H"]$

Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$ is a [printable ascii character](#).

350. Intersection of Two Arrays II

Given two integer arrays nums1 and nums2 , return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

Example 1:

Input: $\text{nums1} = [1, 2, 2, 1]$, $\text{nums2} = [2, 2]$
Output: $[2, 2]$

Example 2:

Input: $\text{nums1} = [4, 9, 5]$, $\text{nums2} = [9, 4, 9, 8, 4]$
Output: $[4, 9]$
Explanation: $[9, 4]$ is also accepted.

Constraints:

- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if `nums1`'s size is small compared to `nums2`'s size? Which algorithm is better?
- What if elements of `nums2` are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

383. Ransom Note

Given two strings `ransomNote` and `magazine`, return `true` if *ransomNote can be constructed from magazine and false otherwise*.

Each letter in `magazine` can only be used once in `ransomNote`.

Example 1:

Input: `ransomNote = "a", magazine = "b"`
Output: `false`

Example 2:

Input: `ransomNote = "aa", magazine = "ab"`
Output: `false`

Example 3:

Input: `ransomNote = "aa", magazine = "aab"`
Output: `true`

Constraints:

- `1 <= ransomNote.length, magazine.length <= 105`
- `ransomNote` and `magazine` consist of lowercase English letters.

387. First Unique Character in a String

Given a string `s`, *find the first non-repeating character in it and return its index*. If it does not exist, return `-1`.

Example 1:

Input: `s = "leetcode"`
Output: `0`

Example 2:

Input: `s = "loveleetcode"`
Output: `2`

Example 3:

Input: `s = "aabb"`
Output: `-1`

Constraints:

- `1 <= s.length <= 105`
- `s` consists of only lowercase English letters.

409. Longest Palindrome

Given a string `s` which consists of lowercase or uppercase letters, return *the length of the **longest palindrome*** that can be built with those letters.

Letters are **case sensitive**, for example, "Aa" is not considered a palindrome here.

Example 1:

Input: `s = "abcccd"`

Output: `7`

Explanation:

One longest palindrome that can be built is "dcccdd", whose length is 7.

Example 2:

Input: `s = "a"`

Output: `1`

Example 3:

Input: `s = "bb"`

Output: `2`

Constraints:

- `1 <= s.length <= 2000`
- `s` consists of lowercase **and/or** uppercase English letters only.

412. Fizz Buzz

Given an integer `n`, return *a string array answer (**1-indexed**)* where:

- `answer[i] == "FizzBuzz"` if `i` is divisible by 3 and 5.
- `answer[i] == "Fizz"` if `i` is divisible by 3.
- `answer[i] == "Buzz"` if `i` is divisible by 5.
- `answer[i] == i` (as a string) if none of the above conditions are true.

Example 1:

Input: `n = 3`

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

Example 2:

Input: `n = 5`

Output: `["1", "2", "Fizz", "4", "Buzz"]`

Example 3:

Input: `n = 15`

Output: `["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz"]`

Constraints:

- `1 <= n <= 104`

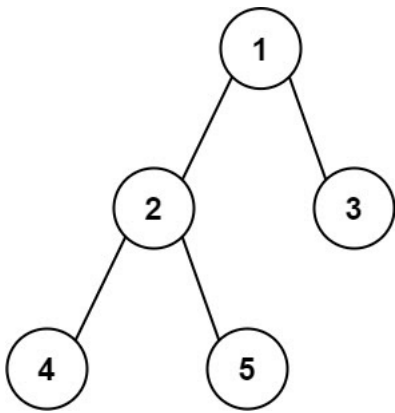
543. Diameter of Binary Tree

Given the root of a binary tree, return *the length of the **diameter** of the tree*.

The **diameter** of a binary tree is the **length** of the longest path between any two nodes in a tree. This path may or may not pass through the **root**.

The **length** of a path between two nodes is represented by the number of edges between them.

Example 1:



Input: root = [1,2,3,4,5]
Output: 3
Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3].

Example 2:

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

Constraints:

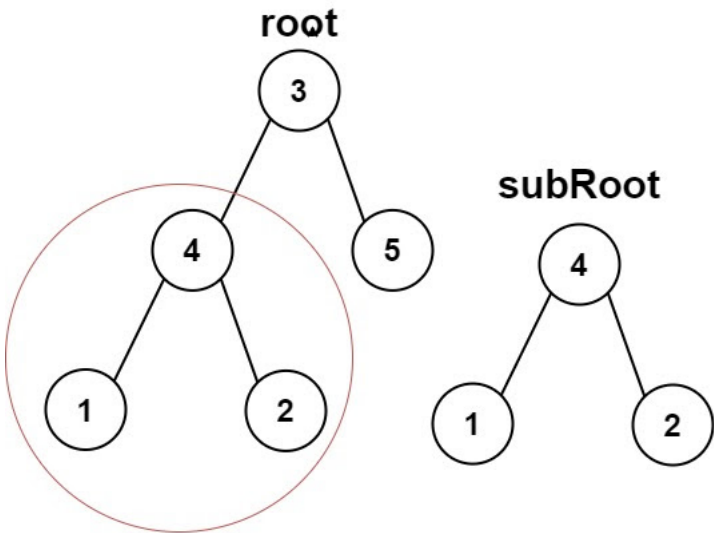
- The number of nodes in the tree is in the range [1, 10⁴].
- -100 <= Node.val <= 100

 572. 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.

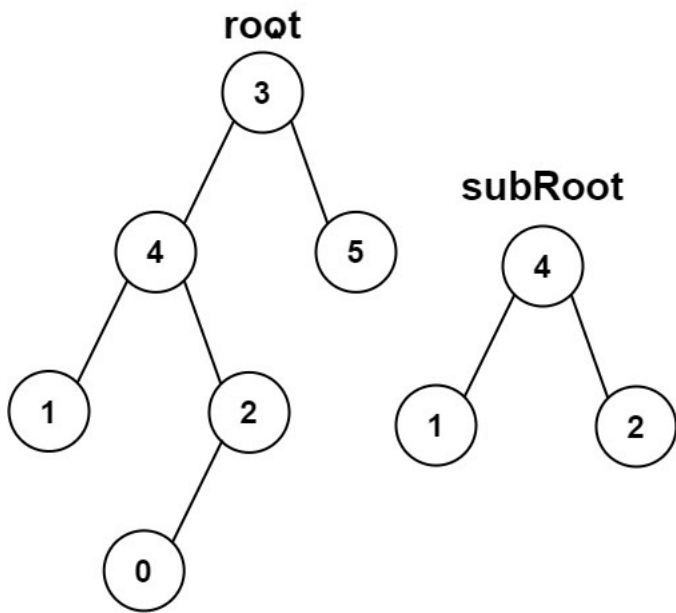
A subtree of a binary tree tree is a tree that consists of a node in tree and all of this node's descendants. The tree tree could also be considered as a subtree of itself.

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].
- $-10^4 \leq \text{root.val} \leq 10^4$
- $-10^4 \leq \text{subRoot.val} \leq 10^4$

 617. Merge Two Binary Trees

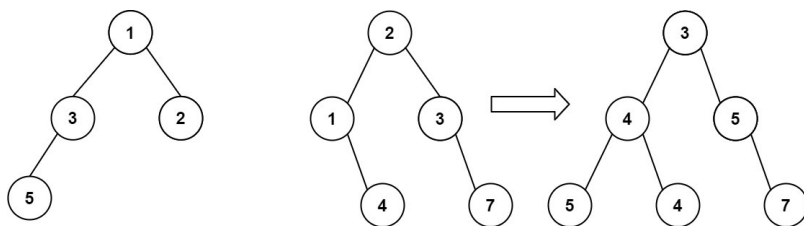
You are given two binary trees root1 and root2.

Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return *the merged tree*.

Note: The merging process must start from the root nodes of both trees.

Example 1:



Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]

Example 2:

Input: root1 = [1], root2 = [1,2]
Output: [2,2]

Constraints:

- The number of nodes in both trees is in the range [0, 2000].
- $-10^4 \leq \text{Node.val} \leq 10^4$

704. Binary Search

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:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 < \text{nums}[i], \text{target} < 10^4$
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

733. Flood Fill

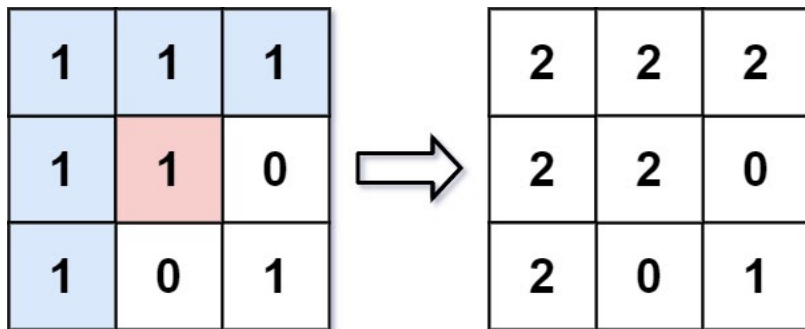
An image is represented by an $m \times n$ integer grid `image` where `image[i][j]` represents the pixel value of the image.

You are also given three integers `sr`, `sc`, and `newColor`. 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 `newColor`.

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`, `newColor = 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, 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 `newColor`. 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`, `newColor = 2`
Output: `[[2,2,2],[2,2,2]]`

Constraints:

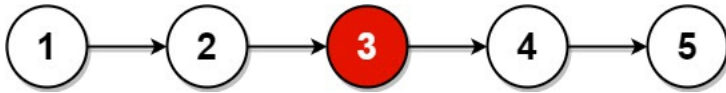
- $m == \text{image.length}$
- $n == \text{image}[i].\text{length}$
- $1 \leq m, n \leq 50$
- $0 \leq \text{image}[i][j], \text{newColor} < 2^{16}$
- $0 \leq \text{sr} < m$
- $0 \leq \text{sc} < n$

876. Middle of the Linked List

Given the head of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle node**.

Example 1:

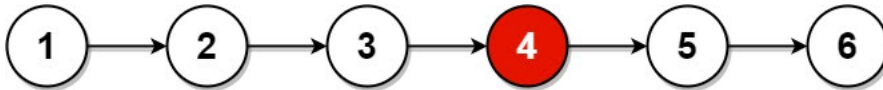


Input: head = [1,2,3,4,5]

Output: [3,4,5]

Explanation: The middle node of the list is node 3.

Example 2:



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

Output: [4,5,6]

Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

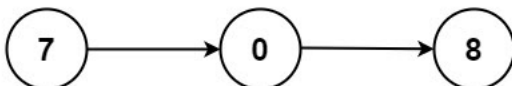
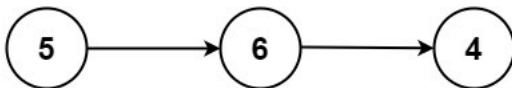
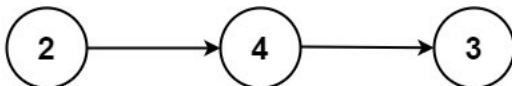
- The number of nodes in the list is in the range [1, 100].
- $1 \leq \text{Node.val} \leq 100$

2. Add Two Numbers

You are given two **non-empty** linked lists representing two non-negative integers. The digits are stored in **reverse order**, and each of their nodes contains a single digit. Add the two numbers and return the sum as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:



Input: l1 = [2,4,3], l2 = [5,6,4]

Output: [7,0,8]

Explanation: 342 + 465 = 807.

Example 2:

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

Output: [0]

Example 3:

Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

Output: [8,9,9,9,0,0,0,1]

Constraints:

- The number of nodes in each linked list is in the range $[1, 100]$.
- $0 \leq \text{Node.val} \leq 9$
- It is guaranteed that the list represents a number that does not have leading zeros.

3. 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* = "bbbbb"

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.

Constraints:

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

5. Longest Palindromic Substring

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

Example 1:

Input: *s* = "babad"

Output: "bab"

Explanation: "aba" is also a valid answer.

Example 2:

Input: *s* = "cbdd"

Output: "bb"

Constraints:

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

7. Reverse Integer

Given a signed 32-bit integer *x*, return *x with its digits reversed*. If reversing *x* causes the value to go outside the signed 32-bit integer range $[-2^{31}, 2^{31} - 1]$, then return 0.

Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

Example 1:

Input: *x* = 123

Output: 321

Example 2:

Input: x = -123
Output: -321

Example 3:

Input: x = 120
Output: 21

Constraints:

- $-2^{31} \leq x \leq 2^{31} - 1$

8. String to Integer (atoi)

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer (similar to C/C++'s `atoi` function).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than -2^{31} should be clamped to -2^{31} , and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result.

Note:

- Only the space character ' ' is considered a whitespace character.
- **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:

Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
 ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
 ^
Step 3: "42" ("42" is read in)
 ^
The parsed integer is 42.
Since 42 is in the range $[-2^{31}, 2^{31} - 1]$, the final result is 42.

Example 2:

Input: s = " -42"
Output: -42
Explanation:
Step 1: "___-42" (leading whitespace is read and ignored)
 ^
Step 2: " -42" ('-' is read, so the result should be negative)
 ^
Step 3: " -42" ("42" is read in)
 ^
The parsed integer is -42.
Since -42 is in the range $[-2^{31}, 2^{31} - 1]$, the final result is -42.

Example 3:

Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
 ^
Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
 ^
Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
 ^
The parsed integer is 4193.
Since 4193 is in the range $[-2^{31}, 2^{31} - 1]$, the final result is 4193.

Constraints:

- $0 \leq s.length \leq 200$
- s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '.'.

11. Container With Most Water

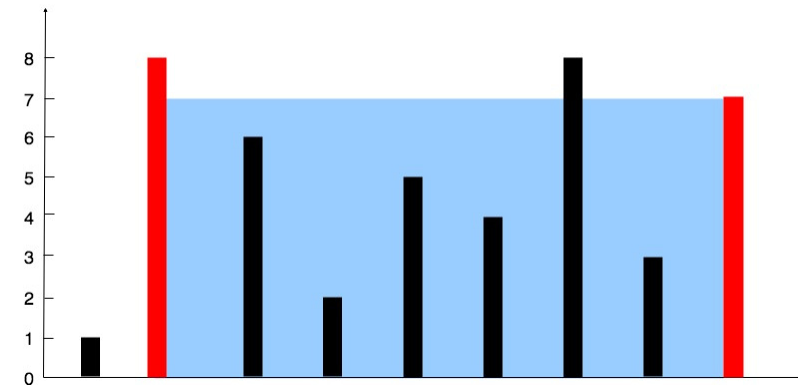
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

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

Example 2:

Input: `height = [1,1]`

Output: 1

Constraints:

- `n == height.length`
- `2 <= n <= 105`
- `0 <= height[i] <= 104`

15. 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 $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] == 0$.

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 <= nums.length <= 3000`
- `-105 <= nums[i] <= 105`

17. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.
A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.



Example 1:
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

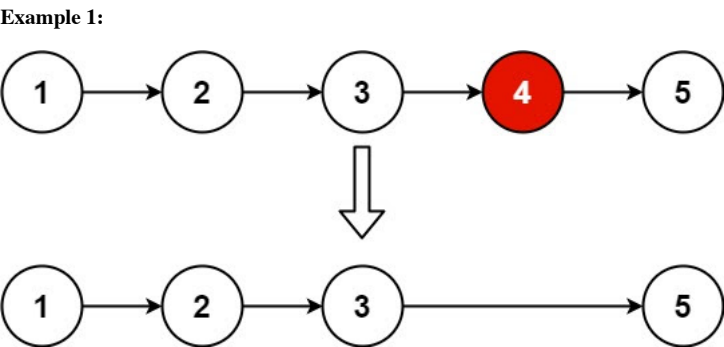
Example 2:
Input: digits = ""
Output: []

Example 3:
Input: digits = "2"
Output: ["a","b","c"]

- Constraints:**
- 0 <= digits.length <= 4
 - digits[i] is a digit in the range ['2', '9'].

19. 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.



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?

22. Generate Parentheses

Given n pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

Example 1:

Input: n = 3
Output: ["((()))","(()())","(())()","()()()","()()()"]

Example 2:

Input: n = 1
Output: ["()"]

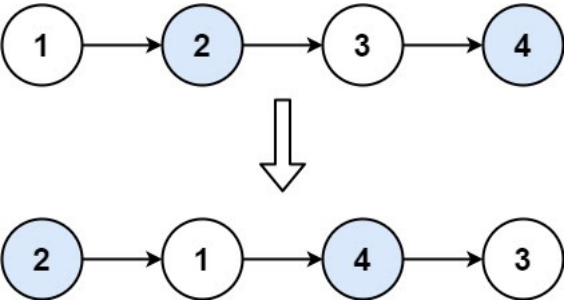
Constraints:

- 1 <= n <= 8

24. Swap Nodes in Pairs

Given a linked list, swap every two adjacent nodes and return its head. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.)

Example 1:



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

Example 2:

Input: head = []
Output: []

Example 3:

Input: head = [1]
Output: [1]

Constraints:

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

29. Divide Two Integers

Given two integers dividend and divisor, divide two integers **without** using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, 8.345 would be truncated to 8 , and -2.7335 would be truncated to -2 .

Return *the quotient after dividing* dividend by divisor.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is **strictly greater than** $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is **strictly less than** -2^{31} , then return -2^{31} .

Example 1:

Input: dividend = 10, divisor = 3
Output: 3
Explanation: $10/3 = 3.33333\dots$ which is truncated to 3.

Example 2:

Input: dividend = 7, divisor = -3
Output: -2
Explanation: $7/-3 = -2.33333\dots$ which is truncated to -2.

Constraints:

- $-2^{31} \leq \text{dividend}$, $\text{divisor} \leq 2^{31} - 1$
- $\text{divisor} \neq 0$

31. Next Permutation

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for $\text{arr} = [1, 2, 3]$, the following are considered permutations of arr : $[1, 2, 3]$, $[1, 3, 2]$, $[3, 1, 2]$, $[2, 3, 1]$.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of $\text{arr} = [1, 2, 3]$ is $[1, 3, 2]$.
- Similarly, the next permutation of $\text{arr} = [2, 3, 1]$ is $[3, 1, 2]$.
- While the next permutation of $\text{arr} = [3, 2, 1]$ is $[1, 2, 3]$ because $[3, 2, 1]$ does not have a lexicographical larger rearrangement.

Given an array of integers *nums*, *find the next permutation of* *nums*.

The replacement must be [in place](#) and use only constant extra memory.

Example 1:

Input: *nums* = $[1, 2, 3]$
Output: $[1, 3, 2]$

Example 2:

Input: *nums* = $[3, 2, 1]$
Output: $[1, 2, 3]$

Example 3:

Input: *nums* = $[1, 1, 5]$
Output: $[1, 5, 1]$

Constraints:

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

33. 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 **possibly rotated** at an unknown pivot index k ($1 \leq k < \text{nums.length}$) such that the resulting array is $[\text{nums}[k], \text{nums}[k+1], \dots, \text{nums}[n-1], \text{nums}[0], \text{nums}[1], \dots, \text{nums}[k-1]]$ (**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 possible 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 an ascending array that is possibly rotated.
- $-10^4 \leq \text{target} \leq 10^4$

34. Find First and Last Position of Element in Sorted Array

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value.

If target is not found in the array, return [-1, -1].

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

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]

Example 3:

Input: nums = [], target = 0
Output: [-1,-1]

Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- nums is a non-decreasing array.
- $-10^9 \leq \text{target} \leq 10^9$

36. Valid Sudoku

Determine if a 9×9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits 1-9 without repetition.
2. Each column must contain the digits 1-9 without repetition.
3. Each of the nine 3×3 sub-boxes of the grid must contain the digits 1-9 without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

```

Input: board =
[["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],
[".","9","8",".",".",".","6","."],
["8",".",".","6",".",".","3"],
["4",".","8","3",".","1"],
["7",".","2",".","6"],
[".","6",".","2","8","."],
[".","4","1","9",".","5"],
[".","8","7","9"]]
Output: true

```

Example 2:

```

Input: board =
[["8","3",".","7",".",".","."],
["6",".","1","9","5",".","."],
[".","9","8",".","6","."],
["8",".","6",".","3"],
["4",".","8","3","1"],
["7",".","2",".","6"],
[".","6",".","2","8","."],
[".","4","1","9",".","5"],
[".","8","7","9"]]
Output: false
Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3

```

Constraints:

- board.length == 9
- board[i].length == 9
- board[i][j] is a digit 1-9 or '.'.

```

*****
38. Count and Say
*****

```

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- countAndSay(1) = "1"
- countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of groups so that each group is a contiguous section all of the **same character**. Then for each group, say the number of characters, then say the character. To convert the saying into a digit string, replace the counts with a number and concatenate every saying.

For example, the saying and conversion for digit string "3322251":

"3322251"

two 3's, three 2's, one 5, and one 1

2 3 + 3 2 + 1 5 + 1 1

"23321511"

Given a positive integer n, return *the* nth *term of the* **count-and-say** *sequence*.

Example 1:

```

Input: n = 1
Output: "1"
Explanation: This is the base case.

```

Example 2:

```

Input: n = 4
Output: "1211"

```

Explanation:
countAndSay(1) = "1"
countAndSay(2) = say "1" = one 1 = "11"
countAndSay(3) = say "11" = two 1's = "21"
countAndSay(4) = say "21" = one 2 + one 1 = "12" + "11" = "1211"

Constraints:

- 1 <= n <= 30

39. Combination Sum

Given an array of **distinct** integers candidates and a target integer target, return *a list of all **unique combinations** of candidates where the chosen numbers sum to target*. You may return the combinations in **any order**.

The **same** number may be chosen from candidates an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.

Example 2:

Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:

Input: candidates = [2], target = 1
Output: []

Constraints:

- 1 <= candidates.length <= 30
- 1 <= candidates[i] <= 200
- All elements of candidates are **distinct**.
- 1 <= target <= 500

45. Jump Game II

Given an array of non-negative integers nums, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

Example 1:

Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

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

Constraints:

- 1 <= nums.length <= 10⁴
- 0 <= nums[i] <= 1000

46. Permutations

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

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

Example 2:

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

Example 3:

Input: `nums = [1]`
Output: `[[1]]`

Constraints:

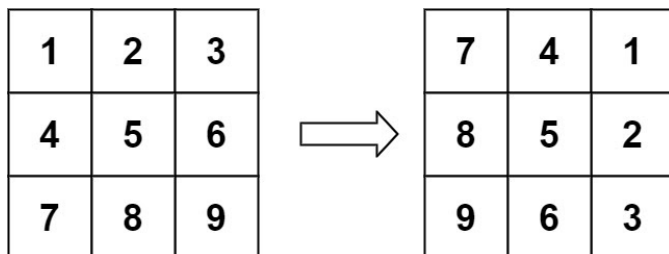
- `1 <= nums.length <= 6`
- `-10 <= nums[i] <= 10`
- All the integers of `nums` are **unique**.

48. Rotate Image

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

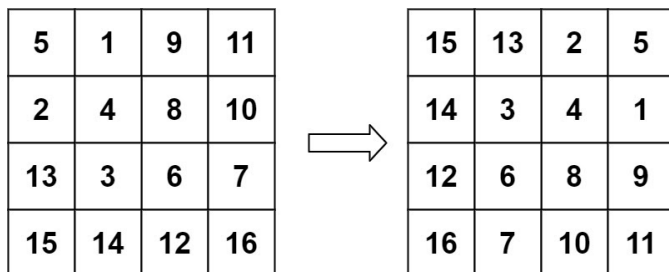
You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

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:



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]]`

Constraints:

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

49. 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:

- $1 \leq \text{strs.length} \leq 10^4$
- $0 \leq \text{strs}[i].\text{length} \leq 100$
- `strs[i]` consists of lowercase English letters.

50. Pow(x, n)

Implement [`pow\(x, n\)`](#), which calculates x raised to the power n (i.e., x^n).

Example 1:

Input: `x = 2.00000, n = 10`
Output: `1024.00000`

Example 2:

Input: `x = 2.10000, n = 3`
Output: `9.26100`

Example 3:

Input: `x = 2.00000, n = -2`
Output: `0.25000`
Explanation: $2^{-2} = 1/2^2 = 1/4 = 0.25$

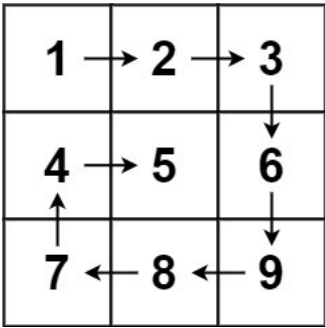
Constraints:

- $-100.0 < x < 100.0$
- $-2^{31} \leq n \leq 2^{31}-1$
- $-10^4 \leq x^n \leq 10^4$

54. Spiral Matrix

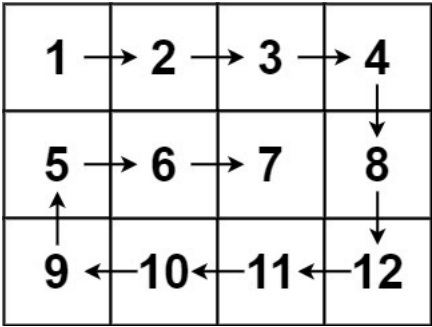
Given an $m \times n$ matrix, return *all elements of the matrix in spiral order*.

Example 1:



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

Example 2:



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

55. 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

56. Merge Intervals

Given an array of intervals where `intervals[i] = [starti, endi]`, 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:

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

57. 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].

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 `starti` in **ascending** order.
- $\text{newInterval.length} == 2$
- $0 \leq \text{start} \leq \text{end} \leq 10^5$

62. Unique Paths

There is a robot on an $m \times n$ grid. The robot is initially located at the **top-left corner** (i.e., `grid[0][0]`). The robot tries to move to the **bottom-right corner** (i.e., `grid[m - 1][n - 1]`). The robot can only move either down or right at any point in time.

Given the two integers `m` and `n`, return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The test cases are generated so that the answer will be less than or equal to $2 * 10^9$.

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

Constraints:

- 1 <= m, n <= 100

64. Minimum Path Sum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:

| | | |
|---|---|---|
| 1 | 3 | 1 |
| 1 | 5 | 1 |
| 4 | 2 | 1 |

Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 -> 3 -> 1 -> 1 -> 1 minimizes the sum.

Example 2:

Input: grid = [[1,2,3],[4,5,6]]
Output: 12

Constraints:

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 200
- 0 <= grid[i][j] <= 100


73. Set Matrix Zeroes

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

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?

74. Search a 2D Matrix

Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

Example 1:

| | | | |
|----|----|----|----|
| 1 | 3 | 5 | 7 |
| 10 | 11 | 16 | 20 |
| 23 | 30 | 34 | 60 |

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Example 2:

| | | | |
|----|----|----|----|
| 1 | 3 | 5 | 7 |
| 10 | 11 | 16 | 20 |
| 23 | 30 | 34 | 60 |

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false

Constraints:

- m == matrix.length
- n == matrix[i].length
- 1 <= m, n <= 100
- $-10^4 \leq \text{matrix}[i][j], \text{target} \leq 10^4$

 75. Sort Colors

Given an array nums with n objects colored red, white, or blue, sort them [in-place](#) so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

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

Example 2:

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

Constraints:

- n == nums.length
- 1 <= n <= 300
- nums[i] is either 0, 1, or 2.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?

 78. Subsets

Given an integer array nums of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

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

Example 2:

Input: nums = [0]
Output: [[],[0]]

Constraints:

- 1 <= nums.length <= 10
- -10 <= nums[i] <= 10
- All the numbers of nums are **unique**.

79. Word Search

Given an m x 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?

91. 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 test cases are generated so that the answer fits 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 = "06"`
Output: 0
Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06").

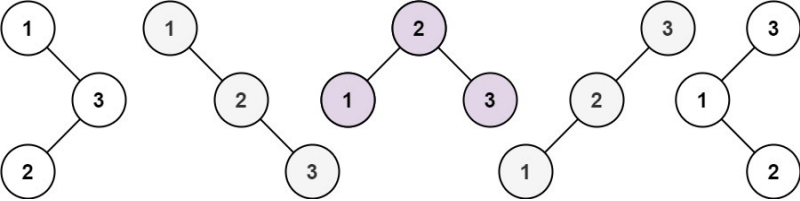
Constraints:

- 1 <= `s.length` <= 100
- `s` contains only digits and may contain leading zero(s).

96. Unique Binary Search Trees

Given an integer `n`, return *the number of structurally unique **BST**'s (binary search trees) which has exactly `n` nodes of unique values from 1 to `n`.*

Example 1:



Input: `n = 3`
Output: 5

Example 2:

Input: `n = 1`
Output: 1

Constraints:

- 1 <= `n` <= 19

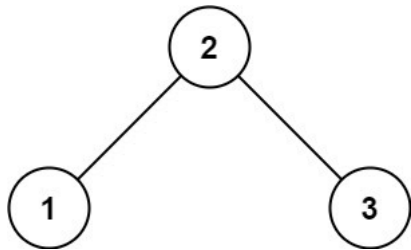
98. Validate Binary Search Tree

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

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

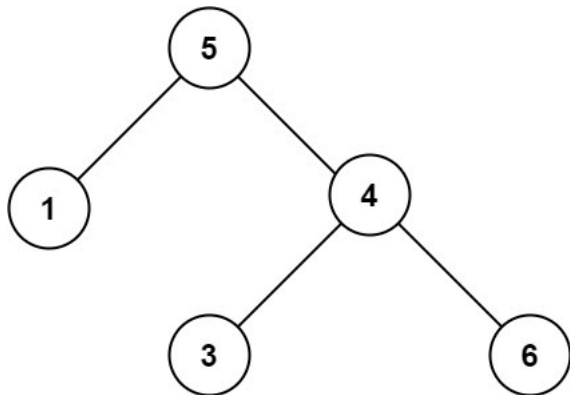
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

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.

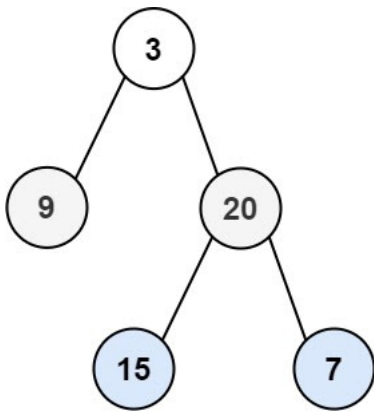
Constraints:

- The number of nodes in the tree is in the range [1, 10⁴].
- -2³¹ <= Node.val <= 2³¹ - 1

102. 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: []

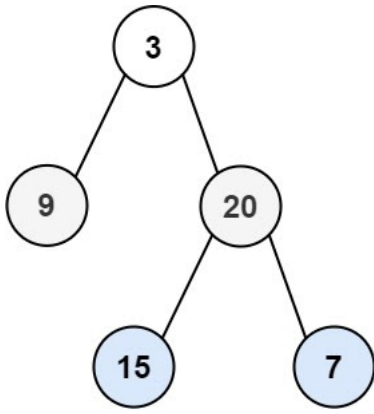
Constraints:

- The number of nodes in the tree is in the range [0, 2000].
- -1000 <= Node.val <= 1000

 103. Binary Tree Zigzag Level Order Traversal

Given the root of a binary tree, return *the zigzag level order traversal of its nodes' values*. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:



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

Example 2:

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

Example 3:

Input: root = []
Output: []

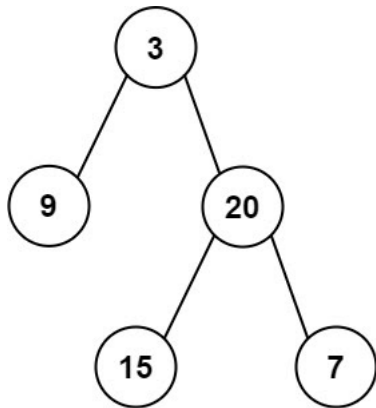
Constraints:

- The number of nodes in the tree is in the range [0, 2000].
- -100 <= Node.val <= 100

105. 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:

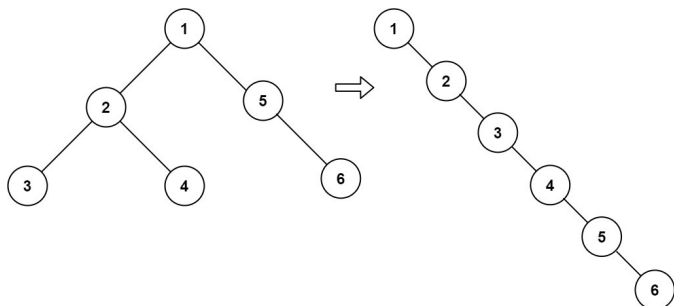
- `1 <= preorder.length <= 3000`
- `inorder.length == preorder.length`
- `-3000 <= preorder[i], inorder[i] <= 3000`
- `preorder` and `inorder` consist of **unique** values.
- Each value of `inorder` also appears in `preorder`.
- `preorder` is **guaranteed** to be the preorder traversal of the tree.
-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

114. Flatten Binary Tree to Linked List

Given the `root` of a binary tree, flatten the tree into a "linked list":

- The "linked list" should use the same `TreeNode` class where the `right` child pointer points to the next node in the list and the `left` child pointer is always `null`.
- The "linked list" should be in the same order as a [pre-order traversal](#) of the binary tree.

Example 1:



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

Example 2:

Input: `root = []`
Output: `[]`

Example 3:

Input: root = [0]
Output: [0]

Constraints:

- The number of nodes in the tree is in the range [0, 2000].
- $-100 \leq \text{Node.val} \leq 100$

Follow up: Can you flatten the tree in-place (with $O(1)$ extra space)?

116. Populating Next Right Pointers in Each Node

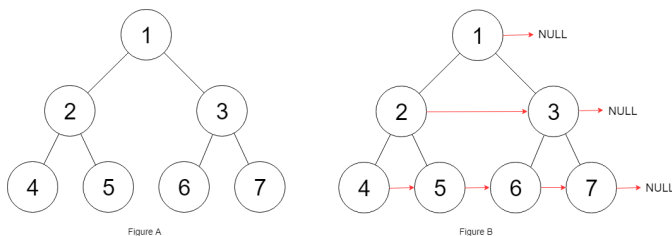
You are given a **perfect binary tree** where all leaves are on the same level, and every parent has two children. The binary tree has the following definition:

```
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 1:



Input: root = [1,2,3,4,5,6,7]
Output: [1,#,2,3,#,4,5,6,7,#]
Explanation: Given the above perfect binary tree (Figure A), your function should populate each next pointer to point to its next right node.

Example 2:

Input: root = []
Output: []

Constraints:

- The number of nodes in the tree is in the range [0, $2^{12} - 1$].
- $-1000 \leq \text{Node.val} \leq 1000$

Follow-up:

- You may only use constant extra space.
- The recursive approach is fine. You may assume implicit stack space does not count as extra space for this problem.

122. Best Time to Buy and Sell Stock II

You are given an integer array prices where prices[i] is the price of a given stock on the i^{th} day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return the *maximum profit* you can achieve.

Example 1:

Input: prices = [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.
Total profit is 4 + 3 = 7.

Example 2:

Input: prices = [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.
Total profit is 4.

Example 3:

Input: prices = [7,6,4,3,1]
Output: 0
Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints:

- 1 <= prices.length <= 3 * 10⁴
- 0 <= prices[i] <= 10⁴

128. 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

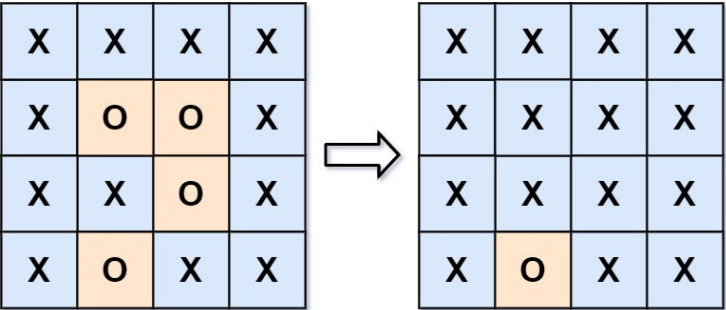
Constraints:

- 0 <= nums.length <= 10⁵
- -10⁹ <= nums[i] <= 10⁹

130. Surrounded Regions

Given an m x n matrix board containing 'x' and 'o', *capture all regions that are 4-directionally surrounded by 'x'*.
A region is **captured** by flipping all 'o's into 'x's in that surrounded region.

Example 1:



Input: board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]
Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]
Explanation: Surrounded regions should not be on the border, which means that any 'O' on the border of the board are not flipped to 'X'. 7

Example 2:

Input: board = [["X"]]
Output: [["X"]]

Constraints:

- m == board.length
- n == board[i].length
- 1 <= m, n <= 200
- board[i][j] is 'X' or 'O'.

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.

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

Example 1:

Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]

Example 2:

Input: s = "a"
Output: [["a"]]

Constraints:

- 1 <= s.length <= 16
- s contains only lowercase English letters.

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

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

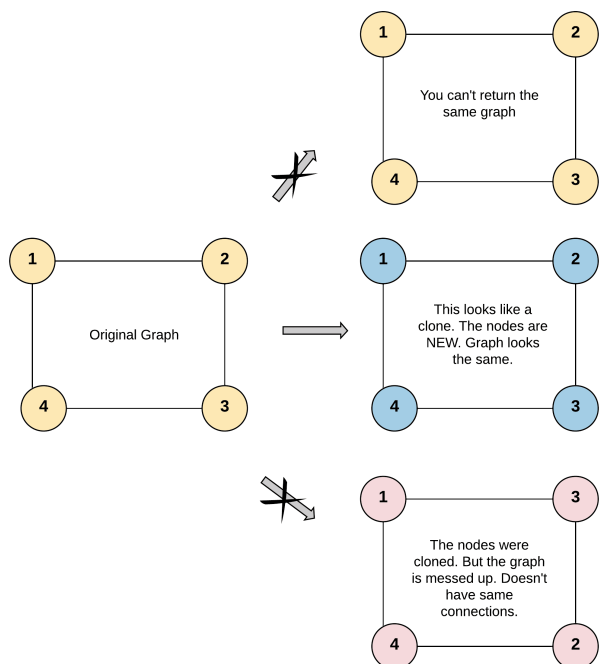
Test case format:

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 is an empty graph, it does not have any nodes.

Constraints:

- The number of nodes in the graph is in the range [0, 100].
- 1 ≤ Node.val ≤ 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.

134. Gas Station

There are n gas stations along a circular route, where the amount of gas at the i^{th} station is $\text{gas}[i]$.

You have a car with an unlimited gas tank and it costs $\text{cost}[i]$ of gas to travel from the i^{th} station to its next $(i + 1)^{\text{th}}$ station. You begin the journey with an empty tank at one of the gas stations.

Given two integer arrays gas and cost , return the starting gas station's index if you can travel around the circuit once in the clockwise direction, otherwise return -1. If there exists a solution, it is **guaranteed** to be **unique**.

Example 1:

Input: $\text{gas} = [1,2,3,4,5]$, $\text{cost} = [3,4,5,1,2]$

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.

Constraints:

- $n == \text{gas.length} == \text{cost.length}$
- $1 \leq n \leq 10^5$
- $0 \leq \text{gas}[i], \text{cost}[i] \leq 10^4$

138. Copy List with Random Pointer

A linked list of length n is given such that each node contains an additional random pointer, which could point to any node in the list, or null.

Construct a [deep copy](#) of the list. The deep copy should consist of exactly n **brand new** nodes, where each new node has its value set to the value of its corresponding original node. Both the next and random pointer of the new nodes should point to new nodes in the copied list such that the pointers in the original list and copied list represent the same list state. **None of the pointers in the new list should point to nodes in the original list.**

For example, if there are two nodes x and y in the original list, where $x.\text{random} \rightarrow y$, then for the corresponding two nodes x and y in the copied list, $x.\text{random} \rightarrow y$.

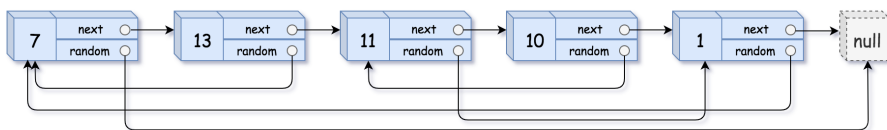
Return *the head of the copied linked list*.

The linked list is represented in the input/output as a list of n nodes. Each node is represented as a pair of $[\text{val}, \text{random_index}]$ where:

- val**: an integer representing `Node.val`
- random_index**: the index of the node (range from 0 to $n-1$) that the random pointer points to, or null if it does not point to any node.

Your code will **only** be given the head of the original linked list.

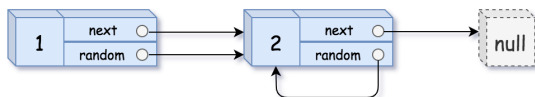
Example 1:



Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]

Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]

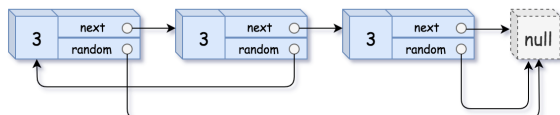
Example 2:



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

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

Example 3:



Input: head = [[3,null],[3,0],[3,null]]

Output: [[3,null],[3,0],[3,null]]

Constraints:

- $0 \leq n \leq 1000$
- $-10^4 \leq \text{Node.val} \leq 10^4$
- `Node.random` is null or is pointing to some node in the linked list.

139. 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:

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

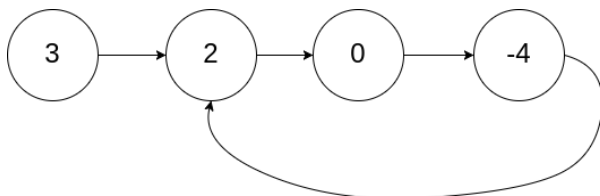
142. Linked List Cycle II

Given the head of a linked list, return *the node where the cycle begins*. If there is no cycle, return `null`.

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 (**0-indexed**). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter**.

Do not modify the linked list.

Example 1:

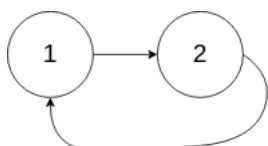


Input: `head = [3,2,0,-4], pos = 1`

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.

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?

143. Reorder List

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

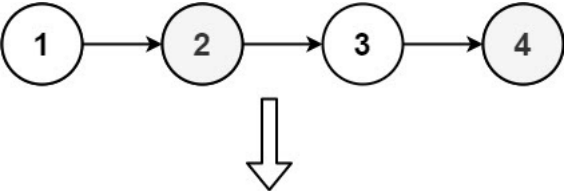
$L_0 \rightarrow L_1 \rightarrow \dots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

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

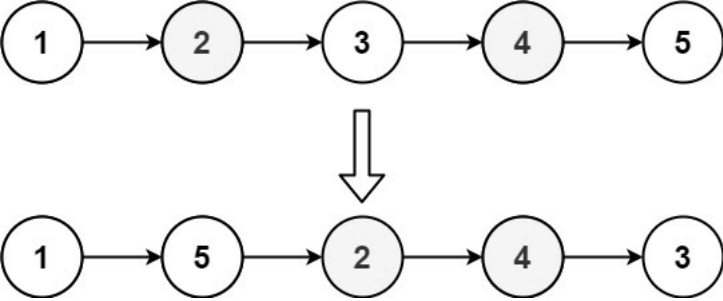
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

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 * 10⁴].
- 1 <= Node.val <= 1000

146. LRU Cache

Design a data structure that follows the constraints of a [Least Recently Used \(LRU\) cache](#).

Implement the LRUCache class:

- LRUCache(int capacity) Initialize the LRU cache with **positive** size capacity.
- int get(int key) Return the value of the key if the key exists, otherwise return -1.
- void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, **evict** the least recently used key.

The functions get and put must each run in $O(1)$ average time complexity.

Example 1:

Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1); // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2); // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1); // return -1 (not found)
lRUCache.get(3); // return 3
lRUCache.get(4); // return 4

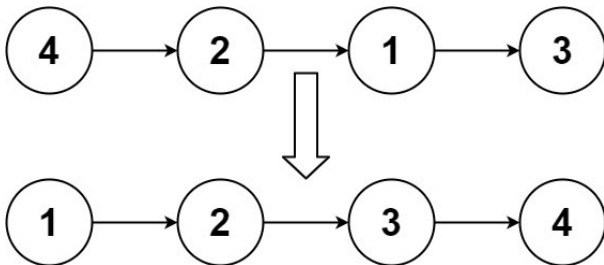
Constraints:

- $1 \leq \text{capacity} \leq 3000$
- $0 \leq \text{key} \leq 10^4$
- $0 \leq \text{value} \leq 10^5$
- At most $2 * 10^5$ calls will be made to get and put.

148. Sort List

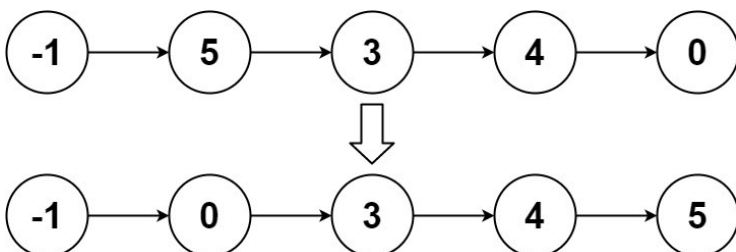
Given the head of a linked list, return *the list after sorting it in ascending order*.

Example 1:



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

Example 2:



Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]

Example 3:

Input: head = []
Output: []

Constraints:

- The number of nodes in the list is in the range $[0, 5 \cdot 10^4]$.
- $-10^5 \leq \text{Node.val} \leq 10^5$

Follow up: Can you sort the linked list in $O(n \log n)$ time and $O(1)$ memory (i.e. constant space)?

150. Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in [Reverse Polish Notation](#).

Valid operators are +, -, *, and /. Each operand may be an integer or another expression.

Note that division between two integers should truncate toward zero.

It is guaranteed that the given RPN expression is always valid. That means the expression would always evaluate to a result, and there will not be any division by zero operation.

Example 1:

Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: $((2 + 1) * 3) = 9$

Example 2:

Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: $(4 + (13 / 5)) = 6$

Example 3:

Input: tokens = ["10","6","9","3","+","-11","*","/","*", "17","+","5","+"]
Output: 22
Explanation: $((10 * (6 / ((9 + 3) * -11))) + 17) + 5$
 $= (((10 * (6 / (12 * -11))) + 17) + 5$
 $= (((10 * (6 / -132)) + 17) + 5$
 $= ((10 * 0) + 17) + 5$
 $= (0 + 17) + 5$
 $= 17 + 5$
 $= 22$

Constraints:

- $1 \leq \text{tokens.length} \leq 10^4$
- $\text{tokens}[i]$ is either an operator: "+", "-", "*", or "/", or an integer in the range $[-200, 200]$.

152. 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*.

The test cases are generated so 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:

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

153. 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:

- $n == \text{nums.length}$
- $1 \leq n \leq 5000$
- $-5000 \leq \text{nums}[i] \leq 5000$
- All the integers of `nums` are **unique**.
- `nums` is sorted and rotated between 1 and n times.

162. Find Peak Element

A peak element is an element that is strictly greater than its neighbors.

Given an integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = $-\infty$` .

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

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: 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.

Constraints:

- `1 <= nums.length <= 1000`
- `-231 <= nums[i] <= 231 - 1`
- `nums[i] != nums[i + 1]` for all valid `i`.

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.

If multiple answers are possible, return **any of them**.

It is **guaranteed** that the length of the answer string is less than 10^4 for all the given inputs.

Example 1:

Input: numerator = 1, denominator = 2

Output: "0.5"

Example 2:

Input: numerator = 2, denominator = 1

Output: "2"

Example 3:

Input: numerator = 4, denominator = 333

Output: "0.(012)"

Constraints:

- `-231 <= numerator, denominator <= 231 - 1`
- `denominator != 0`

172. Factorial Trailing Zeroes

Given an integer `n`, return *the number of trailing zeroes in n!*.

Note that `n! = n * (n - 1) * (n - 2) * ... * 3 * 2 * 1`.

Example 1:

Input: `n = 3`

Output: 0

Explanation: `3! = 6`, no trailing zero.

Example 2:

Input: `n = 5`

Output: 1

Explanation: `5! = 120`, one trailing zero.

Example 3:

Input: `n = 0`

Output: 0

Constraints:

- `0 <= n <= 104`

Follow up: Could you write a solution that works in logarithmic time complexity?

179. Largest Number

Given a list of non-negative integers `nums`, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

Example 1:

Input: `nums = [10,2]`
Output: `"210"`

Example 2:

Input: `nums = [3,30,34,5,9]`
Output: `"9534330"`

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 109`

189. Rotate Array

Given an array, rotate the array to the right by `k` steps, where `k` is non-negative.

Example 1:

Input: `nums = [1,2,3,4,5,6,7], 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: `nums = [-1,-100,3,99], 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]`

Constraints:

- `1 <= nums.length <= 105`
- `-231 <= nums[i] <= 231 - 1`
- `0 <= k <= 105`

Follow up:

- Try to come up with as many solutions as you can. There are at least **three** different ways to solve this problem.
- Could you do it in-place with $O(1)$ extra space?

198. 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.

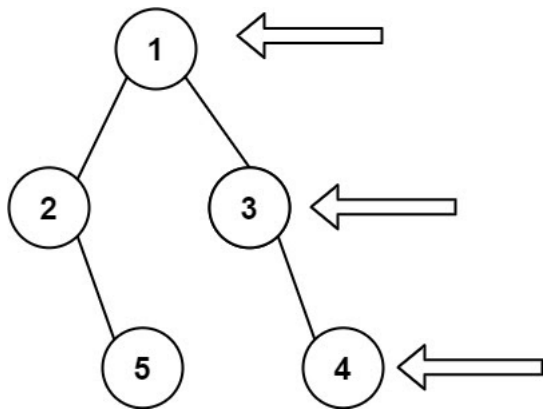
Constraints:

- 1 <= nums.length <= 100
- 0 <= nums[i] <= 400

199. Binary Tree Right Side View

Given the root of 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 1:



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

Example 2:

Input: root = [1,null,3]
Output: [1,3]

Example 3:

Input: root = []
Output: []

Constraints:

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

200. Number of Islands

Given an m x 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"],
 ["1","1","0","0","0"],
 ["1","1","0","0","0"]
]
Output: 1


```
[ "1", "1", "0", "0", "0" ],
[ "0", "0", "1", "0", "0" ],
[ "0", "0", "0", "1", "1" ]
]
```

Output: 3

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is '0' or '1'.

204. Count Primes

Given an integer `n`, return *the number of prime numbers that are strictly less than n*.

Example 1:

Input: `n = 10`
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:

Input: `n = 0`
Output: 0

Example 3:

Input: `n = 1`
Output: 0

Constraints:

- `0 <= n <= 5 * 106`

207. 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**.
-

208. 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 $3 * 10^4$ calls in total will be made to insert, search, and startsWith.

210. Course Schedule II

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] = [a_i, b_i] indicates that you **must** take course b_i first if you want to take course a_i.

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

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return an empty array.

Example 1:

Input: numCourses = 2, prerequisites = [[1,0]]
Output: [0,1]
Explanation: There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1].

Example 2:

Input: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
Output: [0,2,1,3]
Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both courses 1 and 2. Both courses 1 and 2 So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].

Example 3:

Input: numCourses = 1, prerequisites = []
Output: [0]

Constraints:

- $1 \leq \text{numCourses} \leq 2000$
 - $0 \leq \text{prerequisites.length} \leq \text{numCourses} * (\text{numCourses} - 1)$
 - prerequisites[i].length == 2
 - $0 \leq a_i, b_i < \text{numCourses}$
 - $a_i \neq b_i$
 - All the pairs [a_i, b_i] are **distinct**.
-

211. Design Add and Search Words Data Structure

Design a data structure that supports adding new words and finding if a string matches any previously added string.

Implement the `WordDictionary` class:

- `WordDictionary()` Initializes the object.
- `void addWord(word)` Adds word to the data structure, it can be matched later.
- `bool search(word)` Returns `true` if there is any string in the data structure that matches `word` or `false` otherwise. `word` may contain dots `'.'` where dots can be matched with any letter.

Example:

Input
["WordDictionary", "addWord", "addWord", "addWord", "search", "search", "search", "search"]
[[], ["bad"], ["dad"], ["mad"], ["pad"], ["bad"], [".ad"], ["b.."]]
Output
[null, null, null, null, false, true, true, true]

Explanation
`WordDictionary wordDictionary = new WordDictionary();`
`wordDictionary.addWord("bad");`
`wordDictionary.addWord("dad");`
`wordDictionary.addWord("mad");`
`wordDictionary.search("pad"); // return False`
`wordDictionary.search("bad"); // return True`
`wordDictionary.search(".ad"); // return True`
`wordDictionary.search("b.."); // return True`

Constraints:

- `1 <= word.length <= 25`
- `word` in `addWord` consists of lowercase English letters.
- `word` in `search` consist of `'.'` or lowercase English letters.
- There will be at most 3 dots in `word` for `search` queries.
- At most 10^4 calls will be made to `addWord` and `search`.

213. 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`

215. Kth Largest Element in an Array

Given an integer array `nums` and an integer `k`, return *the k^{th} largest element in the array*.

Note that it is the k^{th} largest element in the sorted order, not the k^{th} distinct element.

Example 1:

Input: `nums = [3,2,1,5,6,4]`, `k = 2`
Output: 5

Example 2:

Input: `nums = [3,2,3,1,2,4,5,5,6]`, `k = 4`
Output: 4

Constraints:

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

221. Maximal Square

Given an `m x n` binary matrix filled with 0's and 1's, *find the largest square containing only 1's and return its area*.

Example 1:

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

Input: `matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]`
Output: 4

Example 2:

| | |
|---|---|
| 0 | 1 |
| 1 | 0 |

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

Example 3:

Input: `matrix = [["0"]]`
Output: 0

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 300`
- `matrix[i][j]` is '0' or '1'.

227. Basic Calculator II

Given a string `s` which represents an expression, *evaluate this expression and return its value*.

The integer division should truncate toward zero.

You may assume that the given expression is always valid. All intermediate results will be in the range of $[-2^{31}, 2^{31} - 1]$.

Note: You are not allowed to use any built-in function which evaluates strings as mathematical expressions, such as `eval()`.

Example 1:

Input: `s = "3+2*2"`
Output: 7

Example 2:

Input: `s = " 3/2 "`
Output: 1

Example 3:

Input: `s = " 3+5 / 2 "`
Output: 5

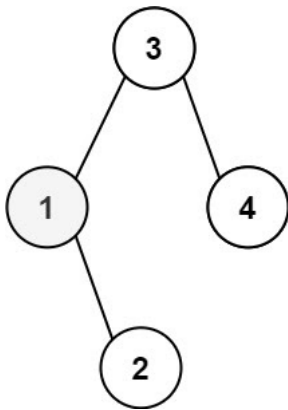
Constraints:

- $1 \leq s.length \leq 3 \times 10^5$
- `s` consists of integers and operators ('+', '-', '*', '/') separated by some number of spaces.
- `s` represents a **valid expression**.
- All the integers in the expression are non-negative integers in the range $[0, 2^{31} - 1]$.
- The answer is **guaranteed** to fit in a **32-bit integer**.

230. Kth Smallest Element in a BST

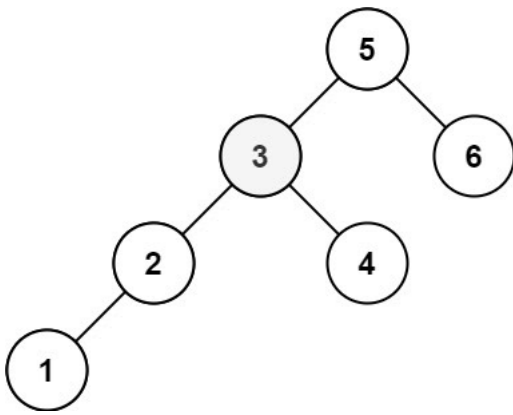
Given the `root` of a binary search tree, and an integer `k`, return the k^{th} smallest value (**1-indexed**) of all the values of the nodes in the tree.

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 <= k <= n <= 10⁴
- 0 <= Node.val <= 10⁴

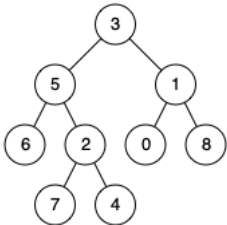
Follow up: If the BST is modified often (i.e., we can do insert and delete operations) and you need to find the kth smallest frequently, how would you optimize?

 236. Lowest Common Ancestor of a Binary Tree

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

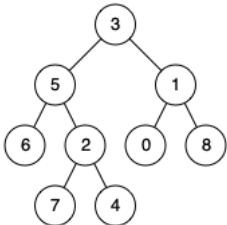
According to the [definition of LCA on Wikipedia](#): “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

Example 1:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.

Example 2:



Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself according to the LCA definition.

Example 3:

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

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 tree.

 238. 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.)

 240. Search a 2D Matrix II

Write an efficient algorithm that searches for a value `target` in an $m \times n$ integer matrix `matrix`. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Example 1:

| | | | | |
|----|----|----|----|----|
| 1 | 4 | 7 | 11 | 15 |
| 2 | 5 | 8 | 12 | 19 |
| 3 | 6 | 9 | 16 | 22 |
| 10 | 13 | 14 | 17 | 24 |
| 18 | 21 | 23 | 26 | 30 |

Input: `matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]]`, `target = 5`
Output: `true`

Example 2:

| | | | | |
|----|----|----|----|----|
| 1 | 4 | 7 | 11 | 15 |
| 2 | 5 | 8 | 12 | 19 |
| 3 | 6 | 9 | 16 | 22 |
| 10 | 13 | 14 | 17 | 24 |
| 18 | 21 | 23 | 26 | 30 |

Input: matrix = [[1,4,7,11,15],[2,5,8,12,19],[3,6,9,16,22],[10,13,14,17,24],[18,21,23,26,30]], target = 20
Output: false

Constraints:

- m == matrix.length
- n == matrix[i].length
- 1 <= n, m <= 300
- $-10^9 \leq \text{matrix}[i][j] \leq 10^9$
- All the integers in each row are **sorted** in ascending order.
- All the integers in each column are **sorted** in ascending order.
- $-10^9 \leq \text{target} \leq 10^9$

279. Perfect Squares

Given an integer n, return *the least number of perfect square numbers that sum to n*.

A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, 1, 4, 9, and 16 are perfect squares while 3 and 11 are not.

Example 1:

Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.

Example 2:

Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.

Constraints:

- 1 <= n <= 10⁴

287. Find the Duplicate Number

Given an array of integers nums containing n + 1 integers where each integer is in the range [1, n] inclusive.

There is only **one repeated number** in nums, return *this repeated number*.

You must solve the problem **without** modifying the array nums and uses only constant extra space.

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq n \leq 10^5$
- `nums.length == n + 1`
- $1 \leq \text{nums}[i] \leq n$
- All the integers in `nums` appear only **once** except for **precisely one integer** which appears **two or more** times.

Follow up:

- How can we prove that at least one duplicate number must exist in `nums`?
- Can you solve the problem in linear runtime complexity?

289. Game of Life

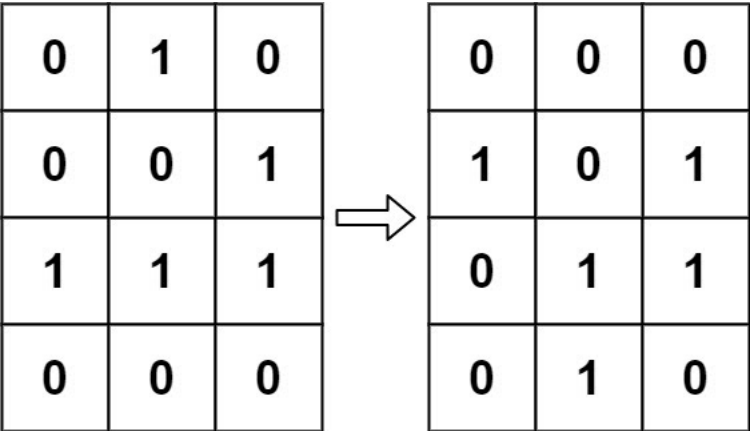
According to [Wikipedia's article](#): "The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

The board is made up of an $m \times n$ grid of cells, where each cell has an initial state: **live** (represented by a 1) or **dead** (represented by a 0). Each cell interacts with its [eight neighbors](#) (horizontal, vertical, diagonal) using the following four rules (taken from the above Wikipedia article):

1. Any live cell with fewer than two live neighbors dies as if caused by under-population.
2. Any live cell with two or three live neighbors lives on to the next generation.
3. Any live cell with more than three live neighbors dies, as if by over-population.
4. Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

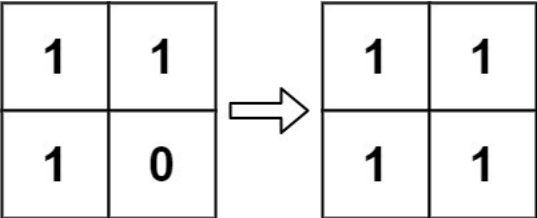
The next state is created by applying the above rules simultaneously to every cell in the current state, where births and deaths occur simultaneously. Given the current state of the $m \times n$ grid board, return *the next state*.

Example 1:



Input: board = [[0,1,0],[0,0,1],[1,1,1],[0,0,0]]
Output: [[0,0,0],[1,0,1],[0,1,1],[0,1,0]]

Example 2:



Input: board = [[1,1],[1,0]]
Output: [[1,1],[1,1]]

Constraints:

- $m == \text{board.length}$
- $n == \text{board}[i].length$
- $1 \leq m, n \leq 25$
- `board[i][j]` is 0 or 1.

Follow up:

- Could you solve it in-place? Remember that the board needs to be updated simultaneously: You cannot update some cells first and then use their updated values to update other cells.
- In this question, we represent the board using a 2D array. In principle, the board is infinite, which would cause problems when the active area encroaches upon the border of the array (i.e., live cells reach the border). How would you address these problems?

300. 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]`

Output: 1

Constraints:

- $1 \leq \text{nums.length} \leq 2500$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

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

310. Minimum Height Trees

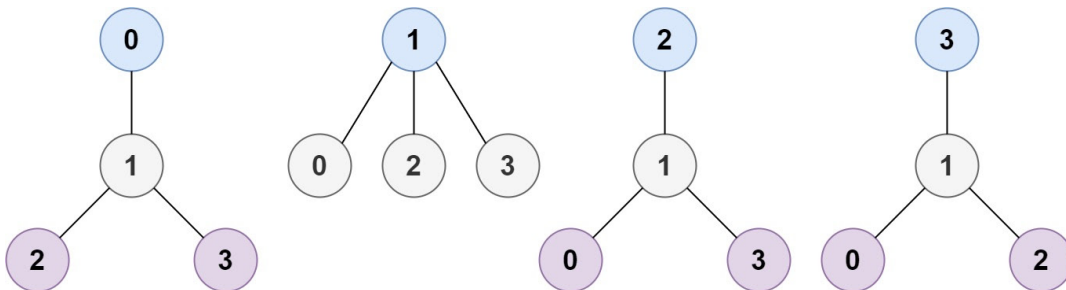
A tree is an undirected graph in which any two vertices are connected by *exactly* one path. In other words, any connected graph without simple cycles is a tree.

Given a tree of n nodes labelled from 0 to $n - 1$, and an array of $n - 1$ edges where `edges[i] = [ai, bi]` indicates that there is an undirected edge between the two nodes a_i and b_i in the tree, you can choose any node of the tree as the root. When you select a node x as the root, the result tree has height h . Among all possible rooted trees, those with minimum height (i.e. $\min(h)$) are called **minimum height trees** (MHTs).

Return a list of all **MHTs'** root labels. You can return the answer in **any order**.

The **height** of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

Example 1:

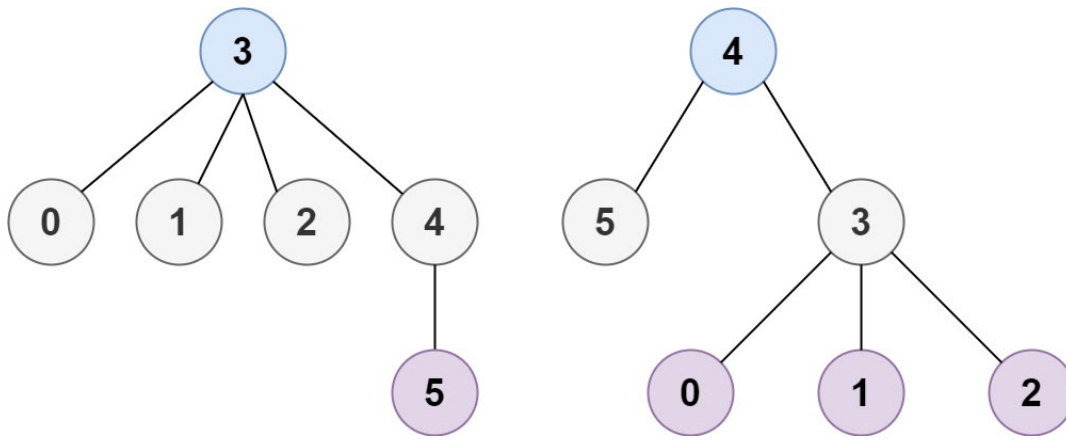


Input: `n = 4, edges = [[1,0],[1,2],[1,3]]`

Output: `[1]`

Explanation: As shown, the height of the tree is 1 when the root is the node with label 1 which is the only MHT.

Example 2:



Input: `n = 6, edges = [[3,0],[3,1],[3,2],[3,4],[5,4]]`
Output: `[3,4]`

Constraints:

- $1 \leq n \leq 2 \cdot 10^4$
- `edges.length == n - 1`
- $0 \leq a_i, b_i < n$
- $a_i \neq b_i$
- All the pairs (a_i, b_i) are distinct.
- The given input is **guaranteed** to be a tree and there will be **no repeated** edges.

 322. 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`

Constraints:

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

 324. Wiggle Sort II

Given an integer array `nums`, reorder it such that `nums[0] < nums[1] > nums[2] < nums[3] ...`

You may assume the input array always has a valid answer.

Example 1:

Input: `nums = [1,5,1,1,6,4]`
Output: `[1,6,1,5,1,4]`

Explanation: [1,4,1,5,1,6] is also accepted.

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 5 \times 10^4$
- $0 \leq \text{nums}[i] \leq 5000$
- It is guaranteed that there will be an answer for the given input nums.

Follow Up: Can you do it in $O(n)$ time and/or **in-place** with $O(1)$ extra space?

328. Odd Even Linked List

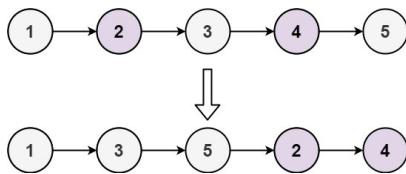
Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return *the reordered list*.

The **first** node is considered **odd**, and the **second** node is **even**, and so on.

Note that the relative order inside both the even and odd groups should remain as it was in the input.

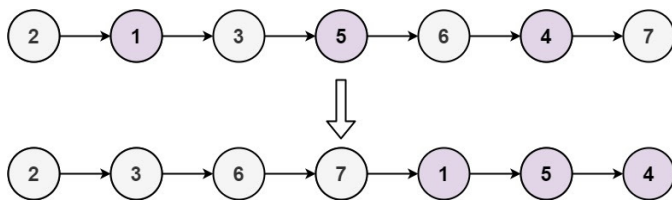
You must solve the problem in $O(1)$ extra space complexity and $O(n)$ time complexity.

Example 1:



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

Example 2:



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

Constraints:

- $n ==$ number of nodes in the linked list
- $0 \leq n \leq 10^4$
- $-10^6 \leq \text{Node.val} \leq 10^6$

334. Increasing Triplet Subsequence

Given an integer array nums, return true if there exists a triple of indices (i, j, k) such that $i < j < k$ and $\text{nums}[i] < \text{nums}[j] < \text{nums}[k]$. If no such indices exists, return false.

Example 1:

Input: nums = [1,2,3,4,5]
Output: true
Explanation: Any triplet where $i < j < k$ is valid.

Example 2:

Input: nums = [5,4,3,2,1]
Output: false
Explanation: No triplet exists.

Example 3:

Input: nums = [2,1,5,0,4,6]
Output: true
Explanation: The triplet (3, 4, 5) is valid because nums[3] == 0 < nums[4] == 4 < nums[5] == 6.

Constraints:

- $1 \leq \text{nums.length} \leq 5 \cdot 10^5$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$

Follow up: Could you implement a solution that runs in $O(n)$ time complexity and $O(1)$ space complexity?

341. Flatten Nested List Iterator

You are given a nested list of integers `nestedList`. Each element is either an integer or a list whose elements may also be integers or other lists. Implement an iterator to flatten it.

Implement the `NestedIterator` class:

- `NestedIterator(List<NestedInteger> nestedList)` Initializes the iterator with the nested list `nestedList`.
- `int next()` Returns the next integer in the nested list.
- `boolean hasNext()` Returns `true` if there are still some integers in the nested list and `false` otherwise.

Your code will be tested with the following pseudocode:

```
initialize iterator with nestedList
res = []
while iterator.hasNext()
    append iterator.next() to the end of res
return res
```

If `res` matches the expected flattened list, then your code will be judged as correct.

Example 1:

Input: nestedList = [[1,1],2,[1,1]]
Output: [1,1,2,1,1]
Explanation: By calling `next` repeatedly until `hasNext` returns false, the order of elements returned by `next` should be: [1,1,2,1,1].

Example 2:

Input: nestedList = [1,[4,[6]]]
Output: [1,4,6]
Explanation: By calling `next` repeatedly until `hasNext` returns false, the order of elements returned by `next` should be: [1,4,6].

Constraints:

- $1 \leq \text{nestedList.length} \leq 500$
- The values of the integers in the nested list is in the range $[-10^6, 10^6]$.

347. 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.

371. 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 <= a, b <= 1000`

378. Kth Smallest Element in a Sorted Matrix

Given an $n \times n$ matrix where each of the rows and columns is sorted in ascending order, return *the k^{th} smallest element in the matrix*.

Note that it is the k^{th} smallest element **in the sorted order**, not the k^{th} **distinct** element.

You must find a solution with a memory complexity better than $O(n^2)$.

Example 1:

Input: `matrix = [[1,5,9],[10,11,13],[12,13,15]]`, `k = 8`
Output: `13`
Explanation: The elements in the matrix are `[1,5,9,10,11,12,13,13,15]`, and the 8^{th} smallest number is `13`

Example 2:

Input: `matrix = [[-5]]`, `k = 1`
Output: `-5`

Constraints:

- `n == matrix.length == matrix[i].length`
- `1 <= n <= 300`
- `-109 <= matrix[i][j] <= 109`
- All the rows and columns of `matrix` are **guaranteed** to be sorted in **non-decreasing order**.
- `1 <= k <= n2`

Follow up:

- Could you solve the problem with a constant memory (i.e., $O(1)$ memory complexity)?
- Could you solve the problem in $O(n)$ time complexity? The solution may be too advanced for an interview but you may find reading [this paper](#) fun.

380. Insert Delete GetRandom O(1)

Implement the `RandomizedSet` class:

- `RandomizedSet()` Initializes the `RandomizedSet` object.

- `bool insert(int val)` Inserts an item `val` into the set if not present. Returns `true` if the item was not present, `false` otherwise.
- `bool remove(int val)` Removes an item `val` from the set if present. Returns `true` if the item was present, `false` otherwise.
- `int getRandom()` Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the **same probability** of being returned.

You must implement the functions of the class such that each function works in **average** $O(1)$ time complexity.

Example 1:

Input

```
["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"]
[[], [1], [2], [2], [], [1], [2], []]
```

Output

```
[null, true, false, true, 2, true, false, 2]
```

Explanation

```
RandomizedSet randomizedSet = new RandomizedSet();
randomizedSet.insert(1); // Inserts 1 to the set. Returns true as 1 was inserted successfully.
randomizedSet.remove(2); // Returns false as 2 does not exist in the set.
randomizedSet.insert(2); // Inserts 2 to the set, returns true. Set now contains [1,2].
randomizedSet.getRandom(); // getRandom() should return either 1 or 2 randomly.
randomizedSet.remove(1); // Removes 1 from the set, returns true. Set now contains [2].
randomizedSet.insert(2); // 2 was already in the set, so return false.
randomizedSet.getRandom(); // Since 2 is the only number in the set, getRandom() will always return 2.
```

Constraints:

- $-2^{31} \leq \text{val} \leq 2^{31} - 1$
- At most $2 * 10^5$ calls will be made to `insert`, `remove`, and `getRandom`.
- There will be **at least one** element in the data structure when `getRandom` is called.

384. Shuffle an Array

Given an integer array `nums`, design an algorithm to randomly shuffle the array. All permutations of the array should be **equally likely** as a result of the shuffling.

Implement the `Solution` class:

- `Solution(int[] nums)` Initializes the object with the integer array `nums`.
- `int[] reset()` Resets the array to its original configuration and returns it.
- `int[] shuffle()` Returns a random shuffling of the array.

Example 1:

Input

```
["Solution", "shuffle", "reset", "shuffle"]
[[[1, 2, 3]], [], [], []]
```

Output

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

Explanation

```
Solution solution = new Solution([1, 2, 3]);
solution.shuffle(); // Shuffle the array [1,2,3] and return its result.
                   // Any permutation of [1,2,3] must be equally likely to be returned.
                   // Example: return [3, 1, 2]
solution.reset();   // Resets the array back to its original configuration [1,2,3]. Return [1, 2, 3]
solution.shuffle(); // Returns the random shuffling of array [1,2,3]. Example: return [1, 3, 2]
```

Constraints:

- $1 \leq \text{nums.length} \leq 50$
- $-10^6 \leq \text{nums}[i] \leq 10^6$
- All the elements of `nums` are **unique**.
- At most 10^4 calls **in total** will be made to `reset` and `shuffle`.

394. Decode String

Given an encoded string, return its decoded string.

The encoding rule is: `k[encoded_string]`, where the `encoded_string` inside the square brackets is being repeated exactly `k` times. Note that `k` is guaranteed to be a positive integer.

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, κ . For example, there will not be input like 3a or 2[4].

Example 1:

Input: s = "3[a]2[bc]"
Output: "aaabcbc"

Example 2:

Input: s = "3[a2[c]]"
Output: "accaccacc"

Example 3:

Input: s = "2[abc]3[cd]ef"
Output: "abccabcccdcdcddef"

Constraints:

- `1 <= s.length <= 30`
- `s` consists of lowercase English letters, digits, and square brackets `'[]'`.
- `s` is guaranteed to be a **valid** input.
- All the integers in `s` are in the range `[1, 300]`.

395. Longest Substring with At Least K Repeating Characters

Given a string `s` and an integer `k`, return *the length of the longest substring of `s` such that the frequency of each character in this substring is greater than or equal to `k`*.

Example 1:

Input: s = "aaabb", k = 3
Output: 3
Explanation: The longest substring is "aaa", as 'a' is repeated 3 times.

Example 2:

Input: s = "ababbc", k = 2
Output: 5
Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

Constraints:

- $1 \leq s.length \leq 10^4$
- s consists of only lowercase English letters.
- $1 \leq k \leq 10^5$

416. Partition Equal Subset Sum

Given a **non-empty** array `nums` containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Example 1:

Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].

Example 2:

Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into equal sum subsets.

Constraints:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 100`

417. Pacific Atlantic Water Flow

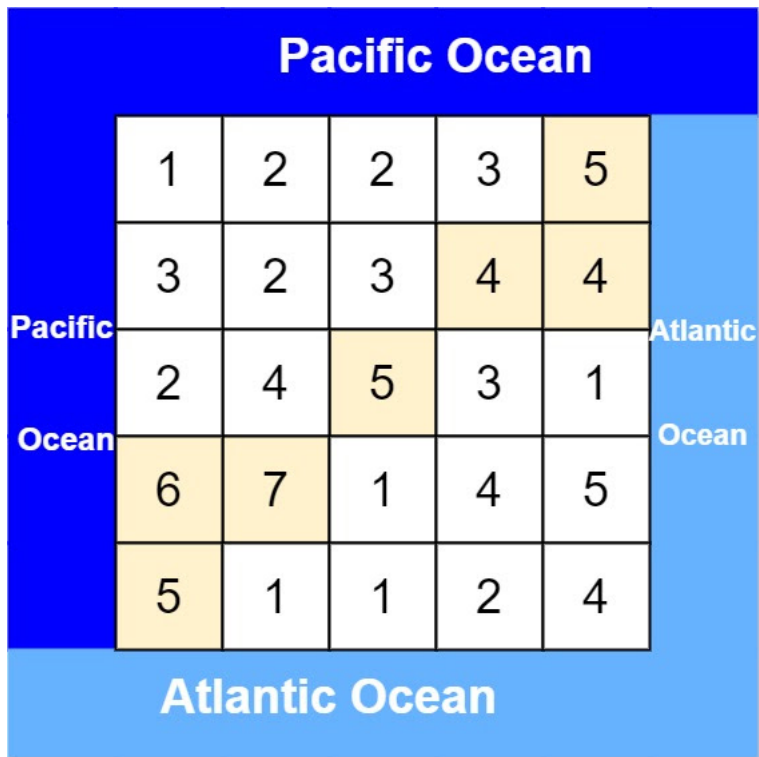
There is an $m \times n$ rectangular island that borders both the **Pacific Ocean** and **Atlantic Ocean**. The **Pacific Ocean** touches the island's left and top edges, and the **Atlantic Ocean** touches the island's right and bottom edges.

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 == heights.length`
- `n == heights[r].length`
- `1 <= m, n <= 200`
- `0 <= heights[r][c] <= 105`

424. 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 = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.

Example 2:

Input: s = "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.
- $0 \leq k \leq s.length$

435. Non-overlapping Intervals

Given an array of intervals intervals where $intervals[i] = [start_i, end_i]$, 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:

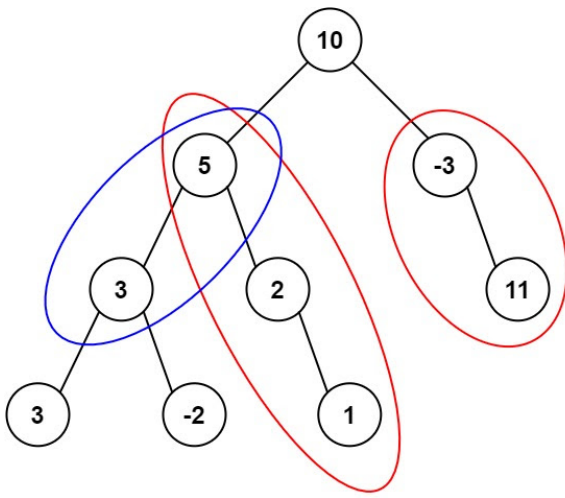
- $1 \leq intervals.length \leq 10^5$
- $intervals[i].length == 2$
- $-5 * 10^4 \leq start_i < end_i \leq 5 * 10^4$

437. Path Sum III

Given the root of a binary tree and an integer targetSum, return *the number of paths where the sum of the values along the path equals targetSum*.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

Example 1:



Input: root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
Output: 3
Explanation: The paths that sum to 8 are shown.

Example 2:

Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: 3

Constraints:

- The number of nodes in the tree is in the range [0, 1000].
- $-10^9 \leq \text{Node.val} \leq 10^9$
- $-1000 \leq \text{targetSum} \leq 1000$

 438. Find All Anagrams in a String

Given two strings *s* and *p*, return *an array of all the start indices of p's anagrams in s*. You may 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: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
 The substring with start index = 0 is "cba", which is an anagram of "abc".
 The substring with start index = 6 is "bac", which is an anagram of "abc".

Example 2:

Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
 The substring with start index = 0 is "ab", which is an anagram of "ab".
 The substring with start index = 1 is "ba", which is an anagram of "ab".
 The substring with start index = 2 is "ab", which is an anagram of "ab".

Constraints:

- $1 \leq \text{s.length}, \text{p.length} \leq 3 * 10^4$
- *s* and *p* consist of lowercase English letters.

 449. Serialize and Deserialize BST

Serialization is 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 search tree**. There is no restriction on how your serialization/deserialization algorithm should work. You need to ensure that a binary search tree can be serialized to a string, and this string can be deserialized to the original tree structure.

The encoded string should be as compact as possible.

Example 1:

Input: root = [2,1,3]
Output: [2,1,3]

Example 2:

Input: root = []
Output: []

Constraints:

- The number of nodes in the tree is in the range $[0, 10^4]$.
- $0 \leq \text{Node.val} \leq 10^4$
- The input tree is **guaranteed** to be a binary search tree.

454. 4Sum II

Given four integer arrays nums1, nums2, nums3, and nums4 all of length n, return the number of tuples (i, j, k, l) such that:

- $0 \leq i, j, k, l < n$
- $\text{nums1}[i] + \text{nums2}[j] + \text{nums3}[k] + \text{nums4}[l] == 0$

Example 1:

Input: nums1 = [1,2], nums2 = [-2,-1], nums3 = [-1,2], nums4 = [0,2]
Output: 2
Explanation:
The two tuples are:
1. (0, 0, 0, 1) -> $\text{nums1}[0] + \text{nums2}[0] + \text{nums3}[0] + \text{nums4}[1] = 1 + (-2) + (-1) + 2 = 0$
2. (1, 1, 0, 0) -> $\text{nums1}[1] + \text{nums2}[1] + \text{nums3}[0] + \text{nums4}[0] = 2 + (-1) + (-1) + 0 = 0$

Example 2:

Input: nums1 = [0], nums2 = [0], nums3 = [0], nums4 = [0]
Output: 1

Constraints:

- $n == \text{nums1.length}$
- $n == \text{nums2.length}$
- $n == \text{nums3.length}$
- $n == \text{nums4.length}$
- $1 \leq n \leq 200$
- $-2^{28} \leq \text{nums1}[i], \text{nums2}[i], \text{nums3}[i], \text{nums4}[i] \leq 2^{28}$

494. Target Sum

You are given an integer array nums and an integer target.

You want to build an **expression** out of nums by adding one of the symbols '+' and '-' before each integer in nums and then concatenate all the integers.

- For example, if $\text{nums} = [2, 1]$, you can add a '+' before 2 and a '-' before 1 and concatenate them to build the expression "+2-1".

Return the number of different **expressions** that you can build, which evaluates to target.

Example 1:

Input: nums = [1,1,1,1,1], target = 3
Output: 5
Explanation: There are 5 ways to assign symbols to make the sum of nums be target 3.
-1 + 1 + 1 + 1 + 1 = 3
+1 - 1 + 1 + 1 + 1 = 3
+1 + 1 - 1 + 1 + 1 = 3
+1 + 1 + 1 - 1 + 1 = 3
+1 + 1 + 1 + 1 - 1 = 3

Example 2:

Input: nums = [1], target = 1
Output: 1

Constraints:

- 1 <= nums.length <= 20
- 0 <= nums[i] <= 1000
- 0 <= sum(nums[i]) <= 1000
- -1000 <= target <= 1000

542. 01 Matrix

Given an m x n binary matrix mat, return *the distance of the nearest 0 for each cell*.
The distance between two adjacent cells is 1.

Example 1:

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Input: mat = [[0,0,0],[0,1,0],[0,0,0]]
Output: [[0,0,0],[0,1,0],[0,0,0]]

Example 2:

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

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

Constraints:

- m == mat.length
- n == mat[i].length
- 1 <= m, n <= 10⁴
- 1 <= m * n <= 10⁴
- mat[i][j] is either 0 or 1.
- There is at least one 0 in mat.

560. Subarray Sum Equals K

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to k*.

Example 1:

Input: nums = [1,1,1], k = 2
Output: 2

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 2 \cdot 10^4$
- $-1000 \leq \text{nums}[i] \leq 1000$
- $-10^7 \leq k \leq 10^7$

621. Task Scheduler

Given a characters array `tasks`, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.

However, there is a non-negative integer `n` that represents the cooldown period between two **same tasks** (the same letter in the array), that is that there must be at least `n` units of time between any two same tasks.

Return *the least number of units of times that the CPU will take to finish all the given tasks*.

Example 1:

Input: tasks = ["A","A","A","B","B","B"], n = 2
Output: 8
Explanation:
A -> B -> idle -> A -> B -> idle -> A -> B
There is at least 2 units of time between any two same tasks.

Example 2:

Input: tasks = ["A","A","A","B","B","B"], n = 0
Output: 6
Explanation: On this case any permutation of size 6 would work since n = 0.
["A","A","A","B","B","B"]
["A","B","A","B","A","B"]
["B","B","B","A","A","A"]
...
And so on.

Example 3:

Input: tasks = ["A","A","A","A","A","A","B","C","D","E","F","G"], n = 2
Output: 16
Explanation:
One possible solution is
A -> B -> C -> A -> D -> E -> A -> F -> G -> A -> idle -> idle -> A -> idle -> idle -> A

Constraints:

- $1 \leq \text{task.length} \leq 10^4$
- `tasks[i]` is upper-case English letter.
- The integer `n` is in the range `[0, 100]`.

647. 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 = "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".

Example 2:

Input: s = "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".

Constraints:

- 1 <= s.length <= 1000
- s consists of lowercase English letters.

721. Accounts Merge

Given a list of accounts where each element accounts[i] is a list of strings, where the first element accounts[i][0] is a name, and the rest of the elements are emails representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some common email to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails in **sorted order**. The accounts themselves can be returned in **any order**.

Example 1:

Input: accounts = [{"John", "johnsmith@mail.com", "john_newyork@mail.com"}, {"John", "johnsmith@mail.com", "john00@mail.com"}, {"Mary", "mary@mail.com"}, {"John", "johnnybravo@mail.com"}]
Output: [{"John", "john00@mail.com", "john_newyork@mail.com", "johnsmith@mail.com"}, {"Mary", "mary@mail.com"}, {"John", "johnnybravo@mail.com"}]
Explanation:
The first and second John's are the same person as they have the common email "johnsmith@mail.com".
The third John and Mary are different people as none of their email addresses are used by other accounts.
We could return these lists in any order, for example the answer [{"Mary", "mary@mail.com"}, {"John", "johnnybravo@mail.com"}, {"John", "john00@mail.com", "john_newyork@mail.com", "johnsmith@mail.com"}] would still be accepted.

Example 2:

Input: accounts = [{"Gabe", "Gabe0@m.co", "Gabe3@m.co", "Gabe1@m.co"}, {"Kevin", "Kevin3@m.co", "Kevin5@m.co", "Kevin0@m.co"}, {"Ethan", "Ethan5@m.co", "Ethan0@m.co", "Ethan4@m.co", "Ethan5@m.co"}, {"Gabe", "Gabe0@m.co", "Gabe1@m.co", "Gabe3@m.co"}, {"Hanzo", "Hanzo0@m.co", "Hanzo1@m.co", "Hanzo2@m.co"}]
Output: [{"Ethan", "Ethan0@m.co", "Ethan4@m.co", "Ethan5@m.co"}, {"Gabe", "Gabe0@m.co", "Gabe1@m.co", "Gabe3@m.co"}, {"Hanzo", "Hanzo0@m.co", "Hanzo1@m.co", "Hanzo2@m.co"}, {"Kevin", "Kevin0@m.co", "Kevin3@m.co", "Kevin5@m.co"}, {"John", "John00@mail.com", "John_newyork@mail.com", "Johnsmith@mail.com"}]

Constraints:

- 1 <= accounts.length <= 1000
- 2 <= accounts[i].length <= 10
- 1 <= accounts[i][j] <= 30
- accounts[i][0] consists of English letters.
- accounts[i][j] (for j > 0) is a valid email.

739. Daily Temperatures

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0 instead.

Example 1:

Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]

Example 2:

Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]

Example 3:

Input: temperatures = [30,60,90]
Output: [1,1,0]

Constraints:

- 1 <= temperatures.length <= 10⁵
- 30 <= temperatures[i] <= 100

763. Partition Labels

You are given a string `s`. We want to partition the string into as many parts as possible so that each letter appears in at most one part.

Note that the partition is done so that after concatenating all the parts in order, the resultant string should be `s`.

Return *a list of integers representing the size of these parts*.

Example 1:

Input: `s = "ababcbacadefegdehijhklij"`
Output: `[9,7,8]`
Explanation:
The partition is "ababcbaca", "defegde", "hijhklij".
This is a partition so that each letter appears in at most one part.
A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it splits `s` into less parts.

Example 2:

Input: `s = "eccbbbbbdec"`
Output: `[10]`

Constraints:

- `1 <= s.length <= 500`
- `s` consists of lowercase English letters.

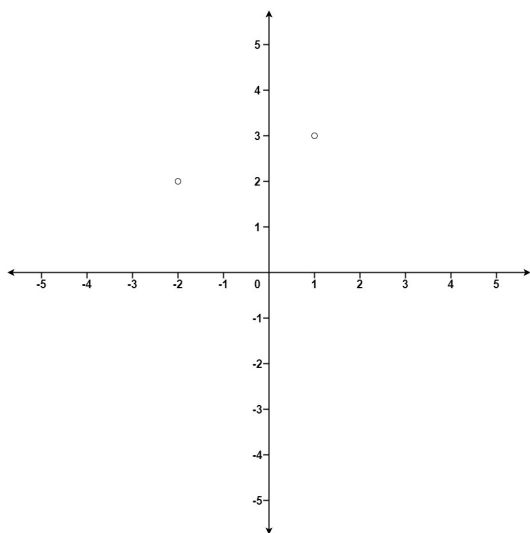
973. K Closest Points to Origin

Given an array of points where `points[i] = [xi, yi]` represents a point on the **X-Y** plane and an integer `k`, return the `k` closest points to the origin `(0, 0)`.

The distance between two points on the **X-Y** plane is the Euclidean distance (i.e., $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$).

You may return the answer in **any order**. The answer is **guaranteed** to be **unique** (except for the order that it is in).

Example 1:



Input: `points = [[1,3],[-2,2]]`, `k = 1`
Output: `[[-2,2]]`
Explanation:
The distance between `(1, 3)` and the origin is `sqrt(10)`.
The distance between `(-2, 2)` and the origin is `sqrt(8)`.
Since `sqrt(8) < sqrt(10)`, `(-2, 2)` is closer to the origin.
We only want the closest `k = 1` points from the origin, so the answer is just `[[-2,2]]`.

Example 2:

Input: `points = [[3,3],[5,-1],[-2,4]]`, `k = 2`
Output: `[[3,3],[-2,4]]`
Explanation: The answer `[[-2,4],[3,3]]` would also be accepted.

Constraints:

- $1 \leq k \leq \text{points.length} \leq 10^4$
- $-10^4 < x_i, y_i < 10^4$

981. Time Based Key-Value Store

Design a time-based key-value data structure that can store multiple values for the same key at different time stamps and retrieve the key's value at a certain timestamp.

Implement the `TimeMap` class:

- `TimeMap()` Initializes the object of the data structure.
- `void set(String key, String value, int timestamp)` Stores the key `key` with the value `value` at the given time `timestamp`.
- `String get(String key, int timestamp)` Returns a value such that `set` was called previously, with `timestamp_prev ≤ timestamp`. If there are multiple such values, it returns the value associated with the largest `timestamp_prev`. If there are no values, it returns `""`.

Example 1:

Input

```
[["TimeMap", "set", "get", "get", "set", "get", "get"]
[[], [{"foo", "bar", 1}, {"foo", 1}, {"foo", 3}, {"foo", "bar2", 4}, {"foo", 4}, {"foo", 5}]]
```

Output

```
[null, null, "bar", "bar", null, "bar2", "bar2"]
```

Explanation

```
TimeMap timeMap = new TimeMap();
timeMap.set("foo", "bar", 1); // store the key "foo" and value "bar" along with timestamp = 1.
timeMap.get("foo", 1); // return "bar"
timeMap.get("foo", 3); // return "bar", since there is no value corresponding to foo at timestamp 3 and timestamp 2, then the only
timeMap.set("foo", "bar2", 4); // store the key "foo" and value "bar2" along with timestamp = 4.
timeMap.get("foo", 4); // return "bar2"
timeMap.get("foo", 5); // return "bar2"
```

Constraints:

- $1 \leq \text{key.length}, \text{value.length} \leq 100$
- key and value consist of lowercase English letters and digits.
- $1 \leq \text{timestamp} \leq 10^7$
- All the timestamps `timestamp` of `set` are strictly increasing.
- At most 2×10^5 calls will be made to `set` and `get`.

994. Rotting Oranges

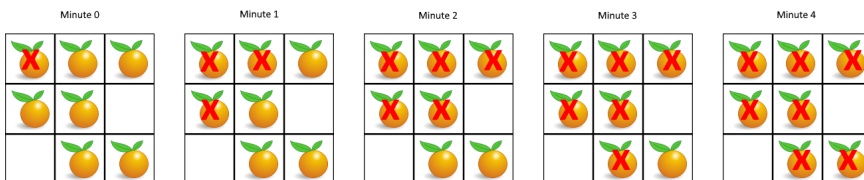
You are given an $m \times n$ grid where each cell can have one of three values:

- 0 representing an empty cell,
- 1 representing a fresh orange, or
- 2 representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange*. If this is impossible, return `-1`.

Example 1:



Input: `grid = [[2,1,1],[1,1,0],[0,1,1]]`
Output: 4

Example 2:

Input: `grid = [[2,1,1],[0,1,1],[1,0,1]]`
Output: -1
Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Example 3:

Input: grid = [[0,2]]
Output: 0
Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

Constraints:

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 10
- grid[i][j] is 0,1,or 2.

1143. 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.

4. Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return **the median** of the two sorted arrays.

The overall run time complexity should be $O(\log(m+n))$.

Example 1:

Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.

Example 2:

Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is $(2 + 3) / 2 = 2.5$.

Constraints:

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- $-10^6 \leq \text{nums1}[i], \text{nums2}[i] \leq 10^6$

10. Regular Expression Matching

Given an input string `s` and a pattern `p`, implement regular expression matching with support for `'.'` and `'*'` where:

- `'.'` Matches any single character.
- `'*'` Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

Example 1:

Input: `s = "aa", p = "a"`
Output: `false`
Explanation: "a" does not match the entire string "aa".

Example 2:

Input: `s = "aa", p = "a*"`
Output: `true`
Explanation: `'*'` means zero or more of the preceding element, `'a'`. Therefore, by repeating `'a'` once, it becomes "aa".

Example 3:

Input: `s = "ab", p = ".*"`
Output: `true`
Explanation: `".*"` means "zero or more (`*`) of any character (`.`)".

Constraints:

- `1 <= s.length <= 20`
- `1 <= p.length <= 30`
- `s` contains only lowercase English letters.
- `p` contains only lowercase English letters, `'.'`, and `'*'`.
- It is guaranteed for each appearance of the character `'*'`, there will be a previous valid character to match.

23. 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 <= k <= 104`
- `0 <= lists[i].length <= 500`
- `-104 <= lists[i][j] <= 104`
- `lists[i]` is sorted in **ascending order**.
- The sum of `lists[i].length` will not exceed `104`.

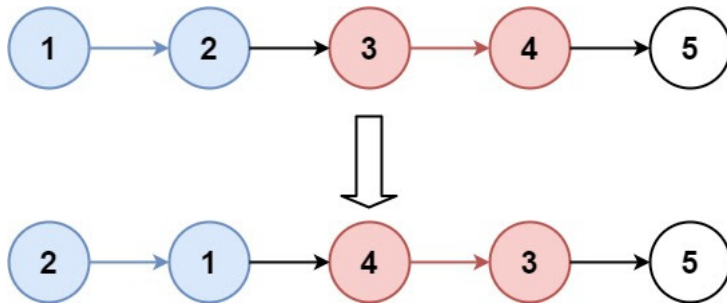
25. Reverse Nodes in k-Group

Given the head of a linked list, reverse the nodes of the list k at a time, and return *the modified list*.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should remain as it is.

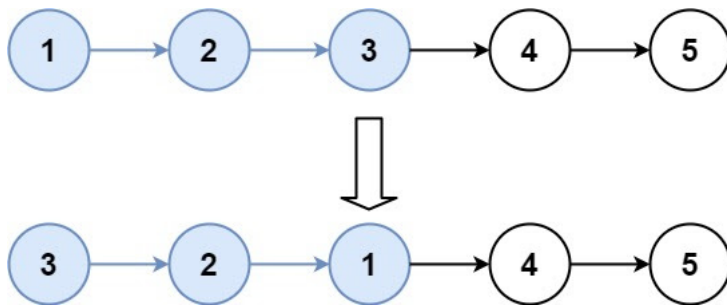
You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:



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

Example 2:



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

Constraints:

- The number of nodes in the list is n .
- $1 \leq k \leq n \leq 5000$
- $0 \leq \text{Node.val} \leq 1000$

Follow-up: Can you solve the problem in $O(1)$ extra memory space?

32. Longest Valid Parentheses

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.

Example 1:

Input: $s = "()"$
Output: 2
Explanation: The longest valid parentheses substring is "()".

Example 2:

Input: $s = ")()()"$
Output: 4
Explanation: The longest valid parentheses substring is "()()".

Example 3:

Input: s = ""
Output: 0

Constraints:

- $0 \leq s.length \leq 3 \times 10^4$
- s[i] is '(', or ')'

41. First Missing Positive

Given an unsorted integer array nums, return the smallest missing positive integer.
You must implement an algorithm that runs in $O(n)$ time and uses constant extra space.

Example 1:

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

Example 2:

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

Example 3:

Input: nums = [7,8,9,11,12]
Output: 1

Constraints:

- $1 \leq nums.length \leq 5 \times 10^5$
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$

42. Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]
Output: 9

Constraints:

- n == height.length
- $1 \leq n \leq 2 \times 10^4$
- $0 \leq height[i] \leq 10^5$

44. Wildcard Matching

Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where:

- '?' Matches any single character.
- '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

Example 1:

Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = "*"
Output: true
Explanation: '*' matches any sequence.

Example 3:

Input: s = "cb", p = "?a"
Output: false
Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.

Constraints:

- 0 <= s.length, p.length <= 2000
- s contains only lowercase English letters.
- p contains only lowercase English letters, '?' or '*'.

72. Edit Distance

Given two strings word1 and word2, return *the minimum number of operations required to convert word1 to word2*.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')

Example 2:

Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')

Constraints:

- 0 <= word1.length, word2.length <= 500
- word1 and word2 consist of lowercase English letters.

76. 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 = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.

Example 3:

Input: s = "a", t = "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:

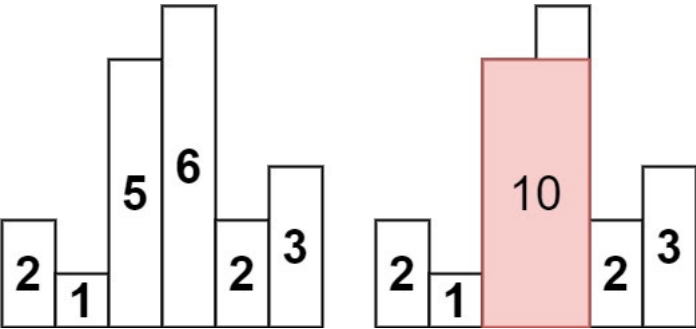
- m == s.length
- n == t.length
- 1 <= m, n <= 10⁵
- s and t consist of uppercase and lowercase English letters.

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

84. Largest Rectangle in Histogram

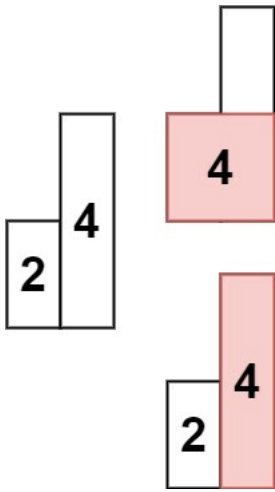
Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return *the area of the largest rectangle in the histogram*.

Example 1:



Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.

Example 2:



Input: heights = [2,4]
Output: 4

Constraints:

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

85. Maximal Rectangle

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return *its area*.

Example 1:

| | | | | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 |

Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]
Output: 6
Explanation: The maximal rectangle is shown in the above picture.

Example 2:

Input: matrix = [["0"]]
Output: 0

Example 3:

Input: matrix = [["1"]]
Output: 1

Constraints:

- rows == matrix.length
- cols == matrix[i].length
- 1 <= row, cols <= 200
- matrix[i][j] is '0' or '1'.

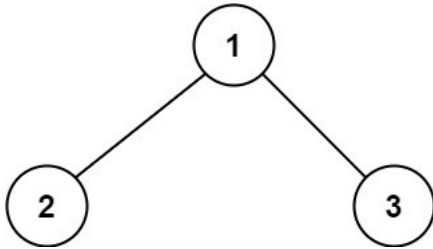
124. Binary Tree Maximum Path Sum

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.

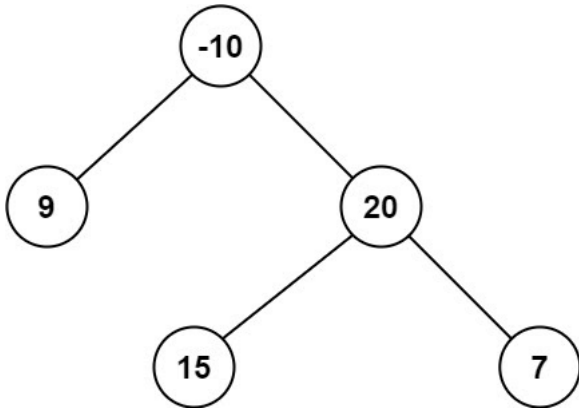
Given the root of a binary tree, return *the maximum path sum of any non-empty path*.

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 * 10⁴].
- -1000 <= Node.val <= 1000

127. Word Ladder

A **transformation sequence** from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s₁ -> s₂ -> ... -> s_k such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- s_k == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return *the number of words in the shortest transformation sequence from beginWord to endWord, or 0 if no such sequence exists*.

Example 1:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> "cog", which is 5 words long.

Example 2:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

Constraints:

- 1 <= beginWord.length <= 10
- endWord.length == beginWord.length
- 1 <= wordList.length <= 5000
- wordList[i].length == beginWord.length
- beginWord, endWord, and wordList[i] consist of lowercase English letters.
- beginWord != endWord
- All the words in wordList are **unique**.

140. Word Break II

Given a string *s* and a dictionary of strings *wordDict*, add spaces in *s* to construct a sentence where each word is a valid dictionary word. Return all such possible sentences in **any order**.

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

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: []

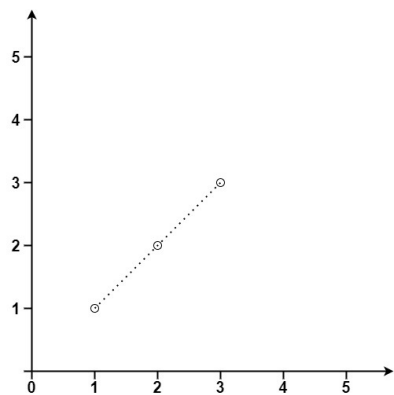
Constraints:

- 1 <= s.length <= 20
- 1 <= wordDict.length <= 1000
- 1 <= wordDict[i].length <= 10
- s and wordDict[i] consist of only lowercase English letters.
- All the strings of wordDict are **unique**.

149. Max Points on a Line

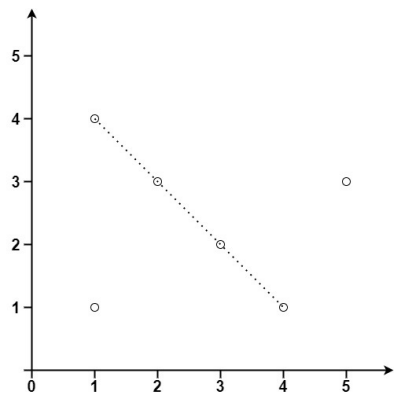
Given an array of points where points[i] = [x_i, y_i] represents a point on the **X-Y** plane, return *the maximum number of points that lie on the same straight line*.

Example 1:



Input: points = [[1,1],[2,2],[3,3]]
Output: 3

Example 2:



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

Constraints:

- 1 <= points.length <= 300
- points[i].length == 2
- -10⁴ <= x_i, y_i <= 10⁴
- All the points are **unique**.

212. Word Search II

Given an m x 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:

| | | | |
|---|---|---|---|
| o | a | a | n |
| e | t | a | e |
| i | h | k | r |
| i | f | l | v |

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 == board.length

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

 218. The Skyline Problem

A city's **skyline** is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return *the skyline formed by these buildings collectively*.

The geometric information of each building is given in the array `buildings` where $\text{buildings}[i] = [\text{left}_i, \text{right}_i, \text{height}_i]$:

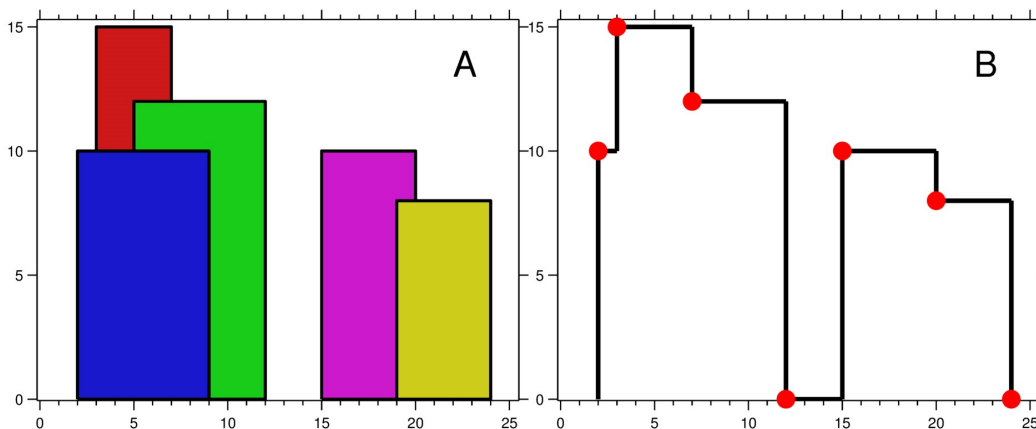
- left_i is the x coordinate of the left edge of the i^{th} building.
- right_i is the x coordinate of the right edge of the i^{th} building.
- height_i is the height of the i^{th} building.

You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

The **skyline** should be represented as a list of "key points" **sorted by their x-coordinate** in the form $[[x_1, y_1], [x_2, y_2], \dots]$. Each key point is the left endpoint of some horizontal segment in the skyline except the last point in the list, which always has a y-coordinate 0 and is used to mark the skyline's termination where the rightmost building ends. Any ground between the leftmost and rightmost buildings should be part of the skyline's contour.

Note: There must be no consecutive horizontal lines of equal height in the output skyline. For instance, $[\dots, [2, 3], [4, 5], [7, 5], [11, 5], [12, 7], \dots]$ is not acceptable; the three lines of height 5 should be merged into one in the final output as such: $[\dots, [2, 3], [4, 5], [12, 7], \dots]$

Example 1:



Input: `buildings = [[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]`

Output: `[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]`

Explanation:

Figure A shows the buildings of the input.

Figure B shows the skyline formed by those buildings. The red points in figure B represent the key points in the output list.

Example 2:

Input: `buildings = [[0,2,3],[2,5,3]]`

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

Constraints:

- $1 \leq \text{buildings.length} \leq 10^4$
- $0 \leq \text{left}_i < \text{right}_i \leq 2^{31} - 1$
- $1 \leq \text{height}_i \leq 2^{31} - 1$
- buildings is sorted by left_i in non-decreasing order.

 224. Basic Calculator

Given a string `s` representing a valid expression, implement a basic calculator to evaluate it, and return *the result of the evaluation*.

Note: You are **not** allowed to use any built-in function which evaluates strings as mathematical expressions, such as `eval()`.

Example 1:

Input: s = "1 + 1"
Output: 2

Example 2:

Input: s = " 2-1 + 2 "
Output: 3

Example 3:

Input: s = "(1+(4+5+2)-3)+(6+8)"
Output: 23

Constraints:

- 1 <= s.length <= 3 * 10⁵
- s consists of digits, '+', '-', '(', ')', and ' '.
- s represents a valid expression.
- '+' is **not** used as a unary operation (i.e., "+1" and "+(2 + 3)" is invalid).
- '-' could be used as a unary operation (i.e., "-1" and "-(2 + 3)" is valid).
- There will be no two consecutive operators in the input.
- Every number and running calculation will fit in a signed 32-bit integer.

239. Sliding Window Maximum

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window*.

Example 1:

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:

| Window position | Max |
|---------------------|-----|
| [1 3 -1] -3 5 3 6 7 | 3 |
| 1 [3 -1 -3] 5 3 6 7 | 3 |
| 1 3 [-1 -3 5] 3 6 7 | 5 |
| 1 3 -1 [-3 5 3] 6 7 | 5 |
| 1 3 -1 -3 [5 3 6] 7 | 6 |
| 1 3 -1 -3 5 [3 6 7] | 7 |

Example 2:

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

Constraints:

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

295. 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?

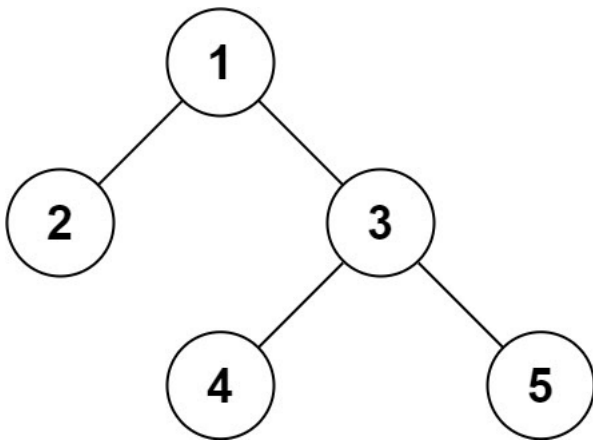
297. Serialize and Deserialize Binary Tree

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: []

Constraints:

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

315. Count of Smaller Numbers After Self

You are given an integer array `nums` and you have to return a new `counts` array. The `counts` array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example 1:

Input: `nums = [5,2,6,1]`
Output: `[2,1,1,0]`
Explanation:
To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.

Example 2:

Input: `nums = [-1]`
Output: `[0]`

Example 3:

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

Constraints:

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

329. Longest Increasing Path in a Matrix

Given an `m x n` integers `matrix`, return *the length of the longest increasing path in matrix*.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

Example 1:

| | | |
|---|---|---|
| 9 | 9 | 4 |
| 6 | 6 | 8 |
| 2 | 1 | 1 |

Input: `matrix = [[9,9,4],[6,6,8],[2,1,1]]`
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].

Example 2:

| | | |
|---|---|---|
| 3 | 4 | 5 |
| 3 | 2 | 6 |
| 2 | 2 | 1 |

Input: `matrix = [[3,4,5],[3,2,6],[2,2,1]]`
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.

Example 3:

Input: `matrix = [[1]]`
Output: 1

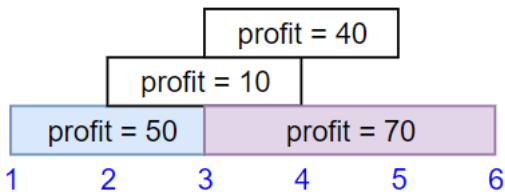
Constraints:

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

1235. Maximum Profit in Job Scheduling

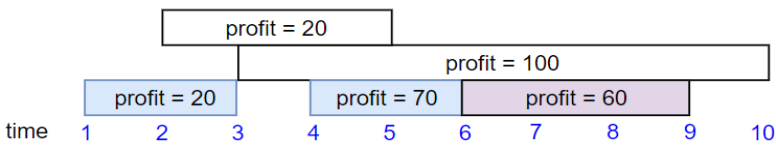
We have `n` jobs, where every job is scheduled to be done from `startTime[i]` to `endTime[i]`, obtaining a profit of `profit[i]`.
You're given the `startTime`, `endTime` and `profit` arrays, return the maximum profit you can take such that there are no two jobs in the subset with overlapping time range.
If you choose a job that ends at time `x` you will be able to start another job that starts at time `x`.

Example 1:



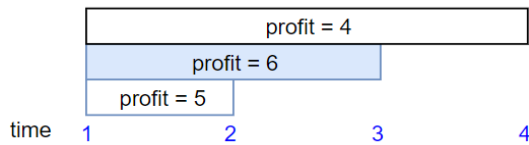
Input: `startTime = [1,2,3,3], endTime = [3,4,5,6], profit = [50,10,40,70]`
Output: 120
Explanation: The subset chosen is the first and fourth job.
Time range `[1-3]+[3-6]`, we get profit of `120 = 50 + 70`.

Example 2:



Input: `startTime = [1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,70,60]`
Output: 150
Explanation: The subset chosen is the first, fourth and fifth job.
Profit obtained `150 = 20 + 70 + 60`.

Example 3:



Input: `startTime = [1,1,1], endTime = [2,3,4], profit = [5,6,4]`
Output: 6

Constraints:

- `1 <= startTime.length == endTime.length == profit.length <= 5 * 104`
- `1 <= startTime[i] < endTime[i] <= 109`
- `1 <= profit[i] <= 104`