**Day 2**

**Array & Linked list**

**Very Easy**

**Q 1 : Majority Elements**

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

The majority element is the element that appears more than ⌊n / 2⌋ 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 == nums.length

1 <= n <= 5 \* 104

-109 <= nums[i] <= 109

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

**Question 2. 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 <= nums.length <= 3 \* 104

-3 \* 104 <= nums[i] <= 3 \* 104

Each element in the array appears twice except for one element which appears only once.

**Question 3** [**Convert Sorted Array to Binary Search Tree**](https://leetcode.com/problems/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.

**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.

**Q4.Merge Two Sorted Lists**

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

Merge the two lists into 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.

**Q5.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, 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?

**Easy**

**Question 1**. **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 <= numRows <= 30

**Question 2. Remove Element**

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. Then return the number of unique elements in nums.

Consider the number of unique elements of nums to be k, to get accepted, you need to do the following things:

Change the array nums such that the first k elements of nums contain the unique elements in the order they were present in nums initially. The remaining elements of nums are not important as well as the size of nums.

Return k.

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.

**Question 3 Baseball Game :**

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings operations, where operations[i] is the ith operation you must apply to the record and is one of the following:

An integer x.

Record a new score of x.

'+'.

Record a new score that is the sum of the previous two scores.

'D'.

Record a new score that is the double of the previous score.

'C'.

Invalidate the previous score, removing it from the record.

Return the sum of all the scores on the record after applying all the operations.

The test cases are generated such that the answer and all intermediate calculations fit in a 32-bit integer and that all operations are valid.

**Example 1:**

Input: ops = ["5","2","C","D","+"]

Output: 30

Explanation:

"5" - Add 5 to the record, record is now [5].

"2" - Add 2 to the record, record is now [5, 2].

"C" - Invalidate and remove the previous score, record is now [5].

"D" - Add 2 \* 5 = 10 to the record, record is now [5, 10].

"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].

The total sum is 5 + 10 + 15 = 30.

**Example 2:**

Input: ops = ["5","-2","4","C","D","9","+","+"]

Output: 27

Explanation:

"5" - Add 5 to the record, record is now [5].

"-2" - Add -2 to the record, record is now [5, -2].

"4" - Add 4 to the record, record is now [5, -2, 4].

"C" - Invalidate and remove the previous score, record is now [5, -2].

"D" - Add 2 \* -2 = -4 to the record, record is now [5, -2, -4].

"9" - Add 9 to the record, record is now [5, -2, -4, 9].

"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].

"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].

The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

**Example 3:**

Input: ops = ["1","C"]

Output: 0

Explanation:

"1" - Add 1 to the record, record is now [1].

"C" - Invalidate and remove the previous score, record is now [].

Since the record is empty, the total sum is 0.

**Constraints:**

1 <= operations.length <= 1000

operations[i] is "C", "D", "+", or a string representing an integer in the range [-3 \* 104, 3 \* 104].

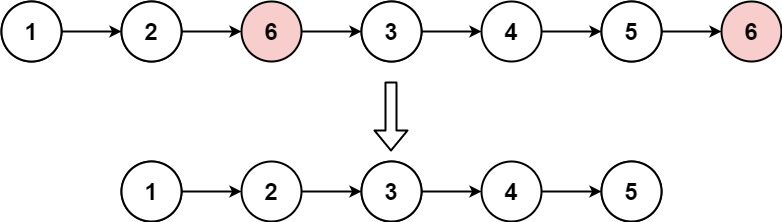
For operation "+", there will always be at least two previous scores on the record.

For operations "C" and "D", there will always be at least one previous score on the record.

**Q4.Remove Linked List Elements**

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

**Example 1:**

****

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

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

**Example 2:**

**Input:** head = [], val = 1

**Output:** []

**Example 3:**

**Input:** head = [7,7,7,7], val = 7

**Output:** []

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

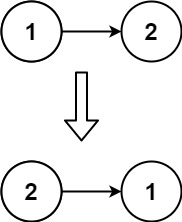
**Example 1:**

****

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

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

**Example 2:**

****

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

**Output:** [2,1]

**Example 3:**

**Input:** head = []

**Output:** []

**Constraints:**

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

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

**Constraints:**

* The number of nodes in the list is in the range [0, 104].
* 1 <= Node.val <= 50
* 0 <= val <= 50

Q5. Reverse Linked List

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

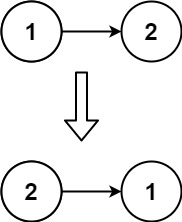
**Example 1:**

****

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

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

**Example 2:**

****

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

**Output:** [2,1]

**Example 3:**

**Input:** head = []

**Output:** []

**Constraints:**

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

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

**Medium:**

Question 1. 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 ith line are (i, 0) and (i, 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 container can contain is 49.

**Example 2:**

Input: height = [1,1]

Output: 1

**Constraints:**

n == height.length

2 <= n <= 105

0 <= height[i] <= 104

**Question 2. Valid Sudoku**

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

Each row must contain the digits 1-9 without repetition.

Each column must contain the digits 1-9 without repetition.

Each of the nine 3 x 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**:

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 sub-box, it is invalid.

**Constraints:**

board.length == 9

board[i].length == 9

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

**Question 3 : Jump Game II**

You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Each element nums[i] represents the maximum length of a forward jump from index i. In other words, if you are at nums[i], you can jump to any nums[i + j] where:

0 <= j <= nums[i] and

i + j < n

Return the minimum number of jumps to reach nums[n - 1]. The test cases are generated such that you can reach nums[n - 1].

**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

It's guaranteed that you can reach nums[n - 1].

Q4.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, just like in Figure B. The serialized output is in level order as connected by the next pointers, with '#' signifying the end of each level.

**Example 2:**

**Input:** root = []

**Output:** []

**Constraints:**

* The number of nodes in the tree is in the range [0, 212 - 1].
* -1000 <= Node.val <= 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.

Q5. Design Circular Queue

Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle, and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Implement the MyCircularQueue class:

* MyCircularQueue(k) Initializes the object with the size of the queue to be k.
* int Front() Gets the front item from the queue. If the queue is empty, return -1.
* int Rear() Gets the last item from the queue. If the queue is empty, return -1.
* boolean enQueue(int value) Inserts an element into the circular queue. Return true if the operation is successful.
* boolean deQueue() Deletes an element from the circular queue. Return true if the operation is successful.
* boolean isEmpty() Checks whether the circular queue is empty or not.
* boolean isFull() Checks whether the circular queue is full or not.

You must solve the problem without using the built-in queue data structure in your programming language.

**Example 1:**

**Input**

["MyCircularQueue", "enQueue", "enQueue", "enQueue", "enQueue", "Rear", "isFull", "deQueue", "enQueue", "Rear"]

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

**Output**

[null, true, true, true, false, 3, true, true, true, 4]

**Explanation**

MyCircularQueue myCircularQueue = new MyCircularQueue(3);

myCircularQueue.enQueue(1); // return True

myCircularQueue.enQueue(2); // return True

myCircularQueue.enQueue(3); // return True

myCircularQueue.enQueue(4); // return False

myCircularQueue.Rear(); // return 3

myCircularQueue.isFull(); // return True

myCircularQueue.deQueue(); // return True

myCircularQueue.enQueue(4); // return True

myCircularQueue.Rear(); // return 4

**Constraints:**

* 1 <= k <= 1000
* 0 <= value <= 1000
* At most 3000 calls will be made to enQueue, deQueue, Front, Rear, isEmpty, and isFull.

**Hard**

**Question 1. Maximum Number of Groups Getting Fresh Donuts**

There is a donuts shop that bakes donuts in batches of batchSize. They have a rule where they must serve all of the donuts of a batch before serving any donuts of the next batch. You are given an integer batchSize and an integer array groups, where groups[i] denotes that there is a group of groups[i] customers that will visit the shop. Each customer will get exactly one donut.

When a group visits the shop, all customers of the group must be served before serving any of the following groups. A group will be happy if they all get fresh donuts. That is, the first customer of the group does not receive a donut that was left over from the previous group.

You can freely rearrange the ordering of the groups. Return the maximum possible number of happy groups after rearranging the groups.

**Example 1:**

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

Output: 4

Explanation: You can arrange the groups as [6,2,4,5,1,3]. Then the 1st, 2nd, 4th, and 6th groups will be happy.

**Example 2:**

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

Output: 4

**Constraints:**

1 <= batchSize <= 9

1 <= groups.length <= 30

1 <= groups[i] <= 109

**Question 2 Cherry Pickup II**

You are given a rows x cols matrix grid representing a field of cherries where grid[i][j] represents the number of cherries that you can collect from the (i, j) cell.

You have two robots that can collect cherries for you:

Robot #1 is located at the top-left corner (0, 0), and

Robot #2 is located at the top-right corner (0, cols - 1).

Return the maximum number of cherries collection using both robots by following the rules below:

From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).

When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.

When both robots stay in the same cell, only one takes the cherries.

Both robots cannot move outside of the grid at any moment.

Both robots should reach the bottom row in grid.

**Example 1:**

Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]

Output: 24

Explanation: Path of robot #1 and #2 are described in color green and blue respectively.

Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12.

Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12.

Total of cherries: 12 + 12 = 24.

**Example 2:**

Input: grid = [[1,0,0,0,0,0,1],[2,0,0,0,0,3,0],[2,0,9,0,0,0,0],[0,3,0,5,4,0,0],[1,0,2,3,0,0,6]]

Output: 28

Explanation: Path of robot #1 and #2 are described in color green and blue respectively.

Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17.

Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11.

Total of cherries: 17 + 11 = 28.

**Constraints:**

rows == grid.length

cols == grid[i].length

2 <= rows, cols <= 70

0 <= grid[i][j] <= 100

**Question 3: Maximum Number of Darts Inside of a Circular Dartboard**

Alice is throwing n darts on a very large wall. You are given an array darts where darts[i] = [xi, yi] is the position of the ith dart that Alice threw on the wall.

Bob knows the positions of the n darts on the wall. He wants to place a dartboard of radius r on the wall so that the maximum number of darts that Alice throws lie on the dartboard.

Given the integer r, return the maximum number of darts that can lie on the dartboard.

**Example 1:**

Input: darts = [[-2,0],[2,0],[0,2],[0,-2]], r = 2

Output: 4

Explanation: Circle dartboard with center in (0,0) and radius = 2 contain all points.

**Example 2:**

Input: darts = [[-3,0],[3,0],[2,6],[5,4],[0,9],[7,8]], r = 5

Output: 5

Explanation: Circle dartboard with center in (0,4) and radius = 5 contain all points except the point (7,8).

**Constraints:**

1 <= darts.length <= 100

darts[i].length == 2

-104 <= xi, yi <= 104

All the darts are unique

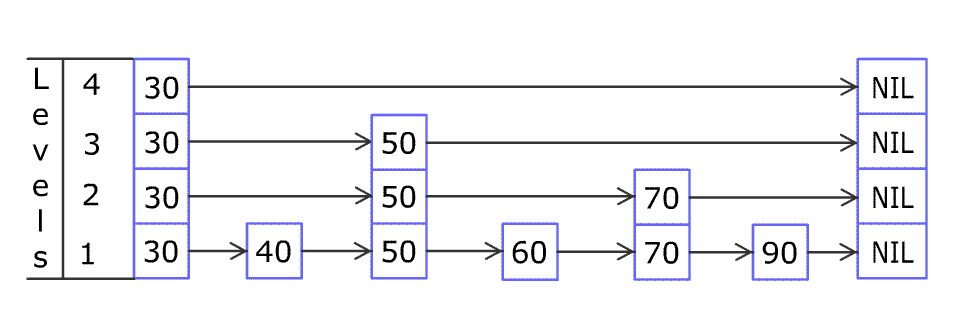
1 <= r <= 5000

**Q4 Design Skiplist**

Design a **Skiplist** without using any built-in libraries.

A **skiplist** is a data structure that takes O(log(n)) time to add, erase and search. Comparing with treap and red-black tree which has the same function and performance, the code length of Skiplist can be comparatively short and the idea behind Skiplists is just simple linked lists.

For example, we have a Skiplist containing [30,40,50,60,70,90] and we want to add 80 and 45 into it. The Skiplist works this way:

  
Artyom Kalinin [CC BY-SA 3.0], via [Wikimedia Commons](https://commons.wikimedia.org/wiki/File:Skip_list_add_element-en.gif)

You can see there are many layers in the Skiplist. Each layer is a sorted linked list. With the help of the top layers, add, erase and search can be faster than O(n). It can be proven that the average time complexity for each operation is O(log(n)) and space complexity is O(n).

See more about Skiplist: <https://en.wikipedia.org/wiki/Skip_list>

Implement the Skiplist class:

* Skiplist() Initializes the object of the skiplist.
* bool search(int target) Returns true if the integer target exists in the Skiplist or false otherwise.
* void add(int num) Inserts the value num into the SkipList.
* bool erase(int num) Removes the value num from the Skiplist and returns true. If num does not exist in the Skiplist, do nothing and return false. If there exist multiple num values, removing any one of them is fine.

Note that duplicates may exist in the Skiplist, your code needs to handle this situation.

**Example 1:**

**Input**

["Skiplist", "add", "add", "add", "search", "add", "search", "erase", "erase", "search"]

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

**Output**

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

**Explanation**

Skiplist skiplist = new Skiplist();

skiplist.add(1);

skiplist.add(2);

skiplist.add(3);

skiplist.search(0); // return False

skiplist.add(4);

skiplist.search(1); // return True

skiplist.erase(0); // return False, 0 is not in skiplist.

skiplist.erase(1); // return True

skiplist.search(1); // return False, 1 has already been erased.

**Constraints:**

* 0 <= num, target <= 2 \* 104
* At most 5 \* 104 calls will be made to search, add, and erase.

Q5. All O`one Data Structure

Design a data structure to store the strings' count with the ability to return the strings with minimum and maximum counts.

Implement the AllOne class:

* AllOne() Initializes the object of the data structure.
* inc(String key) Increments the count of the string key by 1. If key does not exist in the data structure, insert it with count 1.
* dec(String key) Decrements the count of the string key by 1. If the count of key is 0 after the decrement, remove it from the data structure. It is guaranteed that key exists in the data structure before the decrement.
* getMaxKey() Returns one of the keys with the maximal count. If no element exists, return an empty string "".
* getMinKey() Returns one of the keys with the minimum count. If no element exists, return an empty string "".

**Note** that each function must run in O(1) average time complexity.

**Example 1:**

**Input**

["AllOne", "inc", "inc", "getMaxKey", "getMinKey", "inc", "getMaxKey", "getMinKey"]

[[], ["hello"], ["hello"], [], [], ["leet"], [], []]

**Output**

[null, null, null, "hello", "hello", null, "hello", "leet"]

**Explanation**

AllOne allOne = new AllOne();

allOne.inc("hello");

allOne.inc("hello");

allOne.getMaxKey(); // return "hello"

allOne.getMinKey(); // return "hello"

allOne.inc("leet");

allOne.getMaxKey(); // return "hello"

allOne.getMinKey(); // return "leet"

**Constraints:**

* 1 <= key.length <= 10
* key consists of lowercase English letters.
* It is guaranteed that for each call to dec, key is existing in the data structure.
* At most 5 \* 104 calls will be made to inc, dec, getMaxKey, and getMinKey.

**Very Hard**

Question 1. Find Minimum Time to Finish All Jobs

You are given an integer array jobs, where jobs[i] is the amount of time it takes to complete the ith job.There are k workers that you can assign jobs to. Each job should be assigned to exactly one worker. The working time of a worker is the sum of the time it takes to complete all jobs assigned to them. Your goal is to devise an optimal assignment such that the maximum working time of any worker is minimized.

Return the minimum possible maximum working time of any assignment.

**Example 1:**

Input: jobs = [3,2,3], k = 3

Output: 3

Explanation: By assigning each person one job, the maximum time is 3.

**Example 2:**

Input: jobs = [1,2,4,7,8], k = 2

Output: 11

Explanation: Assign the jobs the following way:

Worker 1: 1, 2, 8 (working time = 1 + 2 + 8 = 11)

Worker 2: 4, 7 (working time = 4 + 7 = 11)

The maximum working time is 11.

**Constraints:**

1 <= k <= jobs.length <= 12

1 <= jobs[i] <= 107

**Question 2. Minimum Number of People to Teach**

On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer n, an array languages, and an array friendships where:

There are n languages numbered 1 through n,

languages[i] is the set of languages the i​​​​​​th​​​​ user knows, and

friendships[i] = [u​​​​​​i​​​, v​​​​​​i] denotes a friendship between the users u​​​​​​​​​​​i​​​​​ and vi.

You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of users you need to teach.

Note that friendships are not transitive, meaning if x is a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

**Example 1**:

Input: n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]

Output: 1

Explanation: You can either teach user 1 the second language or user 2 the first language.

**Example 2:**

Input: n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,3]]

Output: 2

Explanation: Teach the third language to users 1 and 3, yielding two users to teach.

**Constraints:**

2 <= n <= 500

languages.length == m

1 <= m <= 500

1 <= languages[i].length <= n

1 <= languages[i][j] <= n

1 <= u​​​​​​i < v​​​​​​i <= languages.length

1 <= friendships.length <= 500

All tuples (u​​​​​i, v​​​​​​i) are unique

languages[i] contains only unique values

**Question 3 Count Ways to Make Array With Product**

You are given a 2D integer array, queries. For each queries[i], where queries[i] = [ni, ki], find the number of different ways you can place positive integers into an array of size ni such that the product of the integers is ki. As the number of ways may be too large, the answer to the ith query is the number of ways modulo 109 + 7.

Return an integer array answer where answer.length == queries.length, and answer[i] is the answer to the ith query.

**Example 1:**

Input: queries = [[2,6],[5,1],[73,660]]

Output: [4,1,50734910]

Explanation: Each query is independent.

[2,6]: There are 4 ways to fill an array of size 2 that multiply to 6: [1,6], [2,3], [3,2], [6,1].

[5,1]: There is 1 way to fill an array of size 5 that multiply to 1: [1,1,1,1,1].

[73,660]: There are 1050734917 ways to fill an array of size 73 that multiply to 660. 1050734917 modulo 109 + 7 = 50734910.

**Example 2:**

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

Output: [1,2,3,10,5]

**Constraints**:

1 <= queries.length <= 104

1 <= ni, ki <= 104

Q4 Maximum Twin Sum of a Linked List

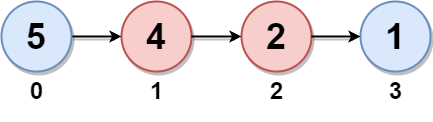
In a linked list of size n, where n is **even**, the ith node (**0-indexed**) of the linked list is known as the **twin** of the (n-1-i)th node, if 0 <= i <= (n / 2) - 1.

* For example, if n = 4, then node 0 is the twin of node 3, and node 1 is the twin of node 2. These are the only nodes with twins for n = 4.

The **twin sum** is defined as the sum of a node and its twin.

Given the head of a linked list with even length, return *the* ***maximum twin sum*** *of the linked list*.

**Example 1:**

****

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

**Output:** 6

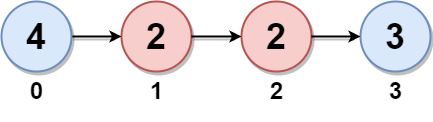
**Explanation:**

Nodes 0 and 1 are the twins of nodes 3 and 2, respectively. All have twin sum = 6.

There are no other nodes with twins in the linked list.

Thus, the maximum twin sum of the linked list is 6.

**Example 2:**

****

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

**Output:** 7

**Explanation:**

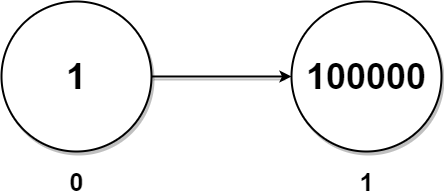
The nodes with twins present in this linked list are:

- Node 0 is the twin of node 3 having a twin sum of 4 + 3 = 7.

- Node 1 is the twin of node 2 having a twin sum of 2 + 2 = 4.

Thus, the maximum twin sum of the linked list is max(7, 4) = 7.

**Example 3:**

****

**Input:** head = [1,100000]

**Output:** 100001

**Explanation:**

There is only one node with a twin in the linked list having twin sum of 1 + 100000 = 100001.

**Constraints:**

* The number of nodes in the list is an **even** integer in the range [2, 105].
* 1 <= Node.val <= 105

Q5. Insert Greatest Common Divisors in Linked List

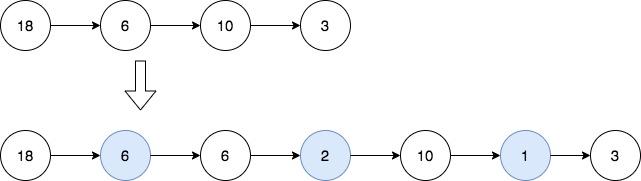
Given the head of a linked list head, in which each node contains an integer value.

Between every pair of adjacent nodes, insert a new node with a value equal to the **greatest common divisor** of them.

Return *the linked list after insertion*.

The **greatest common divisor** of two numbers is the largest positive integer that evenly divides both numbers.

**Example 1:**

****

**Input:** head = [18,6,10,3]

**Output:** [18,6,6,2,10,1,3]

**Explanation:** The 1st diagram denotes the initial linked list and the 2nd diagram denotes the linked list after inserting the new nodes (nodes in blue are the inserted nodes).

- We insert the greatest common divisor of 18 and 6 = 6 between the 1st and the 2nd nodes.

- We insert the greatest common divisor of 6 and 10 = 2 between the 2nd and the 3rd nodes.

- We insert the greatest common divisor of 10 and 3 = 1 between the 3rd and the 4th nodes.

There are no more adjacent nodes, so we return the linked list.

**Example 2:**

****

**Input:** head = [7]

**Output:** [7]

**Explanation:** The 1st diagram denotes the initial linked list and the 2nd diagram denotes the linked list after inserting the new nodes.

There are no pairs of adjacent nodes, so we return the initial linked list.

**Constraints:**

* The number of nodes in the list is in the range [1, 5000].
* 1 <= Node.val <= 1000