目录

[301. Remove Invalid Parentheses 3](#_Toc21248995)

[302. Smallest Rectangle Enclosing Black Pixels 4](#_Toc21248996)

[303. Range Sum Query - Immutable 5](#_Toc21248997)

[304. Range Sum Query 2D - Immutable 7](#_Toc21248998)

[305. Number of Islands II 9](#_Toc21248999)

[306. Additive Number 11](#_Toc21249000)

[307. Range Sum Query - Mutable 13](#_Toc21249001)

[308. Range Sum Query 2D - Mutable 14](#_Toc21249002)

[309. Best Time to Buy and Sell Stock with Cooldown 15](#_Toc21249003)

[310. Minimum Height Trees(●ˇ∀ˇ●) 16](#_Toc21249004)

[311. Sparse Matrix Multiplication 18](#_Toc21249005)

[312. Burst Balloons 20](#_Toc21249006)

[313. Super Ugly Number 21](#_Toc21249007)

[314. Binary Tree Vertical Order Traversal 24](#_Toc21249008)

[315. Count of Smaller Numbers After Self(●ˇ∀ˇ●) 28](#_Toc21249009)

[316. Remove Duplicate Letters 30](#_Toc21249010)

[317. Shortest Distance from All Buildings 31](#_Toc21249011)

[318. Maximum Product of Word Lengths(●ˇ∀ˇ●) 32](#_Toc21249012)

[319. Bulb Switcher 35](#_Toc21249013)

[320. Generalized Abbreviation 37](#_Toc21249014)

[321. Create Maximum Number 38](#_Toc21249015)

[322. Coin Change(●ˇ∀ˇ●) 39](#_Toc21249016)

[323. Number of Connected Components in an Undirected Graph 41](#_Toc21249017)

[324. Wiggle Sort II 43](#_Toc21249018)

[325. Maximum Size Subarray Sum Equals k 44](#_Toc21249019)

[326. Power of Three 46](#_Toc21249020)

[327. Count of Range Sum 47](#_Toc21249021)

[328. Odd Even Linked List 48](#_Toc21249022)

[329. Longest Increasing Path in a Matrix 50](#_Toc21249023)

[330. Patching Array(●ˇ∀ˇ●) 51](#_Toc21249024)

[331. Verify Preorder Serialization of a Binary Tree 53](#_Toc21249025)

[332. Reconstruct Itinerary(●ˇ∀ˇ●) 55](#_Toc21249026)

[333. Largest BST Subtree 57](#_Toc21249027)

[334. Increasing Triplet Subsequence 59](#_Toc21249028)

[335. Self Crossing 61](#_Toc21249029)

[336. Palindrome Pairs 63](#_Toc21249030)

[337. House Robber III(●ˇ∀ˇ●) 65](#_Toc21249031)

[338. Counting Bits(●ˇ∀ˇ●) 67](#_Toc21249032)

[339. Nested List Weight Sum 69](#_Toc21249033)

[340. Longest Substring with At Most K Distinct Characters 72](#_Toc21249034)

[341. Flatten Nested List Iterator 74](#_Toc21249035)

[342. Power of Four 77](#_Toc21249036)

[343. Integer Break 78](#_Toc21249037)

[344. Reverse String 79](#_Toc21249038)

[345. Reverse Vowels of a String 80](#_Toc21249039)

[346. Moving Average from Data Stream 81](#_Toc21249040)

[347. Top K Frequent Elements(●ˇ∀ˇ●) 82](#_Toc21249041)

[348. Design Tic-Tac-Toe 84](#_Toc21249042)

[349. Intersection of Two Arrays 87](#_Toc21249043)

[350. Intersection of Two Arrays II 90](#_Toc21249044)

### 301. Remove Invalid Parentheses

Hard

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

**Note:** The input string may contain letters other than the parentheses ( and ).

**Example 1:**

**Input:** "()())()"

**Output:** ["()()()", "(())()"]

**Example 2:**

**Input:** "(a)())()"

**Output:** ["(a)()()", "(a())()"]

**Example 3:**

**Input:** ")("

**Output:** [""]

### [302. Smallest Rectangle Enclosing Black Pixels](https://leetcode-cn.com/problems/smallest-rectangle-enclosing-black-pixels/)

An image is represented by a binary matrix with 0 as a white pixel and 1 as a black pixel. The black pixels are connected, i.e., there is only one black region. Pixels are connected horizontally and vertically. Given the location (x, y) of one of the black pixels, return the area of the smallest (axis-aligned) rectangle that encloses all black pixels.

**Example:**

**Input:**

[

"0010",

"0110",

"0100"

]

and x = 0, y = 2

**Output:** 6

### 303. Range Sum Query - Immutable

Easy

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

**Example:**

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

**Note:**

1. You may assume that the array does not change.
2. There are many calls to *sumRange* function.

class NumArray **{**

public**:**

NumArray**(**vector**<**int**>** nums**)** **{**

**}**

**};**

class NumArray **{**

public**:**

NumArray**(**vector**<**int**>** nums**)** **{**

int n **=** nums**.**size**();**

sum**.**resize**(**n**+**1**);**

**for** **(**int i **=** 1**;** i **<=** n**;** **++**i**)** **{**

sum**[**i**]** **=** sum**[**i**-**1**]+**nums**[**i**-**1**];**

**}**

**}**

int sumRange**(**int i**,** int j**)** **{**

**return** sum**[**j**+**1**]-**sum**[**i**];**

**}**

private**:**

vector**<**int**>** sum**;**

**};**

### 304. Range Sum Query 2D - Immutable

Medium

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

  
The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Example:**

Given matrix = [

[3, 0, 1, 4, 2],

[5, 6, 3, 2, 1],

[1, 2, 0, 1, 5],

[4, 1, 0, 1, 7],

[1, 0, 3, 0, 5]

]

sumRegion(2, 1, 4, 3) -> 8

sumRegion(1, 1, 2, 2) -> 11

sumRegion(1, 2, 2, 4) -> 12

**Note:**

1. You may assume that the matrix does not change.
2. There are many calls to *sumRegion* function.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

class NumMatrix **{**

public**:**

NumMatrix**(**vector**<**vector**<**int**>>** matrix**)** **{**

**if** **(**matrix**.**empty**())** **return;**

int n **=** matrix**.**size**(),** m **=** matrix**[**0**].**size**();**

sum**.**resize**(**n**,** vector**<**int**>(**m**,** 0**));**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **{**

int rowSum **=** 0**;**

**for** **(**int j **=** 0**;** j **<** m**;** j**++)** **{**

rowSum **+=** matrix**[**i**][**j**];**

**if** **(**i **==** 0**)** sum**[**i**][**j**]** **=** rowSum**;**

**else** sum**[**i**][**j**]** **=** sum**[**i**-**1**][**j**]+**rowSum**;**

**}**

**}**

**}**

int sumRegion**(**int row1**,** int col1**,** int row2**,** int col2**)** **{**

**if** **(**sum**.**empty**())** **return** 0**;**

**else return** get\_sum**(**row2**,** col2**)+**get\_sum**(**row1**-**1**,** col1**-**1**)**

**-**get\_sum**(**row2**,** col1**-**1**)-**get\_sum**(**row1**-**1**,** col2**);**

**}**

private**:**

vector**<**vector**<**int**>>** sum**;**

int get\_sum**(**int i**,** int j**)** **{**

**if** **(**i **<** 0 **||** j **<** 0**)** **return** 0**;**

**else** **return** sum**[**i**][**j**];**

**}**

**};**

### 305. Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each *addLand* operation**. 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:**

**Input:** m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]

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

**Explanation:**

Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

0 0 0

0 0 0

0 0 0

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

1 0 0

0 0 0 Number of islands = 1

0 0 0

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

1 1 0

0 0 0 Number of islands = 1

0 0 0

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

1 1 0

0 0 1 Number of islands = 2

0 0 0

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

1 1 0

0 0 1 Number of islands = 3

0 1 0

**Follow up:**

Can you do it in time complexity O(k log mn), where k is the length of the positions?

class Solution **{**

public**:**

vector**<**int**>** numIslands2**(**int m**,** int n**,** vector**<**vector**<**int**>>&** positions**)** **{**

vector**<**vector**<**int**>>** dirs**{{**0**,** **-**1**},** **{-**1**,** 0**},** **{**0**,** 1**},** **{**1**,** 0**}};**

vector**<**int**>** fa**(**m**\***n**,** **-**1**),** res**;**

// -1 water, other land标签

int cnt **=** 0**;**

**for** **(**auto **&**v **:** positions**)** **{**

int id **=** n**\***v**[**0**]** **+** v**[**1**];**

**if** **(**fa**[**id**]** **==** **-**1**)** **{**

fa**[**id**]** **=** id**;**

**++**cnt**;**

**}**

**for** **(**auto dir **:** dirs**)** **{**

int x **=** v**[**0**]+**dir**[**0**],** y **=** v**[**1**]+**dir**[**1**],** cur\_id **=** n**\***x **+** y**;**

**if** **(**x **<** 0 **||** x **>=** m **||** y **<** 0 **||** y **>=** n **||** fa**[**cur\_id**]** **==** **-**1**)**

**continue;**

int p **=** find**(**fa**,** cur\_id**),** q **=** find**(**fa**,** id**);**

**if** **(**p **!=** q**)** **{**

fa**[**p**]** **=** q**;**

**--**cnt**;**

**}**

**}**

res**.**push\_back**(**cnt**);**

**}**

**return** res**;**

**}**

private**:**

int find**(**vector**<**int**>** **&**fa**,** int x**)** **{**

**return** **(**x **==** fa**[**x**])** **?** x **:** **(**fa**[**x**]** **=** find**(**fa**,** fa**[**x**]));**

**}**

**};**

### 306. Additive Number

Medium

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits '0'-'9', write a function to determine if it's an additive number.

**Note:** Numbers in the additive sequence **cannot** have leading zeros, so sequence 1, 2, 03 or 1, 02, 3 is invalid.

**Example 1:**

**Input:** "112358"

**Output:** true

**Explanation:** The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.

  1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8

**Example 2:**

**Input:** "199100199"

**Output:** true

**Explanation:** The additive sequence is: 1, 99, 100, 199.

  1 + 99 = 100, 99 + 100 = 199

**Follow up:**  
How would you handle overflow for very large input integers?

class Solution **{**

public**:**

bool isAdditiveNumber**(**string num**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool isAdditiveNumber**(**string num**)** **{**

int n **=** num**.**length**();**

**for** **(**int i **=** 1**;** i **<=** n**/**2**;** i**++)** **{**

string x **=** num**.**substr**(**0**,** i**);**

**for** **(**int j **=** i**+**1**;** j **<** n**;** j**++)** **{**

**if** **(**num**[**i**]** **==** '0' **&&** j**-**i **!=** 1**)** **continue;**

string y **=** num**.**substr**(**i**,** j**-**i**);**

**if (**dfs**(**x**,** y**,** j**,** num**,** 2**))** **return** **true;**

**}**

**}**

**return** **false;**

**}**

private**:**

bool dfs**(**string x**,** string y**,** int start**,** string **&**num**,** int cnt**)** **{**

**if** **(**start **==** num**.**length**()** **&&** cnt **>** 2**)** **return** **true;**

string s **=** add**(**x**,** y**);**

**if(**s **!=** num**.**substr**(**start**,** s**.**length**()))** **return** **false;**

**return** dfs**(**y**,** s**,** start**+**s**.**length**(),** num**,** cnt**+**1**);**

**}**

string add**(**string x**,** string y**)** **{**

reverse**(**x**.**begin**(),** x**.**end**());**

reverse**(**y**.**begin**(),** y**.**end**());**

string ret**;**

int c **=** 0**,** p **=** 0**,** q **=** 0**,** n **=** x**.**length**(),** m **=** y**.**length**();**

**while** **(**p **<** n **||** q **<** m**)** **{**

c **+=** **(**p **<** n **?** x**[**p**++]** **-**'0' **:** 0 **)** **+** **(**q **<** m **?** y**[**q**++]** **-**'0' **:** 0 **);**

ret **+=** c **%** 10 **+** '0'**;**

c **/=** 10**;**

**}**

**if** **(**c**)** ret **+=** '1'**;**

reverse**(**ret**.**begin**(),** ret**.**end**());**

**return** ret**;**

**}**

**};**

### 307. Range Sum Query - Mutable

Medium

Given an integer array *nums*, find the sum of the elements between indices *i* and *j* (*i* ≤ *j*), inclusive.

The *update(i, val)* function modifies *nums* by updating the element at index *i* to *val*.

**Example:**

Given nums = [1, 3, 5]

sumRange(0, 2) -> 9

update(1, 2)

sumRange(0, 2) -> 8

**Note:**

1. The array is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRange* function is distributed evenly.

### 308. Range Sum Query 2D - Mutable

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).

The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

**Example:**

Given matrix = [

[3, 0, 1, 4, 2],

[5, 6, 3, 2, 1],

[1, 2, 0, 1, 5],

[4, 1, 0, 1, 7],

[1, 0, 3, 0, 5]

]

sumRegion(2, 1, 4, 3) -> 8

update(3, 2, 2)

sumRegion(2, 1, 4, 3) -> 10

**Note:**

1. The matrix is only modifiable by the *update* function.
2. You may assume the number of calls to *update* and *sumRegion* function is distributed evenly.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

### 309. Best Time to Buy and Sell Stock with Cooldown

Medium

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

* You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
* After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

**Example:**

**Input:** [1,2,3,0,2]

**Output:** 3

**Explanation:** transactions = [buy, sell, cooldown, buy, sell]

### 310. Minimum Height Trees(●ˇ∀ˇ●)

Medium

For an undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

**Format**  
The graph contains n nodes which are labeled from 0 to n - 1. You will be given the number n and a list of undirected edges (each edge is a pair of labels).

You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

**Example 1 :**

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

0

|

1

/ \

2 3

**Output:** [1]

**Example 2 :**

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

0 1 2

\ | /

3

|

4

|

5

**Output:** [3, 4]

class Solution **{**

public**:**

vector**<**int**>** findMinHeightTrees**(**int n**,** vector**<**pair**<**int**,** int**>>&** edges**)** **{**

vector**<**unordered\_set**<**int**>>** adj**(**n**);**

**for** **(**auto **&**p **:** edges**)** **{**

adj**[**p**.**first**].**insert**(**p**.**second**);**

adj**[**p**.**second**].**insert**(**p**.**first**);**

**}**

vector**<**int**>** current**;**

**if** **(**n **==** 1**)** **{**

current**.**push\_back**(**0**);**

**return** current**;**

**}**

// Create first leaf layer

**for** **(**int i **=** 0**;** i **<** adj**.**size**();** **++**i**)** **{**

**if** **(**adj**[**i**].**size**()** **==** 1**)** **{**

current**.**push\_back**(**i**);**

**}**

**}**

// BFS the graph

**while** **(**1**)** **{**

vector**<**int**>** next**;**

**for** **(**int node **:** current**)** **{**

**for** **(**int neighbor **:** adj**[**node**])** **{**

adj**[**neighbor**].**erase**(**node**);**

**if** **(**adj**[**neighbor**].**size**()** **==** 1**)** next**.**push\_back**(**neighbor**);**

**}**

**}**

**if** **(**next**.**empty**())** **return** current**;**

current **=** next**;**

**}**

**}**

**};**

### 311. Sparse Matrix Multiplication

Given two [sparse matrices](https://en.wikipedia.org/wiki/Sparse_matrix) **A** and **B**, return the result of **AB**.

You may assume that **A**'s column number is equal to **B**'s row number.

**Example:**

**Input:**

**A** = [

[ 1, 0, 0],

[-1, 0, 3]

]

**B** = [

[ 7, 0, 0 ],

[ 0, 0, 0 ],

[ 0, 0, 1 ]

]

**Output:**

| 1 0 0 | | 7 0 0 | | 7 0 0 |

**AB** = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |

| 0 0 1 |

class Solution **{**

public**:**

vector**<**vector**<**int**>>** multiply**(**vector**<**vector**<**int**>>** **&**A**,** vector**<**vector**<**int**>>** **&**B**)** **{**

int n **=** A**.**size**(),** l **=** B**.**size**(),** m **=** B**[**0**].**size**();**

vector**<**vector**<**int**>>** res**(**n**,** vector**<**int**>** **(**m**,** 0**));**

vector**<**pair**<**int**,** int**>>** v**[**l**];**

**for** **(**int i **=** 0**;** i **<** l**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** m**;** j**++)** **if** **(**B**[**i**][**j**])** **{**

v**[**i**].**push\_back**({**j**,** B**[**i**][**j**]});**

**}**

**}**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

**for** **(**int k **=** 0**;** k **<** l**;** **++**k**)** **if** **(**A**[**i**][**k**])** **{**

**for** **(**auto **&**pii **:** v**[**k**])** **{**

res**[**i**][**pii**.**first**]** **+=** A**[**i**][**k**]** **\*** pii**.**second**;**

**}**

**}**

**}**

**return** res**;**

**}**

**};**

### 312. Burst Balloons

Hard

Given n balloons, indexed from 0 to n-1. Each balloon is painted with a number on it represented by array nums. You are asked to burst all the balloons. If the you burst balloon i you will get nums[left] \* nums[i] \* nums[right] coins. Here left and right are adjacent indices of i. After the burst, the left and right then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

**Note:**

* You may imagine nums[-1] = nums[n] = 1. They are not real therefore you can not burst them.
* 0 ≤ n ≤ 500, 0 ≤ nums[i] ≤ 100

**Example:**

**Input:** [3,1,5,8]

**Output:** 167

**Explanation:** nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []

  coins = 3\*1\*5 + 3\*5\*8 + 1\*3\*8 + 1\*8\*1 = 167

### 313. Super Ugly Number

Medium

Write a program to find the nth super ugly number.

Super ugly numbers are positive numbers whose all prime factors are in the given prime list primes of size k.

**Example:**

**Input:** n = 12, primes = [2,7,13,19]

**Output:** 32

**Explanation:** [1,2,4,7,8,13,14,16,19,26,28,32] is the sequence of the first 12

super ugly numbers given primes = [2,7,13,19] of size 4.

**Note:**

* 1 is a super ugly number for any given primes.
* The given numbers in primes are in ascending order.
* 0 < k ≤ 100, 0 < n ≤ 106, 0 < primes[i] < 1000.
* The nth super ugly number is guaranteed to fit in a 32-bit signed integer.

class Solution **{**

public**:**

int nthSuperUglyNumber**(**int n**,** vector**<**int**>&** primes**)** **{**

**}**

**};**

class Solution **{**

public**:**

int nthSuperUglyNumber**(**int n**,** vector**<**int**>&** primes**)** **{**

int sz **=** primes**.**size**();**

vector**<**int**>** k**(**sz**,** 0**),** res**(**1**,** 1**);**

**while** **(--**n**)** **{**

int min\_v **=** INT\_MAX**;**

**for** **(**int i **=** 0**;** i **<** sz**;** i**++)** **{**

min\_v **=** min**(**min\_v**,** primes**[**i**]\***res**[**k**[**i**]]);**

**}**

res**.**push\_back**(**min\_v**);**

**for** **(**int i **=** 0**;** i **<** sz**;** i**++)** **{**

**if** **(**primes**[**i**]\***res**[**k**[**i**]]** **==** min\_v**)**

k**[**i**]++;**

**}**

**}**

**return** res**.**back**();**

**}**

**};**

class Solution **{**

public**:**

int nthSuperUglyNumber**(**int n**,** vector**<**int**>&** primes**)** **{**

priority\_queue**<**int**,** vector**<**int**>,** greater**<**int**>>** pq**;**

unordered\_set**<**int**>** My\_set**;**

pq**.**push**(**1**);**

**while** **(--**n**)** **{**

int t **=** pq**.**top**();**

pq**.**pop**();**

**for** **(**int i **:** primes**)** **{**

**if (**INT\_MAX **/** i **<** t**)** **continue;**

i **\*=** t**;**

**if** **(**My\_set**.**count**(**i**))** **continue;**

My\_set**.**insert**(**i**);**

pq**.**push**(**i**);**

**}**

**}**

**return** pq**.**top**();**

**}**

**};**

### 314. Binary Tree Vertical Order Traversal

Given a binary tree, return the *vertical order* traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from **left to right**.

**Examples 1:**

**Input:** [3,9,20,null,null,15,7]

3

/\

/ \

9 20

/\

/ \

15 7

**Output:**

[

[9],

[3,15],

[20],

[7]

]

**Examples 2:**

**Input:** [3,9,8,4,0,1,7]

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

**Output:**

[

[4],

[9],

[3,0,1],

[8],

[7]

]

**Examples 3:**

**Input:** [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5)

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

/\

/ \

5 2

**Output:**

[

[4],

[9,5],

[3,0,1],

[8,2],

[7]

]

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution **{**

public**:**

vector**<**vector**<**int**>>** verticalOrder**(**TreeNode**\*** root**)** **{**

vector**<**vector**<**int**>>** res**;**

**if** **(!**root**)** **return** res**;**

map**<**int**,** vector**<**int**>>** m**;**

queue**<**pair**<**TreeNode**\*,** int**>>** q**;**

q**.**push**({**root**,** 0**});**

**while(!**q**.**empty**())** **{**

auto t **=** q**.**front**();**

q**.**pop**();**

m**[**t**.**second**].**push\_back**(**t**.**first**->**val**);**

**if** **(**t**.**first**->**left**)** q**.**push**({**t**.**first**->**left**,** t**.**second**-**1**});**

**if** **(**t**.**first**->**right**)** q**.**push**({**t**.**first**->**right**,** t**.**second**+**1**});**

**}**

**for** **(**auto **&**i **:** m**)** res**.**push\_back**(**i**.**second**);**

**return** res**;**

**}**

**};**

### 315. Count of Smaller Numbers After Self(●ˇ∀ˇ●)

Hard

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:**

**Input:** [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.

class Solution **{**

public**:**

vector**<**int**>** countSmaller**(**vector**<**int**>&** nums**)** **{**

int n **=** nums**.**size**();**

vector**<**pair**<**int**,** int**>>** v**(**n**);**

vector**<**int**>** res**(**n**,** 0**);**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** v**[**i**]** **=** **{**nums**[**i**],** i**};**

merge\_sort**(**v**.**begin**(),** v**.**end**(),** res**);**

**return** res**;**

**}**

private**:**

void merge\_sort**(**vector**<**pair**<**int**,** int**>>::** iterator l**,** vector**<**pair**<**int**,** int**>>::** iterator r**,** vector**<**int**>** **&**res**)** **{**

**if** **(**l**+**1 **<** r**)** **{**

auto mid **=** l **+** **(**r**-**l**)/**2**;**

merge\_sort**(**l**,** mid**,** res**);**

merge\_sort**(**mid**,** r**,** res**);**

merge**(**l**,** r**,** res**);**

**}**

**}**

void merge**(**vector**<**pair**<**int**,** int**>>::** iterator l**,** vector**<**pair**<**int**,** int**>>::** iterator r**,** vector**<**int**>** **&**res**)** **{**

vector**<**pair**<**int**,** int**>>** t**(**r**-**l**);**

int cnt **=** 0**;**

auto mid **=** l **+** **(**r**-**l**)/**2**,** p **=** l**,** q **=** mid**;**

**while** **(**p **<** mid **||** q **<** r**)** **{**

**if** **(**p **<** mid **&&** q **<** r**)** **{**

**if** **(**q**->**first **>=** p**->**first**)** t**[**cnt**++]** **=** **\***q**++;**

**else** **{**

res**[**p**->**second**]** **+=** r**-**q**;**

t**[**cnt**++]** **=** **\***p**++;**

**}**

**}**

**else** **if** **(**p **<** mid**)** t**[**cnt**++]** **=** **\***p**++;**

**else** t**[**cnt**++]** **=** **\***q**++;**

**}**

int k **=** 0**;**

**for(**auto i **=** l**;** i **<** r**;** i**++)** **\***i **=** t**[**k**++];**

**}**

**};**

### 316. Remove Duplicate Letters

Hard

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

**Example 1:**

**Input:** "bcabc"

**Output:** "abc"

**Example 2:**

**Input:** "cbacdcbc"

**Output:** "acdb"

### 317. Shortest Distance from All Buildings

You want to build a house on an *empty* land which reaches all buildings in the shortest amount of distance. You can only move up, down, left and right. You are given a 2D grid of values **0**, **1** or **2**, where:

* Each **0** marks an empty land which you can pass by freely.
* Each **1** marks a building which you cannot pass through.
* Each **2** marks an obstacle which you cannot pass through.

**Example:**

**Input:** [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

1 - 0 - 2 - 0 - 1

| | | | |

0 - 0 - 0 - 0 - 0

| | | | |

0 - 0 - 1 - 0 - 0

**Output:** 7

**Explanation:** Given three buildings at (0,0), (0,4), (2,2), and an obstacle at (0,2),

the point (1,2) is an ideal empty land to build a house, as the total

  travel distance of 3+3+1=7 is minimal. So return 7.

**Note:**  
There will be at least one building. If it is not possible to build such house according to the above rules, return -1.

### 318. Maximum Product of Word Lengths(●ˇ∀ˇ●)

Medium

Given a string array words, find the maximum value of length(word[i]) \* length(word[j]) where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

**Example 1:**

**Input:** ["abcw","baz","foo","bar","xtfn","abcdef"]

**Output:** 16

**Explanation:** The two words can be "abcw", "xtfn".

**Example 2:**

**Input:** ["a","ab","abc","d","cd","bcd","abcd"]

**Output:** 4

**Explanation:** The two words can be "ab", "cd".

**Example 3:**

**Input:** ["a","aa","aaa","aaaa"]

**Output:** 0

**Explanation:** No such pair of words.

class Solution **{**

public**:**

int maxProduct**(**vector**<**string**> &**words**)** **{**

**}**

**};**

class Solution **{**

public**:**

int maxProduct**(**vector**<**string**> &**words**)** **{**

vector**<**int**>** mask**(**words**.**size**());**

int res **=** 0**;**

**for** **(**int i **=** 0**;** i **<** words**.**size**();** **++**i**)** **{**

**for** **(**char c **:** words**[**i**])**

mask**[**i**]** **|=** 1 **<<** **(**c **-** 'a'**);**

**for** **(**int j **=** 0**;** j **<** i**;** **++**j**)** **if** **(!(**mask**[**i**]** **&** mask**[**j**]))** **{**

res **=** max**(**res**,** int**(**words**[**i**].**size**()\***words**[**j**].**size**()));**

**}**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

int maxProduct**(**vector**<**string**>&** words**)** **{**

unordered\_map**<**int**,** int**>** maxlen**;**

**for** **(**string word **:** words**)** **{**

int mask **=** 0**;**

**for** **(**char c **:** word**)** mask **|=** 1 **<<** **(**c **-** 'a'**);**

maxlen**[**mask**]** **=** max**(**maxlen**[**mask**],** **(**int**)**word**.**size**());**

**}**

int res **=** 0**;**

**for** **(**auto a **:** maxlen**)** **{**

**for** **(**auto b **:** maxlen**)**

**if** **(!(**a**.**first **&** b**.**first**))**

res **=** max**(**res**,** a**.**second **\*** b**.**second**);**

**}**

**return** res

**}**

**};**

### 319. Bulb Switcher

Medium

There are *n* bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the *i*-th round, you toggle every *i* bulb. For the *n*-th round, you only toggle the last bulb. Find how many bulbs are on after *n* rounds.

**Example:**

**Input:** 3

**Output:** 1

**Explanation:**

At first, the three bulbs are **[off, off, off]**.

After first round, the three bulbs are **[on, on, on]**.

After second round, the three bulbs are **[on, off, on]**.

After third round, the three bulbs are **[on, off, off]**.

So you should return 1, because there is only one bulb is on.

class Solution **{**

public**:**

int bulbSwitch**(**int n**)** **{**

// return sqrt(n); /\*这是标答， 以下答案超时\*/

bitset**<**100000000**>** bit**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** **{**

**for** **(**int j **=** i**;** j **<=** n**;** j **+=** i**)** **{**

bit**.**flip**(**j**);**

**}**

**}**

**return** bit**.**count**();**

**}**

**};**

/\*

对于一个叫做bit的bitset：

bit.size() 返回大小（位数）

bit.count() 返回1的个数

bit.any() 返回是否有1

bit.none() 返回是否没有1

bit.set() 全都变成1

bit.set(p) 将第p + 1位变成1（bitset是从第0位开始的！）

bit.set(p, x) 将第p + 1位变成x

bit.reset() 全都变成0

bit.reset(p) 将第p + 1位变成0

bit.flip() 全都取反

bit.flip(p) 将第p + 1位取反

bit.to\_ulong() 返回它转换为unsigned long的结果，如果超出范围则报错

bit.to\_ullong() 返回它转换为unsigned long long的结果，如果超出范围则报错

bit.to\_string() 返回它转换为string的结果

\*/

### 320. Generalized Abbreviation

Write a function to generate the generalized abbreviations of a word.

**Note:**The order of the output does not matter.

**Example:**

**Input:** "word"

**Output:**

["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]

class Solution **{**

public**:**

vector**<**string**>** generateAbbreviations**(**string word**)** **{**

**if** **(**word**.**empty**())** **return** **{**""**};**

vector**<**string**>** pre**{**""**};**

**for(**char **&**c **:** word**)** **{**

vector**<**string**>** cur**;**

**for** **(**auto **&**s **:** pre**)** **{**

cur**.**push\_back**(**s**+**c**);**

**if** **(**isdigit**(**s**.**back**()))** **{**

int i **=** s**.**size**()-**1**;**

**while(**i **>=** 1 **&&** isdigit**(**s**[**i**-**1**]))** i**--;**

cur**.**push\_back**(**s**.**substr**(**0**,** i**)** **+** to\_string**(**stoi**(**s**.**substr**(**i**))+**1**));**

**}**

**else** cur**.**push\_back**(**s**+**"1"**);**

**}**

pre **=** cur**;**

**}**

**return** pre**;**

**}**

**};**

### 321. Create Maximum Number

Hard

Given two arrays of length m and n with digits 0-9 representing two numbers. Create the maximum number of length k <= m + n from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the k digits.

**Note:** You should try to optimize your time and space complexity.

**Example 1:**

**Input:**

nums1 = [3, 4, 6, 5]

nums2 = [9, 1, 2, 5, 8, 3]

k = 5

**Output:**

[9, 8, 6, 5, 3]

**Example 2:**

**Input:**

nums1 = [6, 7]

nums2 = [6, 0, 4]

k = 5

**Output:**

[6, 7, 6, 0, 4]

**Example 3:**

**Input:**

nums1 = [3, 9]

nums2 = [8, 9]

k = 3

**Output:**

[9, 8, 9]

### 322. Coin Change(●ˇ∀ˇ●)

Medium

You are given coins of different denominations and a total amount of money *amount*. Write a function to compute 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.

**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

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

class Solution **{**

public**:**

int coinChange**(**vector**<**int**>&** coins**,** int amount**)** **{**

**}**

**};**

class Solution **{**

public**:**

int coinChange**(**vector**<**int**>&** coins**,** int amount**)** **{**

int Max **=** amount **+** 1**;**

vector**<**int**>** dp**(**amount **+** 1**,** Max**);**

dp**[**0**]** **=** 0**;**

**for** **(**int i **=** 1**;** i **<=** amount**;** i**++)** **{**

**for** **(**int j **=** 0**;** j **<** coins**.**size**();** j**++)** **{**

**if** **(**coins**[**j**]** **<=** i**)** **{**

dp**[**i**]** **=** min**(**dp**[**i**],** dp**[**i**-**coins**[**j**]]+**1**);**

**}**

**}**

**}**

**return** dp**[**amount**]** **>** amount **?** **-**1 **:** dp**[**amount**];**

**}**

**};**

class Solution **{**

public**:**

int coinChange**(**vector**<**int**>&** coins**,** int amount**)** **{**

int Max **=** amount **+** 1**;**

vector**<**int**>** dp**(**amount**+**1**,** Max**);**

dp**[**0**]** **=** 0**;**

**for(**int i **=** 0**;** i **<** coins**.**size**();** i**++)** **{**

**for(**int j **=** coins**[**i**];** j **<=** amount**;** j**++)**

dp**[**j**]** **=** min**(**dp**[**j**],** dp**[**j**-**coins**[**i**]]** **+** 1**);**

**}**

**return** dp**[**amount**]** **==** Max **?** **-**1 **:** dp**[**amount**];**

**}**

**};**

### 323. Number of Connected Components in an Undirected Graph

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

**Example 1:**

**Input:** n = 5 and edges = [[0, 1], [1, 2], [3, 4]]

0 3

| |

1 --- 2 4

**Output:** 2

**Example 2:**

**Input:** n = 5 and edges = [[0, 1], [1, 2], [2, 3], [3, 4]]

0 4

| |

1 --- 2 --- 3

**Output:**1

**Note:**  
You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

class Solution **{**

public**:**

int countComponents**(**int n**,** vector**<**vector**<**int**>>&** edges**)** **{**

vector**<**vector**<**int**>>** g**(**n**);**

vector**<**bool**>** v**(**n**,** **false);**

**for** **(**auto **&**i **:** edges**)** **{**

g**[**i**[**0**]].**push\_back**(**i**[**1**]);**

g**[**i**[**1**]].**push\_back**(**i**[**0**]);**

**}**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **if** **(!**v**[**i**])** **{**

dfs**(**i**,** g**,** v**);**

res**++;**

**}**

**return** res**;**

**}**

private**:**

int res **=** 0**;**

void dfs**(**int i**,** vector**<**vector**<**int**>>** **&**g**,** vector**<**bool**>** **&**v**)** **{**

v**[**i**]** **=** **true;**

**for** **(**auto **&**j **:** g**[**i**])** **if** **(!**v**[**j**])** **{**

dfs**(**j**,** g**,** v**);**

**}**

**}**

**};**

### 324. Wiggle Sort II

Medium

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

**Example 1:**

**Input:** nums = [1, 5, 1, 1, 6, 4]

**Output:** One possible answer is [1, 4, 1, 5, 1, 6].

**Example 2:**

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

**Output:** One possible answer is [2, 3, 1, 3, 1, 2].

**Note:**  
You may assume all input has valid answer.

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

### 325. Maximum Size Subarray Sum Equals k

Given an array *nums* and a target value *k*, find the maximum length of a subarray that sums to *k*. If there isn't one, return 0 instead.

**Note:**  
The sum of the entire *nums* array is guaranteed to fit within the 32-bit signed integer range.

**Example 1:**

**Input:** *nums* = [1, -1, 5, -2, 3], *k* = 3

**Output:** 4

**Explanation:** The subarray [1, -1, 5, -2] sums to 3 and is the longest.

**Example 2:**

**Input:** *nums* = [-2, -1, 2, 1], *k* = 1

**Output:** 2

**Explanation:** The subarray [-1, 2] sums to 1 and is the longest.

**Follow Up:**  
Can you do it in O(*n*) time?

class Solution **{**

public**:**

int maxSubArrayLen**(**vector**<**int**>** **&**nums**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

int maxSubArrayLen**(**vector**<**int**>** **&**nums**,** int k**)** **{**

int sum **=** 0**,** res **=** 0**,** n **=** nums**.**size**();**

unordered\_map**<**int**,** int**>** m**;**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

sum **+=** nums**[**i**];**

**if** **(**sum **==** k**)** res **=** i **+** 1**;**

**else** **if** **(**m**.**count**(**sum **-** k**))** res **=** max**(**res**,** i**-**m**[**sum**-**k**]);**

**if** **(!**m**.**count**(**sum**))** m**[**sum**]** **=** i**;**

**}**

**return** res**;**

**}**

**};**

### 326. Power of Three

Easy

Given an integer, write a function to determine if it is a power of three.

**Example 1:**

**Input:** 27

**Output:** true

**Example 2:**

**Input:** 0

**Output:** false

**Example 3:**

**Input:** 9

**Output:** true

**Example 4:**

**Input:** 45

**Output:** false

**Follow up:**  
Could you do it without using any loop / recursion?

class Solution **{**

public**:**

bool isPowerOfThree**(**int n**)** **{**

**if(**n **<=** 0**)** **return** **false;**

**return** pow**(**3**,** **(**int**)(**log10**(**n**)** **/** log10**(**3**)+**0.5**)** **)** **==** n**;**

// return n > 0 && (1162261467 % n == 0);

**}**

**};**

### 327. Count of Range Sum

Hard

Given an integer array nums, return the number of range sums that lie in [lower, upper] inclusive.  
Range sum S(i, j) is defined as the sum of the elements in nums between indices i and j (i ≤ j), inclusive.

**Note:**  
A naive algorithm of *O*(*n*2) is trivial. You MUST do better than that.

**Example:**

**Input:** *nums* = [-2,5,-1], *lower* = -2, *upper* = 2,

**Output:** 3

**Explanation:** The three ranges are : [0,0], [2,2], [0,2] and their respective sums are: -2, -1, 2.

### 328. Odd Even Linked List

Medium

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

**Example 1:**

**Input:** 1->2->3->4->5->NULL

**Output:** 1->3->5->2->4->NULL

**Example 2:**

**Input:** 2->1->3->5->6->4->7->NULL

**Output:** 2->3->6->7->1->5->4->NULL

**Note:**

* The relative order inside both the even and odd groups should remain as it was in the input.
* The first node is considered odd, the second node even and so on ...

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** oddEvenList**(**ListNode**\*** head**)** **{**

**}**

**};**

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode(int x) : val(x), next(NULL) {}

\* };

\*/

class Solution **{**

public**:**

ListNode**\*** oddEvenList**(**ListNode**\*** head**)** **{**

ListNode **\***even **=** **new** ListNode**(-**1**),** **\***odd **=** **new** ListNode**(-**1**);**

ListNode **\***q **=** even**,** **\***p **=** odd**,** **\***t **=** head**;**

**while** **(**t**)** **{**

p**->**next **=** t**;**

p **=** t**;**

t **=** t**->**next**;**

q**->**next **=** t**;**

q **=** t**;**

**if(**t**)** t **=** t**->**next**;**

**}**

p**->**next **=** even**->**next**;**

**return** odd**->**next**;**

**}**

**};**

### 329. Longest Increasing Path in a Matrix

Hard

85417FavoriteShare

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

**Example 1:**

**Input:** nums =

[

[9,9,4],

[6,6,8],

[2,1,1]

]

**Output:** 4

**Explanation:** The longest increasing path is [1, 2, 6, 9].

**Example 2:**

**Input:** nums =

[

[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.

### 330. Patching Array(●ˇ∀ˇ●)

Hard

Given a sorted positive integer array *nums* and an integer *n*, add/patch elements to the array such that any number in range [1, n] inclusive can be formed by the sum of some elements in the array. Return the minimum number of patches required.

**Example 1:**

**Input:** *nums* = [1,3], *n* = 6

**Output:** 1

**Explanation:**

Combinations of *nums* are [1], [3], [1,3], which form possible sums of: 1, 3, 4.

Now if we add/patch 2 to *nums*, the combinations are: [1], [2], [3], [1,3], [2,3], [1,2,3].

Possible sums are 1, 2, 3, 4, 5, 6, which now covers the range [1, 6].

So we only need 1 patch.

**Example 2:**

**Input:** *nums* = [1,5,10], *n* = 20

**Output:** 2

**Explanation:** The two patches can be [2, 4].

**Example 3:**

**Input:** *nums* = [1,2,2], *n* = 5

**Output:** 0

class Solution **{**

public**:**

int minPatches**(**vector**<**int**>&** nums**,** int n**)** **{**

**}**

**};**

class Solution **{**

public**:**

int minPatches**(**vector**<**int**>&** nums**,** int n**)** **{**

int cnt **=** 0**,** i **=** 0**;**

long long maxNum **=** 1**;**

**while** **(**maxNum **<=** n**)** **{**

**if** **(**i **<** nums**.**size**()** **&&** nums**[**i**]** **<=** maxNum**)**

maxNum **+=** nums**[**i**++];**

**else** **{**

maxNum **\*=** 2**;**

cnt**++;**

**}**

**}**

**return** cnt**;**

**}**

**};**

### 331. Verify Preorder Serialization of a Binary Tree

Medium

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.

9

/ \

3 2

/ \ / \

4 1 # 6

/ \ / \ / \

# # # # # #

For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as "1,,3".

**Example 1:**

**Input:** "9,3,4,#,#,1,#,#,2,#,6,#,#"

**Output:** true

**Example 2:**

**Input:** "1,#"

**Output:** false

**Example 3:**

**Input:** "9,#,#,1"

**Output:** false

class Solution **{**

public**:**

bool isValidSerialization**(**string preorder**)** **{**

stack**<**bool**>** stk**;**

stringstream ss**(**preorder**+**','**);**

char c**;**

**while** **(**ss **>>** c**)** **{**

bool Isdigit **=** isdigit**(**c**);**

**while(**c **!=** ','**)** ss **>>** c**;**

**if** **(!**insert**(**stk**,** Isdigit**))** **return** **false;**

**}**

**return** stk**.**size**()** **==** 1 **&&** stk**.**top**()** **==** **false;**

**}**

private**:**

bool insert**(**stack**<**bool**>** **&**stk**,** bool Isdigit**)** **{**

**if** **(**Isdigit**)** stk**.**push**(true);**

**else** **if** **(!**stk**.**empty**()** **&&** stk**.**top**()** **==** **false)** **{**

stk**.**pop**();**

**if** **(**stk**.**empty**()** **||** stk**.**top**()** **!=** **true)** **return** **false;**

stk**.**pop**();**

**return** insert**(**stk**,** Isdigit**);**

**}**

**else** stk**.**push**(false);**

**return** **true;**

**}**

**};**

### 332. Reconstruct Itinerary(●ˇ∀ˇ●)

Medium

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

**Note:**

1. If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].
2. All airports are represented by three capital letters (IATA code).
3. You may assume all tickets form at least one valid itinerary.

**Example 1:**

**Input:** [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

**Output:** ["JFK", "MUC", "LHR", "SFO", "SJC"]

**Example 2:**

**Input:** [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

**Output:** ["JFK","ATL","JFK","SFO","ATL","SFO"]

**Explanation:** Another possible reconstruction is ["JFK","SFO","ATL","JFK","ATL","SFO"].

  But it is larger in lexical order.

class Solution **{**

public**:**

// fleury's algorithm or Hierholzer algorithm

vector**<**string**>** findItinerary**(**vector**<**vector**<**string**>>** **&**tickets**)** **{**

**for** **(**auto e **:** tickets**)**

graph**[**e**[**0**]].**push**(**e**[**1**]);**

dfs**(**"JFK"**);**

reverse**(**res**.**begin**(),** res**.**end**());**

**return** res**;**

**}**

private**:**

unordered\_map**<**string**,** priority\_queue**<**string**,** vector**<**string**>,** greater**<**string**>>>** graph**;**

vector**<**string**>** res**;**

void dfs**(**string from**)** **{**

auto **&**edges **=** graph**[**from**];**

**while** **(!**edges**.**empty**())** **{**

string to **=** edges**.**top**();**

edges**.**pop**();**

dfs**(**to**);**

**}**

res**.**push\_back**(**from**);**

**}**

**};**

### 333. Largest BST Subtree

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.

**Note:**  
A subtree must include all of its descendants.

**Example:**

**Input:** [10,5,15,1,8,null,7]

10

/ \

5 15

/ \ \

1 8 7

**Output:** 3

**Explanation:** The Largest BST Subtree in this case is the highlighted one.

The return value is the subtree's size, which is 3.

**Follow up:**  
Can you figure out ways to solve it with O(n) time complexity?

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution **{**

public**:**

int largestBSTSubtree**(**TreeNode**\*** root**)** **{**

int sum **=** 0**,** Min **=** INT\_MAX**,** Max **=** INT\_MIN**;**

dfs**(**root**,** Min**,** Max**,** sum**);**

**return** res**;**

**}**

private**:**

int res **=** 0**;**

bool dfs**(**TreeNode **\***root**,** int **&**Min**,** int **&**Max**,** int **&**sum**)** **{**

**if** **(**root **==** **nullptr)** **{**

Min **=** INT\_MAX**,** Max **=** INT\_MIN**,** sum **=** 0**;**

**return** **true;**

**}**

int l\_sum **=** 0**,** r\_sum **=** 0**;**

int l\_min **=** INT\_MAX**,** r\_min **=** INT\_MAX**,** l\_max **=** INT\_MIN**,** r\_max **=** INT\_MIN**;**

bool l\_tree **=** dfs**(**root**->**left**,** l\_min**,** l\_max**,** l\_sum**);**

bool r\_tree **=** dfs**(**root**->**right**,** r\_min**,** r\_max**,** r\_sum**);**

Min **=** min**(**Min**,** min**(**root**->**val**,** l\_min**));**

Max **=** max**(**Max**,** max**(**root**->**val**,** r\_max**));**

**if** **(**l\_tree **&&** r\_tree **&&** l\_max **<** root**->**val **&&** r\_min **>** root**->**val**)** **{**

res **=** max**(**res**,** sum **=** l\_sum **+** r\_sum**+**1**);**

**return** **true;**

**}**

**return** **false;**

**}**

**};**

### 334. Increasing Triplet Subsequence

Medium

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

Formally the function should:

Return true if there exists *i, j, k*   
such that *arr[i]* < *arr[j]* < *arr[k]* given 0 ≤ *i* < *j* < *k* ≤ *n*-1 else return false.

**Note:** Your algorithm should run in O(*n*) time complexity and O(*1*) space complexity.

**Example 1:**

**Input:** [1,2,3,4,5]

**Output:** true

**Example 2:**

**Input:** [5,4,3,2,1]

**Output:** false

class Solution **{**

public**:**

bool increasingTriplet**(**vector**<**int**>&** nums**)** **{**

**}**

**};**

class Solution **{**

public**:**

bool increasingTriplet**(**vector**<**int**>&** nums**)** **{**

int i **=** INT\_MAX**,** j **=** INT\_MAX**;**

**for** **(**auto **&**a **:** nums**)** **{**

**if** **(**a **<=** i**)** i **=** a**;**

**else** **if (**a **<=** j**)** j **=** a**;**

**else** **return** **true;**

**}**

**return** **false;**

**}**

**};**

### 335. Self Crossing

Hard

You are given an array *x* of n positive numbers. You start at point (0,0) and moves x[0] metres to the north, then x[1] metres to the west, x[2] metres to the south, x[3] metres to the east and so on. In other words, after each move your direction changes counter-clockwise.

Write a one-pass algorithm with O(1) extra space to determine, if your path crosses itself, or not.

**Example 1:**

**┌───┐**

**│   │**

**└───┼──>**

**│**

**Input:** [2,1,1,2]

**Output:** true

**Example 2:**

**┌──────┐**

**│      │**

**│**

**│**

**└────────────>**

**Input:** [1,2,3,4]

**Output:** false

**Example 3:**

**┌───┐**

**│   │**

**└───┼>**

**Input:** [1,1,1,1]

**Output:** true

class Solution **{**

public**:**

bool isSelfCrossing**(**vector**<**int**>&** x**)** **{**

x**.**insert**(**x**.**begin**(),** 4**,** 0**);**

int i **=** 4**,** n **=** x**.**size**();**

**for** **(;** i **<** n **&&** x**[**i**]** **>** x**[**i **-** 2**];** i**++);** //逐渐变大

**if** **(**i **==** n**)** **return** **false;**

**if** **(**x**[**i**]** **>=** x**[**i**-**2**]** **-** x**[**i**-**4**])** x**[**i**-**1**]** **-=** x**[**i**-**3**];**

**for** **(**i**++;** i **<** n **&&** x**[**i**]** **<** x**[**i**-**2**];** i**++);** //逐渐变小

**return** i **!=** n**;**

**}**

**};**

### 336. Palindrome Pairs

Hard

Given a list of **unique** words, find all pairs of ***distinct*** indices (i, j) in the given list, so that the concatenation of the two words, i.e. words[i] + words[j] is a palindrome.

**Example 1:**

**Input:** ["abcd","dcba","lls","s","sssll"]

**Output:** [[0,1],[1,0],[3,2],[2,4]]

**Explanation:** The palindromes are ["dcbaabcd","abcddcba","slls","llssssll"]

**Example 2:**

**Input:** ["bat","tab","cat"]

**Output:** [[0,1],[1,0]]

**Explanation:** The palindromes are ["battab","tabbat"]

class Solution **{**

public**:**

vector**<**vector**<**int**>>** palindromePairs**(**vector**<**string**>&** words**)** **{**

vector**<**vector**<**int**>>** res**;**

unordered\_map**<**string**,** int**>** dict**;**

int i**,** j**,** n **=** words**.**size**();**

**for(**i **=** 0**;** i **<** n**;** i**++)** **{**

string t **=** words**[**i**];**

reverse**(**t**.**begin**(),** t**.**end**());**

dict**[**t**]** **=** i**;**

**}**

**for(**i **=** 0**;** i **<** n**;** i**++)** **{**

**for(**j **=** 0**;** j **<** words**[**i**].**size**();** j**++)** **{**

string left **=** words**[**i**].**substr**(**0**,** j**);**

string right **=** words**[**i**].**substr**(**j**);**

**if(**dict**.**count**(**left**)** **&&** dict**[**left**]** **!=** i **&&** isPalindrome**(**right**)){**

res**.**push\_back**({**i**,** dict**[**left**]});**

**if** **(**left**.**empty**())** res**.**push\_back**({**dict**[**left**],** i**});**

**}**

**if** **(**dict**.**count**(**right**)** **&&** dict**[**right**]** **!=** i

**&&** isPalindrome**(**left**))** **{**

res**.**push\_back**({**dict**[**right**],** i**});**

**}**

**}**

**}**

**return** res**;**

**}**

private**:**

bool isPalindrome**(**string s**)** **{**

int start**,** end**,** n **=** s**.**size**();**

**for(**start **=** 0**,** end **=** n **-** 1**;** start **<** end**;** start**++,** end**--)** **{**

**if(**s**[**start**]** **!=** s**[**end**])**

**return** **false;**

**}**

**return** **true;**

**}**

**};**

### 337. House Robber III(●ˇ∀ˇ●)

Medium

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

**Example 1:**

**Input:** [3,2,3,null,3,null,1]

3

/ \

2 3

\ \

3 1

**Output:** 7

**Explanation:** Maximum amount of money the thief can rob = 3 + 3 + 1 = **7**.

**Example 2:**

**Input:** [3,4,5,1,3,null,1]

  3

/ \

4 5

/ \ \

1 3 1

**Output:** 9

**Explanation:** Maximum amount of money the thief can rob = 4 + 5 = **9**.

class Solution **{**

public**:**

int rob**(**TreeNode**\*** root**)** **{**

int l**,** r**;**

**return** tryRob**(**root**,** l**,** r**);**

**}**

private**:**

int tryRob**(**TreeNode **\***root**,** int **&**l**,** int **&**r**)** **{**

// l为左子树最大值（不考虑父亲节点偷不偷）

**if** **(!**root**)** **return** 0**;**

int ll **=** 0**,** lr **=** 0**,** rl **=** 0**,** rr **=** 0**;**

l **=** tryRob**(**root**->**left**,** ll**,** lr**);**

r **=** tryRob**(**root**->**right**,** rl**,** rr**);**

**return** max**(**root**->**val **+** ll **+** lr **+** rl **+** rr**,** l **+** r**);**

**}**

**};**

### 338. Counting Bits(●ˇ∀ˇ●)

Medium

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

**Example 1:**

**Input:** 2

**Output:** [0,1,1]

**Example 2:**

**Input:** 5

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

**Follow up:**

* It is very easy to come up with a solution with run time **O(n\*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
* Space complexity should be **O(n)**.
* Can you do it like a boss? Do it without using any builtin function like **\_\_builtin\_popcount** in c++ or in any other language.

class Solution **{**

public**:**

vector**<**int**>** countBits**(**int num**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** countBits**(**int num**)** **{**

vector**<**int**>** res**(**num **+** 1**,** 0**);**

**for** **(**int i **=** 1**;** i **<=** num**;** **++**i**)** **{**

res**[**i**]** **=** res**[**i **>>** 1**]** **+** **(**i **&** 1**);**

**}**

**return** res**;**

**}**

**};**

### 339. Nested List Weight Sum

Given a nested list of integers, return the sum of all integers in the list weighted by their depth.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

**Example 1:**

**Input:** [[1,1],2,[1,1]]

**Output:** 10

**Explanation:** Four 1's at depth 2, one 2 at depth 1.

**Example 2:**

**Input:** [1,[4,[6]]]

**Output:** 27

**Explanation:** One 1 at depth 1, one 4 at depth 2, and one 6 at depth 3; 1 + 4\*2 + 6\*3 = 27.

/\*\*

\* // This is the interface that allows for creating nested lists.

\* // You should not implement it, or speculate about its implementation

\* class NestedInteger {

\* public:

\* // Constructor initializes an empty nested list.

\* NestedInteger();

\*

\* // Constructor initializes a single integer.

\* NestedInteger(int value);

\*

\* // Return true if this NestedInteger holds a single integer, rather than a nested list.

\* bool isInteger() const;

\*

\* // Return the single integer that this NestedInteger holds, if it holds a single integer

\* // The result is undefined if this NestedInteger holds a nested list

\* int getInteger() const;

\*

\* // Set this NestedInteger to hold a single integer.

\* void setInteger(int value);

\*

\* // Set this NestedInteger to hold a nested list and adds a nested integer to it.

\* void add(const NestedInteger &ni);

\*

\* // Return the nested list that this NestedInteger holds, if it holds a nested list

\* // The result is undefined if this NestedInteger holds a single integer

\* const vector<NestedInteger> &getList() const;

\* };

\*/

class Solution **{**

public**:**

int depthSum**(**vector**<**NestedInteger**>&** nestedList**)** **{**

**}**

**};**

class Solution **{**

public**:**

int depthSum**(**vector**<**NestedInteger**>&** nestedList**)** **{**

**return** dfs**(**1**,** nestedList**);**

**}**

private**:**

int dfs**(**int depth**,** vector**<**NestedInteger**>&** nestedList**)** **{**

int ret **=** 0**;**

**for** **(**auto **&**nest **:** nestedList**)** **{**

**if** **(**nest**.**isInteger**())** ret **+=** depth**\***nest**.**getInteger**();**

**else** ret **+=** dfs**(**depth**+**1**,** nest**.**getList**());**

**}**

**return** ret**;**

**}**

**};**

### 340. Longest Substring with At Most K Distinct Characters

Given a string, find the length of the longest substring T that contains at most *k* distinct characters.

**Example 1:**

**Input:** s = "eceba", k = 2

**Output:** 3

**Explanation:** T is "ece" which its length is 3.

**Example 2:**

**Input:** s = "aa", k = 1

**Output:** 2

**Explanation:** T is "aa" which its length is 2.

class Solution **{**

public**:**

int lengthOfLongestSubstringKDistinct**(**string s**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

int lengthOfLongestSubstringKDistinct**(**string s**,** int k**)** **{**

int res **=** 0**,** left **=** 0**,** n **=** s**.**size**();**

unordered\_map**<**char**,** int**>** m**;**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** **{**

**++**m**[**s**[**i**]];**

**while** **(**m**.**size**()** **>** k**)** **{**

**if** **(--**m**[**s**[**left**]]** **==** 0**)** m**.**erase**(**s**[**left**]);**

**++**left**;**

**}**

res **=** max**(**res**,** i **-** left **+** 1**);**

**}**

**return** res**;**

**}**

**};**

### 341. Flatten Nested List Iterator

Medium

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

**Example 1:**

**Input:** [[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:** [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].

/\*\*

\* // This is the interface that allows for creating nested lists.

\* // You should not implement it, or speculate about its implementation

\* class NestedInteger {

\* public:

\* // Return true if this NestedInteger holds a single integer, rather than a nested list.

\* bool isInteger() const;

\*

\* // Return the single integer that this NestedInteger holds, if it holds a single integer

\* // The result is undefined if this NestedInteger holds a nested list

\* int getInteger() const;

\*

\* // Return the nested list that this NestedInteger holds, if it holds a nested list

\* // The result is undefined if this NestedInteger holds a single integer

\* const vector<NestedInteger> &getList() const;

\* };

\*/

class NestedIterator **{**

public**:**

NestedIterator**(**vector**<**NestedInteger**>** **&**nestedList**)** **{**

begins**.**push**(**nestedList**.**begin**());**

ends**.**push**(**nestedList**.**end**());**

**}**

int next**()** **{**

hasNext**();**

**return** **(**begins**.**top**()++)->**getInteger**();**

**}**

bool hasNext**()** **{**

**while** **(!**begins**.**empty**())** **{**

**if** **(**begins**.**top**()** **==** ends**.**top**())** **{**

begins**.**pop**();**

ends**.**pop**();**

**}** **else** **{**

auto x **=** begins**.**top**();**

**if** **(**x**->**isInteger**())** **return** **true;**

begins**.**top**()++;**

begins**.**push**(**x**->**getList**().**begin**());**

ends**.**push**(**x**->**getList**().**end**());**

**}**

**}**

**return** **false;**

**}**

private**:**

stack**<**vector**<**NestedInteger**>::**iterator**>** begins**,** ends**;**

**};**

class NestedIterator **{**

public**:**

NestedIterator**(**vector**<**NestedInteger**>** **&**nestedList**)** **{**

int n **=** nestedList**.**size**();**

**for(**int i **=** n**-**1**;** i **>=** 0**;** **--**i**)** **{**

stk**.**push**(**nestedList**[**i**]);**

**}**

**}**

int next**()** **{**

int result **=** stk**.**top**().**getInteger**();**

stk**.**pop**();**

**return** result**;**

**}**

bool hasNext**()** **{**

**while** **(!**stk**.**empty**())** **{**

auto cur **=** stk**.**top**();**

**if** **(**cur**.**isInteger**())** **return** **true;**

stk**.**pop**();**

auto **&**adjs **=** cur**.**getList**();**

int n **=** adjs**.**size**();**

**for(**int i **=** n**-**1**;** i **>=** 0**;** **--**i**)** stk**.**push**(**adjs**[**i**]);**

**}**

**return** **false;**

**}**

private**:**

stack**<**NestedInteger**>** stk**;**

**};**

### 342. Power of Four

Easy

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

**Example 1:**

**Input:** 16

**Output:** true

**Example 2:**

**Input:** 5

**Output:** false

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

class Solution **{**

public**:**

bool isPowerOfFour**(**int num**)** **{**

**return** num **>** 0 **&&** **(**num **&** **(**num **-** 1**))** **==** 0 **&&** **(**num **-** 1**)** **%** 3 **==** 0**;**

//return (num > 0) && ((num & (num - 1)) == 0) && ((num & 0x55555555) == num);

**}**

**};**

### 343. Integer Break

Medium

Given a positive integer *n*, break it into the sum of **at least** two positive integers and maximize the product of those integers. Return the maximum product you can get.

**Example 1:**

**Input:** 2

**Output:** 1

**Explanation:** 2 = 1 + 1, 1 × 1 = 1.

**Example 2:**

**Input:** 10

**Output:** 36

**Explanation:** 10 = 3 + 3 + 4, 3 × 3 × 4 = 36.

**Note**: You may assume that *n* is not less than 2 and not larger than 58.

### 344. Reverse String

Easy

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

Do not allocate extra space for another array, you must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

You may assume all the characters consist of [printable ascii characters](https://en.wikipedia.org/wiki/ASCII#Printable_characters).

**Example 1:**

**Input:** ["h","e","l","l","o"]

**Output:** ["o","l","l","e","h"]

**Example 2:**

**Input:** ["H","a","n","n","a","h"]

**Output:** ["h","a","n","n","a","H"]

class Solution **{**

public**:**

void reverseString**(**vector**<**char**>&** s**)** **{**

int i **=** 0**,** j **=** s**.**size**()-**1**;**

**while** **(**i **<** j**)** **{**

swap**(**s**[**i**++],** s**[**j**--]);**

**}**

**}**

**};**

### 345. Reverse Vowels of a String

Easy

Write a function that takes a string as input and reverse only the vowels of a string.

**Example 1:**

**Input:** "hello"

**Output:** "holle"

**Example 2:**

**Input:** "leetcode"

**Output:** "leotcede"

**Note:**  
The vowels does not include the letter "y".

class Solution **{**

public**:**

string reverseVowels**(**string s**)** **{**

int i **=** 0**,** j **=** s**.**size**()-**1**;**

**while** **(**i **<** j**)** **{**

i **=** s**.**find\_first\_of**(**"aeiouAEIOU"**,** i**);**

j **=** s**.**find\_last\_of**(**"aeiouAEIOU"**,** j**);**

**if** **(**i **<** j**)** swap**(**s**[**i**++],** s**[**j**--]);**

**}**

**return** s**;**

**}**

**};**

### 346. Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

**Example:**

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3

class MovingAverage **{**

public**:**

/\*\* Initialize your data structure here. \*/

MovingAverage**(**int size**)** **{**

**this->**size **=** size**;**

**}**

double next**(**int val**)** **{**

q**.**push**(**val**);**

**if** **(**q**.**size**()** **>** size**)** **{**

sum **-=** q**.**front**();**

q**.**pop**();**

**}**

**return** **(**sum **+=** val**)/**q**.**size**();**

**}**

private**:**

int size**;**

double sum **=** 0**;**

queue**<**int**>** q**;**

**};**

### 347. Top K Frequent Elements(●ˇ∀ˇ●)

Medium

Given a non-empty array of integers, return the ***k*** most frequent elements.

**Example 1:**

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

**Output:** [1,2]

**Example 2:**

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

**Output:** [1]

**Note:**

* You may assume *k* is always valid, 1 ≤ *k* ≤ number of unique elements.
* Your algorithm's time complexity **must be** better than O(*n* log *n*), where *n* is the array's size.

class Solution **{**

public**:**

vector**<**int**>** topKFrequent**(**vector**<**int**>&** nums**,** int k**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** topKFrequent**(**vector**<**int**> &**nums**,** int k**)** **{**

unordered\_map**<**int**,** int**>** counts**;**

**for(**auto i **:** nums**)** **++**counts**[**i**];**

vector**<**vector**<**int**>>** buckets**(**nums**.**size**()** **+** 1**);**

**for(**auto **&**k **:** counts**)** buckets**[**k**.**second**].**push\_back**(**k**.**first**);**

vector**<**int**>** res**;**

**for(**int t **=** buckets**.**size**()-**1**;** t **>=** 0**;** t**--)** **{**

**for(**auto i **:** buckets**[**t**])** **{**

res**.**push\_back**(**i**);**

**if** **(**res**.**size**()** **==** k**)** **return** res**;**

**}**

**}**

**return** res**;**

**}**

**};**

### 348. Design Tic-Tac-Toe

Design a Tic-tac-toe game that is played between two players on a *n* x *n* grid.

You may assume the following rules:

1. A move is guaranteed to be valid and is placed on an empty block.
2. Once a winning condition is reached, no more moves is allowed.
3. A player who succeeds in placing *n* of their marks in a horizontal, vertical, or diagonal row wins the game.

**Example:**

Given *n* = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)

|X| | |

| | | | // Player 1 makes a move at (0, 0).

| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 2 makes a move at (0, 2).

| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 1 makes a move at (2, 2).

| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 2 makes a move at (1, 1).

| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 1 makes a move at (2, 0).

|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)

|X| |O|

|O|O| | // Player 2 makes a move at (1, 0).

|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)

|X| |O|

|O|O| | // Player 1 makes a move at (2, 1).

|X|X|X|

**Follow up:**  
Could you do better than O(*n*2) per move() operation?

class TicTacToe **{**

public**:**

/\*\* Initialize your data structure here. \*/

TicTacToe**(**int n**)** **{**

**this->**n **=** n**;**

R**.**resize**(**n**,** 0**);**

C**.**resize**(**n**,** 0**);**

**}**

int move**(**int row**,** int col**,** int player**)** **{**

int add **=** **(**player **==** 2**)** **?** **-**1 **:** 1**;**

**if** **(**judge**(**row**,** add**,** R**)** **||** judge**(**col**,** add**,** C**))** **return** player**;**

**else** **if** **(**row **==** col **&&** judge\_diag**(**0**,** add**))** **return** player**;**

**else** **if** **(**row **+** col **==** n**-**1 **&&** judge\_diag**(**1**,** add**))** **return** player**;**

**else** **return** 0**;**

**}**

private**:**

int n**;**

vector**<**int**>** R**,** C**,** diag**{**0**,** 0**};**

bool judge**(**int row**,** int add**,** vector**<**int**>** **&**R**)** **{**

**if** **(**abs**(**R**[**row**]** **+=** add**)** **==** n**)** **return** **true;**

**else** **return** **false;**

**}**

bool judge\_diag**(**int type**,** int add**)** **{**

**if** **(**abs**(**diag**[**type**]** **+=** add**)** **==** n**)** **return** **true;**

**else** **return** **false;**

**}**

**};**

### 349. Intersection of Two Arrays

Easy

Given two arrays, write a function to compute their intersection.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2]

**Example 2:**

**Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4]

**Output:** [9,4]

**Note:**

* Each element in the result must be unique.
* The result can be in any order.

class Solution **{**

public**:**

vector**<**int**>** intersection**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** intersection**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

sort**(**nums1**.**begin**(),** nums1**.**end**());**

sort**(**nums2**.**begin**(),** nums2**.**end**());**

int n1 **=** **(**int**)**nums1**.**size**(),** n2 **=** **(**int**)**nums2**.**size**();**

int i1 **=** 0**,** i2 **=** 0**;**

vector**<**int**>** res**;**

**while** **(**i1 **<** n1 **&&** i2 **<** n2**){**

**if (**nums1**[**i1**]** **==** nums2**[**i2**])** **{**

res**.**push\_back**(**nums1**[**i1**]);**

i1**++;** i2**++;**

**while** **(**i1 **<** n1 **&&** nums1**[**i1**]** **==** nums1**[**i1**-**1**])** i1**++;**

**while** **(**i2 **<** n2 **&&** nums2**[**i2**]** **==** nums2**[**i2**-**1**])** i2**++;**

**}**

**else** **if (**nums1**[**i1**]** **>** nums2**[**i2**])** **{**

i2**++;**

**while** **(**i2 **<** n2 **&&** nums2**[**i2**]** **==** nums2**[**i2**-**1**])** i2**++;**

**}**

**else** **{**

i1**++;**

**while** **(**i1 **<** n1 **&&** nums1**[**i1**]** **==** nums1**[**i1**-**1**])** i1**++;**

**}**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** intersection**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

vector**<**int**>** res**;**

unordered\_map**<**int**,** bool**>** m**;**

**for** **(**auto i **:** nums1**)** m**[**i**]** **=** **true;**

**for** **(**auto i **:** nums2**)** **if** **(**m**[**i**]){**

res**.**push\_back**(**i**);**

m**[**i**]** **=** **false;**

**}**

**return** res**;**

**}**

**};**

### 350. Intersection of Two Arrays II

Easy

Given two arrays, write a function to compute their intersection.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2,2]

**Example 2:**

**Input:** nums1 = [4,9,5], nums2 = [9,4,9,8,4]

**Output:** [4,9]

**Note:**

* Each element in the result should appear as many times as it shows in both arrays.
* The result can be in any order.

**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?

class Solution **{**

public**:**

vector**<**int**>** intersect**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** intersect**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

sort**(**nums1**.**begin**(),** nums1**.**end**());**

sort**(**nums2**.**begin**(),** nums2**.**end**());**

int n1 **=** **(**int**)**nums1**.**size**(),** n2 **=** **(**int**)**nums2**.**size**();**

int i1 **=** 0**,** i2 **=** 0**;**

vector**<**int**>** res**;**

**while** **(**i1 **<** n1 **&&** i2 **<** n2**){**

**if(**nums1**[**i1**]** **==** nums2**[**i2**])** **{**

res**.**push\_back**(**nums1**[**i1**]);**

i1**++;**

i2**++;**

**}**

**else** **if(**nums1**[**i1**]** **>** nums2**[**i2**])** i2**++;**

**else** i1**++;**

**}**

**return** res**;**

**}**

**};**

class Solution **{**

public**:**

vector**<**int**>** intersect**(**vector**<**int**>&** nums1**,** vector**<**int**>&** nums2**)** **{**

vector**<**int**>** res**;**

unordered\_map**<**int**,** int**>** m**;**

**for** **(**auto i **:** nums1**)** m**[**i**]++;**

**for** **(**auto i **:** nums2**)** **if** **(**m**[**i**]** **>** 0**){**

res**.**push\_back**(**i**);**

m**[**i**]--;**

**}**

**return** res**;**

**}**

**};**