

My LeetCode Submissions - @Prvn21

[Download PDF](#)[Follow @TheShubham99 on GitHub](#)[Star on GitHub](#)[View Source Code](#)

[1473 Find the Longest Substring Containing Vowels in Even Counts \(link\)](#)

Description

Given the string s , return the size of the longest substring containing each vowel an even number of times. That is, 'a', 'e', 'i', 'o', and 'u' must appear an even number of times.

Example 1:

Input: s = "eleetminicoworoep"

Output: 13

Explanation: The longest substring is "leetminicowor" which contains two each of the vowels: e, i and o and zero of the vowels: a a

Example 2:

Input: s = "leetcodeisgreat"

Output: 5

Explanation: The longest substring is "leetc" which contains two e's.

Example 3:

Input: s = "bcbcbc"

Output: 6

Explanation: In this case, the given string "bcbcbc" is the longest because all vowels: a, e, i, o and u appear zero times.

Constraints:

- $1 \leq s.length \leq 5 \times 10^5$
- s contains only lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {number}
 */
var findTheLongestSubstring = function(s) {
    const vowelIndex = { 'a': 0, 'e': 1, 'i': 2, 'o': 3, 'u': 4 }; // Map vowels to bit positions
    let bitmask = 0; // Initial bitmask (all vowels have even counts)
    let maxLen = 0; // The length of the longest valid substring
    const stateMap = new Map();
    stateMap.set(0, -1); // Store the initial state with index -1 (before the string starts)

    for (let i = 0; i < s.length; i++) {
        // If the character is a vowel, toggle the respective bit
        if (vowelIndex.hasOwnProperty(s[i])) {
            bitmask ^= (1 << vowelIndex[s[i]]));
        }

        // Check if we've seen this bitmask before
        if (stateMap.has(bitmask)) {
            maxLen = Math.max(maxLen, i - stateMap.get(bitmask));
        } else {
            // Store the first occurrence of this bitmask
            stateMap.set(bitmask, i);
        }
    }

    return maxLen;
};
```

2503 Longest Subarray With Maximum Bitwise AND ([link](#))

Description

You are given an integer array `nums` of size `n`.

Consider a **non-empty** subarray from `nums` that has the **maximum** possible **bitwise AND**.

- In other words, let k be the maximum value of the bitwise AND of **any** subarray of `nums`. Then, only subarrays with a bitwise AND equal to k should be considered.

Return *the length of the longest such subarray*.

The bitwise AND of an array is the bitwise AND of all the numbers in it.

A **subarray** is a contiguous sequence of elements within an array.

Example 1:

```
Input: nums = [1,2,3,3,2,2]
Output: 2
Explanation:
The maximum possible bitwise AND of a subarray is 3.
The longest subarray with that value is [3,3], so we return 2.
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: 1
Explanation:
The maximum possible bitwise AND of a subarray is 4.
The longest subarray with that value is [4], so we return 1.
```

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $1 \leq \text{nums}[i] \leq 10^6$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var longestSubarray = function(nums) {  
    let maxVal = Math.max(...nums); // Find the maximum value in the array  
    let maxLength = 0; // To track the length of the longest subarray  
    let currentLength = 0; // To count the current sequence of 'maxVal'  
  
    for (let i = 0; i < nums.length; i++) {  
        if (nums[i] === maxVal) {  
            currentLength++; // Increment the current length if the number is equal to maxVal  
            maxLength = Math.max(maxLength, currentLength); // Update maxLength if needed  
        } else {  
            currentLength = 0; // Reset the current length if the number is not maxVal  
        }  
    }  
  
    return maxLength;  
};
```

2507 Number of Common Factors ([link](#))

Description

Given two positive integers a and b , return *the number of common factors of a and b* .

An integer x is a **common factor** of a and b if x divides both a and b .

Example 1:

Input: $a = 12$, $b = 6$

Output: 4

Explanation: The common factors of 12 and 6 are 1, 2, 3, 6.

Example 2:

Input: $a = 25$, $b = 30$

Output: 2

Explanation: The common factors of 25 and 30 are 1, 5.

Constraints:

- $1 \leq a, b \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} a  
 * @param {number} b  
 * @return {number}  
 */  
var commonFactors = function(a, b) {  
    let count = 0;  
    let min = Math.min(a, b); // Find the smaller number of the two  
  
    for (let i = 1; i <= min; i++) {  
        if (a % i === 0 && b % i === 0) {  
            count++; // Increment count if 'i' divides both 'a' and 'b'  
        }  
    }  
  
    return count;  
};
```

[204 Count Primes \(link\)](#)

Description

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

Example 1:

Input: $n = 10$

Output: 4

Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:

Input: $n = 0$

Output: 0

Example 3:

Input: $n = 1$

Output: 0

Constraints:

- $0 \leq n \leq 5 * 10^6$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number} n
 * @return {number}
 */
var countPrimes = function(n) {
    if (n <= 2) return 0;

    // Step 1: Create a boolean array `isPrime` and initialize all entries as `true`
    const isPrime = new Array(n).fill(true);

    // Step 2: Mark 0 and 1 as not prime
    isPrime[0] = isPrime[1] = false;

    // Step 3: Implement the Sieve of Eratosthenes
    for (let i = 2; i * i < n; i++) {
        if (isPrime[i]) {
            for (let j = i * i; j < n; j += i) {
                isPrime[j] = false;
            }
        }
    }

    // Step 4: Count the number of prime numbers
    let primeCount = 0;
    for (let i = 2; i < n; i++) {
        if (isPrime[i]) {
            primeCount++;
        }
    }

    return primeCount;
};
```

[1786 Count the Number of Consistent Strings \(link\)](#)

Description

You are given a string `allowed` consisting of **distinct** characters and an array of strings `words`. A string is **consistent** if all characters in the string appear in the string `allowed`.

Return *the number of consistent strings in the array words*.

Example 1:

```
Input: allowed = "ab", words = ["ad", "bd", "aaab", "baa", "badab"]
Output: 2
Explanation: Strings "aaab" and "baa" are consistent since they only contain characters 'a' and 'b'.
```

Example 2:

```
Input: allowed = "abc", words = ["a", "b", "c", "ab", "ac", "bc", "abc"]
Output: 7
Explanation: All strings are consistent.
```

Example 3:

```
Input: allowed = "cad", words = ["cc", "acd", "b", "ba", "bac", "bad", "ac", "d"]
Output: 4
Explanation: Strings "cc", "acd", "ac", and "d" are consistent.
```

Constraints:

- $1 \leq \text{words.length} \leq 10^4$
- $1 \leq \text{allowed.length} \leq 26$
- $1 \leq \text{words[i].length} \leq 10$

- The characters in `allowed` are **distinct**.
- `words[i]` and `allowed` contain only lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} allowed  
 * @param {string[]} words  
 * @return {number}  
 */  
var countConsistentStrings = function(allowed, words) {  
    // Create a set of allowed characters for quick lookup  
    let allowedSet = new Set(allowed);  
  
    // Initialize the count of consistent strings  
    let count = 0;  
  
    // Iterate through each word in the words array  
    for (let word of words) {  
        let isConsistent = true;  
  
        // Check if each character in the word is in the allowed set  
        for (let char of word) {  
            if (!allowedSet.has(char)) {  
                isConsistent = false;  
                break; // No need to check further if the word is not consistent  
            }  
        }  
  
        // If the word is consistent, increment the count  
        if (isConsistent) {  
            count++;  
        }  
    }  
  
    return count;  
};
```

139 Word Break (link)

Description

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

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

Example 1:

```
Input: s = "leetcode", wordDict = ["leet", "code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
```

Example 2:

```
Input: s = "applepenapple", wordDict = ["apple", "pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
Note that you are allowed to reuse a dictionary word.
```

Example 3:

```
Input: s = "catsandog", wordDict = ["cats", "dog", "sand", "and", "cat"]
Output: false
```

Constraints:

- $1 \leq s.length \leq 300$
- $1 \leq \text{wordDict.length} \leq 1000$
- $1 \leq \text{wordDict}[i].length \leq 20$

- `s` and `wordDict[i]` consist of only lowercase English letters.
- All the strings of `wordDict` are **unique**.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {string[]} wordDict  
 * @return {boolean}  
 */  
var wordBreak = function(s, wordDict) {  
    // Create a set of words for quick lookup  
    let wordSet = new Set(wordDict);  
  
    // Create a dp array with length s.length + 1, initialized to false  
    let dp = new Array(s.length + 1).fill(false);  
  
    // The base case, an empty string can always be segmented  
    dp[0] = true;  
  
    // Iterate over the string length  
    for (let i = 1; i <= s.length; i++) {  
        // For each i, check if there's a j where dp[j] is true and the substring s[j...i] is in the wordDict  
        for (let j = 0; j < i; j++) {  
            if (dp[j] && wordSet.has(s.substring(j, i))) {  
                dp[i] = true;  
                break; // No need to check further if we've found a valid segmentation  
            }  
        }  
    }  
  
    // Return whether the whole string can be segmented  
    return dp[s.length];  
};
```

[79 Word Search \(link\)](#)

Description

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

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

Example 1:

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

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

Example 2:

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

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

Output: true

Example 3:

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

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

Output: false

Constraints:

- $m == \text{board.length}$
- $n = \text{board}[i].length$
- $1 \leq m, n \leq 6$
- $1 \leq \text{word.length} \leq 15$
- board and word consists of only lowercase and uppercase English letters.

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {character[][]} board
 * @param {string} word
 * @return {boolean}
 */
var exist = function(board, word) {
    const rows = board.length;
    const cols = board[0].length;

    function dfs(row, col, index) {
        // Base case: if the entire word is found
        if (index === word.length) return true;

        // Check for boundary conditions and matching characters
        if (row < 0 || row >= rows || col < 0 || col >= cols || board[row][col] !== word[index]) {
            return false;
        }

        // Mark the current cell as visited by setting it to a placeholder character
        const temp = board[row][col];
        board[row][col] = '#'; // Mark as visited

        // Explore all possible directions (up, down, left, right)
        const found = dfs(row + 1, col, index + 1) ||
                      dfs(row - 1, col, index + 1) ||
                      dfs(row, col + 1, index + 1) ||
                      dfs(row, col - 1, index + 1);

        // Restore the cell's original value (backtracking)
        board[row][col] = temp;

        return found;
    }

    // Check each cell in the grid as a potential starting point
    for (let i = 0; i < rows; i++) {
        for (let j = 0; j < cols; j++) {
            if (dfs(i, j, 0)) {

```

```
        return true; // If word is found, return true
    }
}

return false; // If no path finds the word, return false
};
```

1693 Sum of All Odd Length Subarrays ([link](#))

Description

Given an array of positive integers `arr`, return *the sum of all possible odd-length subarrays of arr*.

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

Example 1:

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

Output: 58

Explanation: The odd-length subarrays of arr and their sums are:

[1] = 1

[4] = 4

[2] = 2

[5] = 5

[3] = 3

[1,4,2] = 7

[4,2,5] = 11

[2,5,3] = 10

[1,4,2,5,3] = 15

If we add all these together we get $1 + 4 + 2 + 5 + 3 + 7 + 11 + 10 + 15 = 58$

Example 2:

Input: arr = [1,2]

Output: 3

Explanation: There are only 2 subarrays of odd length, [1] and [2]. Their sum is 3.

Example 3:

Input: arr = [10,11,12]

Output: 66

Constraints:

- $1 \leq \text{arr.length} \leq 100$
- $1 \leq \text{arr}[i] \leq 1000$

Follow up:

Could you solve this problem in $O(n)$ time complexity?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} arr  
 * @return {number}  
 */  
var sumOddLengthSubarrays = function(arr) {  
    let n = arr.length;  
    let totalSum = 0;  
  
    for (let i = 0; i < n; i++) {  
        let contribution = Math.ceil((i + 1) * (n - i) / 2);  
        totalSum += contribution * arr[i];  
    }  
  
    return totalSum;  
};
```

[633 Sum of Square Numbers \(link\)](#)

Description

Given a non-negative integer c , decide whether there're two integers a and b such that $a^2 + b^2 = c$.

Example 1:

```
Input: c = 5
Output: true
Explanation: 1 * 1 + 2 * 2 = 5
```

Example 2:

```
Input: c = 3
Output: false
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} c  
 * @return {boolean}  
 */  
var judgeSquareSum = function(c) {  
    let a = 0;  
    let b = Math.floor(Math.sqrt(c));  
  
    while (a <= b) {  
        let sum = a * a + b * b;  
  
        if (sum === c) {  
            return true;  
        } else if (sum < c) {  
            a++;  
        } else {  
            b--;  
        }  
    }  
  
    return false;  
};
```

2844 Sum of Squares of Special Elements ([link](#))

Description

You are given a **1-indexed** integer array `nums` of length `n`.

An element `nums[i]` of `nums` is called **special** if `i` divides `n`, i.e. `n % i == 0`.

Return *the sum of the squares* of all **special** elements of `nums`.

Example 1:

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

```
Output: 21
```

Explanation: There are exactly 3 special elements in `nums`: `nums[1]` since 1 divides 4, `nums[2]` since 2 divides 4, and `nums[4]` since 4 divides 4. Hence, the sum of the squares of all special elements of `nums` is `nums[1] * nums[1] + nums[2] * nums[2] + nums[4] * nums[4] = 1 * 1 + 2 * 2 + 4 * 4 = 21`.

Example 2:

```
Input: nums = [2,7,1,19,18,3]
```

```
Output: 63
```

Explanation: There are exactly 4 special elements in `nums`: `nums[1]` since 1 divides 6, `nums[2]` since 2 divides 6, `nums[3]` since 3 divides 6, and `nums[6]` since 6 divides 6. Hence, the sum of the squares of all special elements of `nums` is `nums[1] * nums[1] + nums[2] * nums[2] + nums[3] * nums[3] + nums[6] * nums[6] = 1 * 1 + 2 * 2 + 3 * 3 + 6 * 6 = 63`.

Constraints:

- `1 <= nums.length == n <= 50`
- `1 <= nums[i] <= 50`

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var sumOfSquares = function(nums) {  
    const n = nums.length;  
    let sum = 0;  
  
    for (let i = 1; i <= n; i++) {  
        if (n % i === 0) {  
            sum += nums[i - 1] ** 2;  
        }  
    }  
  
    return sum;  
};
```

[1844 Maximum Number of Balls in a Box \(link\)](#)

Description

You are working in a ball factory where you have n balls numbered from `lowLimit` up to `highLimit` **inclusive** (i.e., $n == \text{highLimit} - \text{lowLimit} + 1$), and an infinite number of boxes numbered from 1 to infinity.

Your job at this factory is to put each ball in the box with a number equal to the sum of digits of the ball's number. For example, the ball number 321 will be put in the box number $3 + 2 + 1 = 6$ and the ball number 10 will be put in the box number $1 + 0 = 1$.

Given two integers `lowLimit` and `highLimit`, return *the number of balls in the box with the most balls*.

Example 1:

```
Input: lowLimit = 1, highLimit = 10
Output: 2
Explanation:
Box Number: 1 2 3 4 5 6 7 8 9 10 11 ...
Ball Count: 2 1 1 1 1 1 1 1 0 0 ...
Box 1 has the most number of balls with 2 balls.
```

Example 2:

```
Input: lowLimit = 5, highLimit = 15
Output: 2
Explanation:
Box Number: 1 2 3 4 5 6 7 8 9 10 11 ...
Ball Count: 1 1 1 1 2 2 1 1 1 0 0 ...
Boxes 5 and 6 have the most number of balls with 2 balls in each.
```

Example 3:

```
Input: lowLimit = 19, highLimit = 28
Output: 2
Explanation:
Box Number: 1 2 3 4 5 6 7 8 9 10 11 12 ...
```

```
Ball Count: 0 1 1 1 1 1 1 1 1 2 0 0 ...
Box 10 has the most number of balls with 2 balls.
```

Constraints:

- $1 \leq \text{lowLimit} \leq \text{highLimit} \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} lowLimit  
 * @param {number} highLimit  
 * @return {number}  
 */  
var countBalls = function(lowLimit, highLimit) {  
    const boxMap = {};  
  
    // Function to calculate the sum of digits  
    function sumOfDigits(num) {  
        let sum = 0;  
        while (num > 0) {  
            sum += num % 10;  
            num = Math.floor(num / 10);  
        }  
        return sum;  
    }  
  
    // Count the balls in each box  
    for (let i = lowLimit; i <= highLimit; i++) {  
        const boxNumber = sumOfDigits(i);  
        if (boxMap[boxNumber] === undefined) {  
            boxMap[boxNumber] = 0;  
        }  
        boxMap[boxNumber]++;  
    }  
  
    // Find the maximum number of balls in any box  
    let maxBalls = 0;  
    for (const box in boxMap) {  
        if (boxMap[box] > maxBalls) {  
            maxBalls = boxMap[box];  
        }  
    }  
}
```

```
    return maxBalls;  
};
```

645 Set Mismatch ([link](#))

Description

You have a set of integers s , which originally contains all the numbers from 1 to n . Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in **repetition of one number** and **loss of another number**.

You are given an integer array `nums` representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return *them in the form of an array*.

Example 1:

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

Example 2:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var findErrorNums = function(nums) {  
    const n = nums.length;  
    const expectedSum = n * (n + 1) / 2;  
    const expectedSumOfSquares = n * (n + 1) * (2 * n + 1) / 6;  
  
    let actualSum = 0;  
    let actualSumOfSquares = 0;  
  
    for (let num of nums) {  
        actualSum += num;  
        actualSumOfSquares += num * num;  
    }  
  
    const sumDiff = expectedSum - actualSum; // missing - duplicate  
    const sumSquareDiff = expectedSumOfSquares - actualSumOfSquares; // missing^2 - duplicate^2  
  
    const missing = (sumDiff + sumSquareDiff / sumDiff) / 2;  
    const duplicate = missing - sumDiff;  
  
    return [duplicate, missing];  
};
```

145 Binary Tree Postorder Traversal ([link](#))

Description

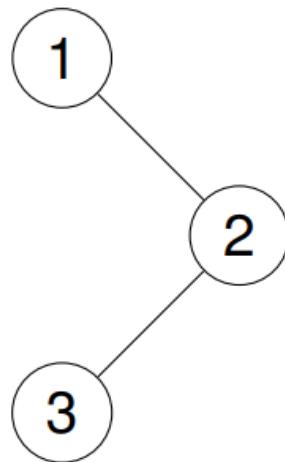
Given the `root` of a binary tree, return *the postorder traversal of its nodes' values*.

Example 1:

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

Output: [3,2,1]

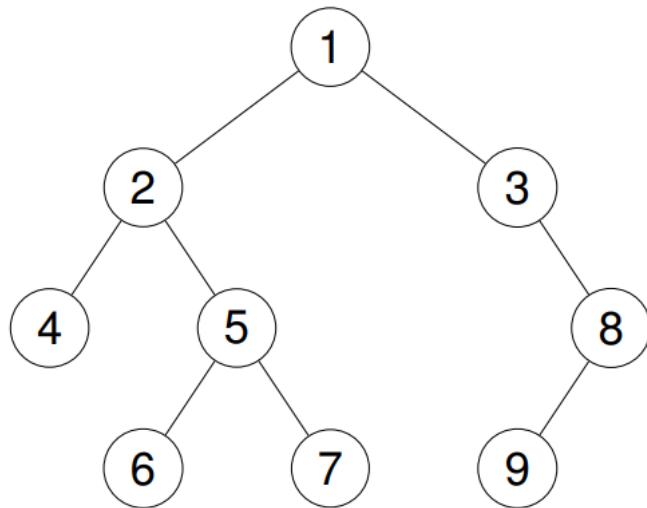
Explanation:



Example 2:

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

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

Explanation:**Example 3:**

Input: root = []

Output: []

Example 4:

Input: root = [1]

Output: [1]

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[]}
 */

// Definition for a binary tree node.
function TreeNode(val, left = null, right = null) {
    this.val = val;
    this.left = left;
    this.right = right;
}

var postorderTraversal = function(root) {
    if (root === null) return [];

    const stack = [];
    const result = [];
    let current = root;

    while (current !== null || stack.length > 0) {
        if (current !== null) {
            stack.push(current);
            result.push(current.val); // Push root's value to result
            current = current.right; // Move to right subtree
        } else {
            current = stack.pop(); // Backtrack to last unvisited node
            current = current.left; // Move to left subtree
        }
    }
}
```

```
    return result.reverse(); // Reverse the result to get left, right, root order
};
```

[592 Fraction Addition and Subtraction \(link\)](#)

Description

Given a string `expression` representing an expression of fraction addition and subtraction, return the calculation result in string format.

The final result should be an [irreducible fraction](#). If your final result is an integer, change it to the format of a fraction that has a denominator 1. So in this case, 2 should be converted to 2/1.

Example 1:

```
Input: expression = "-1/2+1/2"
Output: "0/1"
```

Example 2:

```
Input: expression = "-1/2+1/2+1/3"
Output: "1/3"
```

Example 3:

```
Input: expression = "1/3-1/2"
Output: "-1/6"
```

Constraints:

- The input string only contains '0' to '9', '/', '+' and '-'. So does the output.
- Each fraction (input and output) has the format $\pm\text{numerator}/\text{denominator}$. If the first input fraction or the output is positive, then '+' will be omitted.
- The input only contains valid **irreducible fractions**, where the **numerator** and **denominator** of each fraction will always be in the range [1, 10]. If the denominator is 1, it means this fraction is actually an integer in a fraction format defined above.
- The number of given fractions will be in the range [1, 10].

- The numerator and denominator of the **final result** are guaranteed to be valid and in the range of **32-bit** int.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} expression  
 * @return {string}  
 */  
var fractionAddition = function(expression) {  
    const fractions = expression.match(/[+-]?\d+\/\d+/g); // Match all fractions in the expression  
  
    let numerator = 0;  
    let denominator = 1;  
  
    for (const fraction of fractions) {  
        const [num, denom] = fraction.split('/').map(Number);  
        numerator = numerator * denom + num * denominator;  
        denominator *= denom;  
  
        // Simplify the fraction after each addition/subtraction  
        const gcdValue = gcd(Math.abs(numerator), Math.abs(denominator));  
        numerator /= gcdValue;  
        denominator /= gcdValue;  
    }  
  
    // Ensure the sign is correct  
    if (denominator < 0) {  
        numerator = -numerator;  
        denominator = -denominator;  
    }  
  
    return `${numerator}/${denominator}`;  
}  
  
// Helper function to calculate the greatest common divisor (GCD)  
function gcd(a, b) {  
    return b === 0 ? a : gcd(b, a % b);  
}
```

1054 Complement of Base 10 Integer ([link](#))

Description

The **complement** of an integer is the integer you get when you flip all the 0's to 1's and all the 1's to 0's in its binary representation.

- For example, The integer 5 is "101" in binary and its **complement** is "010" which is the integer 2.

Given an integer n , return *its complement*.

Example 1:

Input: $n = 5$

Output: 2

Explanation: 5 is "101" in binary, with complement "010" in binary, which is 2 in base-10.

Example 2:

Input: $n = 7$

Output: 0

Explanation: 7 is "111" in binary, with complement "000" in binary, which is 0 in base-10.

Example 3:

Input: $n = 10$

Output: 5

Explanation: 10 is "1010" in binary, with complement "0101" in binary, which is 5 in base-10.

Constraints:

- $0 \leq n < 10^9$

Note: This question is the same as 476: <https://leetcode.com/problems/number-complement/>

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var bitwiseComplement = function(n) {  
    // Step 1: Convert the integer to binary string  
    let binary = n.toString(2);  
  
    // Step 2: Flip the bits  
    let flippedBinary = binary.split('').map(bit => bit === '0' ? '1' : '0').join('');  
  
    // Step 3: Convert the flipped binary string to a decimal integer  
    let complement = parseInt(flippedBinary, 2);  
  
    return complement;  
};
```

[476 Number Complement \(link\)](#)

Description

The **complement** of an integer is the integer you get when you flip all the 0's to 1's and all the 1's to 0's in its binary representation.

- For example, The integer 5 is "101" in binary and its **complement** is "010" which is the integer 2.

Given an integer `num`, return *its complement*.

Example 1:

```
Input: num = 5  
Output: 2
```

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

Example 2:

```
Input: num = 1  
Output: 0
```

Explanation: The binary representation of 1 is 1 (no leading zero bits), and its complement is 0. So you need to output 0.

Constraints:

- $1 \leq \text{num} < 2^{31}$

Note: This question is the same as 1009: <https://leetcode.com/problems/complement-of-base-10-integer/>

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {number}  
 */  
var findComplement = function(num) {  
    // Find the number of bits in the binary representation of num  
    const numBits = Math.floor(Math.log2(num)) + 1;  
  
    // Generate a mask with all bits set to 1  
    const mask = (1 << numBits) - 1;  
  
    // Compute the complement  
    return mask ^ num;  
};
```

[664 Strange Printer \(link\)](#)

Description

There is a strange printer with the following two special properties:

- The printer can only print a sequence of **the same character** each time.
- At each turn, the printer can print new characters starting from and ending at any place and will cover the original existing characters.

Given a string s , return *the minimum number of turns the printer needed to print it*.

Example 1:

```
Input: s = "aaabbb"
```

```
Output: 2
```

```
Explanation: Print "aaa" first and then print "bbb".
```

Example 2:

```
Input: s = "aba"
```

```
Output: 2
```

```
Explanation: Print "aaa" first and then print "b" from the second place of the string, which will cover the existing character 'a'.
```

Constraints:

- $1 \leq s.length \leq 100$
- s consists of lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {number}
 */
var strangePrinter = function(s) {
    const n = s.length;
    const dp = Array.from({ length: n }, () => new Array(n).fill(0));

    for (let len = 1; len <= n; len++) {
        for (let i = 0; i + len <= n; i++) {
            let j = i + len - 1;
            dp[i][j] = len; // Start with the worst case where every character needs a separate print
            for (let k = i; k < j; k++) {
                let totalTurns = dp[i][k] + dp[k + 1][j];
                if (s[k] === s[j]) {
                    totalTurns--;
                }
                dp[i][j] = Math.min(dp[i][j], totalTurns);
            }
        }
    }

    return dp[0][n - 1];
};
```

[1240 Stone Game II \(link\)](#)

Description

Alice and Bob continue their games with piles of stones. There are a number of piles **arranged in a row**, and each pile has a positive integer number of stones `piles[i]`. The objective of the game is to end with the most stones.

Alice and Bob take turns, with Alice starting first.

On each player's turn, that player can take **all the stones** in the **first** x remaining piles, where $1 \leq x \leq 2M$. Then, we set $M = \max(M, x)$. Initially, $M = 1$.

The game continues until all the stones have been taken.

Assuming Alice and Bob play optimally, return the maximum number of stones Alice can get.

Example 1:

Input: piles = [2,7,9,4,4]

Output: 10

Explanation:

- If Alice takes one pile at the beginning, Bob takes two piles, then Alice takes 2 piles again. Alice can get $2 + 4 + 4 = 10$ stones in total.
- If Alice takes two piles at the beginning, then Bob can take all three piles left. In this case, Alice get $2 + 7 = 9$ stones in total.

So we return 10 since it's larger.

Example 2:

Input: piles = [1,2,3,4,5,100]

Output: 104

Constraints:

- $1 \leq \text{piles.length} \leq 100$
- $1 \leq \text{piles}[i] \leq 10^4$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} piles  
 * @return {number}  
 */  
var stoneGameII = function(piles) {  
    const n = piles.length;  
    const suffixSum = new Array(n + 1).fill(0);  
  
    // Calculate the suffix sums  
    for (let i = n - 1; i >= 0; i--) {  
        suffixSum[i] = suffixSum[i + 1] + piles[i];  
    }  
  
    // Initialize dp array  
    const dp = Array.from({ length: n }, () => new Array(n + 1).fill(0));  
  
    // Fill dp array  
    for (let i = n - 1; i >= 0; i--) {  
        for (let M = 1; M <= n; M++) {  
            for (let X = 1; X <= 2 * M; X++) {  
                if (i + X >= n) {  
                    dp[i][M] = suffixSum[i]; // Take all remaining stones  
                } else {  
                    dp[i][M] = Math.max(dp[i][M], suffixSum[i] - dp[i + X][Math.max(M, X)]);  
                }  
            }  
        }  
    }  
  
    // The answer is the max stones Alice can get starting from the first pile with M = 1  
    return dp[0][1];  
}
```

[650 2 Keys Keyboard \(link\)](#)

Description

There is only one character 'A' on the screen of a notepad. You can perform one of two operations on this notepad for each step:

- Copy All: You can copy all the characters present on the screen (a partial copy is not allowed).
- Paste: You can paste the characters which are copied last time.

Given an integer n , return *the minimum number of operations to get the character 'A' exactly n times on the screen*.

Example 1:

```
Input: n = 3
Output: 3
Explanation: Initially, we have one character 'A'.
In step 1, we use Copy All operation.
In step 2, we use Paste operation to get 'AA'.
In step 3, we use Paste operation to get 'AAA'.
```

Example 2:

```
Input: n = 1
Output: 0
```

Constraints:

- $1 \leq n \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var minSteps = function(n) {  
    let result = 0;  
    for (let i = 2; i <= n; i++) {  
        while (n % i === 0) {  
            result += i;  
            n /= i;  
        }  
    }  
    return result;  
};
```

3525 Maximum Energy Boost From Two Drinks ([link](#))

Description

You are given two integer arrays `energyDrinkA` and `energyDrinkB` of the same length n by a futuristic sports scientist. These arrays represent the energy boosts per hour provided by two different energy drinks, A and B, respectively.

You want to *maximize* your total energy boost by drinking one energy drink *per hour*. However, if you want to switch from consuming one energy drink to the other, you need to wait for *one hour* to cleanse your system (meaning you won't get any energy boost in that hour).

Return the **maximum** total energy boost you can gain in the next n hours.

Note that you can start consuming *either* of the two energy drinks.

Example 1:

Input: `energyDrinkA` = [1,3,1], `energyDrinkB` = [3,1,1]

Output: 5

Explanation:

To gain an energy boost of 5, drink only the energy drink A (or only B).

Example 2:

Input: `energyDrinkA` = [4,1,1], `energyDrinkB` = [1,1,3]

Output: 7

Explanation:

To gain an energy boost of 7:

- Drink the energy drink A for the first hour.
- Switch to the energy drink B and we lose the energy boost of the second hour.

- Gain the energy boost of the drink B in the third hour.

Constraints:

- $n == \text{energyDrinkA.length} == \text{energyDrinkB.length}$
- $3 \leq n \leq 10^5$
- $1 \leq \text{energyDrinkA}[i], \text{energyDrinkB}[i] \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} energyDrinkA  
 * @param {number[]} energyDrinkB  
 * @return {number}  
 */  
var maxEnergyBoost = function(energyDrinkA, energyDrinkB) {  
    const n = energyDrinkA.length;  
    if (n === 0) return 0;  
  
    // Initialize dp arrays  
    let dpA = new Array(n).fill(0);  
    let dpB = new Array(n).fill(0);  
  
    // Base case  
    dpA[0] = energyDrinkA[0];  
    dpB[0] = energyDrinkB[0];  
  
    for (let i = 1; i < n; i++) {  
        dpA[i] = Math.max(dpA[i - 1] + energyDrinkA[i], (i > 1 ? dpB[i - 2] : 0) + energyDrinkA[i]);  
        dpB[i] = Math.max(dpB[i - 1] + energyDrinkB[i], (i > 1 ? dpA[i - 2] : 0) + energyDrinkB[i]);  
    }  
  
    return Math.max(dpA[n - 1], dpB[n - 1]);  
};
```

[3543 Count Substrings That Satisfy K-Constraint I \(link\)](#)

Description

You are given a **binary** string s and an integer k .

A **binary string** satisfies the **k -constraint** if either of the following conditions holds:

- The number of 0's in the string is at most k .
- The number of 1's in the string is at most k .

Return an integer denoting the number of substrings of s that satisfy the **k -constraint**.

Example 1:

Input: $s = "10101"$, $k = 1$

Output: 12

Explanation:

Every substring of s except the substrings "1010", "10101", and "0101" satisfies the k -constraint.

Example 2:

Input: $s = "1010101"$, $k = 2$

Output: 25

Explanation:

Every substring of s except the substrings with a length greater than 5 satisfies the k -constraint.

Example 3:

Input: $s = "11111"$, $k = 1$

Output: 15**Explanation:**

All substrings of s satisfy the k -constraint.

Constraints:

- $1 \leq s.length \leq 50$
- $1 \leq k \leq s.length$
- $s[i]$ is either '0' or '1'.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @param {number} k
 * @return {number}
 */
var countKConstraintSubstrings = function(s, k) {
    const n = s.length;
    let count = 0;

    for (let i = 0; i < n; i++) {
        let zeros = 0, ones = 0;

        for (let j = i; j < n; j++) {
            if (s[j] === '0') {
                zeros++;
            } else {
                ones++;
            }

            if (zeros <= k || ones <= k) {
                count++;
            } else {
                break;
            }
        }
    }

    return count;
};
```

[3523 Find the Power of K-Size Subarrays II \(link\)](#)

Description

You are given an array of integers `nums` of length `n` and a *positive* integer `k`.

The **power** of an array is defined as:

- Its **maximum** element if *all* of its elements are **consecutive** and **sorted in ascending order**.
- -1 otherwise.

You need to find the **power** of all subarrays of `nums` of size `k`.

Return an integer array `results` of size `n - k + 1`, where `results[i]` is the *power* of `nums[i..(i + k - 1)]`.

Example 1:

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

Output: `[3,4,-1,-1,-1]`

Explanation:

There are 5 subarrays of `nums` of size 3:

- `[1, 2, 3]` with the maximum element 3.
- `[2, 3, 4]` with the maximum element 4.
- `[3, 4, 3]` whose elements are **not** consecutive.
- `[4, 3, 2]` whose elements are **not** sorted.
- `[3, 2, 5]` whose elements are **not** consecutive.

Example 2:

Input: `nums = [2,2,2,2,2]`, `k = 4`

Output: `[-1,-1]`

Example 3:

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

Output: [-1,3,-1,3,-1]

Constraints:

- $1 \leq n == \text{nums.length} \leq 10^5$
- $1 \leq \text{nums}[i] \leq 10^6$
- $1 \leq k \leq n$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @param {number} k
 * @return {number[]}
 */
var resultsArray = function(nums, k) {
    const n = nums.length;
    const result = [];

    let isValid = true;

    // Initial check for the first window
    for (let i = 1; i < k; i++) {
        if (nums[i] !== nums[i - 1] + 1) {
            isValid = false;
            break;
        }
    }

    if (isValid) {
        result.push(nums[k - 1]);
    } else {
        result.push(-1);
    }

    // Sliding the window across the array
    for (let i = k; i < n; i++) {
        // Remove the first element of the previous window and add the next element
        if (isValid && nums[i] === nums[i - 1] + 1) {
            result.push(nums[i]);
        } else {
            // Recheck the entire window
            isValid = true;
            for (let j = i - k + 1; j <= i; j++) {
                if (j > i - k + 1 && nums[j] !== nums[j - 1] + 1) {
                    isValid = false;
                    break;
                }
            }
        }
    }
}
```

```
        }
        result.push(isValid ? nums[i] : -1);
    }

    return result;
};
```

[3522 Find the Power of K-Size Subarrays I \(link\)](#)

Description

You are given an array of integers `nums` of length `n` and a *positive* integer `k`.

The **power** of an array is defined as:

- Its **maximum** element if *all* of its elements are **consecutive** and **sorted in ascending order**.
- -1 otherwise.

You need to find the **power** of all subarrays of `nums` of size `k`.

Return an integer array `results` of size `n - k + 1`, where `results[i]` is the *power* of `nums[i..(i + k - 1)]`.

Example 1:

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

Output: `[3,4,-1,-1,-1]`

Explanation:

There are 5 subarrays of `nums` of size 3:

- `[1, 2, 3]` with the maximum element 3.
- `[2, 3, 4]` with the maximum element 4.
- `[3, 4, 3]` whose elements are **not** consecutive.
- `[4, 3, 2]` whose elements are **not** sorted.
- `[3, 2, 5]` whose elements are **not** consecutive.

Example 2:

Input: `nums = [2,2,2,2,2]`, `k = 4`

Output: `[-1,-1]`

Example 3:

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

Output: [-1,3,-1,3,-1]

Constraints:

- $1 \leq n == \text{nums.length} \leq 500$
- $1 \leq \text{nums}[i] \leq 10^5$
- $1 \leq k \leq n$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {number[]}  
 */  
var resultsArray = function(nums, k) {  
    const results = [];  
    const n = nums.length;  
  
    for (let i = 0; i <= n - k; i++) {  
        const subarray = nums.slice(i, i + k);  
        let isSorted = true;  
  
        // Check if subarray is sorted and consecutive  
        for (let j = 1; j < k; j++) {  
            if (subarray[j] !== subarray[j - 1] + 1) {  
                isSorted = false;  
                break;  
            }  
        }  
  
        // If sorted and consecutive, add the max element to results, else add -1  
        if (isSorted) {  
            results.push(Math.max(...subarray));  
        } else {  
            results.push(-1);  
        }  
    }  
  
    return results;  
};
```

[624 Maximum Distance in Arrays \(link\)](#)

Description

You are given m arrays, where each array is sorted in **ascending order**.

You can pick up two integers from two different arrays (each array picks one) and calculate the distance. We define the distance between two integers a and b to be their absolute difference $|a - b|$.

Return *the maximum distance*.

Example 1:

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

Output: 4

Explanation: One way to reach the maximum distance 4 is to pick 1 in the first or third array and pick 5 in the second array.

Example 2:

Input: arrays = [[1],[1]]

Output: 0

Constraints:

- $m == \text{arrays.length}$
- $2 \leq m \leq 10^5$
- $1 \leq \text{arrays}[i].length \leq 500$
- $-10^4 \leq \text{arrays}[i][j] \leq 10^4$
- $\text{arrays}[i]$ is sorted in **ascending order**.
- There will be at most 10^5 integers in all the arrays.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[][]} arrays  
 * @return {number}  
 */  
var maxDistance = function(arrays) {  
    let minVal = arrays[0][0];  
    let maxVal = arrays[0][arrays[0].length - 1];  
    let maxDistance = 0;  
  
    for (let i = 1; i < arrays.length; i++) {  
        let currentMin = arrays[i][0];  
        let currentMax = arrays[i][arrays[i].length - 1];  
  
        // Calculate the distance with the current min and max values  
        maxDistance = Math.max(maxDistance, Math.abs(currentMax - minVal), Math.abs(maxVal - currentMin));  
  
        // Update the min and max values  
        minVal = Math.min(minVal, currentMin);  
        maxVal = Math.max(maxVal, currentMax);  
    }  
  
    return maxDistance;  
};
```

1341 Split a String in Balanced Strings (link)

Description

Balanced strings are those that have an equal quantity of 'L' and 'R' characters.

Given a **balanced** string s , split it into some number of substrings such that:

- Each substring is balanced.

Return *the maximum number of balanced strings you can obtain*.

Example 1:

Input: $s = "RLRRLLRLRL"$

Output: 4

Explanation: s can be split into "RL", "RRLL", "RL", "RL", each substring contains same number of 'L' and 'R'.

Example 2:

Input: $s = "RLRRRLLRLL"$

Output: 2

Explanation: s can be split into "RL", "RRRLLRLL", each substring contains same number of 'L' and 'R'.

Note that s cannot be split into "RL", "RR", "RL", "LR", "LL", because the 2nd and 5th substrings are not balanced.

Example 3:

Input: $s = "LLLLRRRR"$

Output: 1

Explanation: s can be split into "LLLLRRRR".

Constraints:

- $2 \leq s.length \leq 1000$
- $s[i]$ is either 'L' or 'R'.
- s is a **balanced** string.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var balancedStringSplit = function(s) {  
    let balance = 0;  
    let count = 0;  
  
    for (let i = 0; i < s.length; i++) {  
        if (s[i] === 'L') {  
            balance++;  
        } else if (s[i] === 'R') {  
            balance--;  
        }  
  
        // When balance is zero, we have a balanced substring  
        if (balance === 0) {  
            count++;  
        }  
    }  
  
    return count;  
};
```

[2881 Split Strings by Separator \(link\)](#)

Description

Given an array of strings `words` and a character `separator`, **split** each string in `words` by `separator`.

Return *an array of strings containing the new strings formed after the splits, excluding empty strings.*

Notes

- `separator` is used to determine where the split should occur, but it is not included as part of the resulting strings.
- A split may result in more than two strings.
- The resulting strings must maintain the same order as they were initially given.

Example 1:

Input: words = ["one.two.three", "four.five", "six"], separator = "."

Output: ["one", "two", "three", "four", "five", "six"]

Explanation: In this example we split as follows:

"one.two.three" splits into "one", "two", "three"

"four.five" splits into "four", "five"

"six" splits into "six"

Hence, the resulting array is ["one", "two", "three", "four", "five", "six"].

Example 2:

Input: words = ["\$easy\$", "\$problem\$"], separator = "\$"

Output: ["easy", "problem"]

Explanation: In this example we split as follows:

"\$easy\$" splits into "easy" (excluding empty strings)

"\$problem\$" splits into "problem" (excluding empty strings)

Hence, the resulting array is ["easy", "problem"].

Example 3:

Input: words = ["|||"], separator = "|"

Output: []

Explanation: In this example the resulting split of "|||" will contain only empty strings, so we return an empty array [].

Constraints:

- $1 \leq \text{words.length} \leq 100$
- $1 \leq \text{words[i].length} \leq 20$
- characters in words[i] are either lowercase English letters or characters from the string ". ,|#\$@"
- separator is a character from the string ". ,|#\$@" (excluding the quotes)

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string[]} words  
 * @param {character} separator  
 * @return {string[]}  
 */  
var splitWordsBySeparator = function(words, separator) {  
    let result = [];  
  
    words.forEach(word => {  
        // Split the word by the separator and filter out empty strings  
        let splitWords = word.split(separator).filter(part => part !== "");  
        // Add the non-empty parts to the result array  
        result.push(...splitWords);  
    });  
  
    return result;  
};
```

719 Find K-th Smallest Pair Distance ([link](#))

Description

The **distance of a pair** of integers a and b is defined as the absolute difference between a and b .

Given an integer array nums and an integer k , return *the k^{th} smallest distance among all the pairs* $\text{nums}[i]$ and $\text{nums}[j]$ where $0 \leq i < j < \text{nums.length}$.

Example 1:

```
Input: nums = [1,3,1], k = 1
Output: 0
Explanation: Here are all the pairs:
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
Then the 1st smallest distance pair is (1,1), and its distance is 0.
```

Example 2:

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

Example 3:

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

Constraints:

- $n == \text{nums.length}$

- $2 \leq n \leq 10^4$
- $0 \leq \text{nums}[i] \leq 10^6$
- $1 \leq k \leq n * (n - 1) / 2$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @param {number} k
 * @return {number}
 */
var smallestDistancePair = function(nums, k) {
    // Step 1: Sort the array
    nums.sort((a, b) => a - b);

    // Step 2: Binary search for the smallest distance
    let left = 0;
    let right = nums[nums.length - 1] - nums[0];

    while (left < right) {
        let mid = Math.floor((left + right) / 2);
        // Step 3: Count pairs with distance <= mid
        let count = 0;
        let j = 0;

        for (let i = 0; i < nums.length; i++) {
            while (j < nums.length && nums[j] - nums[i] <= mid) {
                j++;
            }
            count += j - i - 1; // j - i - 1 gives the number of pairs with i < j and nums[j] - nums[i] <= mid
        }

        // Step 4: Adjust the binary search range
        if (count >= k) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    return left;
}
```

```
};
```

[179 Largest Number \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 10^9$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {string}  
 */  
var largestNumber = function(nums) {  
    // Convert all numbers to strings for easy comparison  
    let numsStr = nums.map(String);  
  
    // Sort the numbers based on the concatenated value  
    numsStr.sort((a, b) => (b + a) - (a + b));  
  
    // Join all the strings to form the largest number  
    let largestNum = numsStr.join('');  
  
    // Handle the case where the largest number is 0  
    return largestNum[0] === '0' ? '0' : largestNum;  
};
```

[3188 Find Champion I \(link\)](#)

Description

There are n teams numbered from 0 to $n - 1$ in a tournament.

Given a **0-indexed** 2D boolean matrix grid of size $n * n$. For all i, j that $0 \leq i, j \leq n - 1$ and $i \neq j$ team i is **stronger** than team j if $\text{grid}[i][j] == 1$, otherwise, team j is **stronger** than team i .

Team a will be the **champion** of the tournament if there is no team b that is stronger than team a .

Return *the team that will be the champion of the tournament*.

Example 1:

Input: $\text{grid} = [[0,1],[0,0]]$

Output: 0

Explanation: There are two teams in this tournament.

$\text{grid}[0][1] == 1$ means that team 0 is stronger than team 1 . So team 0 will be the champion.

Example 2:

Input: $\text{grid} = [[0,0,1],[1,0,1],[0,0,0]]$

Output: 1

Explanation: There are three teams in this tournament.

$\text{grid}[1][0] == 1$ means that team 1 is stronger than team 0 .

$\text{grid}[1][2] == 1$ means that team 1 is stronger than team 2 .

So team 1 will be the champion.

Constraints:

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

- $2 \leq n \leq 100$
- $\text{grid}[i][j]$ is either 0 or 1.
- For all i $\text{grid}[i][i]$ is 0.
- For all i, j that $i \neq j$, $\text{grid}[i][j] \neq \text{grid}[j][i]$.
- The input is generated such that if team a is stronger than team b and team b is stronger than team c, then team a is stronger than team c.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[][]} grid  
 * @return {number}  
 */  
var findChampion = function(grid) {  
    const n = grid.length;  
  
    for (let i = 0; i < n; i++) {  
        let isChampion = true;  
        for (let j = 0; j < n; j++) {  
            if (i !== j && grid[j][i] === 1) {  
                isChampion = false;  
                break;  
            }  
        }  
        if (isChampion) {  
            return i;  
        }  
    }  
  
    return -1; // This should not happen given the problem constraints  
};
```

[371 Sum of Two Integers \(link\)](#)

Description

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

Example 1:

```
Input: a = 1, b = 2
Output: 3
```

Example 2:

```
Input: a = 2, b = 3
Output: 5
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} a  
 * @param {number} b  
 * @return {number}  
 */  
var getSum = function(a, b) {  
    while (b !== 0) {  
        // Carry now contains common set bits of a and b  
        let carry = a & b;  
  
        // Sum of bits of a and b where at least one of the bits is not set  
        a = a ^ b;  
  
        // Carry is shifted by one so that adding it to a gives the required sum  
        b = carry << 1;  
    }  
  
    return a;  
};
```

[3429 Special Array I \(link\)](#)

Description

An array is considered **special** if every pair of its adjacent elements contains two numbers with different parity.

You are given an array of integers `nums`. Return `true` if `nums` is a **special** array, otherwise, return `false`.

Example 1:

Input: `nums = [1]`

Output: `true`

Explanation:

There is only one element. So the answer is `true`.

Example 2:

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

Output: `true`

Explanation:

There is only two pairs: $(2,1)$ and $(1,4)$, and both of them contain numbers with different parity. So the answer is `true`.

Example 3:

Input: `nums = [4,3,1,6]`

Output: `false`

Explanation:

`nums[1]` and `nums[2]` are both odd. So the answer is `false`.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {boolean}  
 */  
var isArraySpecial = function(nums) {  
    // Loop through the array up to the second-to-last element  
    for (let i = 0; i < nums.length - 1; i++) {  
        // Check if the current element and the next element have the same parity  
        if (nums[i] % 2 === nums[i + 1] % 2) {  
            // If they have the same parity, return false  
            return false;  
        }  
    }  
    // If all adjacent elements have different parity, return true  
    return true;  
};
```

279 Perfect Squares ([link](#))

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq n \leq 10^4$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var numSquares = function(n) {  
    // Create an array to store the minimum number of perfect squares that sum to each number from 0 to n  
    const dp = Array(n + 1).fill(Infinity);  
    dp[0] = 0; // Base case: 0 can be formed with 0 perfect squares  
  
    // Iterate over all numbers from 1 to n  
    for (let i = 1; i <= n; i++) {  
        // Try every square number less than or equal to i  
        for (let j = 1; j * j <= i; j++) {  
            dp[i] = Math.min(dp[i], dp[i - j * j] + 1);  
        }  
    }  
  
    // The answer is the minimum number of perfect squares that sum to n  
    return dp[n];  
};
```

18 4Sum (link)

Description

Given an array `nums` of n integers, return *an array of all the unique quadruplets* $[nums[a], \text{ } nums[b], \text{ } nums[c], \text{ } nums[d]]$ such that:

- $0 \leq a, b, c, d < n$
- $a, b, c,$ and d are **distinct**.
- $nums[a] + nums[b] + nums[c] + nums[d] == \text{target}$

You may return the answer in **any order**.

Example 1:

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

Example 2:

```
Input: nums = [2,2,2,2,2], target = 8
Output: [[2,2,2,2]]
```

Constraints:

- $1 \leq \text{nums.length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number[][][]}  
 */  
var fourSum = function(nums, target) {  
    const result = [];  
    if (nums.length < 4) return result;  
  
    // Step 1: Sort the array  
    nums.sort((a, b) => a - b);  
  
    // Step 2: Iterate through the array with four pointers  
    for (let i = 0; i < nums.length - 3; i++) {  
        // Avoid duplicates for the first number  
        if (i > 0 && nums[i] === nums[i - 1]) continue;  
  
        for (let j = i + 1; j < nums.length - 2; j++) {  
            // Avoid duplicates for the second number  
            if (j > i + 1 && nums[j] === nums[j - 1]) continue;  
  
            let left = j + 1;  
            let right = nums.length - 1;  
  
            while (left < right) {  
                const sum = nums[i] + nums[j] + nums[left] + nums[right];  
  
                if (sum === target) {  
                    result.push([nums[i], nums[j], nums[left], nums[right]]);  
  
                    // Move the left pointer and avoid duplicates  
                    while (left < right && nums[left] === nums[left + 1]) left++;  
                    left++;  
  
                    // Move the right pointer and avoid duplicates  
                    while (left < right && nums[right] === nums[right - 1]) right--;  
                    right--;  
                } else if (sum < target) {  
                    left++;  
                } else {  
                    right--;  
                }  
            }  
        }  
    }  
};
```

```
        left++;
    } else {
        right--;
    }
}
return result;
};
```

347 Top K Frequent Elements ([link](#))

Description

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

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `k` is in the range $[1, \text{the number of unique elements in the array}]$.
- It is **guaranteed** that the answer is **unique**.

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @param {number} k
 * @return {number[]}
 */
var topKFrequent = function(nums, k) {
    const frequencyMap = new Map();

    // Step 1: Count the frequency of each element
    for (let num of nums) {
        frequencyMap.set(num, (frequencyMap.get(num) || 0) + 1);
    }

    // Step 2: Create an array of buckets where index is the frequency
    const bucket = Array(nums.length + 1).fill().map(() => []);

    for (let [num, freq] of frequencyMap) {
        bucket[freq].push(num);
    }

    // Step 3: Gather the top k frequent elements
    const result = [];
    for (let i = bucket.length - 1; i >= 0 && result.length < k; i--) {
        if (bucket[i].length > 0) {
            result.push(...bucket[i]);
        }
    }

    return result.slice(0, k);
};
```

[273 Integer to English Words \(link\)](#)

Description

Convert a non-negative integer `num` to its English words representation.

Example 1:

```
Input: num = 123
Output: "One Hundred Twenty Three"
```

Example 2:

```
Input: num = 12345
Output: "Twelve Thousand Three Hundred Forty Five"
```

Example 3:

```
Input: num = 1234567
Output: "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {string}  
 */  
  
const ones = ["", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine"];  
const teens = ["Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"];  
const tens = ["", "", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"];  
const thousands = ["", "Thousand", "Million", "Billion"];  
  
function numberToWords(num) {  
    if (num === 0) return "Zero";  
  
    let words = '';  
  
    function helper(num) {  
        if (num === 0) return "";  
        else if (num < 10) return ones[num] + " ";  
        else if (num < 20) return teens[num - 10] + " ";  
        else if (num < 100) return tens[Math.floor(num / 10)] + " " + helper(num % 10);  
        else return ones[Math.floor(num / 100)] + " Hundred " + helper(num % 100);  
    }  
  
    for (let i = 0; num > 0; i++) {  
        if (num % 1000 !== 0) {  
            words = helper(num % 1000) + thousands[i] + " " + words;  
        }  
        num = Math.floor(num / 1000);  
    }  
  
    return words.trim();  
}
```

10 Regular Expression Matching (link)

Description

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

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

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

Example 1:

Input: $s = "aa"$, $p = "a"$

Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: $s = "aa"$, $p = "a*$

Output: true

Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:

Input: $s = "ab"$, $p = ".*"$

Output: true

Explanation: ".*" means "zero or more (*) of any character (.)".

Constraints:

- $1 \leq s.length \leq 20$

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @param {string} p
 * @return {boolean}
 */
var isMatch = function(s, p) {
    // Initialize a 2D array with false values
    const dp = Array(s.length + 1).fill(null).map(() => Array(p.length + 1).fill(false));

    // Base case: empty string and empty pattern match
    dp[0][0] = true;

    // Handle patterns like a*, a*b*, etc. at the start
    for (let j = 1; j <= p.length; j++) {
        if (p[j - 1] === '*') {
            dp[0][j] = dp[0][j - 2];
        }
    }

    for (let i = 1; i <= s.length; i++) {
        for (let j = 1; j <= p.length; j++) {
            if (p[j - 1] === s[i - 1] || p[j - 1] === '.') {
                // Current characters match, carry over the previous result
                dp[i][j] = dp[i - 1][j - 1];
            } else if (p[j - 1] === '*') {
                // Consider zero occurrence of the preceding element
                dp[i][j] = dp[i][j - 2];

                // Consider one or more occurrences of the preceding element
                if (p[j - 2] === s[i - 1] || p[j - 2] === '.') {
                    dp[i][j] = dp[i][j] || dp[i - 1][j];
                }
            }
        }
    }
}
```

```
    return dp[s.length()][p.length];  
};
```

[219 Contains Duplicate II \(link\)](#)

Description

Given an integer array `nums` and an integer `k`, return `true` if there are two **distinct indices** `i` and `j` in the array such that `nums[i] == nums[j]` and `abs(i - j) <= k`.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $0 \leq k \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {boolean}  
 */  
var containsNearbyDuplicate = function(nums, k) {  
    let indexMap = new Map();  
  
    for (let i = 0; i < nums.length; i++) {  
        if (indexMap.has(nums[i])) {  
            // Check if the previous index and current index are within k distance  
            if (i - indexMap.get(nums[i]) <= k) {  
                return true;  
            }  
        }  
        // Update the index of the current number  
        indexMap.set(nums[i], i);  
    }  
  
    return false;  
};
```

[167 Two Sum II - Input Array Is Sorted \(link\)](#)

Description

Given a **1-indexed** array of integers `numbers` that is already **sorted in non-decreasing order**, find two numbers such that they add up to a specific target number. Let these two numbers be `numbers[index1]` and `numbers[index2]` where $1 \leq \text{index}_1 < \text{index}_2 \leq \text{numbers.length}$.

Return *the indices of the two numbers, `index1` and `index2`, added by one as an integer array [index₁, index₂] of length 2*.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

```
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].
```

Example 3:

```
Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].
```

Constraints:

- $2 \leq \text{numbers.length} \leq 3 * 10^4$
- $-1000 \leq \text{numbers}[i] \leq 1000$
- **numbers is sorted in non-decreasing order.**
- $-1000 \leq \text{target} \leq 1000$
- The tests are generated such that there is **exactly one solution**.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} numbers  
 * @param {number} target  
 * @return {number[]}  
 */  
var twoSum = function(numbers, target) {  
    let left = 0;  
    let right = numbers.length - 1;  
  
    while (left < right) {  
        const sum = numbers[left] + numbers[right];  
        if (sum === target) {  
            return [left + 1, right + 1];  
        } else if (sum < target) {  
            left++;  
        } else {  
            right--;  
        }  
    }  
  
    return []; // Should never reach here since there's always one solution  
};
```

1756 Minimum Deletions to Make String Balanced ([link](#))

Description

You are given a string s consisting only of characters 'a' and 'b'.

You can delete any number of characters in s to make s **balanced**. s is **balanced** if there is no pair of indices (i, j) such that $i < j$ and $s[i] = 'b'$ and $s[j] = 'a'$.

Return *the minimum number of deletions needed to make s balanced*.

Example 1:

Input: $s = "aababbab"$

Output: 2

Explanation: You can either:

Delete the characters at 0-indexed positions 2 and 6 ("aab**abbab**" -> "aaabbb"), or

Delete the characters at 0-indexed positions 3 and 6 ("aab**abbab**" -> "aabbba").

Example 2:

Input: $s = "bbaaaaabb"$

Output: 2

Explanation: The only solution is to delete the first two characters.

Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$ is 'a' or 'b'.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var minimumDeletions = function(s) {  
    let aCount = 0;  
    let bCount = 0;  
  
    // Count the number of 'a's in the entire string  
    for (const char of s) {  
        if (char === 'a') aCount++;  
    }  
  
    let minDeletions = aCount; // All 'a's are deleted if we choose to delete all 'a's  
  
    for (const char of s) {  
        if (char === 'a') {  
            aCount--; // Remove current 'a' from consideration  
        } else {  
            bCount++; // Count the number of 'b's that should be kept  
        }  
  
        // Minimum deletions to make balanced: deleting all remaining 'a's + deleting all seen 'b's  
        minDeletions = Math.min(minDeletions, aCount + bCount);  
    }  
  
    return minDeletions;  
};
```

[1511 Count Number of Teams \(link\)](#)

Description

There are n soldiers standing in a line. Each soldier is assigned a **unique** rating value.

You have to form a team of 3 soldiers amongst them under the following rules:

- Choose 3 soldiers with index (i, j, k) with rating $(\text{rating}[i], \text{rating}[j], \text{rating}[k])$.
- A team is valid if: $(\text{rating}[i] < \text{rating}[j] < \text{rating}[k])$ or $(\text{rating}[i] > \text{rating}[j] > \text{rating}[k])$ where $(0 \leq i < j < k < n)$.

Return the number of teams you can form given the conditions. (soldiers can be part of multiple teams).

Example 1:

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

Output: 3

Explanation: We can form three teams given the conditions. (2,3,4), (5,4,1), (5,3,1).

Example 2:

Input: rating = [2,1,3]

Output: 0

Explanation: We can't form any team given the conditions.

Example 3:

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

Output: 4

Constraints:

- `n == rating.length`
- `3 <= n <= 1000`
- `1 <= rating[i] <= 105`
- All the integers in `rating` are **unique**.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} rating  
 * @return {number}  
 */  
var numTeams = function(rating) {  
    const n = rating.length;  
    let count = 0;  
  
    for (let j = 0; j < n; j++) {  
        let lessLeft = 0, greaterLeft = 0, lessRight = 0, greaterRight = 0;  
  
        for (let i = 0; i < j; i++) {  
            if (rating[i] < rating[j]) {  
                lessLeft++;  
            } else if (rating[i] > rating[j]) {  
                greaterLeft++;  
            }  
        }  
  
        for (let k = j + 1; k < n; k++) {  
            if (rating[k] < rating[j]) {  
                lessRight++;  
            } else if (rating[k] > rating[j]) {  
                greaterRight++;  
            }  
        }  
  
        count += lessLeft * greaterRight + greaterLeft * lessRight;  
    }  
  
    return count;  
};
```

[162 Find Peak Element \(\[link\]\(#\)\)](#)

Description

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

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

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

Example 1:

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

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

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

Output: 5

Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$
- $\text{nums}[i] \neq \text{nums}[i + 1]$ for all valid i .

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var findPeakElement = function(nums) {  
    let left = 0;  
    let right = nums.length - 1;  
  
    while (left < right) {  
        const mid = Math.floor((left + right) / 2);  
  
        if (nums[mid] > nums[mid + 1]) {  
            right = mid; // Peak is on the left side (including mid)  
        } else {  
            left = mid + 1; // Peak is on the right side (excluding mid)  
        }  
    }  
  
    // left and right converge to the peak element  
    return left;  
};
```

1 Two Sum ([link](#))

Description

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

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

You can return the answer in any order.

Example 1:

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

Output: `[0,1]`

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

Example 2:

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

Output: `[1,2]`

Example 3:

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

Output: `[0,1]`

Constraints:

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- **Only one valid answer exists.**

Follow-up: Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number[]}  
 */  
var twoSum = function(nums, target) {  
    // Create a hash map to store the complement and its index  
    const complementMap = new Map();  
  
    // Loop through each element in the array  
    for (let index = 0; index < nums.length; index++) {  
        const num = nums[index];  
        // Calculate the complement  
        const complement = target - num;  
  
        // Check if the complement is already in the hash map  
        if (complementMap.has(complement)) {  
            // Return the indices of the complement and the current number  
            return [complementMap.get(complement), index];  
        }  
  
        // If not, add the current number and its index to the hash map  
        complementMap.set(num, index);  
    }  
  
    // If no solution is found, return an empty array (based on problem constraints, this shouldn't happen)  
    return [];  
};
```

[67 Add Binary_\(link\)](#)

Description

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

Example 1:

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

Example 2:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} a
 * @param {string} b
 * @return {string}
 */
var addBinary = function(a, b) {
    let result = '' // Initialize the result string
    let carry = 0; // Initialize the carry to 0
    let i = a.length - 1; // Start from the end of string a
    let j = b.length - 1; // Start from the end of string b

    // Iterate while there are characters left in either string or there is a carry
    while (i >= 0 || j >= 0 || carry) {
        let sum = carry; // Start with the carry from the previous iteration

        // If there are still characters left in string a, add its value to sum
        if (i >= 0) {
            sum += a[i] - '0'; // Convert character to integer
            i--; // Move to the next character in string a
        }

        // If there are still characters left in string b, add its value to sum
        if (j >= 0) {
            sum += b[j] - '0'; // Convert character to integer
            j--; // Move to the next character in string b
        }

        // Calculate the current digit and carry
        result = (sum % 2) + result; // Append the current digit to the result
        carry = Math.floor(sum / 2); // Calculate the new carry
    }

    return result; // Return the resulting binary string
};
```

31 Next Permutation (link)

Description

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

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3]`, `[1,3,2]`, `[2, 1, 3]`, `[2, 3, 1]`, `[3,1,2]`, `[3,2,1]`.

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

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

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

The replacement must be **in place** and use only constant extra memory.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instead.
 */
var nextPermutation = function(nums) {
    let i = nums.length - 2;

    // Step 1: Find the first decreasing element
    while (i >= 0 && nums[i] >= nums[i + 1]) {
        i--;
    }

    if (i >= 0) {
        let j = nums.length - 1;

        // Step 2: Find the element to swap with
        while (j >= 0 && nums[j] <= nums[i]) {
            j--;
        }

        // Step 3: Swap the elements
        [nums[i], nums[j]] = [nums[j], nums[i]];
    }

    // Step 4: Reverse the suffix
    reverse(nums, i + 1);

    return nums;
}

function reverse(nums, start) {
    let end = nums.length - 1;
    while (start < end) {
        [nums[start], nums[end]] = [nums[end], nums[start]];
        start++;
        end--;
    }
}
```

```
}
```

[46 Permutations \(link\)](#)

Description

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- $1 \leq \text{nums.length} \leq 6$
- $-10 \leq \text{nums}[i] \leq 10$
- All the integers of `nums` are **unique**.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[][]}  
 */  
var permute = function(nums) {  
    let result = [];  
  
    function generate(arr, n) {  
        if (n === 1) {  
            result.push([...arr]);  
            return;  
        }  
  
        for (let i = 0; i < n; i++) {  
            generate(arr, n - 1);  
            if (n % 2 === 0) {  
                [arr[i], arr[n - 1]] = [arr[n - 1], arr[i]]; // Swap elements  
            } else {  
                [arr[0], arr[n - 1]] = [arr[n - 1], arr[0]]; // Swap elements  
            }  
        }  
    }  
  
    generate(nums, nums.length);  
    return result;  
};
```

[451 Sort Characters By Frequency \(link\)](#)

Description

Given a string s , sort it in **decreasing order** based on the **frequency** of the characters. The **frequency** of a character is the number of times it appears in the string.

Return *the sorted string*. If there are multiple answers, return *any of them*.

Example 1:

```
Input: s = "tree"
Output: "eert"
Explanation: 'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
```

Example 2:

```
Input: s = "cccaaa"
Output: "aaaccc"
Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.
Note that "cacaca" is incorrect, as the same characters must be together.
```

Example 3:

```
Input: s = "Aabb"
Output: "bbAa"
Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

Constraints:

- $1 \leq s.length \leq 5 * 10^5$
- s consists of uppercase and lowercase English letters and digits.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
var frequencySort = function(s) {  
    // Step 1: Create a frequency map  
    const frequencyMap = {};  
    for (let char of s) {  
        frequencyMap[char] = (frequencyMap[char] || 0) + 1;  
    }  
  
    // Step 2: Convert the frequency map to an array of [char, frequency] pairs  
    const charArray = Object.entries(frequencyMap);  
  
    // Step 3: Sort the array by frequency in descending order  
    charArray.sort((a, b) => b[1] - a[1]);  
  
    // Step 4: Build the resulting string  
    let result = '';  
    for (let [char, frequency] of charArray) {  
        result += char.repeat(frequency);  
    }  
  
    return result;  
};
```

[1741 Sort Array by Increasing Frequency](#) ([link](#))

Description

Given an array of integers `nums`, sort the array in **increasing** order based on the frequency of the values. If multiple values have the same frequency, sort them in **decreasing** order.

Return the *sorted array*.

Example 1:

```
Input: nums = [1,1,2,2,2,3]
Output: [3,1,1,2,2,2]
Explanation: '3' has a frequency of 1, '1' has a frequency of 2, and '2' has a frequency of 3.
```

Example 2:

```
Input: nums = [2,3,1,3,2]
Output: [1,3,3,2,2]
Explanation: '2' and '3' both have a frequency of 2, so they are sorted in decreasing order.
```

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var frequencySort = function(nums) {  
    // Step 1: Create a frequency map  
    const frequencyMap = {};  
    for (let num of nums) {  
        frequencyMap[num] = (frequencyMap[num] || 0) + 1;  
    }  
  
    // Step 2: Sort the array with a custom comparator  
    return nums.sort((a, b) => {  
        if (frequencyMap[a] === frequencyMap[b]) {  
            return b - a; // If frequencies are the same, sort in decreasing order  
        }  
        return frequencyMap[a] - frequencyMap[b]; // Otherwise, sort by increasing frequency  
    });  
};
```

[2502 Sort the People \(link\)](#)

Description

You are given an array of strings `names`, and an array `heights` that consists of **distinct** positive integers. Both arrays are of length `n`.

For each index `i`, `names[i]` and `heights[i]` denote the name and height of the i^{th} person.

Return `names` sorted in **descending** order by the people's heights.

Example 1:

```
Input: names = ["Mary", "John", "Emma"], heights = [180,165,170]
```

```
Output: ["Mary", "Emma", "John"]
```

```
Explanation: Mary is the tallest, followed by Emma and John.
```

Example 2:

```
Input: names = ["Alice", "Bob", "Bob"], heights = [155,185,150]
```

```
Output: ["Bob", "Alice", "Bob"]
```

```
Explanation: The first Bob is the tallest, followed by Alice and the second Bob.
```

Constraints:

- `n == names.length == heights.length`
- $1 \leq n \leq 10^3$
- $1 \leq \text{names}[i].\text{length} \leq 20$
- $1 \leq \text{heights}[i] \leq 10^5$
- `names[i]` consists of lower and upper case English letters.
- All the values of `heights` are distinct.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string[]} names  
 * @param {number[]} heights  
 * @return {string[]}  
 */  
var sortPeople = function(names, heights) {  
    // Combine names and heights into an array of objects  
    const people = names.map((name, index) => ({  
        name: name,  
        height: heights[index]  
    }));  
  
    // Sort the array of objects by height in descending order  
    people.sort((a, b) => b.height - a.height);  
  
    // Extract and return the sorted names  
    return people.map(person => person.name);  
};
```

[778 Reorganize String \(link\)](#)

Description

Given a string s , rearrange the characters of s so that any two adjacent characters are not the same.

Return *any possible rearrangement of s or return "" if not possible.*

Example 1:

```
Input: s = "aab"
Output: "aba"
```

Example 2:

```
Input: s = "aaab"
Output: ""
```

Constraints:

- $1 \leq s.length \leq 500$
- s consists of lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {string}
 */
function reorganizeString(s) {
    const charCount = {};

    // Count the frequency of each character
    for (const char of s) {
        if (charCount[char]) {
            charCount[char]++;
        } else {
            charCount[char] = 1;
        }
    }

    // Create an array of [char, frequency] and sort by frequency in descending order
    const sortedChars = Object.keys(charCount).map(char => [char, charCount[char]]);
    sortedChars.sort((a, b) => b[1] - a[1]);

    // If the most frequent character's count is more than half the length of the string plus one, return ''
    if (sortedChars[0][1] > Math.ceil(s.length / 2)) {
        return '';
    }

    const result = new Array(s.length);
    let index = 0;

    // Place the most frequent characters first in even indices
    for (const [char, count] of sortedChars) {
        for (let i = 0; i < count; i++) {
            if (index >= s.length) {
                index = 1; // Start filling from odd indices
            }
            result[index] = char;
            index += 2;
        }
    }
}
```

```
    return result.join('');
}

// Example usage:
console.log(reorganizeString("aab")); // Output: "aba" or "baa"
console.log(reorganizeString("aaab")); // Output: ""
```

[53 Maximum Subarray \(link\)](#)

Description

Given an integer array `nums`, find the subarray with the largest sum, and return *its sum*.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.
```

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var maxSubArray = function(nums) {  
    let maxSoFar = nums[0];  
    let maxEndingHere = nums[0];  
  
    for (let i = 1; i < nums.length; i++) {  
        maxEndingHere = Math.max(nums[i], maxEndingHere + nums[i]);  
        maxSoFar = Math.max(maxSoFar, maxEndingHere);  
    }  
  
    return maxSoFar;  
};
```

15 3Sum (link)

Description

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

Notice that the solution set must not contain duplicate triplets.

Example 1:

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

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

Explanation:

`nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.`

`nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.`

`nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.`

The distinct triplets are `[-1,0,1]` and `[-1,-1,2]`.

Notice that the order of the output and the order of the triplets does not matter.

Example 2:

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

Output: `[]`

Explanation: The only possible triplet does not sum up to 0.

Example 3:

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

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

Explanation: The only possible triplet sums up to 0.

Constraints:

- $3 \leq \text{nums.length} \leq 3000$
- $-10^5 \leq \text{nums}[i] \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[][]}  
 */  
var threeSum = function(nums) {  
    const result = [];  
    if (nums === null || nums.length < 3) {  
        return result;  
    }  
  
    // Sort the array  
    nums.sort((a, b) => a - b);  
  
    for (let i = 0; i < nums.length - 2; i++) {  
        // Avoid duplicate triplets  
        if (i > 0 && nums[i] === nums[i - 1]) {  
            continue;  
        }  
  
        let left = i + 1;  
        let right = nums.length - 1;  
  
        while (left < right) {  
            const sum = nums[i] + nums[left] + nums[right];  
  
            if (sum === 0) {  
                result.push([nums[i], nums[left], nums[right]]);  
  
                // Avoid duplicates for left pointer  
                while (left < right && nums[left] === nums[left + 1]) {  
                    left++;  
                }  
  
                // Avoid duplicates for right pointer  
                while (left < right && nums[right] === nums[right - 1]) {  
                    right--;  
                }  
            }  
        }  
    }  
};
```

```
        left++;
        right--;
    } else if (sum < 0) {
        left++;
    } else {
        right--;
    }
}

return result;
};
```

9 Palindrome Number ([link](#))

Description

Given an integer x , return `true` if x is a **palindrome**, and `false` otherwise.

Example 1:

```
Input: x = 121
Output: true
Explanation: 121 reads as 121 from left to right and from right to left.
```

Example 2:

```
Input: x = -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.
```

Example 3:

```
Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.
```

Constraints:

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

Follow up: Could you solve it without converting the integer to a string?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} x  
 * @return {boolean}  
 */  
var isPalindrome = function(x) {  
    var p = x.toString().split('').reverse().join('');  
    if(x==p){  
        return true;  
    }else{  
        return false;  
    }  
};
```

[134 Gas Station \(link\)](#)

Description

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

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

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

Example 1:

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

Output: 3

Explanation:

Start at station 3 (index 3) and fill up with 4 unit of gas. Your tank = $0 + 4 = 4$

Travel to station 4. Your tank = $4 - 1 + 5 = 8$

Travel to station 0. Your tank = $8 - 2 + 1 = 7$

Travel to station 1. Your tank = $7 - 3 + 2 = 6$

Travel to station 2. Your tank = $6 - 4 + 3 = 5$

Travel to station 3. The cost is 5. Your gas is just enough to travel back to station 3.

Therefore, return 3 as the starting index.

Example 2:

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

Output: -1

Explanation:

You can't start at station 0 or 1, as there is not enough gas to travel to the next station.

Let's start at station 2 and fill up with 4 unit of gas. Your tank = $0 + 4 = 4$

Travel to station 0. Your tank = $4 - 3 + 2 = 3$

Travel to station 1. Your tank = $3 - 3 + 3 = 3$

You cannot travel back to station 2, as it requires 4 unit of gas but you only have 3.

Therefore, you can't travel around the circuit once no matter where you start.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} gas
 * @param {number[]} cost
 * @return {number}
 */
var canCompleteCircuit = function(gas, cost) {
    const n = gas.length;
    let totalTank = 0; // To track the total gas remaining after the entire circuit
    let currTank = 0; // To track the gas remaining in the tank during the journey
    let startingStation = 0; // The starting station index

    for (let i = 0; i < n; i++) {
        totalTank += gas[i] - cost[i];
        currTank += gas[i] - cost[i];

        // If at any point currTank is negative, it means we can't start the journey from the previous startingStation to this point
        if (currTank < 0) {
            // Reset the starting station to the next station
            startingStation = i + 1;
            // Reset currTank
            currTank = 0;
        }
    }

    // If totalTank is negative, it means we can't complete the circuit from any station.
    return totalTank >= 0 ? startingStation : -1;
};
```

11 Container With Most Water (link)

Description

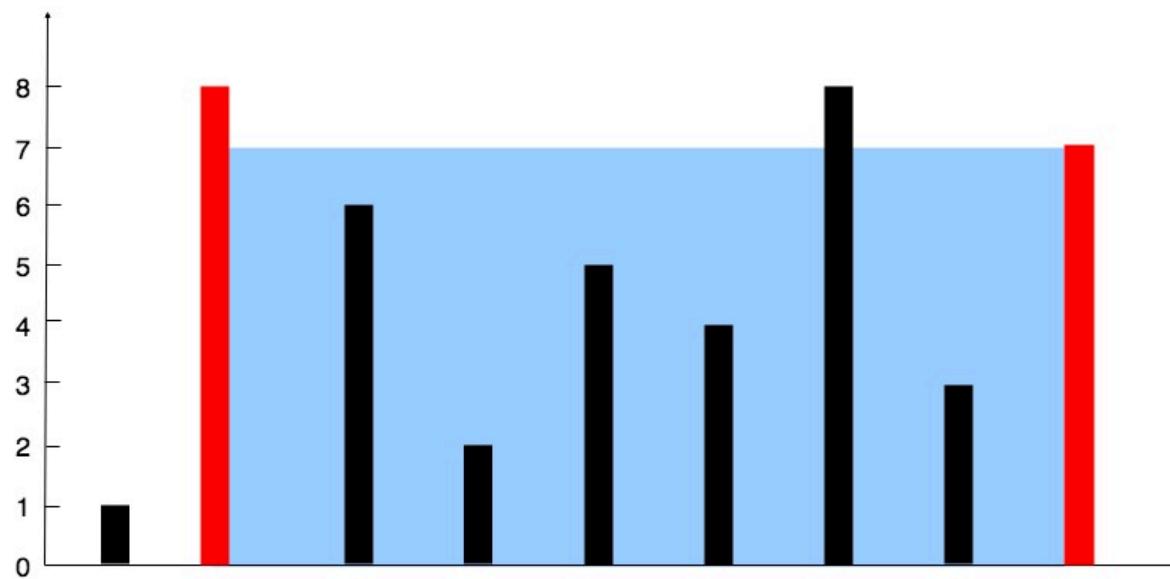
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the i^{th} line are $(i, 0)$ and $(i, \text{height}[i])$.

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

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

Notice that you may not slant the container.

Example 1:



Input: `height = [1,8,6,2,5,4,8,3,7]`
Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) is 72.

Example 2:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} height  
 * @return {number}  
 */  
var maxArea = function(height) {  
    let maxArea = 0;  
    let left = 0;  
    let right = height.length - 1;  
  
    while (left < right) {  
        let minHeight = Math.min(height[left], height[right]);  
        let width = right - left;  
        let area = minHeight * width;  
        maxArea = Math.max(maxArea, area);  
  
        if (height[left] < height[right]) {  
            left++;  
        } else {  
            right--;  
        }  
    }  
  
    return maxArea;  
};
```

33 Search in Rotated Sorted Array ([link](#))

Description

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

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

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- $1 \leq \text{nums.length} \leq 5000$

- $-10^4 \leq \text{nums}[i] \leq 10^4$
- All values of `nums` are **unique**.
- `nums` is an ascending array that is possibly rotated.
- $-10^4 \leq \text{target} \leq 10^4$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number}  
 */  
var search = function(nums, target) {  
    for(let i = 0; i < nums.length; i++){  
        if(nums[i] === target){  
            return i;  
        }  
    }  
    return -1;  
};
```

1711 Find Valid Matrix Given Row and Column Sums ([link](#))

Description

You are given two arrays `rowSum` and `colSum` of non-negative integers where `rowSum[i]` is the sum of the elements in the i^{th} row and `colSum[j]` is the sum of the elements of the j^{th} column of a 2D matrix. In other words, you do not know the elements of the matrix, but you do know the sums of each row and column.

Find any matrix of **non-negative** integers of size `rowSum.length` \times `colSum.length` that satisfies the `rowSum` and `colSum` requirements.

Return a *2D array representing any matrix that fulfills the requirements*. It's guaranteed that **at least one** matrix that fulfills the requirements exists.

Example 1:

```
Input: rowSum = [3,8], colSum = [4,7]
Output: [[3,0],
         [1,7]]
Explanation:
0th row: 3 + 0 = 3 == rowSum[0]
1st row: 1 + 7 = 8 == rowSum[1]
0th column: 3 + 1 = 4 == colSum[0]
1st column: 0 + 7 = 7 == colSum[1]
The row and column sums match, and all matrix elements are non-negative.
Another possible matrix is: [[1,2],
                           [3,5]]
```

Example 2:

```
Input: rowSum = [5,7,10], colSum = [8,6,8]
Output: [[0,5,0],
         [6,1,0],
         [2,0,8]]
```

Constraints:

- $1 \leq \text{rowSum.length}, \text{colSum.length} \leq 500$
- $0 \leq \text{rowSum}[i], \text{colSum}[i] \leq 10^8$
- $\text{sum}(\text{rowSum}) == \text{sum}(\text{colSum})$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} rowSum  
 * @param {number[]} colSum  
 * @return {number[][]}  
 */  
var restoreMatrix = function(rowSum, colSum) {  
    const m = rowSum.length;  
    const n = colSum.length;  
    const matrix = Array.from({ length: m }, () => Array(n).fill(0));  
  
    for (let i = 0; i < m; i++) {  
        for (let j = 0; j < n; j++) {  
            // Calculate the value to place in the matrix  
            const value = Math.min(rowSum[i], colSum[j]);  
            matrix[i][j] = value;  
  
            // Update rowSum and colSum  
            rowSum[i] -= value;  
            colSum[j] -= value;  
        }  
    }  
  
    return matrix;  
};
```

[12 Integer to Roman \(link\)](#)

Description

Seven different symbols represent Roman numerals with the following values:

Symbol Value

I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Roman numerals are formed by appending the conversions of decimal place values from highest to lowest. Converting a decimal place value into a Roman numeral has the following rules:

- If the value does not start with 4 or 9, select the symbol of the maximal value that can be subtracted from the input, append that symbol to the result, subtract its value, and convert the remainder to a Roman numeral.
- If the value starts with 4 or 9 use the **subtractive form** representing one symbol subtracted from the following symbol, for example, 4 is 1 (I) less than 5 (V): IV and 9 is 1 (I) less than 10 (X): IX. Only the following subtractive forms are used: 4 (IV), 9 (IX), 40 (XL), 90 (XC), 400 (CD) and 900 (CM).
- Only powers of 10 (I, X, C, M) can be appended consecutively at most 3 times to represent multiples of 10. You cannot append 5 (V), 50 (L), or 500 (D) multiple times. If you need to append a symbol 4 times use the **subtractive form**.

Given an integer, convert it to a Roman numeral.

Example 1:

Input: num = 3749

Output: "MMMDCCXLIX"

Explanation:

3000 = MMM as 1000 (M) + 1000 (M) + 1000 (M)

700 = DCC as 500 (D) + 100 (C) + 100 (C)

40 = XL as 10 (X) less of 50 (L)

9 = IX as 1 (I) less of 10 (X)

Note: 49 is not 1 (I) less of 50 (L) because the conversion is based on decimal places

Example 2:

Input: num = 58

Output: "LVIII"

Explanation:

50 = L

8 = VIII

Example 3:

Input: num = 1994

Output: "MCMXCIV"

Explanation:

1000 = M

900 = CM

90 = XC

4 = IV

Constraints:

- $1 \leq \text{num} \leq 3999$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {string}  
 */  
var intToRoman = function(num) {  
    const romanSymbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I"];  
    const romanValues = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1];  
  
    let result = '';  
  
    for (let i = 0; i < romanSymbols.length; i++) {  
        while (num >= romanValues[i]) {  
            result += romanSymbols[i];  
            num -= romanValues[i];  
        }  
    }  
  
    return result;  
};
```

[394 Decode String \(link\)](#)

Description

Given an encoded string, return its decoded string.

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

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k . For example, there will not be input like `3a` or `2[4]`.

The test cases are generated so that the length of the output will never exceed 10^5 .

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {string}
 */
function isOpening(char) {
    return '[' == char;
}
function isClosing(char) {
    return ']' == char;
}
function isDigit(char) {
    const result = parseInt(char);
    if (Number.isNaN(result)) {
        return false;
    } else {
        return true;
    }
}
function getString(stack) {
    let arr = [];
    while (stack[stack.length - 1] != '[') {
        arr.push(stack.pop());
    }
    stack.pop();
    arr.reverse();
    return arr.join("");
}
function repeatString(str, stack, count) {
    let s = "";
    while (count--) {
        s += str;
    }
    stack.push(s);
}
var decodeString = function (s) {
    let number = [], stack = [];
    for (let i = 0; i < s.length; i++) {
        if (isOpening(s[i])) {
```

```
    stack.push(s[i]);
} else if (isClosing(s[i])) {
    const str = getString(stack);
    repeatString(str, stack, number.pop());
} else if (isDigit(s[i])) {
    let str = "";
    while (isDigit(s[i])) {
        str += (s[i]);
        i += 1;
        // console.log(str);
    }
    i -= 1;
    number.push(parseInt(str)); // what is more than one digit
} else {
    stack.push(s[i]);
}
}
return stack.join('');
};
```

1496 Lucky Numbers in a Matrix ([link](#))

Description

Given an $m \times n$ matrix of **distinct** numbers, return *all lucky numbers in the matrix in any order*.

A **lucky number** is an element of the matrix such that it is the minimum element in its row and maximum in its column.

Example 1:

```
Input: matrix = [[3,7,8],[9,11,13],[15,16,17]]
```

```
Output: [15]
```

```
Explanation: 15 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Example 2:

```
Input: matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]]
```

```
Output: [12]
```

```
Explanation: 12 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Example 3:

```
Input: matrix = [[7,8],[1,2]]
```

```
Output: [7]
```

```
Explanation: 7 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Constraints:

- $m == \text{mat.length}$
- $n == \text{mat[i].length}$
- $1 \leq n, m \leq 50$
- $1 \leq \text{matrix}[i][j] \leq 10^5$.

- All elements in the matrix are distinct.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[][]} matrix
 * @return {number[]}
 */
var luckyNumbers = function(matrix) {
    const m = matrix.length;
    const n = matrix[0].length;
    let rowMins = new Array(m).fill(Infinity);
    let colMaxs = new Array(n).fill(-Infinity);

    // Find the minimum element in each row
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            if (matrix[i][j] < rowMins[i]) {
                rowMins[i] = matrix[i][j];
            }
        }
    }

    // Find the maximum element in each column
    for (let j = 0; j < n; j++) {
        for (let i = 0; i < m; i++) {
            if (matrix[i][j] > colMaxs[j]) {
                colMaxs[j] = matrix[i][j];
            }
        }
    }

    // Find all lucky numbers
    let luckyNumbers = [];
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            if (matrix[i][j] === rowMins[i] && matrix[i][j] === colMaxs[j]) {
                luckyNumbers.push(matrix[i][j]);
            }
        }
    }
}
```

```
    return luckyNumbers;  
};
```

[1128 Remove All Adjacent Duplicates In String \(link\)](#)

Description

You are given a string s consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

Example 1:

```
Input: s = "abbaca"
```

```
Output: "ca"
```

```
Explanation:
```

For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only possible move. The res

Example 2:

```
Input: s = "azxxzy"
```

```
Output: "ay"
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
var removeDuplicates = function(s) {  
    const stack = [];  
    for (const char of s) {  
        stack[stack.length - 1] === char ? stack.pop() : stack.push(char);  
    }  
    return stack.join('');  
};
```

30 Substring with Concatenation of All Words ([link](#))

Description

You are given a string s and an array of strings words . All the strings of words are of **the same length**.

A **concatenated string** is a string that exactly contains all the strings of any permutation of words concatenated.

- For example, if $\text{words} = ["ab", "cd", "ef"]$, then "abcdef", "abefcd", "cdabef", "cdefab", "efabcd", and "efcdab" are all concatenated strings. "acdbef" is not a concatenated string because it is not the concatenation of any permutation of words .

Return an array of *the starting indices* of all the concatenated substrings in s . You can return the answer in **any order**.

Example 1:

Input: $s = \text{"barfoothefoobarman"}$, $\text{words} = [\text{"foo"}, \text{"bar"}]$

Output: [0,9]

Explanation:

The substring starting at 0 is "barfoo". It is the concatenation of $["\text{bar"}, \text{"foo"}]$ which is a permutation of words .
The substring starting at 9 is "foobar". It is the concatenation of $[\text{"foo"}, \text{"bar"}]$ which is a permutation of words .

Example 2:

Input: $s = \text{"wordgoodgoodgoodbestword"}$, $\text{words} = [\text{"word"}, \text{"good"}, \text{"best"}, \text{"word"}]$

Output: []

Explanation:

There is no concatenated substring.

Example 3:

Input: $s = \text{"barfoofoobarthefoobarman"}$, $\text{words} = [\text{"bar"}, \text{"foo"}, \text{"the"}]$

Output: [6,9,12]**Explanation:**

The substring starting at 6 is "foobarthe". It is the concatenation of ["foo", "bar", "the"].

The substring starting at 9 is "barthefoo". It is the concatenation of ["bar", "the", "foo"].

The substring starting at 12 is "thefoobar". It is the concatenation of ["the", "foo", "bar"].

Constraints:

- $1 \leq s.length \leq 10^4$
- $1 \leq words.length \leq 5000$
- $1 \leq words[i].length \leq 30$
- s and $words[i]$ consist of lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {string[]} words  
 * @return {number[]}  
 */  
var findSubstring = function(s, words) {  
    if (s.length === 0 || words.length === 0) return [];  
  
    const wordLength = words[0].length;  
    const numWords = words.length;  
    const concatLength = wordLength * numWords;  
    const result = [];  
  
    const wordCount = {};  
    for (const word of words) {  
        if (wordCount[word] !== undefined) {  
            wordCount[word]++;  
        } else {  
            wordCount[word] = 1;  
        }  
    }  
  
    for (let i = 0; i <= s.length - concatLength; i++) {  
        const seen = {};  
        let j = 0;  
        while (j < numWords) {  
            const word = s.substring(i + j * wordLength, i + (j + 1) * wordLength);  
            if (wordCount[word] !== undefined) {  
                if (seen[word] !== undefined) {  
                    seen[word]++;  
                } else {  
                    seen[word] = 1;  
                }  
  
                if (seen[word] > wordCount[word]) break;  
            } else {  
                break;  
            }  
            j++;  
        }  
        if (j === numWords) result.push(i);  
    }  
};
```

```
        j++;
    }
    if (j === numWords) {
        result.push(i);
    }
}

return result;
};
```

6 Zigzag Conversion ([link](#))

Description

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N  
A P L S I I G  
Y   I   R
```

And then read line by line: "PAHNAPLSIIGYIR"

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string s, int numRows);
```

Example 1:

```
Input: s = "PAYPALISHIRING", numRows = 3  
Output: "PAHNAPLSIIGYIR"
```

Example 2:

```
Input: s = "PAYPALISHIRING", numRows = 4  
Output: "PINALSIGYAHRPI"  
Explanation:  
P       I       N  
A       L       S       I       G  
Y       A       H       R  
P       I
```

Example 3:

Input: s = "A", numRows = 1
Output: "A"

Constraints:

- $1 \leq s.length \leq 1000$
- s consists of English letters (lower-case and upper-case), ',', and '.'.
- $1 \leq numRow \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {number} numRows  
 * @return {string}  
 */  
var convert = function(s, numRows) {  
    if (numRows === 1 || s.length <= numRows) {  
        return s;  
    }  
  
    let rows = new Array(Math.min(numRows, s.length)).fill("");  
    let curRow = 0;  
    let goingDown = false;  
  
    for (let char of s) {  
        rows[curRow] += char;  
        if (curRow === 0 || curRow === numRows - 1) {  
            goingDown = !goingDown;  
        }  
        curRow += goingDown ? 1 : -1;  
    }  
  
    return rows.join("");  
};
```

[41 First Missing Positive \(link\)](#)

Description

Given an unsorted integer array `nums`. Return the *smallest positive integer* that is *not present* in `nums`.

You must implement an algorithm that runs in $O(n)$ time and uses $O(1)$ auxiliary space.

Example 1:

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

Output: 3

Explanation: The numbers in the range [1,2] are all in the array.

Example 2:

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

Output: 2

Explanation: 1 is in the array but 2 is missing.

Example 3:

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

Output: 1

Explanation: The smallest positive integer 1 is missing.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var firstMissingPositive = function(nums) {  
    const n = nums.length;  
  
    // Place each number in its correct position  
    for (let i = 0; i < n; i++) {  
        while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] !== nums[i]) {  
            // Swap nums[i] with nums[nums[i] - 1]  
            let correctIndex = nums[i] - 1;  
            [nums[i], nums[correctIndex]] = [nums[correctIndex], nums[i]];  
        }  
    }  
  
    // Find the first index which doesn't have the correct number  
    for (let i = 0; i < n; i++) {  
        if (nums[i] !== i + 1) {  
            return i + 1;  
        }  
    }  
  
    // If all numbers are in the correct position, return n + 1  
    return n + 1;  
};
```

[39 Combination Sum \(link\)](#)

Description

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

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

The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
```

```
Output: [[2,2,3],[7]]
```

Explanation:

2 and 3 are candidates, and $2 + 2 + 3 = 7$. Note that 2 can be used multiple times.

7 is a candidate, and $7 = 7$.

These are the only two combinations.

Example 2:

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

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

Example 3:

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

```
Output: []
```

Constraints:

- $1 \leq \text{candidates.length} \leq 30$
- $2 \leq \text{candidates}[i] \leq 40$
- All elements of candidates are **distinct**.
- $1 \leq \text{target} \leq 40$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} candidates  
 * @param {number} target  
 * @return {number[][][]}  
 */  
var combinationSum = function(candidates, target) {  
    const result = [];  
  
    const backtrack = (remaining, start, current) => {  
        if (remaining === 0) {  
            result.push([...current]);  
            return;  
        }  
  
        for (let i = start; i < candidates.length; i++) {  
            if (candidates[i] <= remaining) {  
                current.push(candidates[i]);  
                backtrack(remaining - candidates[i], i, current);  
                current.pop();  
            }  
        }  
    };  
  
    backtrack(target, 0, []);  
    return result;  
};
```

1298 Reverse Substrings Between Each Pair of Parentheses ([link](#))

Description

You are given a string s that consists of lower case English letters and brackets.

Reverse the strings in each pair of matching parentheses, starting from the innermost one.

Your result should **not** contain any brackets.

Example 1:

```
Input: s = "(abcd)"  
Output: "dcba"
```

Example 2:

```
Input: s = "(u(love)i)"  
Output: "iloveu"  
Explanation: The substring "love" is reversed first, then the whole string is reversed.
```

Example 3:

```
Input: s = "(ed(et(oc))el)"  
Output: "leetcode"  
Explanation: First, we reverse the substring "oc", then "etco", and finally, the whole string.
```

Constraints:

- $1 \leq s.length \leq 2000$
- s only contains lower case English characters and parentheses.
- It is guaranteed that all parentheses are balanced.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String reverseParentheses(String s) {
        Stack<Integer> stack = new Stack<>();
        char[] chars = s.toCharArray();
        int n = chars.length;

        for (int i = 0; i < n; i++) {
            if (chars[i] == '(') {
                stack.push(i);
            } else if (chars[i] == ')') {
                int start = stack.pop();
                reverseSubstring(chars, start + 1, i - 1);
            }
        }

        StringBuilder result = new StringBuilder();
        for (char c : chars) {
            if (c != '(' && c != ')') {
                result.append(c);
            }
        }

        return result.toString();
    }

    public static void reverseSubstring(char[] arr, int start, int end) {
        while (start < end) {
            char temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
}
```

}

[1951 Find the Winner of the Circular Game \(link\)](#)

Description

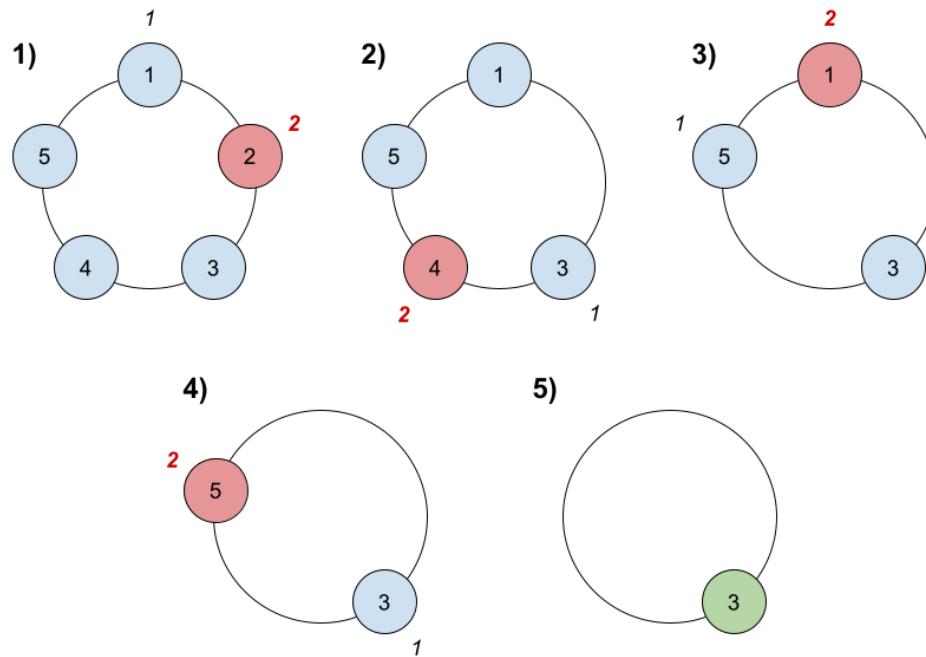
There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in **clockwise order**. More formally, moving clockwise from the i^{th} friend brings you to the $(i+1)^{\text{th}}$ friend for $1 \leq i < n$, and moving clockwise from the n^{th} friend brings you to the 1^{st} friend.

The rules of the game are as follows:

1. **Start** at the 1^{st} friend.
2. Count the next k friends in the clockwise direction **including** the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step 2 **starting** from the friend **immediately clockwise** of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, n , and an integer k , return *the winner of the game*.

Example 1:



Input: $n = 5$, $k = 2$

Output: 3

Explanation: Here are the steps of the game:

- 1) Start at friend 1.
- 2) Count 2 friends clockwise, which are friends 1 and 2.
- 3) Friend 2 leaves the circle. Next start is friend 3.
- 4) Count 2 friends clockwise, which are friends 3 and 4.
- 5) Friend 4 leaves the circle. Next start is friend 5.
- 6) Count 2 friends clockwise, which are friends 5 and 1.
- 7) Friend 1 leaves the circle. Next start is friend 3.
- 8) Count 2 friends clockwise, which are friends 3 and 5.
- 9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

Example 2:

Input: $n = 6$, $k = 5$

Output: 1

Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

Constraints:

- $1 \leq k \leq n \leq 500$

Follow up:

Could you solve this problem in linear time with constant space?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @param {number} k  
 * @return {number}  
 */  
function findTheWinner(n, k) {  
    // Create an array representing the circle of friends  
    let friends = [];  
    for (let i = 1; i <= n; i++) {  
        friends.push(i);  
    }  
  
    let current = 0; // Start with the first friend  
  
    while (friends.length > 1) {  
        // Calculate the index of the friend to be eliminated  
        current = (current + k - 1) % friends.length;  
        // Remove the eliminated friend from the array  
        friends.splice(current, 1);  
    }  
  
    // Return the last remaining friend (winner)  
    return friends[0];  
}
```

[3396 Valid Word \(link\)](#)

Description

A word is considered **valid** if:

- It contains a **minimum** of 3 characters.
- It contains only digits (0-9), and English letters (uppercase and lowercase).
- It includes **at least** one **vowel**.
- It includes **at least** one **consonant**.

You are given a string `word`.

Return `true` if `word` is valid, otherwise, return `false`.

Notes:

- 'a', 'e', 'i', 'o', 'u', and their upercases are **vowels**.
- A **consonant** is an English letter that is not a vowel.

Example 1:

Input: word = "234Adas"

Output: true

Explanation:

This word satisfies the conditions.

Example 2:

Input: word = "b3"

Output: false

Explanation:

The length of this word is fewer than 3, and does not have a vowel.

Example 3:

Input: word = "a3\$e"

Output: false

Explanation:

This word contains a '\$' character and does not have a consonant.

Constraints:

- $1 \leq \text{word.length} \leq 20$
- word consists of English uppercase and lowercase letters, digits, '@', '#', and '\$'.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean isValid(String word) {
        if (word.length() < 3) {
            return false;
        }

        boolean hasVowel = false;
        boolean hasConsonant = false;

        for (char c : word.toCharArray()) {
            if (!Character.isLetterOrDigit(c)) {
                return false;
            }
            if (isVowel(c)) {
                hasVowel = true;
            } else if (isConsonant(c)) {
                hasConsonant = true;
            }
        }

        return hasVowel && hasConsonant;
    }

    public static boolean isVowel(char c) {
        c = Character.toLowerCase(c);
        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';
    }

    public static boolean isConsonant(char c) {
        return Character.isLetter(c) && !isVowel(c);
    }
}
```

16 3Sum Closest (link)

Description

Given an integer array `nums` of length `n` and an integer `target`, find three integers in `nums` such that the sum is closest to `target`.

Return *the sum of the three integers*.

You may assume that each input would have exactly one solution.

Example 1:

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

Output: 2

Explanation: The sum that is closest to the target is 2. $(-1 + 2 + 1 = 2)$.

Example 2:

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

Output: 0

Explanation: The sum that is closest to the target is 0. $(0 + 0 + 0 = 0)$.

Constraints:

- $3 \leq \text{nums.length} \leq 500$
- $-1000 \leq \text{nums}[i] \leq 1000$
- $-10^4 \leq \text{target} \leq 10^4$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int closestSum = nums[0] + nums[1] + nums[2]; // Initialize with the first three elements

        for (int i = 0; i < nums.length - 2; i++) {
            int left = i + 1;
            int right = nums.length - 1;

            while (left < right) {
                int currentSum = nums[i] + nums[left] + nums[right];

                if (currentSum == target) {
                    return currentSum; // Found the exact sum
                }

                // Update the closest sum if the current one is closer to the target
                if (Math.abs(currentSum - target) < Math.abs(closestSum - target)) {
                    closestSum = currentSum;
                }

                if (currentSum < target) {
                    left++;
                } else {
                    right--;
                }
            }
        }

        return closestSum;
    }
}
```

350 Intersection of Two Arrays II ([link](#))

Description

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

Example 1:

```
Input: 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]
Explanation: [9,4] is also accepted.
```

Constraints:

- $1 \leq \text{nums1.length}, \text{nums2.length} \leq 1000$
- $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$

Follow up:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        Map<Integer, Integer> countMap = new HashMap<>();
        List<Integer> intersection = new ArrayList<>();

        // Count occurrences of each element in nums1
        for (int num : nums1) {
            countMap.put(num, countMap.getOrDefault(num, 0) + 1);
        }

        // Find common elements in nums2 and add to the result
        for (int num : nums2) {
            if (countMap.containsKey(num) && countMap.get(num) > 0) {
                intersection.add(num);
                countMap.put(num, countMap.get(num) - 1);
            }
        }

        // Convert the result list to an array
        int[] result = new int[intersection.size()];
        for (int i = 0; i < intersection.size(); i++) {
            result[i] = intersection.get(i);
        }

        return result;
    }
}
```

[3349 Maximum Length Substring With Two Occurrences \(link\)](#)

Description

Given a string s , return the **maximum** length of a substring such that it contains *at most two occurrences* of each character.

Example 1:

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

Output: 4

Explanation:

The following substring has a length of 4 and contains at most two occurrences of each character: "bcbbbcbba".

Example 2:

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

Output: 2

Explanation:

The following substring has a length of 2 and contains at most two occurrences of each character: "aaaa".

Constraints:

- $2 \leq s.\text{length} \leq 100$
- s consists only of lowercase English letters.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int maximumLengthSubstring(String s) {
        Map<Character, Integer> charCount = new HashMap<>();
        int left = 0, right = 0, maxLength = 0;

        while (right < s.length()) {
            char rightChar = s.charAt(right);
            charCount.put(rightChar, charCount.getOrDefault(rightChar, 0) + 1);

            while (charCount.get(rightChar) > 2) {
                char leftChar = s.charAt(left);
                charCount.put(leftChar, charCount.get(leftChar) - 1);
                left++;
            }

            maxLength = Math.max(maxLength, right - left + 1);
            right++;
        }

        return maxLength;
    }
}
```

3324 Split the Array (link)

Description

You are given an integer array `nums` of **even** length. You have to split the array into two parts `nums1` and `nums2` such that:

- `nums1.length == nums2.length == nums.length / 2.`
- `nums1` should contain **distinct** elements.
- `nums2` should also contain **distinct** elements.

Return `true` if it is possible to split the array, and `false` otherwise.

Example 1:

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

Output: `true`

Explanation: One of the possible ways to split `nums` is `nums1 = [1,2,3]` and `nums2 = [1,2,4]`.

Example 2:

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

Output: `false`

Explanation: The only possible way to split `nums` is `nums1 = [1,1]` and `nums2 = [1,1]`. Both `nums1` and `nums2` do not contain distinct elements.

Constraints:

- `1 <= nums.length <= 100`
- `nums.length % 2 == 0`
- `1 <= nums[i] <= 100`

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean isPossibleToSplit(int[] nums) {
        int n = nums.length;
        if (n % 2 != 0) {
            return false; // The array length should be even.
        }

        int halfLength = n / 2;

        // Count occurrences of each element
        Map<Integer, Integer> countMap = new HashMap<>();
        for (int num : nums) {
            countMap.put(num, countMap.getOrDefault(num, 0) + 1);
        }

        // Check if any element occurs more than twice
        for (int count : countMap.values()) {
            if (count > 2) {
                return false;
            }
        }

        // Check if there are enough distinct elements
        Set<Integer> distinctElements = new HashSet<>(countMap.keySet());
        if (distinctElements.size() < halfLength) {
            return false;
        }

        return true;
    }
}
```

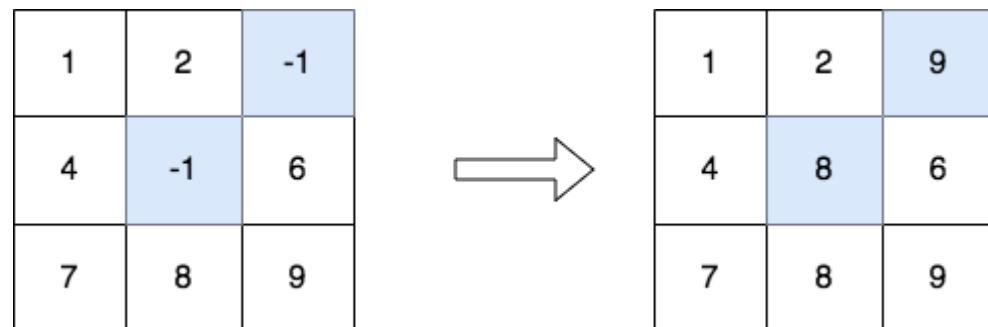
[3330 Modify the Matrix \(link\)](#)

Description

Given a **0-indexed** $m \times n$ integer matrix `matrix`, create a new **0-indexed** matrix called `answer`. Make `answer` equal to `matrix`, then replace each element with the value `-1` with the **maximum** element in its respective column.

Return *the matrix* `answer`.

Example 1:



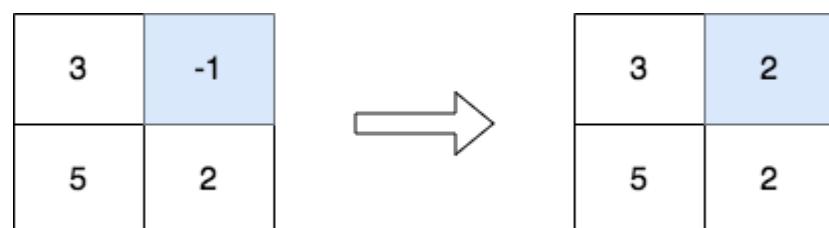
Input: `matrix = [[1,2,-1],[4,-1,6],[7,8,9]]`

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

Explanation: The diagram above shows the elements that are changed (in blue).

- We replace the value in the cell `[1][1]` with the maximum value in the column 1, that is 8.
- We replace the value in the cell `[0][2]` with the maximum value in the column 2, that is 9.

Example 2:



Input: matrix = [[3,-1],[5,2]]

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

Explanation: The diagram above shows the elements that are changed (in blue).

Constraints:

- $m == \text{matrix.length}$
- $n == \text{matrix}[i].length$
- $2 \leq m, n \leq 50$
- $-1 \leq \text{matrix}[i][j] \leq 100$
- The input is generated such that each column contains at least one non-negative integer.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[][] modifiedMatrix(int[][][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;

        // Create the answer matrix as a copy of the original matrix
        int[][] answer = new int[m][n];
        for (int i = 0; i < m; i++) {
            System.arraycopy(matrix[i], 0, answer[i], 0, n);
        }

        // Find the maximum element in each column
        int[] maxInColumn = new int[n];
        for (int j = 0; j < n; j++) {
            int max = Integer.MIN_VALUE;
            for (int i = 0; i < m; i++) {
                if (matrix[i][j] > max) {
                    max = matrix[i][j];
                }
            }
            maxInColumn[j] = max;
        }

        // Replace -1 with the maximum element in its respective column
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (matrix[i][j] == -1) {
                    answer[i][j] = maxInColumn[j];
                }
            }
        }

        return answer;
    }
}
```

3309 Count Prefix and Suffix Pairs I ([link](#))

Description

You are given a **0-indexed** string array `words`.

Let's define a **boolean** function `isPrefixAndSuffix` that takes two strings, `str1` and `str2`:

- `isPrefixAndSuffix(str1, str2)` returns true if `str1` is **both** a prefix and a suffix of `str2`, and false otherwise.

For example, `isPrefixAndSuffix("aba", "ababa")` is true because "aba" is a prefix of "ababa" and also a suffix, but `isPrefixAndSuffix("abc", "abcd")` is false.

Return *an integer denoting the number of index pairs* (i, j) *such that* $i < j$, *and* `isPrefixAndSuffix(words[i], words[j])` *is* true.

Example 1:

```
Input: words = ["a", "aba", "ababa", "aa"]
Output: 4
Explanation: In this example, the counted index pairs are:
i = 0 and j = 1 because isPrefixAndSuffix("a", "aba") is true.
i = 0 and j = 2 because isPrefixAndSuffix("a", "ababa") is true.
i = 0 and j = 3 because isPrefixAndSuffix("a", "aa") is true.
i = 1 and j = 2 because isPrefixAndSuffix("aba", "ababa") is true.
Therefore, the answer is 4.
```

Example 2:

```
Input: words = ["pa", "papa", "ma", "mama"]
Output: 2
Explanation: In this example, the counted index pairs are:
i = 0 and j = 1 because isPrefixAndSuffix("pa", "papa") is true.
i = 2 and j = 3 because isPrefixAndSuffix("ma", "mama") is true.
Therefore, the answer is 2.
```

Example 3:

Input: words = ["abab", "ab"]

Output: 0

Explanation: In this example, the only valid index pair is i = 0 and j = 1, and isPrefixAndSuffix("abab", "ab") is false. Therefore, the answer is 0.

Constraints:

- $1 \leq \text{words.length} \leq 50$
- $1 \leq \text{words[i].length} \leq 10$
- words[i] consists only of lowercase English letters.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int countPrefixSuffixPairs(String[] words) {
        int count = 0;
        int n = words.length;

        // Iterate through all pairs (i, j) with i < j
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (isPrefixAndSuffix(words[i], words[j])) {
                    count++;
                }
            }
        }

        return count;
    }

    public static boolean isPrefixAndSuffix(String str1, String str2) {
        int len1 = str1.length();
        int len2 = str2.length();
        if (len1 > len2) {
            return false;
        }
        // Check if str1 is a prefix of str2
        boolean isPrefix = str2.startsWith(str1);
        // Check if str1 is a suffix of str2
        boolean isSuffix = str2.endsWith(str1);
        return isPrefix && isSuffix;
    }
}
```

3242 Count Elements With Maximum Frequency ([link](#))

Description

You are given an array `nums` consisting of **positive** integers.

Return *the total frequencies of elements in `nums` such that those elements all have the maximum frequency*.

The **frequency** of an element is the number of occurrences of that element in the array.

Example 1:

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

Output: 4

Explanation: The elements 1 and 2 have a frequency of 2 which is the maximum frequency in the array.
So the number of elements in the array with maximum frequency is 4.

Example 2:

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

Output: 5

Explanation: All elements of the array have a frequency of 1 which is the maximum.
So the number of elements in the array with maximum frequency is 5.

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int maxFrequencyElements(int[] nums) {
        // Create a map to store the frequency of each element
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        // Calculate the frequency of each element
        for (int num : nums) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }

        // Find the maximum frequency
        int maxFrequency = 0;
        for (int freq : frequencyMap.values()) {
            if (freq > maxFrequency) {
                maxFrequency = freq;
            }
        }

        // Count the total frequencies of elements with the maximum frequency
        int total = 0;
        for (int freq : frequencyMap.values()) {
            if (freq == maxFrequency) {
                total += freq;
            }
        }

        return total;
    }
}
```

[3446 Find the Number of Good Pairs I \(link\)](#)

Description

You are given 2 integer arrays `nums1` and `nums2` of lengths `n` and `m` respectively. You are also given a **positive** integer `k`.

A pair (i, j) is called **good** if `nums1[i]` is divisible by `nums2[j] * k` ($0 \leq i \leq n - 1, 0 \leq j \leq m - 1$).

Return the total number of **good** pairs.

Example 1:

Input: `nums1 = [1,3,4]`, `nums2 = [1,3,4]`, `k = 1`

Output: 5

Explanation:

The 5 good pairs are $(0, 0)$, $(1, 0)$, $(1, 1)$, $(2, 0)$, and $(2, 2)$.

Example 2:

Input: `nums1 = [1,2,4,12]`, `nums2 = [2,4]`, `k = 3`

Output: 2

Explanation:

The 2 good pairs are $(3, 0)$ and $(3, 1)$.

Constraints:

- $1 \leq n, m \leq 50$
- $1 \leq \text{nums1}[i], \text{nums2}[j] \leq 50$
- $1 \leq k \leq 50$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int numberOfPairs(int[] nums1, int[] nums2, int k) {
        int count = 0;

        for (int i = 0; i < nums1.length; i++) {
            for (int j = 0; j < nums2.length; j++) {
                if (nums1[i] % (nums2[j] * k) == 0) {
                    count++;
                }
            }
        }

        return count;
    }
}
```

[3447 Clear Digits \(link\)](#)

Description

You are given a string s .

Your task is to remove **all** digits by doing this operation repeatedly:

- Delete the *first* digit and the **closest non-digit** character to its *left*.

Return the resulting string after removing all digits.

Example 1:

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

Output: "abc"

Explanation:

There is no digit in the string.

Example 2:

Input: $s = \text{"cb34"}$

Output: "\u2022"

Explanation:

First, we apply the operation on $s[2]$, and s becomes "c4" .

Then we apply the operation on $s[1]$, and s becomes "\u2022" .

Constraints:

- $1 \leq s.length \leq 100$
- s consists only of lowercase English letters and digits.
- The input is generated such that it is possible to delete all digits.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String clearDigits(String s) {
        StringBuilder sb = new StringBuilder(s);

        while (true) {
            int firstDigitIndex = -1;

            // Find the index of the first digit
            for (int i = 0; i < sb.length(); i++) {
                if (Character.isDigit(sb.charAt(i))) {
                    firstDigitIndex = i;
                    break;
                }
            }

            // If no digit is found, break the loop
            if (firstDigitIndex == -1) {
                break;
            }

            // Find the closest non-digit character to the left of the first digit
            int closestNonDigitIndex = -1;
            for (int i = firstDigitIndex - 1; i >= 0; i--) {
                if (!Character.isDigit(sb.charAt(i))) {
                    closestNonDigitIndex = i;
                    break;
                }
            }

            // Remove the first digit and the closest non-digit character to its left
            if (closestNonDigitIndex != -1) {
                sb.deleteCharAt(firstDigitIndex);
                sb.deleteCharAt(closestNonDigitIndex);
            } else {
                // If there is no non-digit character to the left, just remove the digit
                sb.deleteCharAt(firstDigitIndex);
            }
        }
    }
}
```

```
        return sb.toString();  
    }  
}
```

3421 Count Pairs That Form a Complete Day I ([link](#))

Description

Given an integer array `hours` representing times in `hours`, return an integer denoting the number of pairs i, j where $i < j$ and $\text{hours}[i] + \text{hours}[j]$ forms a **complete day**.

A **complete day** is defined as a time duration that is an **exact multiple** of 24 hours.

For example, 1 day is 24 hours, 2 days is 48 hours, 3 days is 72 hours, and so on.

Example 1:

Input: `hours = [12,12,30,24,24]`

Output: 2

Explanation:

The pairs of indices that form a complete day are $(0, 1)$ and $(3, 4)$.

Example 2:

Input: `hours = [72,48,24,3]`

Output: 3

Explanation:

The pairs of indices that form a complete day are $(0, 1)$, $(0, 2)$, and $(1, 2)$.

Constraints:

- $1 \leq \text{hours.length} \leq 100$
- $1 \leq \text{hours}[i] \leq 10^9$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int countCompleteDayPairs(int[] hours) {
        int count = 0;

        for (int i = 0; i < hours.length; i++) {
            for (int j = i + 1; j < hours.length; j++) {
                if ((hours[i] + hours[j]) % 24 == 0) {
                    count++;
                }
            }
        }

        return count;
    }
}
```

[14 Longest Common Prefix \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq \text{strs.length} \leq 200$
- $0 \leq \text{strs[i].length} \leq 200$
- strs[i] consists of only lowercase English letters.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs == null || strs.length == 0) {
            return "";
        }

        String prefix = strs[0];
        for (int i = 1; i < strs.length; i++) {
            while (strs[i].indexOf(prefix) != 0) {
                prefix = prefix.substring(0, prefix.length() - 1);
                if (prefix.isEmpty()) {
                    return "";
                }
            }
        }
        return prefix;
    }
}
```

121 Best Time to Buy and Sell Stock ([link](#))

Description

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

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

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

Example 1:

Input: `prices = [7,1,5,3,6,4]`

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

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

Example 2:

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

Output: 0

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

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length == 0) {
            return 0;
        }

        int minPrice = Integer.MAX_VALUE;
        int maxProfit = 0;

        for (int price : prices) {
            if (price < minPrice) {
                minPrice = price;
            } else if (price - minPrice > maxProfit) {
                maxProfit = price - minPrice;
            }
        }

        return maxProfit;
    }
}
```

32 Longest Valid Parentheses ([link](#))

Description

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int longestValidParentheses(String s) {
        Stack<Integer> stack = new Stack<>();
        int maxLength = 0;
        stack.push(-1); // Initialize with a dummy index

        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c == '(') {
                stack.push(i);
            } else { // c == ')'
                stack.pop();
                if (stack.isEmpty()) {
                    stack.push(i);
                } else {
                    maxLength = Math.max(maxLength, i - stack.peek());
                }
            }
        }

        return maxLength;
    }
}
```

3 Longest Substring Without Repeating Characters ([link](#))

Description

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

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        if (n == 0) return 0;

        HashMap<Character, Integer> map = new HashMap<>();
        int maxLength = 0;
        int start = 0;

        for (int end = 0; end < n; end++) {
            char currentChar = s.charAt(end);
            if (map.containsKey(currentChar)) {
                // Move the start pointer to the right of the last occurrence of currentChar
                start = Math.max(start, map.get(currentChar) + 1);
            }
            // Update the index of the current character
            map.put(currentChar, end);
            // Update the maximum length of the substring
            maxLength = Math.max(maxLength, end - start + 1);
        }

        return maxLength;
    }
}
```

[50 Pow\(x, n\) \(link\)](#)

Description

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

- $-100.0 < x < 100.0$
- $-2^{31} \leq n \leq 2^{31}-1$
- n is an integer.
- Either x is not zero or $n > 0$.
- $-10^4 \leq x^n \leq 10^4$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {  
    public double myPow(double x, int n) {  
  
        double ans = Math.pow(x,n);  
        return ans;  
  
    }  
}
```

1056 Capacity To Ship Packages Within D Days ([link](#))

Description

A conveyor belt has packages that must be shipped from one port to another within `days` days.

The i^{th} package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `days` days.

Example 1:

```
Input: weights = [1,2,3,4,5,6,7,8,9,10], days = 5
```

```
Output: 15
```

Explanation: A ship capacity of 15 is the minimum to ship all the packages in 5 days like this:

1st day: 1, 2, 3, 4, 5

2nd day: 6, 7

3rd day: 8

4th day: 9

5th day: 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 14 and splitting the packages into parts like (2, 3, 4, 5, 6, 7, 8, 9) is not allowed.



Example 2:

```
Input: weights = [3,2,2,4,1,4], days = 3
```

```
Output: 6
```

Explanation: A ship capacity of 6 is the minimum to ship all the packages in 3 days like this:

1st day: 3, 2

2nd day: 2, 4

3rd day: 1, 4

Example 3:

Input: weights = [1,2,3,1,1], days = 4

Output: 3

Explanation:

1st day: 1

2nd day: 2

3rd day: 3

4th day: 1, 1

Constraints:

- $1 \leq \text{days} \leq \text{weights.length} \leq 5 * 10^4$
- $1 \leq \text{weights}[i] \leq 500$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int shipWithinDays(int[] weights, int days) {
        int max = 0;
        int sum = 0;

        for(int val : weights){
            sum += val;
            max = Math.max(max, val);
        }

        if(weights.length == days){
            return max;
        }

        int low = max;
        int high = sum;
        int ans = 0;

        while(low <= high){
            int mid = low + (high-low)/2;
            if(isPossible(weights, mid, days)){
                ans = mid;
                high = mid-1;

            }else{
                low = mid+1;
            }
        }
        return ans;
    }

    public static boolean isPossible(int[] weights, int mid, int days){
        int d=1;
        int sum =0;
        for(int i=0; i <weights.length; i++){
            sum += weights[i];
            if(sum > mid){
                d++;
                sum = weights[i];
            }
        }
        return d <= days;
    }
}
```

```
        }
    }
    return d <= days;
}
```

128 Longest Consecutive Sequence ([link](#))

Description

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

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

Example 1:

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

Output: 4

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

Example 2:

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

Output: 9

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int longestConsecutive(int[] nums) {
        int n = nums.length;
        if(n==0){
            return 0;
        }

        Arrays.sort(nums);

        int count = 1;
        int max = 0;

        for(int i=1; i< n; i++){
            if(nums[i] != nums[i-1]){
                if(nums[i] == nums[i-1] + 1){
                    count++;
                }

                }else{
                    max = Math.max(max, count);
                    count = 1;
                }
            }
        }

        return Math.max(max, count);
    }
}
```

[238 Product of Array Except Self \(link\)](#)

Description

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

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

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

Example 1:

```
Input: nums = [1,2,3,4]
Output: [24,12,8,6]
```

Example 2:

```
Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]
```

Constraints:

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

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
        Scanner scanner = new Scanner(System.in);

        // Input
        int N = nums.length;

        // Output - Product of array except the element at that index
        int[] productArray = new int[N];

        // Calculate product of all elements to the left of each index
        int leftProduct = 1;
        for (int i = 0; i < N; i++) {
            productArray[i] = leftProduct;
            leftProduct *= nums[i];
        }

        // Calculate product of all elements to the right of each index
        int rightProduct = 1;
        for (int i = N - 1; i >= 0; i--) {
            productArray[i] *= rightProduct;
            rightProduct *= nums[i];
        }

        // Print the product of array except the element at that index
        return (productArray);
    }
}
```

[735 Asteroid Collision \(link\)](#)

Description

We are given an array `asteroids` of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:

Input: `asteroids = [5,10,-5]`

Output: `[5,10]`

Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:

Input: `asteroids = [8,-8]`

Output: `[]`

Explanation: The 8 and -8 collide exploding each other.

Example 3:

Input: `asteroids = [10,2,-5]`

Output: `[10]`

Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Constraints:

- $2 \leq \text{asteroids.length} \leq 10^4$
- $-1000 \leq \text{asteroids}[i] \leq 1000$
- $\text{asteroids}[i] \neq 0$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[] asteroidCollision(int[] asteroids) {
        Stack<Integer> st = new Stack<>();
        for(int i = 0; i < asteroids.length; i++){
            if(asteroids[i] > 0){
                st.push(asteroids[i]);
            }else{
                while(st.size() > 0 && st.peek() > 0 && st.peek() < -1*asteroids[i]){
                    st.pop();
                }
                if(st.size() > 0 && st.peek() == -1*asteroids[i]){
                    st.pop();
                }else if(st.isEmpty() || st.peek() < 0){
                    st.push(asteroids[i]);
                }
            }
        }
        int[] ans = new int[st.size()];
        int i = st.size()-1;
        while(st.size() > 0){
            int ele = st.pop();
            ans[i] = ele;
            i--;
        }
        return ans;
    }
}
```

1463 The K Weakest Rows in a Matrix ([link](#))

Description

You are given an $m \times n$ binary matrix `mat` of 1's (representing soldiers) and 0's (representing civilians). The soldiers are positioned **in front** of the civilians. That is, all the 1's will appear to the **left** of all the 0's in each row.

A row i is **weaker** than a row j if one of the following is true:

- The number of soldiers in row i is less than the number of soldiers in row j .
- Both rows have the same number of soldiers and $i < j$.

Return *the indices of the k weakest rows in the matrix ordered from weakest to strongest*.

Example 1:

```
Input: mat =  
[[1,1,0,0,0],  
 [1,1,1,1,0],  
 [1,0,0,0,0],  
 [1,1,0,0,0],  
 [1,1,1,1,1]],  
k = 3  
Output: [2,0,3]
```

Explanation:

The number of soldiers in each row is:

- Row 0: 2
- Row 1: 4
- Row 2: 1
- Row 3: 2
- Row 4: 5

The rows ordered from weakest to strongest are [2,0,3,1,4].

Example 2:

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

```
[1,1,1,1],  
[1,0,0,0],  
[1,0,0,0]],  
k = 2
```

Output: [0,2]

Explanation:

The number of soldiers in each row is:

- Row 0: 1
- Row 1: 4
- Row 2: 1
- Row 3: 1

The rows ordered from weakest to strongest are [0,2,3,1].

Constraints:

- m == mat.length
- n == mat[i].length
- 2 <= n, m <= 100
- 1 <= k <= m
- matrix[i][j] is either 0 or 1.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[] kWeakestRows(int[][] mat, int k) {
        int[] weakestRows = new int[k];
        int[] counts = new int[mat.length];

        for (int i = 0; i < mat.length; i++) {
            counts[i] = countSoldiers(mat[i]);
        }

        for (int i = 0; i < k; i++) {
            int minCountIndex = findMinIndex(counts);
            weakestRows[i] = minCountIndex;
            counts[minCountIndex] = Integer.MAX_VALUE; // Mark as visited
        }

        return weakestRows;
    }

    public static int countSoldiers(int[] row) {
        int count = 0;
        for (int num : row) {
            if (num == 1) {
                count++;
            } else {
                break;
            }
        }
        return count;
    }

    public static int findMinIndex(int[] counts) {
        int minIndex = 0;
        for (int i = 1; i < counts.length; i++) {
            if (counts[i] < counts[minIndex]) {
                minIndex = i;
            }
        }
        return minIndex;
    }
}
```

```
    }  
}
```

[3397 Find the Integer Added to Array I \(link\)](#)

Description

You are given two arrays of equal length, `nums1` and `nums2`.

Each element in `nums1` has been increased (or decreased in the case of negative) by an integer, represented by the variable `x`.

As a result, `nums1` becomes **equal** to `nums2`. Two arrays are considered **equal** when they contain the same integers with the same frequencies.

Return the integer `x`.

Example 1:

Input: `nums1` = [2,6,4], `nums2` = [9,7,5]

Output: 3

Explanation:

The integer added to each element of `nums1` is 3.

Example 2:

Input: `nums1` = [10], `nums2` = [5]

Output: -5

Explanation:

The integer added to each element of `nums1` is -5.

Example 3:

Input: `nums1` = [1,1,1,1], `nums2` = [1,1,1,1]

Output: 0**Explanation:**

The integer added to each element of `nums1` is 0.

Constraints:

- $1 \leq \text{nums1.length} == \text{nums2.length} \leq 100$
- $0 \leq \text{nums1}[i], \text{nums2}[i] \leq 1000$
- The test cases are generated in a way that there is an integer x such that `nums1` can become equal to `nums2` by adding x to each element of `nums1`.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int addedInteger(int[] nums1, int[] nums2) {
        int ans = 0;
        Arrays.sort(nums1);
        Arrays.sort(nums2);
        for(int i = 0; i < nums1.length; i++){
            ans =  nums2[i] - nums1[i];
        }
        return ans;
    }
}
```

[2128 Reverse Prefix of Word \(link\)](#)

Description

Given a **0-indexed** string `word` and a character `ch`, **reverse** the segment of `word` that starts at index `0` and ends at the index of the **first occurrence** of `ch` (**inclusive**). If the character `ch` does not exist in `word`, do nothing.

- For example, if `word` = "abcdefd" and `ch` = "d", then you should **reverse** the segment that starts at `0` and ends at `3` (**inclusive**). The resulting string will be "dcbaefd".

Return *the resulting string*.

Example 1:

Input: word = "abcdefd", ch = "d"

Output: "dcbaefd"

Explanation: The first occurrence of "d" is at index 3.

Reverse the part of word from 0 to 3 (inclusive), the resulting string is "dcbaefd".

Example 2:

Input: word = "xyxzxe", ch = "z"

Output: "zxyxxe"

Explanation: The first and only occurrence of "z" is at index 3.

Reverse the part of word from 0 to 3 (inclusive), the resulting string is "zxyxxe".

Example 3:

Input: word = "abcd", ch = "z"

Output: "abcd"

Explanation: "z" does not exist in word.

You should not do any reverse operation, the resulting string is "abcd".

Constraints:

- $1 \leq \text{word.length} \leq 250$
- `word` consists of lowercase English letters.
- `ch` is a lowercase English letter.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String reversePrefix(String word, char ch) {
        int j = word.indexOf(ch);
        if (j != -1) {
            return new StringBuilder(word.substring(0, j + 1)).reverse().toString() + word.substring(j + 1);
        }
        return word;
    }
}
```

[1762 Furthest Building You Can Reach \(link\)](#)

Description

You are given an integer array `heights` representing the heights of buildings, some bricks, and some ladders.

You start your journey from building `0` and move to the next building by possibly using bricks or ladders.

While moving from building `i` to building `i+1` (**0-indexed**),

- If the current building's height is **greater than or equal** to the next building's height, you do **not** need a ladder or bricks.
- If the current building's height is **less than** the next building's height, you can either use **one ladder** or $(h[i+1] - h[i])$ **bricks**.

Return the furthest building index (0-indexed) you can reach if you use the given ladders and bricks optimally.

Example 1:



Input: heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1

Output: 4

Explanation: Starting at building 0, you can follow these steps:

- Go to building 1 without using ladders nor bricks since $4 \geq 2$.
- Go to building 2 using 5 bricks. You must use either bricks or ladders because $2 < 7$.
- Go to building 3 without using ladders nor bricks since $7 \geq 6$.

- Go to building 4 using your only ladder. You must use either bricks or ladders because $6 < 9$. It is impossible to go beyond building 4 because you do not have any more bricks or ladders.

Example 2:

```
Input: heights = [4,12,2,7,3,18,20,3,19], bricks = 10, ladders = 2
Output: 7
```

Example 3:

```
Input: heights = [14,3,19,3], bricks = 17, ladders = 0
Output: 3
```

Constraints:

- $1 \leq \text{heights.length} \leq 10^5$
- $1 \leq \text{heights}[i] \leq 10^6$
- $0 \leq \text{bricks} \leq 10^9$
- $0 \leq \text{ladders} \leq \text{heights.length}$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int furthestBuilding(int[] heights, int bricks, int ladders) {
        PriorityQueue<Integer> p = new PriorityQueue<>((x, y) -> y - x);

        int i = 0, diff = 0;
        for (i = 0; i < heights.length - 1; i++) {
            diff = heights[i + 1] - heights[i];

            if (diff <= 0) {
                continue;
            }

            bricks -= diff;
            p.offer(diff);

            if (bricks < 0) {
                bricks += p.poll();
                ladders--;
            }

            if (ladders < 0) {
                break;
            }
        }

        return i;
    }
}
```

[409 Longest Palindrome \(link\)](#)

Description

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

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

Example 1:

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

Output: 7

Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

Example 2:

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

Output: 1

Explanation: The longest palindrome that can be built is "a", whose length is 1.

Constraints:

- $1 \leq s.\text{length} \leq 2000$
- s consists of lowercase **and/or** uppercase English letters only.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int longestPalindrome(String str) {
        // Create a map to store character frequencies
        Map<Character, Integer> charFreq = new HashMap<>();

        // Count the frequency of each character
        for (char c : str.toCharArray()) {
            charFreq.put(c, charFreq.getOrDefault(c, 0) + 1);
        }

        // Initialize variables to track palindrome length and flag for odd frequency character
        int palindromeLength = 0;
        boolean hasOddFrequency = false;

        // Iterate through the character frequencies
        for (int freq : charFreq.values()) {
            // If frequency is even, add it directly to palindrome length
            // If frequency is odd, subtract 1 and add to palindrome length, set flag for odd frequency character
            if (freq % 2 == 0) {
                palindromeLength += freq;
            } else {
                palindromeLength += freq - 1;
                hasOddFrequency = true;
            }
        }

        // If there is any character with odd frequency, add 1 to the palindrome length
        if (hasOddFrequency) {
            palindromeLength++;
        }

        return palindromeLength;
    }
}
```

[217 Contains Duplicate \(link\)](#)

Description

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

Example 1:

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

Output: `true`

Explanation:

The element 1 occurs at the indices 0 and 3.

Example 2:

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

Output: `false`

Explanation:

All elements are distinct.

Example 3:

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

Output: `true`

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$

- $-10^9 \leq \text{nums}[i] \leq 10^9$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int num : nums) {
            if (map.containsKey(num)) {
                return true;
            }
            map.put(num, 1);
        }
        return false;
    }
}
```

[205 Isomorphic Strings \(link\)](#)

Description

Given two strings s and t , *determine if they are isomorphic*.

Two strings s and t are isomorphic if the characters in s can be replaced to get t .

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Example 1:

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

Output: true

Explanation:

The strings s and t can be made identical by:

- Mapping ' e ' to ' a '.
- Mapping ' g ' to ' d '.

Example 2:

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

Output: false

Explanation:

The strings s and t can not be made identical as ' o ' needs to be mapped to both ' a ' and ' r '.

Example 3:

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

Output: true

Constraints:

- $1 \leq s.length \leq 5 * 10^4$
- $t.length == s.length$
- s and t consist of any valid ascii character.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {string} t  
 * @return {boolean}  
 */  
var isIsomorphic = function(s, t) {  
    if (s.length !== t.length) {  
        return false;  
    }  
  
    let objS = {};  
    let objT = {};  
    for (let i = 0; i < s.length; i++) {  
  
        if (objS[s[i]] !== objT[t[i]]) {  
            return false;  
        }  
  
        objS[s[i]] = i;  
        objT[t[i]] = i;  
  
    }  
    return true;  
};
```

58 Length of Last Word (link)

Description

Given a string s consisting of words and spaces, return *the length of the last word in the string*.

A **word** is a maximal substring consisting of non-space characters only.

Example 1:

Input: $s = \text{"Hello World"}$

Output: 5

Explanation: The last word is "World" with length 5.

Example 2:

Input: $s = \text{" fly me to the moon "}$

Output: 4

Explanation: The last word is "moon" with length 4.

Example 3:

Input: $s = \text{"luffy is still joyboy"}$

Output: 6

Explanation: The last word is "joyboy" with length 6.

Constraints:

- $1 \leq s.length \leq 10^4$
- s consists of only English letters and spaces ' '.
- There will be at least one word in s .

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int lengthOfLastWord(String s) {
        String str = s.trim();
        int count = 0;
        for(int i = str.length()-1; i>=0; i--){
            if(str.charAt(i) == ' '){
                break;
            }
            count++;
        }
        return count;
    }
}
```

[485 Max Consecutive Ones \(link\)](#)

Description

Given a binary array `nums`, return *the maximum number of consecutive 1's in the array*.

Example 1:

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

Output: 3

Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3.

Example 2:

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

Output: 2

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- `nums[i]` is either 0 or 1.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int maxCount = 0;
        int currentCount = 0;

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 1) {
                currentCount++;
                maxCount = Math.max(maxCount, currentCount);
            } else {
                // If the current element is not 1, reset the currentCount by 0 ;
                currentCount = 0;
            }
        }
        return maxCount;
    }
}
```

[20 Valid Parentheses \(link\)](#)

Description

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

An input string is valid if:

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

Example 1:

Input: $s = "()"$

Output: true

Example 2:

Input: $s = "()\[]{}"$

Output: true

Example 3:

Input: $s = "(])"$

Output: false

Example 4:

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

Output: true

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean isValid(String str) {
        Stack <Character> st = new Stack<>();

        for(int i =0; i < str.length(); i++){
            char curr = str.charAt(i);

            if(
                (st.size() > 0 && curr == ')' && st.peek() == '(') ||
                (st.size() > 0 && curr == ']' && st.peek() == '[') ||
                (st.size() > 0 && curr == '}' && st.peek() == '{')
            ){
                st.pop();
            }else{
                st.push(curr);
            }
        }
        return st.size() == 0;
    }
}
```

[442 Find All Duplicates in an Array \(link\)](#)

Description

Given an integer array `nums` of length `n` where all the integers of `nums` are in the range $[1, n]$ and each integer appears **once or twice**, return *an array of all the integers that appears twice*.

You must write an algorithm that runs in $O(n)$ time and uses only *constant* auxiliary space, excluding the space needed to store the output

Example 1:

```
Input: nums = [4,3,2,7,8,2,3,1]
Output: [2,3]
```

Example 2:

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

Example 3:

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

Constraints:

- `n == nums.length`
- $1 \leq n \leq 10^5$
- $1 \leq \text{nums}[i] \leq n$
- Each element in `nums` appears **once or twice**.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public List<Integer> findDuplicates(int[] nums) {
        ArrayList<Integer> arr = new ArrayList<Integer>();
        Arrays.sort(nums);
        for(int i = 1; i < nums.length; i++){
            if(nums[i] == nums[i-1]){
                arr.add( nums[i]);
            }
        }
        return arr;
    }
}
```

713 Subarray Product Less Than K ([link](#))

Description

Given an array of integers `nums` and an integer `k`, return *the number of contiguous subarrays where the product of all the elements in the subarray is strictly less than k*.

Example 1:

```
Input: nums = [10,5,2,6], k = 100
Output: 8
Explanation: The 8 subarrays that have product less than 100 are:
[10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6]
Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.
```

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 3 * 10^4$
- $1 \leq \text{nums}[i] \leq 1000$
- $0 \leq k \leq 10^6$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int numSubarrayProductLessThanK(int[] arr, int k) {
        int n = arr.length;
        int count = 0;
        // Nested loop to generate all subarrays
        for (int i = 0; i < n; i++) {
            int prod = 1;

            for (int j = i; j < n; j++) {
                prod *= arr[j];
                if (prod < k) {
                    count++;
                } else {
                    break; // If product exceeds k, break the inner loop
                }
            }
        }
        return count;
    }
}
```

152 Maximum Product Subarray ([link](#))

Description

Given an integer array `nums`, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

Example 1:

```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

```
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int maxProduct(int[] arr) {
        int n = arr.length;
        int maxSoFar = 1;
        int minSoFar = 1;
        int maxProd = Integer.MIN_VALUE; // default value -infinity

        for (int i = 0; i < n; i++) {
            int curr = arr[i];
            int temp = maxSoFar;
            maxSoFar = Math.max(curr, Math.max(curr * maxSoFar, curr * minSoFar));
            minSoFar = Math.min(curr, Math.min(curr * temp, curr * minSoFar));

            maxProd = Math.max(maxProd, maxSoFar);
        }
        return maxProd;
    }
}
```

[287 Find the Duplicate Number \(link\)](#)

Description

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

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

Follow up:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int findDuplicate(int[] nums) {
        Arrays.sort(nums);
        for(int i = 0; i<nums.length; i++){
            if(nums[i]== nums[i+1]){
                return nums[i];
            }
        }
        return -1;
    }
}
```

961 Long Pressed Name ([link](#))

Description

Your friend is typing his name into a keyboard. Sometimes, when typing a character c , the key might get *long pressed*, and the character will be typed 1 or more times.

You examine the typed characters of the keyboard. Return `True` if it is possible that it was your friends name, with some characters (possibly none) being long pressed.

Example 1:

```
Input: name = "alex", typed = "aaleex"  
Output: true  
Explanation: 'a' and 'e' in 'alex' were long pressed.
```

Example 2:

```
Input: name = "saeed", typed = "ssaaedd"  
Output: false  
Explanation: 'e' must have been pressed twice, but it was not in the typed output.
```

Constraints:

- $1 \leq \text{name.length}, \text{typed.length} \leq 1000$
- `name` and `typed` consist of only lowercase English letters.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean isLongPressedName(String name, String targetTypedName) {
        int i = 0; // Pointer for name
        int j = 0; // Pointer for targetTypedName

        // Iterate through both strings
        while (j < targetTypedName.length()) {
            if (i < name.length() && name.charAt(i) == targetTypedName.charAt(j)) {
                i++;
                j++;
            }
            else if (j > 0 && targetTypedName.charAt(j) == targetTypedName.charAt(j - 1)) {
                j++;
            }
            else {
                return false;
            }
        }
        return i == name.length();
    }
}
```

5 Longest Palindromic Substring ([link](#))

Description

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

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String longestPalindrome(String s) {
        String str = LongestPalindromicSubstring (s);
        return str;
    }

    public static boolean isPalindrome(String str) {
        int i = 0;
        int j = str.length()-1;
        while(i<=j){
            if(str.charAt(i) != str.charAt(j) ){
                return false;
            }
            i++;
            j--;
        }
        return true;
    }

    public static String LongestPalindromicSubstring(String str) {
        int n = str.length();
        // System.out.println("Total number of substrings: " + (n * (n + 1) / 2));
        String [] palindromeSub = new String[n];
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j <= n; j++) {
                String s = str.substring(i, j);

                boolean check = isPalindrome(s);

                // Output the result
                if (check) {
                    palindromeSub[i] = s;
                }
            }
        }
        int maxLength = 0;
```

```
String LongestPalindromicSubs = "";

for(String ele : palindromeSub){
    if(ele.length() > maxLength){
        maxLength = Math.max(maxLength, ele.length());
        LongestPalindromicSubs = ele;
    }
}

return (LongestPalindromicSubs);
}
```

[696 Count Binary Substrings \(link\)](#)

Description

Given a binary string s , return the number of non-empty substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

Example 1:

Input: $s = "00110011"$

Output: 6

Explanation: There are 6 substrings that have equal number of consecutive 1's and 0's: "0011", "01", "1100", "10", "0011", and "01"

Notice that some of these substrings repeat and are counted the number of times they occur.

Also, "00110011" is not a valid substring because all the 0's (and 1's) are not grouped together.

Example 2:

Input: $s = "10101"$

Output: 4

Explanation: There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

Constraints:

- $1 \leq s.length \leq 10^5$
- $s[i]$ is either '0' or '1'.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int countBinarySubstrings(String str) {
        int n = str.length();
        int i = 0;
        int ans = 0;

        while (i < n) {
            int countZero = 0;
            int countOne = 0;

            if (str.charAt(i) == '0') {
                while (i < n && str.charAt(i) == '0') {
                    countZero++;
                    i++;
                }
                int j = i;
                while (j < n && str.charAt(j) == '1') {
                    countOne++;
                    j++;
                }
            } else {
                while (i < n && str.charAt(i) == '1') {
                    countOne++;
                    i++;
                }
                int j = i;
                while (j < n && str.charAt(j) == '0') {
                    countZero++;
                    j++;
                }
            }
            ans += Math.min(countZero, countOne);
        }
        return ans;
    }
}
```

```
}
```

[125 Valid Palindrome \(link\)](#)

Description

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

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

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

Example 3:

```
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters.
Since an empty string reads the same forward and backward, it is a palindrome.
```

Constraints:

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public boolean isPalindrome(String s) {
        int i = 0;
        int j = s.length()-1;
        while(i <= j){
            if(Character.isLetterOrDigit(s.charAt(i)) && Character.isLetterOrDigit(s.charAt(j))){
                if(Character.toLowerCase(s.charAt(i)) != Character.toLowerCase(s.charAt(j))){
                    return false;
                }else{
                    i++;
                    j--;
                }
            }else if(!Character.isLetterOrDigit(s.charAt(i))){
                i++;
            }else if(!Character.isLetterOrDigit(s.charAt(j))){
                j--;
            }
        }
        return true;
    }
}
```

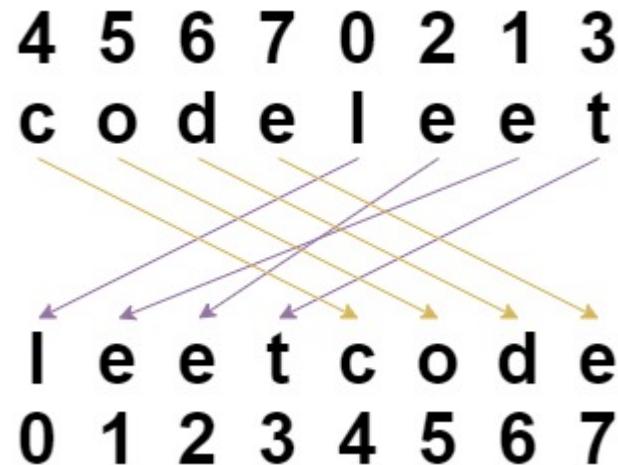
[1651 Shuffle String \(link\)](#)

Description

You are given a string s and an integer array indices of the **same length**. The string s will be shuffled such that the character at the i^{th} position moves to $\text{indices}[i]$ in the shuffled string.

Return *the shuffled string*.

Example 1:



Input: $s = \text{"codeleet"}$, $\text{indices} = [4,5,6,7,0,2,1,3]$

Output: "leetcode"

Explanation: As shown, "codeleet" becomes "leetcode" after shuffling.

Example 2:

Input: $s = \text{"abc"}$, $\text{indices} = [0,1,2]$

Output: "abc"

Explanation: After shuffling, each character remains in its position.

Constraints:

- `s.length == indices.length == n`
- `1 <= n <= 100`
- `s` consists of only lowercase English letters.
- `0 <= indices[i] < n`
- All values of `indices` are **unique**.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public String restoreString(String s, int[] indices) {
        int len = s.length();
        String[] strArr = s.split("");
        String[] shuffleStr = new String[len];
        for(int i=0; i<s.length(); i++){
            shuffleStr[indices[i]] = strArr[i];
        }
        String ansShuffle = "";
        for(String ele : shuffleStr){
            if(ele != null) ansShuffle += ele;
        }
        return ansShuffle;
    }
}
```

[2571 Find the Pivot Integer \(link\)](#)

Description

Given a positive integer n , find the **pivot integer** x such that:

- The sum of all elements between 1 and x inclusively equals the sum of all elements between x and n inclusively.

Return *the pivot integer* x . If no such integer exists, return -1 . It is guaranteed that there will be at most one pivot index for the given input.

Example 1:

Input: $n = 8$

Output: 6

Explanation: 6 is the pivot integer since: $1 + 2 + 3 + 4 + 5 + 6 = 6 + 7 + 8 = 21$.

Example 2:

Input: $n = 1$

Output: 1

Explanation: 1 is the pivot integer since: $1 = 1$.

Example 3:

Input: $n = 4$

Output: -1

Explanation: It can be proved that no such integer exist.

Constraints:

- $1 \leq n \leq 1000$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int pivotInteger(int n) {
        if(n==1) return n;
        int totalSum = n * (n + 1) / 2;
        int leftSum = 0;
        int rightSum = totalSum - 1; // Adjusting rightSum to exclude n itself
        int pivot = -1;

        for (int x = 1; x <= n; x++) {
            leftSum += x;
            rightSum -= (x + 1); // Adjusting rightSum by adding x + 1 to account for the exclusion of n

            if (leftSum == rightSum) {
                pivot = x + 1; // Adding 1 to account for the index starting from 1
                break;
            }
        }

        return pivot;
    }
}
```

[151 Reverse Words in a String \(link\)](#)

Description

Given an input string s , reverse the order of the **words**.

A **word** is defined as a sequence of non-space characters. The **words** in s will be separated by at least one space.

Return a *string of the words in reverse order concatenated by a single space*.

Note that s may contain leading or trailing spaces or multiple spaces between two words. The returned string should only have a single space separating the words. Do not include any extra spaces.

Example 1:

```
Input: s = "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: s = " hello world "
Output: "world hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: s = "a good example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string.
```

Constraints:

- $1 \leq s.length \leq 10^4$
- s contains English letters (upper-case and lower-case), digits, and spaces ' '.
- There is **at least one** word in s .

Follow-up: If the string data type is mutable in your language, can you solve it **in-place** with $O(1)$ extra space?

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {  
    public String reverseWords(String s) {  
        String[] strArr = s.trim().split(" ");  
        String str = "";  
        for(int i = strArr.length-1; i >= 0; i--){  
            if(strArr[i] != ""){  
                str += strArr[i] + " ";  
            }  
        }  
        return str.trim();  
    }  
}
```

[2634 Minimum Common Value \(link\)](#)

Description

Given two integer arrays `nums1` and `nums2`, sorted in non-decreasing order, return *the minimum integer common* to both arrays. If there is no common integer amongst `nums1` and `nums2`, return -1.

Note that an integer is said to be **common** to `nums1` and `nums2` if both arrays have **at least one** occurrence of that integer.

Example 1:

Input: `nums1 = [1,2,3]`, `nums2 = [2,4]`

Output: 2

Explanation: The smallest element common to both arrays is 2, so we return 2.

Example 2:

Input: `nums1 = [1,2,3,6]`, `nums2 = [2,3,4,5]`

Output: 2

Explanation: There are two common elements in the array 2 and 3 out of which 2 is the smallest, so 2 is returned.

Constraints:

- $1 \leq \text{nums1.length}, \text{nums2.length} \leq 10^5$
- $1 \leq \text{nums1}[i], \text{nums2}[j] \leq 10^9$
- Both `nums1` and `nums2` are sorted in **non-decreasing** order.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int getCommon(int[] nums1, int[] nums2) {
        int pointer1 = 0, pointer2 = 0;

        while (pointer1 < nums1.length && pointer2 < nums2.length) {
            if (nums1[pointer1] == nums2[pointer2]) {
                return nums1[pointer1];
            } else if (nums1[pointer1] < nums2[pointer2]) {
                pointer1++;
            } else {
                pointer2++;
            }
        }

        return -1; // If no common element found
    }
}
```

3227 Find Missing and Repeated Values ([link](#))

Description

You are given a **0-indexed** 2D integer matrix `grid` of size $n * n$ with values in the range $[1, n^2]$. Each integer appears **exactly once** except a which appears **twice** and b which is **missing**. The task is to find the repeating and missing numbers a and b .

Return a **0-indexed** integer array `ans` of size 2 where `ans[0]` equals to a and `ans[1]` equals to b .

Example 1:

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

Output: [2,4]

Explanation: Number 2 is repeated and number 4 is missing so the answer is [2,4].

Example 2:

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

Output: [9,5]

Explanation: Number 9 is repeated and number 5 is missing so the answer is [9,5].

Constraints:

- $2 \leq n == \text{grid.length} == \text{grid[i].length} \leq 50$
- $1 \leq \text{grid}[i][j] \leq n * n$
- For all x that $1 \leq x \leq n * n$ there is exactly one x that is not equal to any of the grid members.
- For all x that $1 \leq x \leq n * n$ there is exactly one x that is equal to exactly two of the grid members.
- For all x that $1 \leq x \leq n * n$ except two of them there is exatly one pair of i, j that $0 \leq i, j \leq n - 1$ and $\text{grid}[i][j] == x$.

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int[] findMissingAndRepeatedValues(int[][] grid) {
        int n = grid.length;
        int[] count = new int[n * n + 1];

        int repeating = -1;
        int missing = -1;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                int num = grid[i][j];
                count[num]++;
            }
        }

        for (int i = 1; i <= n * n; i++) {
            if (count[i] == 2) {
                repeating = i;
            } else if (count[i] == 0) {
                missing = i;
            }
        }

        return new int[]{repeating, missing};
    }
}
```

[169 Majority Element \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

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

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

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int majorityElement(int[] arr) {
        // Sort the array
        Arrays.sort(arr);
        int n = arr.length;

        // Iterate through the sorted array and count occurrences of each element

        for (int i = 0; i < n;) {
            int j = i;
            while (j < n && arr[i] == arr[j]) {
                j++;
            }

            int count = j - i;
            if (count > n / 2) {
                return (arr[i]);
            }
            i = j;
        }
        return -1;
    }
}
```

628 Maximum Product of Three Numbers ([link](#))

Description

Given an integer array `nums`, find three numbers whose product is maximum and return the maximum product.

Example 1:

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

Example 2:

```
Input: nums = [1,2,3,4]
Output: 24
```

Example 3:

```
Input: nums = [-1,-2,-3]
Output: -6
```

Constraints:

- $3 \leq \text{nums.length} \leq 10^4$
- $-1000 \leq \text{nums}[i] \leq 1000$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {  
    public int maximumProduct(int[] nums) {  
        Arrays.sort(nums);  
        int n = nums.length;  
        // Maximum product can be either the product of the three largest positive integers  
        // or the product of the two smallest negative integers and the largest positive integer.  
        return Math.max(nums[n-1] * nums[n-2] * nums[n-3], nums[0] * nums[1] * nums[n-1]);  
    }  
}
```

1406 Subtract the Product and Sum of Digits of an Integer ([link](#))

Description

Given an integer number n , return the difference between the product of its digits and the sum of its digits.

Example 1:

```
Input: n = 234
Output: 15
Explanation:
Product of digits = 2 * 3 * 4 = 24
Sum of digits = 2 + 3 + 4 = 9
Result = 24 - 9 = 15
```

Example 2:

```
Input: n = 4421
Output: 21
Explanation:
Product of digits = 4 * 4 * 2 * 1 = 32
Sum of digits = 4 + 4 + 2 + 1 = 11
Result = 32 - 11 = 21
```

Constraints:

- $1 \leq n \leq 10^5$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int subtractProductAndSum(int n) {
        int productOfDigit = 1;
        int sumOfDigit = 0;

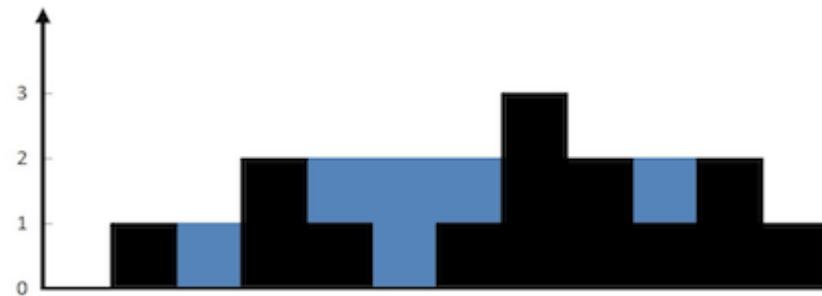
        while(n > 0){
            int digit = n%10;
            System.out.println(digit+" ");
            productOfDigit *= digit;
            sumOfDigit += digit;
            n = n/10;
        }
        int ans = productOfDigit - sumOfDigit;
        // System.out.println(productOfDigit +" "+sumOfDigit);
        return ans;
    }
}
```

42 Trapping Rain Water (link)

Description

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

Example 1:



Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

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

Example 2:

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

Output: 9

Constraints:

- $n == \text{height.length}$

- $1 \leq n \leq 2 * 10^4$
- $0 \leq \text{height}[i] \leq 10^5$

(scroll down for solution)

Solution

Language: java

Status: Accepted

```
class Solution {
    public int trap(int[] height) {
        if (height == null || height.length <= 2) {
            return 0;
        }

        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0;
        int result = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                leftMax = Math.max(leftMax, height[left]);
                result += leftMax - height[left];
                left++;
            } else {
                rightMax = Math.max(rightMax, height[right]);
                result += rightMax - height[right];
                right--;
            }
        }

        return result;
    }
}
```

[78 Subsets \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

- $1 \leq \text{nums.length} \leq 10$
- $-10 \leq \text{nums}[i] \leq 10$
- All the numbers of `nums` are **unique**.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[][]}  
 */  
var subsets = function(nums) {  
    let result = [];  
    result.push([]);  
  
    for(let i =0; i<nums.length; i++){  
        let current = nums[i];  
  
        let len = result.length;  
        for(let j =0; j<len; j++){  
            let set1 = result[j].slice(0);  
            set1.push(current);  
            result.push(set1);  
        }  
    }  
  
    return result;  
};
```

[3172 Divisible and Non-divisible Sums Difference \(link\)](#)

Description

You are given positive integers n and m .

Define two integers as follows:

- num1 : The sum of all integers in the range $[1, n]$ (both **inclusive**) that are **not divisible** by m .
- num2 : The sum of all integers in the range $[1, n]$ (both **inclusive**) that are **divisible** by m .

Return *the integer* $\text{num1} - \text{num2}$.

Example 1:

Input: $n = 10$, $m = 3$

Output: 19

Explanation: In the given example:

- Integers in the range $[1, 10]$ that are not divisible by 3 are $[1, 2, 4, 5, 7, 8, 10]$, num1 is the sum of those integers = 37.
- Integers in the range $[1, 10]$ that are divisible by 3 are $[3, 6, 9]$, num2 is the sum of those integers = 18.

We return $37 - 18 = 19$ as the answer.

Example 2:

Input: $n = 5$, $m = 6$

Output: 15

Explanation: In the given example:

- Integers in the range $[1, 5]$ that are not divisible by 6 are $[1, 2, 3, 4, 5]$, num1 is the sum of those integers = 15.
- Integers in the range $[1, 5]$ that are divisible by 6 are $[]$, num2 is the sum of those integers = 0.

We return $15 - 0 = 15$ as the answer.

Example 3:

Input: $n = 5$, $m = 1$

Output: -15

Explanation: In the given example:

- Integers in the range [1, 5] that are not divisible by 1 are [], num1 is the sum of those integers = 0.
- Integers in the range [1, 5] that are divisible by 1 are [1,2,3,4,5], num2 is the sum of those integers = 15.
We return 0 - 15 = -15 as the answer.

Constraints:

- $1 \leq n, m \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @param {number} m  
 * @return {number}  
 */  
var differenceOfSums = function(n, m) {  
    let num1 = 0;  
    let num2 = 0;  
    for (let i = 1; i <= n; i++) {  
        if (i % m === 0) {  
            num1 = num1 + i;  
        }  
        if (i % m != 0) {  
            num2 = num2 + i;  
        }  
    }  
    return num2 - num1;  
};
```

2999 Check if Strings Can be Made Equal With Operations I (link)

Description

You are given two strings s_1 and s_2 , both of length 4, consisting of **lowercase** English letters.

You can apply the following operation on any of the two strings **any** number of times:

- Choose any two indices i and j such that $j - i = 2$, then **swap** the two characters at those indices in the string.

Return `true` if you can make the strings s_1 and s_2 equal, and `false` otherwise.

Example 1:

Input: $s_1 = "abcd"$, $s_2 = "cdab"$

Output: true

Explanation: We can do the following operations on s_1 :

- Choose the indices $i = 0$, $j = 2$. The resulting string is $s_1 = "cbad"$.
- Choose the indices $i = 1$, $j = 3$. The resulting string is $s_1 = "cdab" = s_2$.

Example 2:

Input: $s_1 = "abcd"$, $s_2 = "dacb"$

Output: false

Explanation: It is not possible to make the two strings equal.

Constraints:

- $s_1.length == s_2.length == 4$
- s_1 and s_2 consist only of lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s1  
 * @param {string} s2  
 * @return {boolean}  
 */  
var canBeEqual = function(s1, s2) {  
    return ((s1[0] == s2[0] && s1[2] == s2[2]) ||  
            (s1[0] == s2[2] && s1[2] == s2[0])) &&  
        ((s1[1] == s2[1] && s1[3] == s2[3]) ||  
            (s1[1] == s2[3] && s1[3] == s2[1]));  
};
```

2876 Number of Employees Who Met the Target ([link](#))

Description

There are n employees in a company, numbered from 0 to $n - 1$. Each employee i has worked for $\text{hours}[i]$ hours in the company.

The company requires each employee to work for **at least** target hours.

You are given a **0-indexed** array of non-negative integers hours of length n and a non-negative integer target .

Return *the integer denoting the number of employees who worked at least target hours*.

Example 1:

```
Input: hours = [0,1,2,3,4], target = 2
Output: 3
Explanation: The company wants each employee to work for at least 2 hours.
- Employee 0 worked for 0 hours and didn't meet the target.
- Employee 1 worked for 1 hours and didn't meet the target.
- Employee 2 worked for 2 hours and met the target.
- Employee 3 worked for 3 hours and met the target.
- Employee 4 worked for 4 hours and met the target.
There are 3 employees who met the target.
```

Example 2:

```
Input: hours = [5,1,4,2,2], target = 6
Output: 0
Explanation: The company wants each employee to work for at least 6 hours.
There are 0 employees who met the target.
```

Constraints:

- $1 \leq n == \text{hours.length} \leq 50$

- $0 \leq \text{hours}[i], \text{target} \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} hours  
 * @param {number} target  
 * @return {number}  
 */  
var numberOfWorkersWhoMetTarget = function(hours, target) {  
    let count = 0;  
    for (let i = 0; i < hours.length; i++) {  
        if (hours[i] >= target) {  
            count++;  
        }  
    }  
    return count;  
//    console.log("first :", count);  
};
```

3045 Minimum Right Shifts to Sort the Array ([link](#))

Description

You are given a **0-indexed** array `nums` of length `n` containing **distinct** positive integers. Return *the minimum number of right shifts required to sort* `nums` *and -1 if this is not possible.*

A **right shift** is defined as shifting the element at index `i` to index `(i + 1) % n`, for all indices.

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: 2
Explanation:
After the first right shift, nums = [2,3,4,5,1].
After the second right shift, nums = [1,2,3,4,5].
Now nums is sorted; therefore the answer is 2.
```

Example 2:

```
Input: nums = [1,3,5]
Output: 0
Explanation: nums is already sorted therefore, the answer is 0.
```

Example 3:

```
Input: nums = [2,1,4]
Output: -1
Explanation: It's impossible to sort the array using right shifts.
```

Constraints:

- `1 <= nums.length <= 100`

- $1 \leq \text{nums}[i] \leq 100$
- `nums` contains distinct integers.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var minimumRightShifts = function (nums) {  
    let sorted = [...nums].sort((a, b) => a - b).join("");  
    let maxTry = nums.length;  
    let count = 0;  
    while (maxTry >= 0) {  
        console.log(sorted, nums);  
        if (sorted == nums.join("")) return count;  
        nums.unshift(nums.pop());  
        count++;  
        maxTry--;  
    }  
    return -1;  
};  
// minimumRightShifts([3, 4, 5, 1, 2]);
```

941 Sort Array By Parity (link)

Description

Given an integer array `nums`, move all the even integers at the beginning of the array followed by all the odd integers.

Return *any array* that satisfies this condition.

Example 1:

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

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

Explanation: The outputs `[4,2,3,1]`, `[2,4,1,3]`, and `[4,2,1,3]` would also be accepted.

Example 2:

Input: `nums = [0]`

Output: `[0]`

Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $0 \leq \text{nums}[i] \leq 5000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var sortArrayByParity = function(nums) {  
    let output = [];  
    for (let i = 0; i < nums.length; i++) {  
        if (nums[i] % 2 === 0) {  
            output.push(nums[i]);  
        }  
    }  
    for (let i = 0; i < nums.length; i++) {  
        if (nums[i] % 2 !== 0) {  
            output.push(nums[i]);  
        }  
    }  
    return output  
//    console.log("output :", output);  
};
```

[874 Backspace String Compare \(link\)](#)

Description

Given two strings s and t , return `true` if they are equal when both are typed into empty text editors. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

Example 1:

```
Input: s = "ab#c", t = "ad#c"
Output: true
Explanation: Both s and t become "ac".
```

Example 2:

```
Input: s = "ab##", t = "c#d#"
Output: true
Explanation: Both s and t become "".
```

Example 3:

```
Input: s = "a#c", t = "b"
Output: false
Explanation: s becomes "c" while t becomes "b".
```

Constraints:

- $1 \leq s.length, t.length \leq 200$
- s and t only contain lowercase letters and '#' characters.

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {string} t  
 * @return {boolean}  
 */  
var backspaceCompare = function(s, t) {  
    let sArr = [];  
    for (let char of s) {  
        if (char === "#") {  
            sArr.pop();  
        } else {  
            sArr.push(char);  
        }  
    }  
    let s1 = sArr.join("");  
  
    let tArr = [];  
    for (let char of t) {  
        if (char === "#") {  
            tArr.pop();  
        } else {  
            tArr.push(char);  
        }  
    }  
    let t1 = tArr.join("");  
  
    return s1 === t1;  
    // console.log("result :", s1 === t1, s1, t1);  
};
```

2783 Nested Array Generator ([link](#))

Description

Given a **multi-dimensional array** of integers, return a generator object which yields integers in the same order as **inorder traversal**.

A **multi-dimensional array** is a recursive data structure that contains both integers and other **multi-dimensional arrays**.

inorder traversal iterates over each array from left to right, yielding any integers it encounters or applying **inorder traversal** to any arrays it encounters.

Example 1:

```
Input: arr = [[[6]],[1,3],[]]
Output: [6,1,3]
Explanation:
const generator = inorderTraversal(arr);
generator.next().value; // 6
generator.next().value; // 1
generator.next().value; // 3
generator.next().done; // true
```

Example 2:

```
Input: arr = []
Output: []
Explanation: There are no integers so the generator doesn't yield anything.
```

Constraints:

- $0 \leq \text{arr.flat().length} \leq 10^5$
- $0 \leq \text{arr.flat()[i]} \leq 10^5$
- $\text{maxNestingDepth} \leq 10^5$

Can you solve this without creating a new flattened version of the array?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {Array} arr
 * @return {Generator}
 */
var inorderTraversal = function*(arr) {

    let flatArr = arr.flat(Infinity);
    flatArr.reverse()
    while(flatArr.length) {
        yield flatArr.pop()
    }

};

/*
 * const gen = inorderTraversal([1, [2, 3]]);
 * gen.next().value; // 1
 * gen.next().value; // 2
 * gen.next().value; // 3
*/

```

2470 Removing Stars From a String ([link](#))

Description

You are given a string s , which contains stars $*$.

In one operation, you can:

- Choose a star in s .
- Remove the closest **non-star** character to its **left**, as well as remove the star itself.

Return *the string after all stars have been removed*.

Note:

- The input will be generated such that the operation is always possible.
- It can be shown that the resulting string will always be unique.

Example 1:

Input: $s = \text{"leet}**\text{cod}*e"$

Output: "lecoe"

Explanation: Performing the removals from left to right:

- The closest character to the 1st star is 't' in "leet**cod*e". s becomes "lee*cod*e".
 - The closest character to the 2nd star is 'e' in "lee*cod*e". s becomes "lecod*e".
 - The closest character to the 3rd star is 'd' in "lecod*e". s becomes "lecoe".
- There are no more stars, so we return "lecoe".

Example 2:

Input: $s = \text{"erase}*****"$

Output: ""

Explanation: The entire string is removed, so we return an empty string.

Constraints:

- $1 \leq s.length \leq 10^5$
- s consists of lowercase English letters and stars *.
- The operation above can be performed on s .

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
const removeStars = ( s, r = [] ) =>  
    s.split``.forEach( char => char == `*` ? r.pop() : r.push(char) ) || r.join``  
  
// var removeStars = function (s) {  
//     let strArr = [];  
//     for (const char of s) {  
//         if (char == "*") strArr.pop();  
//         else strArr.push(char);  
//     }  
//     // return strArr.join("");  
//     console.log(strArr.join("") );  
// };
```

[2775 Generate Fibonacci Sequence \(link\)](#)

Description

Write a generator function that returns a generator object which yields the **fibonacci sequence**.

The **fibonacci sequence** is defined by the relation $x_n = x_{n-1} + x_{n-2}$.

The first few numbers of the series are 0, 1, 1, 2, 3, 5, 8, 13.

Example 1:

```
Input: callCount = 5
Output: [0,1,1,2,3]
Explanation:
const gen = fibGenerator();
gen.next().value; // 0
gen.next().value; // 1
gen.next().value; // 1
gen.next().value; // 2
gen.next().value; // 3
```

Example 2:

```
Input: callCount = 0
Output: []
Explanation: gen.next() is never called so nothing is outputted
```

Constraints:

- $0 \leq \text{callCount} \leq 50$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @return {Generator<number>}  
 */  
  
var fibGenerator = function* (callCount) {  
    let current = 0;  
    let next = 1;  
  
    while (true) {  
        yield current;  
  
        [current, next] = [next, current + next];  
    }  
};  
  
/**  
 * const gen = fibGenerator();  
 * gen.next().value; // 0  
 * gen.next().value; // 1  
 */
```

[1127 Last Stone Weight \(link\)](#)

Description

You are given an array of integers `stones` where `stones[i]` is the weight of the i^{th} stone.

We are playing a game with the stones. On each turn, we choose the **heaviest two stones** and smash them together. Suppose the heaviest two stones have weights x and y with $x \leq y$. The result of this smash is:

- If $x == y$, both stones are destroyed, and
- If $x != y$, the stone of weight x is destroyed, and the stone of weight y has new weight $y - x$.

At the end of the game, there is **at most one** stone left.

Return *the weight of the last remaining stone*. If there are no stones left, return 0.

Example 1:

Input: stones = [2,7,4,1,8,1]

Output: 1

Explanation:

We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then,
we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then,
we combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone.

Example 2:

Input: stones = [1]

Output: 1

Constraints:

- `1 <= stones.length <= 30`
- `1 <= stones[i] <= 1000`

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} stones  
 * @return {number}  
 */  
var lastStoneWeight = function (stones) {  
  
    if (stones.length < 2) return stones;  
  
    stones.sort((a, b) => a - b);  
  
    let removeLast1 = stones.pop();  
    let removeLast2 = stones.pop();  
    stones.push(Math.abs(removeLast1 - removeLast2));  
  
    return lastStoneWeight(stones);  
};
```

2917 Count Pairs Whose Sum is Less than Target ([link](#))

Description

Given a **0-indexed** integer array `nums` of length `n` and an integer `target`, return *the number of pairs* (i, j) where $0 \leq i < j < n$ and $\text{nums}[i] + \text{nums}[j] < \text{target}$.

Example 1:

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

Output: 3

Explanation: There are 3 pairs of indices that satisfy the conditions in the statement:

- $(0, 1)$ since $0 < 1$ and $\text{nums}[0] + \text{nums}[1] = 0 < \text{target}$
- $(0, 2)$ since $0 < 2$ and $\text{nums}[0] + \text{nums}[2] = 1 < \text{target}$
- $(0, 4)$ since $0 < 4$ and $\text{nums}[0] + \text{nums}[4] = 0 < \text{target}$

Note that $(0, 3)$ is not counted since $\text{nums}[0] + \text{nums}[3]$ is not strictly less than the target.

Example 2:

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

Output: 10

Explanation: There are 10 pairs of indices that satisfy the conditions in the statement:

- $(0, 1)$ since $0 < 1$ and $\text{nums}[0] + \text{nums}[1] = -4 < \text{target}$
- $(0, 3)$ since $0 < 3$ and $\text{nums}[0] + \text{nums}[3] = -8 < \text{target}$
- $(0, 4)$ since $0 < 4$ and $\text{nums}[0] + \text{nums}[4] = -13 < \text{target}$
- $(0, 5)$ since $0 < 5$ and $\text{nums}[0] + \text{nums}[5] = -7 < \text{target}$
- $(0, 6)$ since $0 < 6$ and $\text{nums}[0] + \text{nums}[6] = -3 < \text{target}$
- $(1, 4)$ since $1 < 4$ and $\text{nums}[1] + \text{nums}[4] = -5 < \text{target}$
- $(3, 4)$ since $3 < 4$ and $\text{nums}[3] + \text{nums}[4] = -9 < \text{target}$
- $(3, 5)$ since $3 < 5$ and $\text{nums}[3] + \text{nums}[5] = -3 < \text{target}$
- $(4, 5)$ since $4 < 5$ and $\text{nums}[4] + \text{nums}[5] = -8 < \text{target}$
- $(4, 6)$ since $4 < 6$ and $\text{nums}[4] + \text{nums}[6] = -4 < \text{target}$

Constraints:

- $1 \leq \text{nums.length} == n \leq 50$
- $-50 \leq \text{nums}[i], \text{target} \leq 50$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number}  
 */  
var countPairs = function(nums, target) {  
    let countOutput = 0;  
    for (let i = 0; i < nums.length; i++) {  
        for (let j = i + 1; j < nums.length; j++) {  
            if (nums[i] + nums[j] < target && j > i) {  
                countOutput++;  
            }  
        }  
    }  
    return countOutput  
//    console.log("output :", countOutput);  
};
```

2977 Check if a String Is an Acronym of Words ([link](#))

Description

Given an array of strings `words` and a string `s`, determine if `s` is an **acronym** of words.

The string `s` is considered an acronym of `words` if it can be formed by concatenating the **first** character of each string in `words` **in order**. For example, "ab" can be formed from ["apple", "banana"], but it can't be formed from ["bear", "aardvark"].

Return `true` if `s` is an acronym of `words`, and `false` otherwise.

Example 1:

```
Input: words = ["alice","bob","charlie"], s = "abc"
```

```
Output: true
```

```
Explanation: The first character in the words "alice", "bob", and "charlie" are 'a', 'b', and 'c', respectively. Hence, s = "abc" is
```

Example 2:

```
Input: words = ["an","apple"], s = "a"
```

```
Output: false
```

```
Explanation: The first character in the words "an" and "apple" are 'a' and 'a', respectively.
```

```
The acronym formed by concatenating these characters is "aa".
```

```
Hence, s = "a" is not the acronym.
```

Example 3:

```
Input: words = ["never","gonna","give","up","on","you"], s = "ngguoy"
```

```
Output: true
```

```
Explanation: By concatenating the first character of the words in the array, we get the string "ngguoy".
```

```
Hence, s = "ngguoy" is the acronym.
```

Constraints:

- $1 \leq \text{words.length} \leq 100$
- $1 \leq \text{words}[i].length \leq 10$
- $1 \leq s.length \leq 100$
- $\text{words}[i]$ and s consist of lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string[]} words  
 * @param {string} s  
 * @return {boolean}  
 */  
var isAcronym = function(words, s) {  
    let firstWord = "";  
    for (let i = 0; i < words.length; i++) {  
        firstWord += words[i][0];  
    }  
    return s === firstWord;  
    // console.log("first :", firstWord);  
};
```

[953 Reverse Only Letters \(\[link\]\(#\)\)](#)

Description

Given a string s , reverse the string according to the following rules:

- All the characters that are not English letters remain in the same position.
- All the English letters (lowercase or uppercase) should be reversed.

Return s *after reversing it*.

Example 1:

```
Input: s = "ab-cd"
Output: "dc-ba"
```

Example 2:

```
Input: s = "a-bC-dEf-ghIj"
Output: "j-Ih-gfE-dCba"
```

Example 3:

```
Input: s = "Test1ng-Leet=code-Q!"
Output: "Qedo1ct-eeLg=ntse-T!"
```

Constraints:

- $1 \leq s.length \leq 100$
- s consists of characters with ASCII values in the range [33, 122].
- s does not contain '\"' or '\\'.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
  
var reverseOnlyLetters = function(s) {  
    let arr = s.split('');  
    let i = 0;  
    let j = s.length-1;  
    while(i < j){  
        if(!arr[i].match(/[a-zA-Z]/i)){  
            i++  
        }  
        else if(!arr[j].match(/[a-zA-Z]/i)){  
            j--  
        }  
        else{  
            [arr[i], arr[j]] = [arr[j], arr[i]]  
            i++  
            j--  
        }  
    }  
    return arr.join('')  
};
```

345 Reverse Vowels of a String ([link](#))

Description

Given a string s , reverse only all the vowels in the string and return it.

The vowels are '`a`', '`e`', '`i`', '`o`', and '`u`', and they can appear in both lower and upper cases, more than once.

Example 1:

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

Output: "AceCreIm"

Explanation:

The vowels in s are $[\text{'I}', \text{'e'}, \text{'e'}, \text{'A'}]$. On reversing the vowels, s becomes "AceCreIm" .

Example 2:

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

Output: "leotcede"

Constraints:

- $1 \leq s.length \leq 3 * 10^5$
- s consist of **printable ASCII** characters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {string}
 */
var reverseVowels = function (s) {
    let vowelArr = [];
    let sArr = s.split("");

    for (let i = sArr.length - 1; i >= 0; i--) {
        if (
            sArr[i] === "a" ||
            sArr[i] === "e" ||
            sArr[i] === "i" ||
            sArr[i] === "o" ||
            sArr[i] === "u" ||
            sArr[i] === "A" ||
            sArr[i] === "E" ||
            sArr[i] === "I" ||
            sArr[i] === "O" ||
            sArr[i] === "U"
        ) {
            vowelArr.push(sArr[i]);
        }
    }
    let output = [];
    let count = 0;
    for (let i = 0; i < sArr.length; i++) {
        if (
            sArr[i] === "a" ||
            sArr[i] === "e" ||
            sArr[i] === "i" ||
            sArr[i] === "o" ||
            sArr[i] === "u" ||
            sArr[i] === "A" ||
            sArr[i] === "E" ||
            sArr[i] === "I" ||
            sArr[i] === "O" ||
            sArr[i] === "U"
        )
```

```
    ) {
        output.push(vowelArr[count]);
        count++;
        continue;
    }
    output.push(sArr[i]);
}
return output.join("");
// console.log("output :", output.join(""));
};
```

[2886 Faulty Keyboard \(link\)](#)

Description

Your laptop keyboard is faulty, and whenever you type a character '`i`' on it, it reverses the string that you have written. Typing other characters works as expected.

You are given a **0-indexed** string `s`, and you type each character of `s` using your faulty keyboard.

Return *the final string that will be present on your laptop screen*.

Example 1:

```
Input: s = "string"
Output: "rtsng"
Explanation:
After typing first character, the text on the screen is "s".
After the second character, the text is "st".
After the third character, the text is "str".
Since the fourth character is an 'i', the text gets reversed and becomes "rts".
After the fifth character, the text is "rtsn".
After the sixth character, the text is "rtsng".
Therefore, we return "rtsng".
```

Example 2:

```
Input: s = "poionter"
Output: "ponter"
Explanation:
After the first character, the text on the screen is "p".
After the second character, the text is "po".
Since the third character you type is an 'i', the text gets reversed and becomes "op".
Since the fourth character you type is an 'i', the text gets reversed and becomes "po".
After the fifth character, the text is "pon".
After the sixth character, the text is "pont".
After the seventh character, the text is "ponte".
```

After the eighth character, the text is "ponter".
Therefore, we return "ponter".

Constraints:

- $1 \leq s.length \leq 100$
- s consists of lowercase English letters.
- $s[0] \neq 'i'$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {string}  
 */  
  
var finalString = function (s) {  
  
    let sArr = s.split("");  
    let output = [];  
  
    for (let i = 0; i < sArr.length; i++) {  
        if (sArr[i] === "i") {  
            output.reverse();  
            continue;  
        }  
        output.push(sArr[i]);  
    }  
    return output.join("")  
};
```

[412 Fizz Buzz \(link\)](#)

Description

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

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

Example 1:

```
Input: n = 3
Output: ["1", "2", "Fizz"]
```

Example 2:

```
Input: n = 5
Output: ["1", "2", "Fizz", "4", "Buzz"]
```

Example 3:

```
Input: n = 15
Output: ["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz"]
```

Constraints:

- $1 \leq n \leq 10^4$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number} n
 * @return {string[]}
 */
var fizzBuzz = function(n) {
    let arr = []
    for(let i = 1; i <= n; i++){
        if(i % 3 === 0 && i % 5 === 0){
            arr.push("FizzBuzz")
        }else if(i % 5 ===0 ){
            arr.push("Buzz")
        }else if(i % 3 === 0){
            arr.push("Fizz")
        }else{
            arr.push(i.toString())
        }
    }
    return arr
    // console.log(arr)
};

};
```

[2551 Apply Operations to an Array \(link\)](#)

Description

You are given a **0-indexed** array `nums` of size `n` consisting of **non-negative** integers.

You need to apply $n - 1$ operations to this array where, in the i^{th} operation (**0-indexed**), you will apply the following on the i^{th} element of `nums`:

- If $\text{nums}[i] == \text{nums}[i + 1]$, then multiply $\text{nums}[i]$ by 2 and set $\text{nums}[i + 1]$ to 0. Otherwise, you skip this operation.

After performing **all** the operations, **shift** all the 0's to the **end** of the array.

- For example, the array `[1,0,2,0,0,1]` after shifting all its 0's to the end, is `[1,2,1,0,0,0]`.

Return *the resulting array*.

Note that the operations are applied **sequentially**, not all at once.

Example 1:

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

Output: `[1,4,2,0,0,0]`

Explanation: We do the following operations:

- $i = 0$: $\text{nums}[0]$ and $\text{nums}[1]$ are not equal, so we skip this operation.
- $i = 1$: $\text{nums}[1]$ and $\text{nums}[2]$ are equal, we multiply $\text{nums}[1]$ by 2 and change $\text{nums}[2]$ to 0. The array becomes `[1,4,0,1,1,0]`.
- $i = 2$: $\text{nums}[2]$ and $\text{nums}[3]$ are not equal, so we skip this operation.
- $i = 3$: $\text{nums}[3]$ and $\text{nums}[4]$ are equal, we multiply $\text{nums}[3]$ by 2 and change $\text{nums}[4]$ to 0. The array becomes `[1,4,0,2,0,0]`.
- $i = 4$: $\text{nums}[4]$ and $\text{nums}[5]$ are equal, we multiply $\text{nums}[4]$ by 2 and change $\text{nums}[5]$ to 0. The array becomes `[1,4,0,2,0,0]`. After that, we shift the 0's to the end, which gives the array `[1,4,2,0,0,0]`.

Example 2:

Input: `nums = [0,1]`

Output: `[1,0]`

Explanation: No operation can be applied, we just shift the 0 to the end.

Constraints:

- $2 \leq \text{nums.length} \leq 2000$
- $0 \leq \text{nums}[i] \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {number[]}
 */
var applyOperations = function(nums) {

    for(let i=0; i<nums.length; i++){
        if(nums[i]===nums[i+1]){
            nums[i]= nums[i]*2
            nums[i+1] = nums[i+1]*0
        }
    }

    let length = nums.length
    for(let i=0; i<length; i++){
        if(nums[i]===0){
            nums.splice(i,1);
            nums.push(0);

            i--;
            length--;
        }
    }
    return nums;
};
```

[283 Move Zeroes \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {void} Do not return anything, modify nums in-place instead.  
 */  
var moveZeroes = function(nums) {  
  
    let length = nums.length  
  
    for(let i=0; i<length; i++){  
  
        if(nums[i]===0){  
            nums.splice(i,1);  
            nums.push(0);  
  
            i--;  
            length--;  
        }  
    }  
}
```

367 Valid Perfect Square ([link](#))

Description

Given a positive integer num, return true if num is a perfect square or false otherwise.

A **perfect square** is an integer that is the square of an integer. In other words, it is the product of some integer with itself.

You must not use any built-in library function, such as `sqrt`.

Example 1:

```
Input: num = 16
Output: true
Explanation: We return true because 4 * 4 = 16 and 4 is an integer.
```

Example 2:

```
Input: num = 14
Output: false
Explanation: We return false because 3.742 * 3.742 = 14 and 3.742 is not an integer.
```

Constraints:

- $1 \leq \text{num} \leq 2^{31} - 1$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {boolean}  
 */  
var isPerfectSquare = function(num) {  
    for(let i=0; i<=num/i; i++){  
        if(i*i===num){  
            return true;  
        }  
    }  
    return false;  
};
```

28 Find the Index of the First Occurrence in a String [\(link\)](#)

Description

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

Example 1:

```
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
```

Example 2:

```
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
```

Constraints:

- $1 \leq \text{haystack.length}, \text{needle.length} \leq 10^4$
- `haystack` and `needle` consist of only lowercase English characters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} haystack  
 * @param {string} needle  
 * @return {number}  
 */  
var strStr = function(haystack, needle) {  
    // let check = haystack.includes(needle);  
    // if (check) {  
    //     for (let i = 0; i < haystack.length; i++) {  
    //         if (haystack[i]===needle[i]) {  
    //             return i  
    //         }  
    //     }  
    //     console.log("check :", check);  
    // } else {  
    //     return -1;  
    // }  
    return haystack.indexOf(needle)  
};
```

[2727 Number of Senior Citizens \(link\)](#)

Description

You are given a **0-indexed** array of strings `details`. Each element of `details` provides information about a given passenger compressed into a string of length 15. The system is such that:

- The first ten characters consist of the phone number of passengers.
- The next character denotes the gender of the person.
- The following two characters are used to indicate the age of the person.
- The last two characters determine the seat allotted to that person.

Return *the number of passengers who are strictly more than 60 years old*.

Example 1:

```
Input: details = ["7868190130M7522", "5303914400F9211", "9273338290F4010"]
```

```
Output: 2
```

```
Explanation: The passengers at indices 0, 1, and 2 have ages 75, 92, and 40. Thus, there are 2 people who are over 60 years old.
```

Example 2:

```
Input: details = ["1313579440F2036", "2921522980M5644"]
```

```
Output: 0
```

```
Explanation: None of the passengers are older than 60.
```

Constraints:

- $1 \leq \text{details.length} \leq 100$
- $\text{details}[i].length == 15$
- $\text{details}[i]$ consists of digits from '0' to '9'.
- $\text{details}[i][10]$ is either 'M' or 'F' or 'O'.
- The phone numbers and seat numbers of the passengers are distinct.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string[]} details  
 * @return {number}  
 */  
var countSeniors = function(details) {  
    let counter = 0;  
    for(let i = 0; i < details.length; i++){  
        if(details[i][11].concat(details[i][12]) > 60){  
            counter++  
        }  
    }  
    return counter  
};  
  
// return details.filter(el => el.slice(-4, -2) > 60).length;
```

2748 Calculate Delayed Arrival Time ([link](#))

Description

You are given a positive integer `arrivalTime` denoting the arrival time of a train in hours, and another positive integer `delayedTime` denoting the amount of delay in hours.

Return *the time when the train will arrive at the station*.

Note that the time in this problem is in 24-hours format.

Example 1:

```
Input: arrivalTime = 15, delayedTime = 5
```

```
Output: 20
```

```
Explanation: Arrival time of the train was 15:00 hours. It is delayed by 5 hours. Now it will reach at 15+5 = 20 (20:00 hours).
```

Example 2:

```
Input: arrivalTime = 13, delayedTime = 11
```

```
Output: 0
```

```
Explanation: Arrival time of the train was 13:00 hours. It is delayed by 11 hours. Now it will reach at 13+11=24 (Which is denoted by 0 in 24-hour format).
```

Constraints:

- $1 \leq \text{arrivalTime} < 24$
- $1 \leq \text{delayedTime} \leq 24$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} arrivalTime  
 * @param {number} delayedTime  
 * @return {number}  
 */  
var findDelayedArrivalTime = function(arrivalTime, delayedTime) {  
    let time = arrivalTime + delayedTime;  
    let output = time >= 24 ? time - 24 : time;  
    // console.log("output :", output);  
    return output;  
};
```

[2819 Remove Trailing Zeros From a String \(link\)](#)

Description

Given a **positive** integer `num` represented as a string, return *the integer num without trailing zeros as a string*.

Example 1:

Input: num = "51230100"

Output: "512301"

Explanation: Integer "51230100" has 2 trailing zeros, we remove them and return integer "512301".

Example 2:

Input: num = "123"

Output: "123"

Explanation: Integer "123" has no trailing zeros, we return integer "123".

Constraints:

- $1 \leq \text{num.length} \leq 1000$
- `num` consists of only digits.
- `num` doesn't have any leading zeros.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} num  
 * @return {string}  
 */  
var removeTrailingZeros = function(num) {  
    let output = num.replace(/0+$/ , "");  
    return output;  
};
```

1019 Squares of a Sorted Array ([link](#))

Description

Given an integer array `nums` sorted in **non-decreasing** order, return *an array of the squares of each number sorted in non-decreasing order*.

Example 1:

```
Input: nums = [-4,-1,0,3,10]
Output: [0,1,9,16,100]
Explanation: After squaring, the array becomes [16,1,0,9,100].
After sorting, it becomes [0,1,9,16,100].
```

Example 2:

```
Input: nums = [-7,-3,2,3,11]
Output: [4,9,9,49,121]
```

Constraints:

- $1 \leq \text{nums.length} \leq 10^4$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `nums` is sorted in **non-decreasing** order.

Follow up: Squaring each element and sorting the new array is very trivial, could you find an $O(n)$ solution using a different approach?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var sortedSquares = function(nums) {  
    let output = [];  
    for (let i = 0; i < nums.length; i++) {  
        let squr = nums[i] * nums[i];  
        output.push(sqr);  
    }  
    output.sort((a, b) => a - b);  
    return output;  
};
```

2805 Array Wrapper ([link](#))

Description

Create a class `ArrayWrapper` that accepts an array of integers in its constructor. This class should have two features:

- When two instances of this class are added together with the `+` operator, the resulting value is the sum of all the elements in both arrays.
- When the `String()` function is called on the instance, it will return a comma separated string surrounded by brackets. For example, `[1,2,3]`.

Example 1:

```
Input: nums = [[1,2],[3,4]], operation = "Add"
Output: 10
Explanation:
const obj1 = new ArrayWrapper([1,2]);
const obj2 = new ArrayWrapper([3,4]);
obj1 + obj2; // 10
```

Example 2:

```
Input: nums = [[23,98,42,70]], operation = "String"
Output: "[23,98,42,70]"
Explanation:
const obj = new ArrayWrapper([23,98,42,70]);
String(obj); // "[23,98,42,70]"
```

Example 3:

```
Input: nums = [[[],[]]], operation = "Add"
Output: 0
Explanation:
const obj1 = new ArrayWrapper([]);
```

```
const obj2 = new ArrayWrapper([]);  
obj1 + obj2; // 0
```

Constraints:

- $0 \leq \text{nums.length} \leq 1000$
- $0 \leq \text{nums}[i] \leq 1000$
- Note: `nums` is the array passed to the constructor

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 */  
var ArrayWrapper = function(nums) {  
    this.nums = nums;  
};  
  
ArrayWrapper.prototype.valueOf = function() {  
    return this.nums.reduce((acc,num) => acc + num,0);  
}  
  
ArrayWrapper.prototype.toString = function() {  
    return `[${this.nums.join(',')}]`;  
}  
  
/**  
 * const obj1 = new ArrayWrapper([1,2]);  
 * const obj2 = new ArrayWrapper([3,4]);  
 * obj1 + obj2; // 10  
 * String(obj1); // "[1,2]"  
 * String(obj2); // "[3,4]"  
 */
```

[88 Merge Sorted Array \(link\)](#)

Description

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

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

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

Example 1:

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

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

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

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

Example 2:

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

Output: `[1]`

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

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

Example 3:

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

Output: `[1]`

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

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

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

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums1  
 * @param {number} m  
 * @param {number[]} nums2  
 * @param {number} n  
 * @return {void} Do not return anything, modify nums1 in-place instead.  
 */  
var merge = function(nums1, m, nums2, n) {  
    nums1.splice(m)  
    nums2.splice(n)  
    nums1.push(...nums2)  
    return nums1.sort((a, b) => a - b)  
};
```

[2684 Determine the Winner of a Bowling Game \(link\)](#)

Description

You are given two **0-indexed** integer arrays `player1` and `player2`, representing the number of pins that player 1 and player 2 hit in a bowling game, respectively.

The bowling game consists of n turns, and the number of pins in each turn is exactly 10.

Assume a player hits x_i pins in the i^{th} turn. The value of the i^{th} turn for the player is:

- $2x_i$ if the player hits 10 pins **in either $(i - 1)^{\text{th}}$ or $(i - 2)^{\text{th}}$ turn.**
- Otherwise, it is x_i .

The **score** of the player is the sum of the values of their n turns.

Return

- 1 if the score of player 1 is more than the score of player 2,
- 2 if the score of player 2 is more than the score of player 1, and
- 0 in case of a draw.

Example 1:

Input: `player1 = [5,10,3,2]`, `player2 = [6,5,7,3]`

Output: 1

Explanation:

The score of player 1 is $5 + 10 + 2*3 + 2*2 = 25$.

The score of player 2 is $6 + 5 + 7 + 3 = 21$.

Example 2:

Input: player1 = [3,5,7,6], player2 = [8,10,10,2]

Output: 2

Explanation:

The score of player 1 is $3 + 5 + 7 + 6 = 21$.

The score of player 2 is $8 + 10 + 2 \cdot 10 + 2 \cdot 2 = 42$.

Example 3:

Input: player1 = [2,3], player2 = [4,1]

Output: 0

Explanation:

The score of player1 is $2 + 3 = 5$.

The score of player2 is $4 + 1 = 5$.

Example 4:

Input: player1 = [1,1,1,10,10,10,10], player2 = [10,10,10,10,1,1,1]

Output: 2

Explanation:

The score of player1 is $1 + 1 + 1 + 10 + 2 \cdot 10 + 2 \cdot 10 + 2 \cdot 10 = 73$.

The score of player2 is $10 + 2 \cdot 10 + 2 \cdot 10 + 2 \cdot 10 + 2 \cdot 1 + 2 \cdot 1 + 1 = 75$.

Constraints:

- $n == \text{player1.length} == \text{player2.length}$
- $1 \leq n \leq 1000$
- $0 \leq \text{player1}[i], \text{player2}[i] \leq 10$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} player1  
 * @param {number[]} player2  
 * @return {number}  
 */  
var isWinner = function(player1, player2) {  
    let score = 0;  
    for (let i = 0; i < player1.length; i++) {  
        let p1Score = player1[i];  
        let p2Score = player2[i];  
        if (player1[i - 1] == 10 || player1[i - 2] == 10) p1Score *= 2;  
        if (player2[i - 1] == 10 || player2[i - 2] == 10) p2Score *= 2;  
        score += p1Score - p2Score;  
    }  
    let result = score > 0 ? 1 : score === 0 ? 0 : 2;  
    return result  
//    console.log("result :", result);  
};
```

2767 Maximum Sum With Exactly K Elements ([link](#))

Description

You are given a **0-indexed** integer array `nums` and an integer `k`. Your task is to perform the following operation **exactly k** times in order to maximize your score:

1. Select an element `m` from `nums`.
2. Remove the selected element `m` from the array.
3. Add a new element with a value of `m + 1` to the array.
4. Increase your score by `m`.

Return *the maximum score you can achieve after performing the operation exactly k times*.

Example 1:

```
Input: nums = [1,2,3,4,5], k = 3
Output: 18
```

Explanation: We need to choose exactly 3 elements from `nums` to maximize the sum.
For the first iteration, we choose 5. Then sum is 5 and `nums` = [1,2,3,4,6]
For the second iteration, we choose 6. Then sum is 5 + 6 and `nums` = [1,2,3,4,7]
For the third iteration, we choose 7. Then sum is 5 + 6 + 7 = 18 and `nums` = [1,2,3,4,8]
So, we will return 18.
It can be proven, that 18 is the maximum answer that we can achieve.

Example 2:

```
Input: nums = [5,5,5], k = 2
Output: 11
```

Explanation: We need to choose exactly 2 elements from `nums` to maximize the sum.
For the first iteration, we choose 5. Then sum is 5 and `nums` = [5,5,6]
For the second iteration, we choose 6. Then sum is 5 + 6 = 11 and `nums` = [5,5,7]
So, we will return 11.
It can be proven, that 11 is the maximum answer that we can achieve.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {number}  
 */  
var maximizeSum = function(nums, k) {  
    let max = Math.max(...nums);  
    let sum = max;  
    for(let i=1; i<k; i++){  
        sum+= max+i;  
    }  
    return sum;  
};
```

[2752 Sum Multiples \(link\)](#)

Description

Given a positive integer n , find the sum of all integers in the range $[1, n]$ **inclusive** that are divisible by 3, 5, or 7.

Return *an integer denoting the sum of all numbers in the given range satisfying the constraint.*

Example 1:

Input: $n = 7$

Output: 21

Explanation: Numbers in the range $[1, 7]$ that are divisible by 3, 5, or 7 are 3, 5, 6, 7. The sum of these numbers is 21.

Example 2:

Input: $n = 10$

Output: 40

Explanation: Numbers in the range $[1, 10]$ that are divisible by 3, 5, or 7 are 3, 5, 6, 7, 9, 10. The sum of these numbers is 40.

Example 3:

Input: $n = 9$

Output: 30

Explanation: Numbers in the range $[1, 9]$ that are divisible by 3, 5, or 7 are 3, 5, 6, 7, 9. The sum of these numbers is 30.

Constraints:

- $1 \leq n \leq 10^3$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var sumOfMultiples = function(n) {  
    let res = 0;  
    for (let i = 0; i <= n; i++) {  
        if (i % 3 === 0 || i % 5 === 0 || i % 7 === 0) {  
            res = res + i;  
        }  
    }  
    return res  
};
```

[2800 Minimum String Length After Removing Substrings \(link\)](#)

Description

You are given a string s consisting only of **uppercase** English letters.

You can apply some operations to this string where, in one operation, you can remove **any** occurrence of one of the substrings "AB" or "CD" from s .

Return *the minimum possible length of the resulting string that you can obtain*.

Note that the string concatenates after removing the substring and could produce new "AB" or "CD" substrings.

Example 1:

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

Output: 2

Explanation: We can do the following operations:

- Remove the substring "ABFCACDB", so $s = \text{"FCACDB"}$.
- Remove the substring "FCACDB", so $s = \text{"FCAB"}$.
- Remove the substring "FCAB", so $s = \text{"FC"}$.

So the resulting length of the string is 2.

It can be shown that it is the minimum length that we can obtain.

Example 2:

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

Output: 5

Explanation: We cannot do any operations on the string so the length remains the same.

Constraints:

- $1 \leq s.length \leq 100$
- s consists only of uppercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var minLength = function(s) {  
  
    while(s.includes("AB") || s.includes("CD")){  
  
        s = s.replace("AB","");
        s = s.replace("CD","");
    }  
    return s.length  
};
```

[2859 Add Two Promises \(link\)](#)

Description

Given two promises `promise1` and `promise2`, return a new promise. `promise1` and `promise2` will both resolve with a number. The returned promise should resolve with the sum of the two numbers.

Example 1:

```
Input:  
promise1 = new Promise(resolve => setTimeout(() => resolve(2), 20)),  
promise2 = new Promise(resolve => setTimeout(() => resolve(5), 60))  
Output: 7  
Explanation: The two input promises resolve with the values of 2 and 5 respectively. The returned promise should resolve with a value of 7.
```

Example 2:

```
Input:  
promise1 = new Promise(resolve => setTimeout(() => resolve(10), 50)),  
promise2 = new Promise(resolve => setTimeout(() => resolve(-12), 30))  
Output: -2  
Explanation: The two input promises resolve with the values of 10 and -12 respectively. The returned promise should resolve with a value of -2.
```

Constraints:

- `promise1` and `promise2` are promises that resolve with a number

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {Promise} promise1  
 * @param {Promise} promise2  
 * @return {Promise}  
 */  
var addTwoPromises = async function(promise1, promise2) {  
    return (await promise1 + await promise2);  
};  
  
/**  
 * addTwoPromises(Promise.resolve(2), Promise.resolve(2))  
 *   .then(console.log); // 4  
 */
```

[2864 Is Object Empty \(link\)](#)

Description

Given an object or an array, return if it is empty.

- An empty object contains no key-value pairs.
- An empty array contains no elements.

You may assume the object or array is the output of `JSON.parse`.

Example 1:

Input: obj = {"x": 5, "y": 42}

Output: false

Explanation: The object has 2 key-value pairs so it is not empty.

Example 2:

Input: obj = {}

Output: true

Explanation: The object doesn't have any key-value pairs so it is empty.

Example 3:

Input: obj = [null, false, 0]

Output: false

Explanation: The array has 3 elements so it is not empty.

Constraints:

- `obj` is a valid JSON object or array

- $2 \leq \text{JSON.stringify}(\text{obj}).\text{length} \leq 10^5$

Can you solve it in O(1) time?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {Object | Array} obj  
 * @return {boolean}  
 */  
  
var isEmpty = function(obj) {  
  
    for (var x in obj) {  
        if (obj.hasOwnProperty(x)) return false;  
    }  
    return true;  
  
};
```

[2824 Check if The Number is Fascinating \(link\)](#)

Description

You are given an integer n that consists of exactly 3 digits.

We call the number n **fascinating** if, after the following modification, the resulting number contains all the digits from 1 to 9 **exactly** once and does not contain any 0's:

- **Concatenate** n with the numbers $2 * n$ and $3 * n$.

Return `true` if n is *fascinating*, or `false` otherwise.

Concatenating two numbers means joining them together. For example, the concatenation of 121 and 371 is 121371.

Example 1:

Input: $n = 192$

Output: true

Explanation: We concatenate the numbers $n = 192$ and $2 * n = 384$ and $3 * n = 576$. The resulting number is 192384576. This number contains all digits from 1 to 9 exactly once.

Example 2:

Input: $n = 100$

Output: false

Explanation: We concatenate the numbers $n = 100$ and $2 * n = 200$ and $3 * n = 300$. The resulting number is 100200300. This number does not contain all digits from 1 to 9 exactly once.

Constraints:

- $100 \leq n \leq 999$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {boolean}  
 */  
var isFascinating = function(n) {  
    let conc = n.toString().concat(n*2).concat(n*3)  
    console.log(conc)  
  
    if (conc.includes('0')) return false  
    if (conc.length !== 9) return false  
    if ((new Set(conc)).size !== 9) return false  
  
    return true  
}  
  
// const isFascinating = (n) => "123456789" === `${n}${n * 2}${n * 3}`.split("").sort().join("");
```

[2386 Min Max Game \(link\)](#)

Description

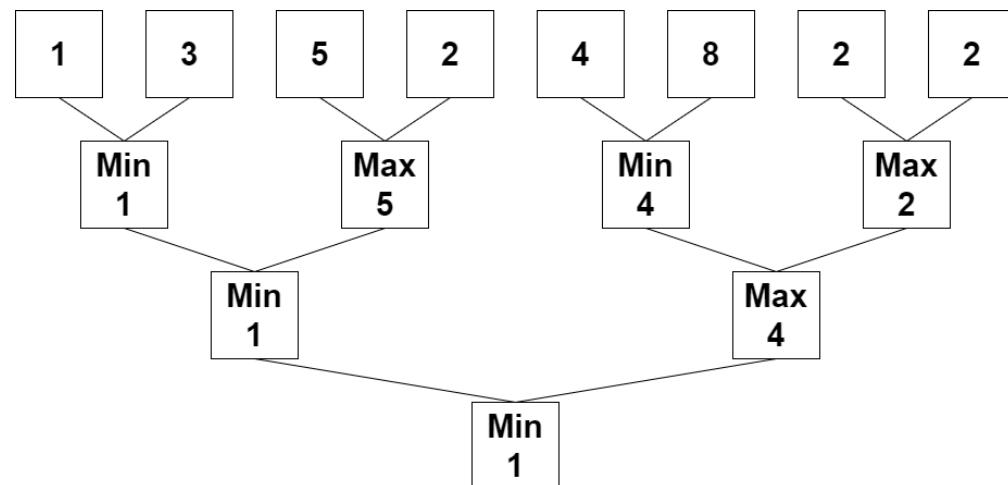
You are given a **0-indexed** integer array `nums` whose length is a power of 2.

Apply the following algorithm on `nums`:

1. Let n be the length of `nums`. If $n == 1$, **end** the process. Otherwise, **create a new 0-indexed integer array `newNums` of length $n / 2$.**
2. For every **even** index i where $0 \leq i < n / 2$, **assign** the value of `newNums[i]` as $\min(\text{nums}[2 * i], \text{nums}[2 * i + 1])$.
3. For every **odd** index i where $0 \leq i < n / 2$, **assign** the value of `newNums[i]` as $\max(\text{nums}[2 * i], \text{nums}[2 * i + 1])$.
4. **Replace** the array `nums` with `newNums`.
5. **Repeat** the entire process starting from step 1.

Return *the last number that remains in `nums` after applying the algorithm*.

Example 1:



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

Output: 1

Explanation: The following arrays are the results of applying the algorithm repeatedly.

```
First: nums = [1,5,4,2]
Second: nums = [1,4]
Third: nums = [1]
1 is the last remaining number, so we return 1.
```

Example 2:

```
Input: nums = [3]
Output: 3
Explanation: 3 is already the last remaining number, so we return 3.
```

Constraints:

- $1 \leq \text{nums.length} \leq 1024$
- $1 \leq \text{nums}[i] \leq 10^9$
- nums.length is a power of 2.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var minMaxGame = function (nums) {  
    if(nums.length === 1) return nums[0]  
  
    let result = [...nums];  
  
    while (result.length !== 2) {  
        let tempResult = []  
        for (let i = 0; i < result.length; i += 4) {  
            tempResult.push(Math.min(result[i], result[i + 1]))  
            tempResult.push(Math.max(result[i + 2], result[i + 3]))  
        }  
        result = tempResult  
    }  
  
    return Math.min(result[0],result[1])  
};
```

[2836 Neither Minimum nor Maximum \(link\)](#)

Description

Given an integer array `nums` containing **distinct positive** integers, find and return **any** number from the array that is neither the **minimum** nor the **maximum** value in the array, or `-1` if there is no such number.

Return *the selected integer*.

Example 1:

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

Output: `2`

Explanation: In this example, the minimum value is `1` and the maximum value is `4`. Therefore, either `2` or `3` can be valid answers.

Example 2:

Input: `nums = [1,2]`

Output: `-1`

Explanation: Since there is no number in `nums` that is neither the maximum nor the minimum, we cannot select a number that satisfies

Example 3:

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

Output: `2`

Explanation: Since `2` is neither the maximum nor the minimum value in `nums`, it is the only valid answer.

Constraints:

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

- All values in `nums` are distinct

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var findNonMinOrMax = function(nums) {  
    nums.sort((a,b)=> a-b)  
  
    if(nums.length == 2 ||  nums.length == 1) return -1  
    else return nums[1]  
};
```

2820 Return Length of Arguments Passed ([link](#))

Description

Write a function `argumentsLength` that returns the count of arguments passed to it.

Example 1:

```
Input: args = [5]
Output: 1
Explanation:
argumentsLength(5); // 1
```

One value was passed to the function so it should return 1.

Example 2:

```
Input: args = [{}, null, "3"]
Output: 3
Explanation:
argumentsLength({}, null, "3"); // 3
```

Three values were passed to the function so it should return 3.

Constraints:

- `args` is a valid JSON array
- $0 \leq \text{args.length} \leq 100$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @return {number}  
 */  
var argumentsLength = function(...args) {  
    return (args.length)  
};  
  
/**  
 * argumentsLength(1, 2, 3); // 3  
 */
```

[2746 Filter Elements from Array \(link\)](#)

Description

Given an integer array `arr` and a filtering function `fn`, return a filtered array `filteredArr`.

The `fn` function takes one or two arguments:

- `arr[i]` - number from the `arr`
- `i` - index of `arr[i]`

`filteredArr` should only contain the elements from the `arr` for which the expression `fn(arr[i], i)` evaluates to a **truthy** value. A **truthy** value is a value where `Boolean(value)` returns `true`.

Please solve it without the built-in `Array.filter` method.

Example 1:

Input: arr = [0,10,20,30], fn = function greaterThan10(n) { return n > 10; }

Output: [20,30]

Explanation:

`const newArray = filter(arr, fn); // [20, 30]`

The function filters out values that are not greater than 10

Example 2:

Input: arr = [1,2,3], fn = function firstIndex(n, i) { return i === 0; }

Output: [1]

Explanation:

`fn` can also accept the index of each element

In this case, the function removes elements not at index 0

Example 3:

Input: arr = [-2,-1,0,1,2], fn = function plusOne(n) { return n + 1 }
Output: [-2,0,1,2]
Explanation:
Falsey values such as 0 should be filtered out

Constraints:

- $0 \leq \text{arr.length} \leq 1000$
- $-10^9 \leq \text{arr}[i] \leq 10^9$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} arr  
 * @param {Function} fn  
 * @return {number[]}  
 */  
var filter = function(arr, fn) {  
    let i = 0;  
    return arr.filter(item => fn(item, i++))  
};
```

2747 Apply Transform Over Each Element in Array ([link](#))

Description

Given an integer array `arr` and a mapping function `fn`, return a new array with a transformation applied to each element.

The returned array should be created such that `returnedArray[i] = fn(arr[i], i)`.

Please solve it without the built-in `Array.map` method.

Example 1:

```
Input: arr = [1,2,3], fn = function plusone(n) { return n + 1; }  
Output: [2,3,4]
```

Explanation:

```
const newArray = map(arr, plusone); // [2,3,4]  
The function increases each value in the array by one.
```

Example 2:

```
Input: arr = [1,2,3], fn = function plusI(n, i) { return n + i; }  
Output: [1,3,5]  
Explanation: The function increases each value by the index it resides in.
```

Example 3:

```
Input: arr = [10,20,30], fn = function constant() { return 42; }  
Output: [42,42,42]  
Explanation: The function always returns 42.
```

Constraints:

- $0 \leq \text{arr.length} \leq 1000$
- $-10^9 \leq \text{arr}[i] \leq 10^9$
- fn returns a number

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} arr  
 * @param {Function} fn  
 * @return {number[]}  
 */  
var map = function(arr, fn) {  
  
    let i = 0;  
    return arr.map(item => fn(item, i++))  
};
```

[2761 Array Reduce Transformation \(link\)](#)

Description

Given an integer array `nums`, a reducer function `fn`, and an initial value `init`, return the final result obtained by executing the `fn` function on each element of the array, sequentially, passing in the return value from the calculation on the preceding element.

This result is achieved through the following operations: `val = fn(init, nums[0])`, `val = fn(val, nums[1])`, `val = fn(val, nums[2])`, ... until every element in the array has been processed. The ultimate value of `val` is then returned.

If the length of the array is 0, the function should return `init`.

Please solve it without using the built-in `Array.reduce` method.

Example 1:

```
Input:  
nums = [1,2,3,4]  
fn = function sum(accum, curr) { return accum + curr; }  
init = 0  
Output: 10  
Explanation:  
initially, the value is init=0.  
(0) + nums[0] = 1  
(1) + nums[1] = 3  
(3) + nums[2] = 6  
(6) + nums[3] = 10  
The final answer is 10.
```

Example 2:

```
Input:  
nums = [1,2,3,4]  
fn = function sum(accum, curr) { return accum + curr * curr; }  
init = 100  
Output: 130  
Explanation:
```

```
initially, the value is init=100.  
(100) + nums[0] * nums[0] = 101  
(101) + nums[1] * nums[1] = 105  
(105) + nums[2] * nums[2] = 114  
(114) + nums[3] * nums[3] = 130  
The final answer is 130.
```

Example 3:

Input:
nums = []
fn = function sum(accum, curr) { return 0; }
init = 25

Output: 25

Explanation: For empty arrays, the answer is always init.

Constraints:

- $0 \leq \text{nums.length} \leq 1000$
- $0 \leq \text{nums}[i] \leq 1000$
- $0 \leq \text{init} \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {Function} fn  
 * @param {number} init  
 * @return {number}  
 */  
var reduce = function(nums, fn, init) {  
    var a = init;  
  
    for(let i=0; i<nums.length; i++){  
        a = fn(a, nums[i]);  
    }  
  
    return a;  
};
```

2809 Create Hello World Function ([link](#))

Description

Write a function `createHelloWorld`. It should return a new function that always returns "Hello World".

Example 1:

```
Input: args = []
Output: "Hello World"
Explanation:
const f = createHelloWorld();
f(); // "Hello World"
```

The function returned by `createHelloWorld` should always return "Hello World".

Example 2:

```
Input: args = [{}],null,42]
Output: "Hello World"
Explanation:
const f = createHelloWorld();
f({}, null, 42); // "Hello World"
```

Any arguments could be passed to the function but it should still always return "Hello World".

Constraints:

- $0 \leq \text{args.length} \leq 10$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @return {Function}  
 */  
var createHelloWorld = function() {  
    return function(...args) {  
        return "Hello World"  
    }  
};  
// const f = createHelloWorld();  
// f();  
  
/**  
 * const f = createHelloWorld();  
 * f(); // "Hello World"  
 */
```

[2749 Promise Time Limit \(link\)](#)

Description

Given an asynchronous function `fn` and a time `t` in milliseconds, return a new **time limited** version of the input function. `fn` takes arguments provided to the **time limited** function.

The **time limited** function should follow these rules:

- If the `fn` completes within the time limit of `t` milliseconds, the **time limited** function should resolve with the result.
- If the execution of the `fn` exceeds the time limit, the **time limited** function should reject with the string "Time Limit Exceeded".

Example 1:

```
Input:  
fn = async (n) => {  
    await new Promise(res => setTimeout(res, 100));  
    return n * n;  
}  
inputs = [5]  
t = 50  
Output: {"rejected": "Time Limit Exceeded", "time": 50}  
Explanation:  
const limited = timeLimit(fn, t)  
const start = performance.now()  
let result;  
try {  
    const res = await limited(...inputs)  
    result = {"resolved": res, "time": Math.floor(performance.now() - start)};  
} catch (err) {  
    result = {"rejected": err, "time": Math.floor(performance.now() - start)};  
}  
console.log(result) // Output
```

The provided function is set to resolve after 100ms. However, the time limit is set to 50ms. It rejects at t=50ms because the time

Example 2:

Input:

```
fn = async (n) => {
    await new Promise(res => setTimeout(res, 100));
    return n * n;
}
```

```
inputs = [5]
```

```
t = 150
```

Output: {"resolved":25,"time":100}

Explanation:

The function resolved $5 * 5 = 25$ at $t=100\text{ms}$. The time limit is never reached.

Example 3:**Input:**

```
fn = async (a, b) => {
    await new Promise(res => setTimeout(res, 120));
    return a + b;
}
```

```
inputs = [5,10]
```

```
t = 150
```

Output: {"resolved":15,"time":120}

Explanation:

The function resolved $5 + 10 = 15$ at $t=120\text{ms}$. The time limit is never reached.

Example 4:**Input:**

```
fn = async () => {
    throw "Error";
}
```

```
inputs = []
```

```
t = 1000
```

Output: {"rejected":"Error","time":0}

Explanation:

The function immediately throws an error.

Constraints:

- $0 \leq \text{inputs.length} \leq 10$

- $0 \leq t \leq 1000$
- fn returns a promise

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {Function} fn  
 * @param {number} t  
 * @return {Function}  
 */  
var timeLimit = function(fn, t) {  
    return async function(...args) {  
  
        return new Promise((resolve,reject) => {  
            setTimeout(() => {  
                reject("Time Limit Exceeded");  
            }, t);  
            fn(...args).then(resolve).catch(reject);  
        })  
    }  
};  
  
/**  
 * const limited = timeLimit((t) => new Promise(res => setTimeout(res, t)), 100);  
 * limited(150).catch(console.log) // "Time Limit Exceeded" at t=100ms  
*/
```

242 Valid Anagram ([link](#))

Description

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

Example 1:

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

Output: true

Example 2:

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

Output: false

Constraints:

- $1 \leq s.\text{length}, t.\text{length} \leq 5 * 10^4$
- s and t consist of lowercase English letters.

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @param {string} t  
 * @return {boolean}  
 */  
var isAnagram = function(s, t) {  
    let sort_s = s.split("").sort().join();  
    let sort_t = t.split("").sort().join();  
  
    return sort_s === sort_t  
};
```

[1013 Fibonacci Number \(link\)](#)

Description

The **Fibonacci numbers**, commonly denoted $F(n)$ form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$\begin{aligned}F(0) &= 0, \quad F(1) = 1 \\F(n) &= F(n - 1) + F(n - 2), \text{ for } n > 1.\end{aligned}$$

Given n , calculate $F(n)$.

Example 1:

Input: $n = 2$
Output: 1
Explanation: $F(2) = F(1) + F(0) = 1 + 0 = 1.$

Example 2:

Input: $n = 3$
Output: 2
Explanation: $F(3) = F(2) + F(1) = 1 + 1 = 2.$

Example 3:

Input: $n = 4$
Output: 3
Explanation: $F(4) = F(3) + F(2) = 2 + 1 = 3.$

Constraints:

- $0 \leq n \leq 30$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var fib = function(n) {  
    // if(n<=1){  
    //     return n  
    // }  
  
    return n<=1 ? n: fib(n-1) + fib(n-2)  
};
```

2427 First Letter to Appear Twice ([link](#))

Description

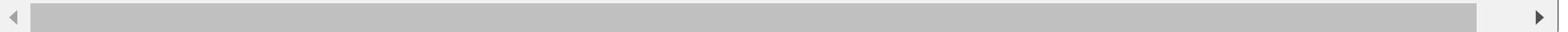
Given a string s consisting of lowercase English letters, return *the first letter to appear twice*.

Note:

- A letter a appears twice before another letter b if the **second** occurrence of a is before the **second** occurrence of b .
- s will contain at least one letter that appears twice.

Example 1:

```
Input: s = "abccbaacz"
Output: "c"
Explanation:
The letter 'a' appears on the indexes 0, 5 and 6.
The letter 'b' appears on the indexes 1 and 4.
The letter 'c' appears on the indexes 2, 3 and 7.
The letter 'z' appears on the index 8.
The letter 'c' is the first letter to appear twice, because out of all the letters the index of its second occurrence is the smalles
```



Example 2:

```
Input: s = "abcdd"
Output: "d"
Explanation:
The only letter that appears twice is 'd' so we return 'd'.
```

Constraints:

- $2 \leq s.length \leq 100$

- s consists of lowercase English letters.
- s has at least one repeated letter.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {character}  
 */  
  
var repeatedCharacter = function(s) {  
    const obj = {};  
    for(let i = 0; i < s.length; i++) {  
        if(!obj[s[i]]) {  
            obj[s[i]] = 1  
        }else {  
            return s[i];  
        }  
    }  
};
```

[387 First Unique Character in a String \(link\)](#)

Description

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

Example 1:

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

Output: 0

Explanation:

The character '`l`' at index 0 is the first character that does not occur at any other index.

Example 2:

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

Output: 2

Example 3:

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

Output: -1

Constraints:

- $1 \leq s.\text{length} \leq 10^5$
- s consists of only lowercase English letters.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var firstUniqChar = function(s) {  
    for ( let i = 0 ; i < s.length; i++) {  
        if (s.indexOf(s[i]) === s.lastIndexOf(s[i]))  
            return i  
    }  
    return -1  
}
```

[2561 Number of Distinct Averages \(link\)](#)

Description

You are given a **0-indexed** integer array `nums` of **even** length.

As long as `nums` is **not** empty, you must repetitively:

- Find the minimum number in `nums` and remove it.
- Find the maximum number in `nums` and remove it.
- Calculate the average of the two removed numbers.

The **average** of two numbers a and b is $(a + b) / 2$.

- For example, the average of 2 and 3 is $(2 + 3) / 2 = 2.5$.

Return *the number of distinct averages calculated using the above process*.

Note that when there is a tie for a minimum or maximum number, any can be removed.

Example 1:

```
Input: nums = [4,1,4,0,3,5]
Output: 2
Explanation:
1. Remove 0 and 5, and the average is (0 + 5) / 2 = 2.5. Now, nums = [4,1,4,3].
2. Remove 1 and 4. The average is (1 + 4) / 2 = 2.5, and nums = [4,3].
3. Remove 3 and 4, and the average is (3 + 4) / 2 = 3.5.
Since there are 2 distinct numbers among 2.5, 2.5, and 3.5, we return 2.
```

Example 2:

```
Input: nums = [1,100]
Output: 1
Explanation:
There is only one average to be calculated after removing 1 and 100, so we return 1.
```

Constraints:

- $2 \leq \text{nums.length} \leq 100$
- nums.length is even.
- $0 \leq \text{nums}[i] \leq 100$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {number}
 */
var distinctAverages = function (nums) {
    let sortNums = nums.sort((a, b) => a - b);
    let avrArr = [];
    for (let i = 0; i < sortNums.length; i++) {
        let min = sortNums.shift();
        let max = sortNums.pop();
        var averageIs = (min + max) / 2;
        avrArr.push(averageIs);
        i = 0;
    }
    for (let i = 0; i < avrArr.length; i++) {
        for (let j = avrArr.length - 1; j >= i + 1; j--) {
            if (avrArr[i] == avrArr[j]) {
                avrArr.splice(j, 1);
            }
        }
    }
    return avrArr.length
    console.log(avrArr, "aftr");
};
```

26 Remove Duplicates from Sorted Array ([link](#))

Description

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:

- $1 \leq \text{nums.length} \leq 3 * 10^4$
- $-100 \leq \text{nums}[i] \leq 100$
- **nums** is sorted in **non-decreasing** order.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var removeDuplicates = function(nums) {  
    for (let i = 0; i < nums.length; i++) {  
        for (let j = nums.length - 1; j >= i + 1; j--) {  
            if (nums[i] == nums[j]) {  
                nums.splice(j,1);  
            }  
        }  
    }  
    // return nums;  
    console.log(nums);  
};  
removeDuplicates([1,1,2])
```

[2542 Average Value of Even Numbers That Are Divisible by Three \(link\)](#)

Description

Given an integer array `nums` of **positive** integers, return *the average value of all even integers that are divisible by 3*.

Note that the **average** of `n` elements is the **sum** of the `n` elements divided by `n` and **rounded down** to the nearest integer.

Example 1:

Input: `nums = [1,3,6,10,12,15]`

Output: 9

Explanation: 6 and 12 are even numbers that are divisible by 3. $(6 + 12) / 2 = 9$.

Example 2:

Input: `nums = [1,2,4,7,10]`

Output: 0

Explanation: There is no single number that satisfies the requirement, so return 0.

Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $1 \leq \text{nums}[i] \leq 1000$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var averageValue = function (nums) {  
    let sum = 0;  
    let n = 0;  
    for (let i = 0; i < nums.length; i++) {  
        if (nums[i] % 2 == 0 && nums[i] % 3 == 0) {  
            sum += nums[i];  
            n++;  
        }  
    }  
    return sum == 0 ? sum : parseInt(sum/n)  
};  
// averageValue([1, 3, 6, 10, 12, 15]);  
// averageValue([1, 2, 4, 7, 10]);
```

2298 Count Integers With Even Digit Sum ([link](#))

Description

Given a positive integer `num`, return *the number of positive integers less than or equal to num whose digit sums are even*.

The **digit sum** of a positive integer is the sum of all its digits.

Example 1:

```
Input: num = 4
Output: 2
Explanation:
```

The only integers less than or equal to 4 whose digit sums are even are 2 and 4.

Example 2:

```
Input: num = 30
Output: 14
Explanation:
```

The 14 integers less than or equal to 30 whose digit sums are even are 2, 4, 6, 8, 11, 13, 15, 17, 19, 20, 22, 24, 26, and 28.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number} num
 * @return {number}
 */
var countEven = function (num) {
    let numArr = [];
    for (i = 1; i <= num; i++) {
        numArr.push(i);
    }
    let numSpltArr = [];
    for (i = 0; i < numArr.length; i++) {
        let numsplt = numArr[i].toString().split("");
        numSpltArr.push(numsplt);
    }
    // console.log(numSpltArr);

    let numEven = [];
    for (let i = 0; i < numSpltArr.length; i++) {
        var numSArr = numSpltArr[i];

        if (numSArr.length == 1 && numSArr[0] % 2 == 0) {
            numEven.push(numSArr[0]);
        }
        if (numSArr.length > 1) {
            let sum = 0;

            for (let j = 0; j < numSArr.length; j++) {
                sum += +numSArr[j];
            }

            // console.log(sum, "sum");
            if (sum % 2 == 0) {
                numEven.push(sum);
            }
        }
    }
    // console.log(numEven, numEven.length);
    return numEven.length
}
```

```
};  
// countEven(4);
```

1031 Add to Array-Form of Integer ([link](#))

Description

The **array-form** of an integer `num` is an array representing its digits in left to right order.

- For example, for `num = 1321`, the array form is `[1,3,2,1]`.

Given `num`, the **array-form** of an integer, and an integer `k`, return *the array-form of the integer* `num + k`.

Example 1:

```
Input: num = [1,2,0,0], k = 34
Output: [1,2,3,4]
Explanation: 1200 + 34 = 1234
```

Example 2:

```
Input: num = [2,7,4], k = 181
Output: [4,5,5]
Explanation: 274 + 181 = 455
```

Example 3:

```
Input: num = [2,1,5], k = 806
Output: [1,0,2,1]
Explanation: 215 + 806 = 1021
```

Constraints:

- $1 \leq \text{num.length} \leq 10^4$
- $0 \leq \text{num}[i] \leq 9$

- `num` does not contain any leading zeros except for the zero itself.
- $1 \leq k \leq 10^4$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} num  
 * @param {number} k  
 * @return {number[]}  
 */  
var addToArrayForm = function (num, k) {  
    let numToNum = BigInt(num.join("")) + BigInt(k);  
    return numToNum.toString().split("").map(Number);  
};
```

2284 Smallest Value of the Rearranged Number ([link](#))

Description

You are given an integer `num`. **Rearrange** the digits of `num` such that its value is **minimized** and it does not contain **any** leading zeros.

Return *the rearranged number with minimal value*.

Note that the sign of the number does not change after rearranging the digits.

Example 1:

```
Input: num = 310
Output: 103
```

Explanation: The possible arrangements for the digits of 310 are 013, 031, 103, 130, 301, 310.
The arrangement with the smallest value that does not contain any leading zeros is 103.

Example 2:

```
Input: num = -7605
Output: -7650
```

Explanation: Some possible arrangements for the digits of -7605 are -7650, -6705, -5076, -0567.
The arrangement with the smallest value that does not contain any leading zeros is -7650.

Constraints:

- $-10^{15} \leq \text{num} \leq 10^{15}$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
var smallestNumber = function (num) {
    if (num >= 0) {
        let numPos = num.toString().split("").sort((a, b) => a - b);

        let numPos2 = [];
        var output;

        for (let i = 0; i < numPos.length; i++) {
            numPos2.push(Number(numPos[i]));
        }

        if (numPos2[0] !== 0) {
            output = Number(numPos2.join(""))
        }

        return output ;
    }

    if (numPos2[0] == 0) {
        for (let i = 0; i < numPos2.length; i++) {
            if (numPos2[i] > 0) {
                let temp = numPos2[0];
                numPos2[0] = numPos2[i];
                numPos2[i] = temp;
                break;
            }
        }
        output = numPos2.join("");
        return Number(output);
    }
} else {
    let result = num
        .toString()
        .split("")
        .sort((a, b) => b - a);

    if (result[result.length - 1] == "-") {
        result.pop();
        result.unshift("-");
    }
}
```

```
        }
        result = result.join("");
        return Number(result);
    }
};

// let num = 270;
// console.log(smallestNumber(num));
```

[958 Sort Array By Parity II \(link\)](#)

Description

Given an array of integers `nums`, half of the integers in `nums` are **odd**, and the other half are **even**.

Sort the array so that whenever `nums[i]` is odd, `i` is **odd**, and whenever `nums[i]` is even, `i` is **even**.

Return *any answer array that satisfies this condition*.

Example 1:

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

Output: `[4,5,2,7]`

Explanation: `[4,7,2,5]`, `[2,5,4,7]`, `[2,7,4,5]` would also have been accepted.

Example 2:

Input: `nums = [2,3]`

Output: `[2,3]`

Constraints:

- $2 \leq \text{nums.length} \leq 2 * 10^4$
- `nums.length` is even.
- Half of the integers in `nums` are even.
- $0 \leq \text{nums}[i] \leq 1000$

Follow Up: Could you solve it in-place?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var sortArrayByParityII = function (nums) {  
    let oddArr = [];  
    let evenArr = [];  
    let finalArr = [];  
  
    for(let i=0; i<nums.length; i++){  
        if(nums[i]%2 === 0){  
            evenArr.push(nums[i])  
        }else{  
            oddArr.push(nums[i])  
        }  
    }  
    // let oddSort = oddArr.sort((a,b) => b-a);  
    // let evenSort = evenArr.sort((a,b) => a-b);  
  
    for(let i =0; i< evenArr.length; i++){  
        finalArr.push(evenArr[i]);  
        finalArr.push(oddArr[i]);  
    }  
  
    return finalArr  
};  
  
// sortArrayByParityII([3, 1, 2, 4]);
```

[75 Sort Colors \(link\)](#)

Description

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

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

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

Example 1:

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

Example 2:

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

Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 300$
- $\text{nums}[i]$ is either 0, 1, or 2.

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {void} Do not return anything, modify nums in-place instead.  
 */  
var sortColors = function(nums) {  
    nums.sort((a,b)=> a-b)  
};
```

[153 Find Minimum in Rotated Sorted Array \(link\)](#)

Description

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

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

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

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

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

Example 1:

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

Output: 1

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

Example 2:

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

Output: 0

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

Example 3:

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

Output: 11

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var findMin = function (nums) {  
    nums.sort(function (a, b) {  
        return a - b;  
    });  
    return nums[0]  
};  
// console.log(nums[0]);  
// findMin([3, 4, 5, 1, 2]);
```

8 String to Integer (atoi) (link)

Description

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer.

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

1. **Whitespace**: Ignore any leading whitespace (" ").
2. **Signedness**: Determine the sign by checking if the next character is '-' or '+', assuming positivity is neither present.
3. **Conversion**: Read the integer by skipping leading zeros until a non-digit character is encountered or the end of the string is reached. If no digits were read, then the result is 0.
4. **Rounding**: If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then round the integer to remain in the range. Specifically, integers less than -2^{31} should be rounded to -2^{31} , and integers greater than $2^{31} - 1$ should be rounded to $2^{31} - 1$.

Return the integer as the final result.

Example 1:

Input: s = "42"

Output: 42

Explanation:

The underlined characters are what is read in and the caret is the current reader position.

Step 1: "42" (no characters read because there is no leading whitespace)
 ^

Step 2: "42" (no characters read because there is neither a '-' nor '+')
 ^

Step 3: "42" ("42" is read in)
 ^

Example 2:

Input: s = "-042"

Output: -42**Explanation:**

```
Step 1: "___-042" (leading whitespace is read and ignored)
         ^
Step 2: "   -042" ('-' is read, so the result should be negative)
         ^
Step 3: "   -042" ("042" is read in, leading zeros ignored in the result)
```

Example 3:**Input:** s = "1337c0d3"**Output:** 1337**Explanation:**

```
Step 1: "1337c0d3" (no characters read because there is no leading whitespace)
         ^
Step 2: "1337c0d3" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "1337c0d3" ("1337" is read in; reading stops because the next character is a non-digit)
```

Example 4:**Input:** s = "0-1"**Output:** 0**Explanation:**

```
Step 1: "0-1" (no characters read because there is no leading whitespace)
         ^
Step 2: "0-1" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "0-1" ("0" is read in; reading stops because the next character is a non-digit)
```

Example 5:

Input: s = "words and 987"

Output: 0

Explanation:

Reading stops at the first non-digit character 'w'.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} s  
 * @return {number}  
 */  
var myAtoi = function(s) {  
  
    // Matches characters which is integers OR  
    // Matches characters which starts with either + or -  
    // Also matches only the space character which is at the start  
    let numText = s.match(/(^(\ +?|)\d+|(^(\ +?|)(-\|\+)\d+))/)  
  
    let num = parseInt(numText)  
  
    if (num < (-2)**31) {  
        num = (-2)**31  
    } else if ( num > (2)**31 - 1) {  
        num = 2**31-1  
    }  
  
    return isNaN(num)?0:num  
};
```

[2639 Separate the Digits in an Array \(link\)](#)

Description

Given an array of positive integers `nums`, return *an array answer that consists of the digits of each integer in `nums` after separating them in the same order they appear in `nums`.*

To separate the digits of an integer is to get all the digits it has in the same order.

- For example, for the integer 10921, the separation of its digits is [1,0,9,2,1].

Example 1:

```
Input: nums = [13,25,83,77]
Output: [1,3,2,5,8,3,7,7]
Explanation:
- The separation of 13 is [1,3].
- The separation of 25 is [2,5].
- The separation of 83 is [8,3].
- The separation of 77 is [7,7].
answer = [1,3,2,5,8,3,7,7]. Note that answer contains the separations in the same order.
```

Example 2:

```
Input: nums = [7,1,3,9]
Output: [7,1,3,9]
Explanation: The separation of each integer in nums is itself.
answer = [7,1,3,9].
```

Constraints:

- $1 \leq \text{nums.length} \leq 1000$
- $1 \leq \text{nums}[i] \leq 10^5$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var separateDigits = function (nums) {  
    let separate = nums.toString().split("");  
  
    let output = [];  
    for (let i = 0; i < separate.length; i++) {  
        if (  
            separate[i] == "0" ||  
            separate[i] == "1" ||  
            separate[i] == "2" ||  
            separate[i] == "3" ||  
            separate[i] == "4" ||  
            separate[i] == "5" ||  
            separate[i] == "6" ||  
            separate[i] == "7" ||  
            separate[i] == "8" ||  
            separate[i] == "9"  
        ) {  
            let jointNum = +separate[i];  
            output.push(jointNum);  
        }  
    }  
    return output  
//    console.log(output);  
};
```

[2283 Sort Even and Odd Indices Independently \(link\)](#)

Description

You are given a **0-indexed** integer array `nums`. Rearrange the values of `nums` according to the following rules:

1. Sort the values at **odd indices** of `nums` in **non-increasing** order.
 - For example, if `nums` = [4,1,2,3] before this step, it becomes [4,3,2,1] after. The values at odd indices 1 and 3 are sorted in non-increasing order.
2. Sort the values at **even indices** of `nums` in **non-decreasing** order.
 - For example, if `nums` = [4,1,2,3] before this step, it becomes [2,1,4,3] after. The values at even indices 0 and 2 are sorted in non-decreasing order.

Return *the array formed after rearranging the values of* `nums`.

Example 1:

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

Output: [2,3,4,1]

Explanation:

First, we sort the values present at odd indices (1 and 3) in non-increasing order.

So, `nums` changes from [4,1,2,3] to [4,3,2,1].

Next, we sort the values present at even indices (0 and 2) in non-decreasing order.

So, `nums` changes from [4,3,2,1] to [2,3,4,1].

Thus, the array formed after rearranging the values is [2,3,4,1].

Example 2:

Input: `nums` = [2,1]

Output: [2,1]

Explanation:

Since there is exactly one odd index and one even index, no rearrangement of values takes place.

The resultant array formed is [2,1], which is the same as the initial array.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var sortEvenOdd = function (nums) {  
    let oddArr = [];  
    let evenArr = [];  
    let finalArr = [];  
  
    for (let i = 0; i < nums.length; i++) {  
        if (i % 2 === 0) {  
            evenArr.push(nums[i]);  
        } else {  
            oddArr.push(nums[i]);  
        }  
    }  
  
    let oddSort = oddArr.sort((a, b) => b - a);  
    let evenSort = evenArr.sort((a, b) => a - b);  
  
    for (let i = 0; i < oddArr.length; i++) {  
        finalArr.push(evenSort[i]);  
        finalArr.push(oddSort[i]);  
    }  
  
    if (evenSort.length > oddSort.length) {  
        finalArr.push(evenSort[evenSort.length - 1]);  
    }  
  
    // console.log(finalArr);  
    return finalArr  
};  
// sortEvenOdd([4, 1, 2, 3]);
```

[948 Sort an Array \(link\)](#)

Description

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem **without using any built-in** functions in $O(n \log(n))$ time complexity and with the smallest space complexity possible.

Example 1:

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

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

Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of o

Example 2:

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

Output: `[0,0,1,1,2,5]`

Explanation: Note that the values of `nums` are not necessarily unique.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number[]}  
 */  
var sortArray = function(nums) {  
  
    nums.sort(function(a,b){return a-b})  
    return nums  
};
```

[2589 Maximum Value of a String in an Array \(link\)](#)

Description

The **value** of an alphanumeric string can be defined as:

- The **numeric** representation of the string in base 10, if it comprises of digits **only**.
- The **length** of the string, otherwise.

Given an array `strs` of alphanumeric strings, return *the maximum value of any string in `strs`*.

Example 1:

Input: `strs = ["alic3", "bob", "3", "4", "00000"]`

Output: 5

Explanation:

- "alic3" consists of both letters and digits, so its value is its length, i.e. 5.
- "bob" consists only of letters, so its value is also its length, i.e. 3.
- "3" consists only of digits, so its value is its numeric equivalent, i.e. 3.
- "4" also consists only of digits, so its value is 4.
- "00000" consists only of digits, so its value is 0.

Hence, the maximum value is 5, of "alic3".

Example 2:

Input: `strs = ["1", "01", "001", "0001"]`

Output: 1

Explanation:

Each string in the array has value 1. Hence, we return 1.

Constraints:

- $1 \leq \text{strs.length} \leq 100$
- $1 \leq \text{strs[i].length} \leq 9$

- `strs[i]` consists of only lowercase English letters and digits.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string[]} strs  
 * @return {number}  
 */  
var maximumValue = function (strs) {  
    let max = 0;  
    for (let i = 0; i < strs.length; i++) {  
        if (/^\d+$/.test(strs[i])) {  
            toNumber = parseInt(strs[i]);  
        } else {  
            toNumber = strs[i].length;  
        }  
        if (toNumber > max) {  
            max = toNumber;  
        }  
    }  
    // console.log(max);  
    return max;  
};  
// maximumValue(["alic3", "bob", "3", "4", "r00000"]);
```

7 Reverse Integer ([link](#))

Description

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

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

Example 1:

```
Input: x = 123
Output: 321
```

Example 2:

```
Input: x = -123
Output: -321
```

Example 3:

```
Input: x = 120
Output: 21
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} x  
 * @return {number}  
 */  
var reverse = function (x) {  
    let result = x.toString().split("").reverse();  
  
    for (let i = 0; i < result.length; i++) {  
        if (result[0] == "0") {  
            result.shift();  
        }  
    }  
  
    if (result[result.length - 1] == "-") {  
        result.pop();  
        result.unshift("-");  
    }  
    result = result.join("");  
    if(result <= -2147483651 || result >= 2147483651){  
        return 0;  
    }  
    return result  
};  
  
// reverse(-124000);
```

4 Median of Two Sorted Arrays ([link](#))

Description

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

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

Example 1:

Input: `nums1 = [1,3]`, `nums2 = [2]`

Output: `2.00000`

Explanation: merged array = `[1,2,3]` and median is 2.

Example 2:

Input: `nums1 = [1,2]`, `nums2 = [3,4]`

Output: `2.50000`

Explanation: merged array = `[1,2,3,4]` and median is $(2 + 3) / 2 = 2.5$.

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number}
 */
var findMedianSortedArrays = function (nums1, nums2) {
    let numsMerge = [...nums1, ...nums2];

    let nums = numsMerge.sort(function (a, b) {
        return a - b;
    });
    var mid = 0;

    if (nums.length % 2 == 0) {
        mid = [nums[[nums.length / 2] - 1] + nums[nums.length / 2]] / 2;
    } else if (nums.length % 2 != 0) {
        mid = nums[[nums.length - 1] / 2];
    }
    // console.log(mid);
    return mid
};
```

[2630 Alternating Digit Sum \(link\)](#)

Description

You are given a positive integer n . Each digit of n has a sign according to the following rules:

- The **most significant digit** is assigned a **positive** sign.
- Each other digit has an opposite sign to its adjacent digits.

Return *the sum of all digits with their corresponding sign*.

Example 1:

```
Input: n = 521
Output: 4
Explanation: (+5) + (-2) + (+1) = 4.
```

Example 2:

```
Input: n = 111
Output: 1
Explanation: (+1) + (-1) + (+1) = 1.
```

Example 3:

```
Input: n = 886996
Output: 0
Explanation: (+8) + (-8) + (+6) + (-9) + (+9) + (-6) = 0.
```

Constraints:

- $1 \leq n \leq 10^9$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var alternateDigitSum = function(n) {  
    let sumPosi =0;  
    let arr = n.toString().split("");  
  
    for(let i=0; i<arr.length; i++){  
        if(i%2==0){  
            sumPosi += +arr[i]  
        }else{  
            sumPosi -= +arr[i]  
        }  
    }  
    return sumPosi;  
};
```

34 Find First and Last Position of Element in Sorted Array ([link](#))

Description

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

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var searchRange = function (nums, target) {
    let output = [-1,-1];
    for(let i=0; i<nums.length; i++){
        if(nums[i]==target){
            output[0]=i
            break;
        }
    }
    for(let j=nums.length-1; j>=0; j--){
        if(nums[j]==target){
            output[1]=j
            break;
        }
    }
    return output
};
// searchRange([5, 7, 7, 8, 8, 10], 8);
```

81 Search in Rotated Sorted Array II ([link](#))

Description

There is an integer array `nums` sorted in non-decreasing order (not necessarily with **distinct** values).

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

Given the array `nums` **after** the rotation and an integer target, return true *if target is in* `nums`, or false *if it is not in* `nums`.

You must decrease the overall operation steps as much as possible.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- `nums` is guaranteed to be rotated at some pivot.
- $-10^4 \leq \text{target} \leq 10^4$

Follow up: This problem is similar to [Search in Rotated Sorted Array](#), but `nums` may contain **duplicates**. Would this affect the runtime complexity? How and why?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @param {number} target
 * @return {boolean}
 */
var search = function(nums, target) {
    let arr = nums.sort(function(a,b){ return a-b})
    let start =0
    let end = arr.length-1;

    while(start<=end){
        let mid = parseInt((start+end)/2)
        if(arr[mid]==target){
            return true
        }else if(arr[mid]<target){
            start = mid+1
        }else {
            end = mid-1
        }

    }
    return false
};
```

1476 Count Negative Numbers in a Sorted Matrix ([link](#))

Description

Given a $m \times n$ matrix `grid` which is sorted in non-increasing order both row-wise and column-wise, return *the number of negative numbers in grid*.

Example 1:

```
Input: grid = [[4,3,2,-1],[3,2,1,-1],[1,1,-1,-2],[-1,-1,-2,-3]]  
Output: 8  
Explanation: There are 8 negatives number in the matrix.
```

Example 2:

```
Input: grid = [[3,2],[1,0]]  
Output: 0
```

Constraints:

- $m == \text{grid.length}$
- $n == \text{grid}[i].length$
- $1 \leq m, n \leq 100$
- $-100 \leq \text{grid}[i][j] \leq 100$

Follow up: Could you find an $O(n + m)$ solution?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[][]} grid  
 * @return {number}  
 */  
var countNegatives = function (grid) {  
    let arr = [];  
    let neg = 0;  
    for (let element of grid) {  
        arr.push(...element);  
    }  
    // console.log(arr);  
    for (let elem of arr) {  
        if (elem < 0) {  
            neg++;  
        }  
    }  
    return neg;  
};
```

2624 Difference Between Element Sum and Digit Sum of an Array ([link](#))

Description

You are given a positive integer array `nums`.

- The **element sum** is the sum of all the elements in `nums`.
- The **digit sum** is the sum of all the digits (not necessarily distinct) that appear in `nums`.

Return *the absolute difference between the element sum and digit sum of `nums`*.

Note that the absolute difference between two integers x and y is defined as $|x - y|$.

Example 1:

```
Input: nums = [1,15,6,3]
Output: 9
Explanation:
The element sum of nums is 1 + 15 + 6 + 3 = 25.
The digit sum of nums is 1 + 1 + 5 + 6 + 3 = 16.
The absolute difference between the element sum and digit sum is |25 - 16| = 9.
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: 0
Explanation:
The element sum of nums is 1 + 2 + 3 + 4 = 10.
The digit sum of nums is 1 + 2 + 3 + 4 = 10.
The absolute difference between the element sum and digit sum is |10 - 10| = 0.
```

Constraints:

- $1 \leq \text{nums.length} \leq 2000$

- `1 <= nums[i] <= 2000`

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var differenceOfSum = function (nums) {  
    let sumEle = 0;  
    for (let i = 0; i < nums.length; i++) {  
        sumEle = sumEle + nums[i];  
    }  
    let sumDigit = 0;  
    let nums1 = nums.join("").split("");  
    for (let j = 0; j < nums1.length; j++) {  
        sumDigit += parseInt(+nums1[j]);  
    }  
    let difference = sumEle - sumDigit;  
    // console.log(difference);  
    return difference;  
};  
differenceOfSum([1, 15, 6, 3]);
```

[792 Binary Search \(link\)](#)

Description

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

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

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number}  
 */  
var search = function(nums, target) {  
  
    let start = 0;  
    let end = nums.length-1;  
    while(start<=end){  
        let mid =parseInt((start + end )/2);  
        if(nums[mid]==target){  
            return mid;  
        }else if(nums[mid]<target){  
            start = mid + 1;  
        }else{  
            end = mid - 1;  
        }  
    }  
    return -1;  
};
```

2614 Maximum Count of Positive Integer and Negative Integer ([link](#))

Description

Given an array `nums` sorted in **non-decreasing** order, return *the maximum between the number of positive integers and the number of negative integers*.

- In other words, if the number of positive integers in `nums` is `pos` and the number of negative integers is `neg`, then return the maximum of `pos` and `neg`.

Note that `0` is neither positive nor negative.

Example 1:

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

Output: `3`

Explanation: There are 3 positive integers and 3 negative integers. The maximum count among them is 3.

Example 2:

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

Output: `3`

Explanation: There are 2 positive integers and 3 negative integers. The maximum count among them is 3.

Example 3:

Input: `nums = [5, 20, 66, 1314]`

Output: `4`

Explanation: There are 4 positive integers and 0 negative integers. The maximum count among them is 4.

Constraints:

- $1 \leq \text{nums.length} \leq 2000$
- $-2000 \leq \text{nums}[i] \leq 2000$
- **nums is sorted in a non-decreasing order.**

Follow up: Can you solve the problem in $O(\log(n))$ time complexity?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var maximumCount = function (nums) {  
    let pos = 0;  
    let neg = 0;  
    for (let i = 0; i < nums.length; i++) {  
        if (nums[i] > 0) {  
            pos++;  
        } else if (nums[i] < 0) {  
            neg++;  
        }  
    }  
    if (pos > neg) {  
        return pos  
    } else if (pos < neg) {  
        return neg  
    } else {  
        return pos  
    }  
};
```

[154 Find Minimum in Rotated Sorted Array II \(link\)](#)

Description

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

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

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

Given the sorted rotated array `nums` that may contain **duplicates**, return *the minimum element of this array*.

You must decrease the overall operation steps as much as possible.

Example 1:

```
Input: nums = [1,3,5]
Output: 1
```

Example 2:

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

Constraints:

- $n == \text{nums.length}$
- $1 \leq n \leq 5000$
- $-5000 \leq \text{nums}[i] \leq 5000$
- `nums` is sorted and rotated between 1 and n times.

Follow up: This problem is similar to [Find Minimum in Rotated Sorted Array](#), but `nums` may contain **duplicates**. Would this affect the runtime complexity? How and why?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var findMin = function (nums) {  
    nums.sort(function (a, b) {  
        return a - b;  
    });  
    // console.log(nums[0]);  
    return nums[0]  
};
```

[268 Missing Number \(link\)](#)

Description

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

Example 1:

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

```
Output: 2
```

```
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it o
```

Example 2:

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

```
Output: 2
```

```
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it o
```

Example 3:

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

```
Output: 8
```

```
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it o
```

Constraints:

- $n == \text{nums.length}$

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @return {number}  
 */  
var missingNumber = function(nums) {  
    nums = nums.sort((a,b) => a-b)  
  
    for(let i=0;i<nums.length;i++){  
  
        if(i != nums[i]){  
            return i  
        }  
    }  
  
    return nums.length  
};  
// missingNumber([2, 5, 1, 3]);
```

[263 Ugly Number \(link\)](#)

Description

An **ugly number** is a positive integer whose prime factors are limited to 2, 3, and 5.

Given an integer n , return `true` if n is an **ugly number**.

Example 1:

```
Input: n = 6
Output: true
Explanation: 6 = 2 × 3
```

Example 2:

```
Input: n = 1
Output: true
Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.
```

Example 3:

```
Input: n = 14
Output: false
Explanation: 14 is not ugly since it includes the prime factor 7.
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {boolean}  
 */  
var isUgly = function (n) {  
    if (n == 1) {  
        return true  
    } else if (n == 0) {  
        return false  
    }  
    // check n is devideable by 2,3,5 untill n==1 , when n==1 then result come out true, otherwise result will be false =====>  
    while (n != 1) {  
        if (n % 2 == 0) {  
            n = n / 2  
        } else if (n % 3 == 0) {  
            n = n / 3  
        } else if (n % 5 == 0) {  
            n = n / 5  
        } else {  
            return false  
        }  
    }  
    return true  
};
```

344 Reverse String_(link)

Description

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

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

Example 1:

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

Example 2:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {character[]} s  
 * @return {void} Do not return anything, modify s in-place instead.  
 */  
var reverseString = function(s) {  
    s.reverse()  
};
```

[231 Power of Two \(link\)](#)

Description

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

An integer n is a power of two, if there exists an integer x such that $n == 2^x$.

Example 1:

```
Input: n = 1
Output: true
Explanation: 20 = 1
```

Example 2:

```
Input: n = 16
Output: true
Explanation: 24 = 16
```

Example 3:

```
Input: n = 3
Output: false
```

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number} n
 * @return {boolean}
 */

// 2 Type Solutions Answer =====>>>>
var isPowerOfTwo = function (n) {
    if (n < 1) {
        return false
    }
    if (n == 1) {
        return true
    }
    if (n % 2 == 1) {
        return false
    }
    return isPowerOfTwo(n / 2)
}

// 2nd Solutions =====>>>>>>>>
// var isPowerOfTwo = function (n) {
//     let twoPowerArr = [1];
//     let num = 2;
//     let result = false;
//     for (let i = 1; i <= n / 3 + 1; i++) {
//         var twoPower = 1;
//         for (let j = 1; j <= i; j++) {
//             twoPower = twoPower * num;
//         }
//         twoPowerArr.push(twoPower);
//     }
// }
```

```
// }  
// for (let k = 0; k < twoPowerArr.length; k++) {  
//   if (twoPowerArr[k] == n) {  
//     result = true;  
//   }  
// }  
// return result  
// };  
// // isPowerOfTwo(27);
```

[342 Power of Four \(link\)](#)

Description

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

An integer n is a power of four, if there exists an integer x such that $n == 4^x$.

Example 1:

```
Input: n = 16
Output: true
```

Example 2:

```
Input: n = 5
Output: false
```

Example 3:

```
Input: n = 1
Output: true
```

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number} n
 * @return {boolean}
 */
// 2 Type Solutions Answer =====>>>>
var isPowerOfFour = function(n) {
    if (n < 1) {
        return false
    }
    if (n == 1) {
        return true
    }
    if (n % 4 == 1) {
        return false
    }
    return isPowerOfFour(n / 4)
}

// 
// 
// 
// 2nd Solutions =====>>>>>>>>
// var isPowerOfFour = function (n) {
//     let num = 4;
//     let fourPowerArr = [1];
//     let result = false;
//     for (let i = 1; i <= n / 3 + 1; i++) {
//         var fourPower = 1;
//         for (let j = 1; j <= i; j++) {
```

```
//        fourPower = fourPower * num;
//    }
//    fourPowerArr.push(fourPower);
//
//}
// console.log(fourPowerArr, "fourPower Arr");
for (let k = 0; k < fourPowerArr.length; k++) {
    if (fourPowerArr[k] == n) {
        result = true;
    }
}
// console.log(result);
return result
};

isPowerOfFour(27);
```

326 Power of Three (link)

Description

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

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

Example 1:

```
Input: n = 27
Output: true
Explanation: 27 = 33
```

Example 2:

```
Input: n = 0
Output: false
Explanation: There is no x where 3x = 0.
```

Example 3:

```
Input: n = -1
Output: false
Explanation: There is no x where 3x = (-1).
```

Constraints:

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

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {boolean}  
 */  
var isPowerOfThree = function(n) {  
    if(n==0) return false  
    while(n%3 ===0){n= n/3}  
    return n==1  
};
```

[228 Summary Ranges \(link\)](#)

Description

You are given a **sorted unique** integer array `nums`.

A **range** $[a, b]$ is the set of all integers from a to b (inclusive).

Return *the smallest sorted list of ranges that cover all the numbers in the array exactly*. That is, each element of `nums` is covered by exactly one of the ranges, and there is no integer x such that x is in one of the ranges but not in `nums`.

Each range $[a, b]$ in the list should be output as:

- " $a \rightarrow b$ " if $a \neq b$
- " a " if $a == b$

Example 1:

```
Input: nums = [0,1,2,4,5,7]
Output: ["0->2", "4->5", "7"]
Explanation: The ranges are:
[0,2] --> "0->2"
[4,5] --> "4->5"
[7,7] --> "7"
```

Example 2:

```
Input: nums = [0,2,3,4,6,8,9]
Output: ["0", "2->4", "6", "8->9"]
Explanation: The ranges are:
[0,0] --> "0"
[2,4] --> "2->4"
[6,6] --> "6"
[8,9] --> "8->9"
```

Constraints:

- $0 \leq \text{nums.length} \leq 20$
- $-2^{31} \leq \text{nums}[i] \leq 2^{31} - 1$
- All the values of `nums` are **unique**.
- `nums` is sorted in ascending order.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {string[]}
 */
var summaryRanges = function (nums) {
  let arr = [];
  let renge = [];
  for (let i = 0; i < nums.length; i++) {
    if (nums[i] + 1 === nums[i + 1]) {
      arr.push(nums[i]);
    } else if (nums[i] + 1 !== nums[i + 1]) {
      arr.push(nums[i]);
      renge.push(arr);
      arr = [];
    } else {
      renge.push(nums[i]);
    }
  }
  // console.log(renge, "ren");
  for (let j = 0; j < renge.length; j++) {
    if (renge[j].length > 1) {
      renge[j] = renge[j][0] + "->" + renge[j][renge[j].length - 1];
    } else if ((renge[j].length = 1)) {
      renge[j] = `${renge[j]}`;
    }
  }
  return renge;
  // console.log(renge, "rengeee");
};
```

[414 Third Maximum Number \(link\)](#)

Description

Given an integer array `nums`, return *the third distinct maximum number in this array. If the third maximum does not exist, return the maximum number.*

Example 1:

```
Input: nums = [3,2,1]
Output: 1
Explanation:
The first distinct maximum is 3.
The second distinct maximum is 2.
The third distinct maximum is 1.
```

Example 2:

```
Input: nums = [1,2]
Output: 2
Explanation:
The first distinct maximum is 2.
The second distinct maximum is 1.
The third distinct maximum does not exist, so the maximum (2) is returned instead.
```

Example 3:

```
Input: nums = [2,2,3,1]
Output: 1
Explanation:
The first distinct maximum is 3.
The second distinct maximum is 2 (both 2's are counted together since they have the same value).
The third distinct maximum is 1.
```

Constraints:

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

Follow up: Can you find an $O(n)$ solution?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {number}
 */
var thirdMax = function (nums) {
    nums.sort(function (a, b) {
        return a - b;
    });
    // console.log(nums, "before");

    for (let i = 0; i < nums.length; i++) {
        for (let j = nums.length - 1; j >= i + 1; j--) {
            if (nums[i] == nums[j]) {
                nums.splice(j, 1);
            }
        }
    }
    let thirdMaxElm = nums[nums.length - 3];
    if (thirdMaxElm == undefined) {
        thirdMaxElm = nums[nums.length - 1];
    } else if (nums[nums.length - 2] == undefined) {
        thirdMaxElm = nums[nums.length - 1];
    } else if (nums[nums.length - 1] == undefined) {
        thirdMaxElm = 0;
    }
    // console.log(nums, "after");
    // console.log(thirdMaxElm);
    return thirdMaxElm;
};
// thirdMax([2, 4, 5, 10, 6, 2, 8, 7]);
```

[258 Add Digits \(link\)](#)

Description

Given an integer `num`, repeatedly add all its digits until the result has only one digit, and return it.

Example 1:

```
Input: num = 38
Output: 2
Explanation: The process is
38 --> 3 + 8 --> 11
11 --> 1 + 1 --> 2
Since 2 has only one digit, return it.
```

Example 2:

```
Input: num = 0
Output: 0
```

Constraints:

- $0 \leq \text{num} \leq 2^{31} - 1$

Follow up: Could you do it without any loop/recursion in $O(1)$ runtime?

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {number}  
 */  
var addDigits = function (num) {  
    let arr = num.toString().split("");  
    while (arr.length != 1) {  
        let sum = arr.reduce((a, b) => Number(a) + Number(b), 0);  
        arr = sum.toString().split("");  
    }  
    return (Number(arr[0]));  
};
```

[415 Add Strings \(link\)](#)

Description

Given two non-negative integers, `num1` and `num2` represented as string, return *the sum of num1 and num2 as a string*.

You must solve the problem without using any built-in library for handling large integers (such as `BigInteger`). You must also not convert the inputs to integers directly.

Example 1:

```
Input: num1 = "11", num2 = "123"
Output: "134"
```

Example 2:

```
Input: num1 = "456", num2 = "77"
Output: "533"
```

Example 3:

```
Input: num1 = "0", num2 = "0"
Output: "0"
```

Constraints:

- $1 \leq \text{num1.length}, \text{num2.length} \leq 10^4$
- `num1` and `num2` consist of only digits.
- `num1` and `num2` don't have any leading zeros except for the zero itself.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {string} num1  
 * @param {string} num2  
 * @return {string}  
 */  
var addStrings = function (num1, num2) {  
    let sum = BigInt(num1) + BigInt(num2);  
    let result = "" + sum;  
    return result  
};
```

[202 Happy Number \(link\)](#)

Description

Write an algorithm to determine if a number n is happy.

A **happy number** is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
- Those numbers for which this process **ends in 1** are happy.

Return `true` if n is a happy number, and `false` if not.

Example 1:

```
Input: n = 19
Output: true
Explanation:
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

Example 2:

```
Input: n = 2
Output: false
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {boolean}  
 */  
const isHappy = (n) => {  
    let temp = n;  
  
    for (let i = 0; i < 10000; i++) {  
        let str = String(temp);  
        let sum = 0;  
  
        for (let j = 0; j < str.length; j++) {  
            let num = +str[j];  
            let square = num * num;  
            sum += square;  
        }  
        if (sum === 1) return true;  
        temp = sum;  
    }  
    return false;  
};
```

[136 Single Number \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {number[]} nums
 * @return {number}
 */
var singleNumber = function(nums) {
    let result = {}
    for(let i of nums){
        result[i]?delete result[i]:result[i]=1;
    }
    return Object.keys(result)[0]
};

/*
const singleNumber = (nums) => {
    let storeObj = {};

    for (let i = 0; i < nums.length; i++) {
        storeObj[nums[i]] = (storeObj[nums[i]] || 0) + 1;
    }

    for (let key in storeObj) {
        if (storeObj[key] === 1) {
            return key;
        }
    }
};
*/

```

2608 Count the Digits That Divide a Number ([link](#))

Description

Given an integer `num`, return *the number of digits in num that divide num*.

An integer `val` divides `nums` if `nums % val == 0`.

Example 1:

```
Input: num = 7
Output: 1
Explanation: 7 divides itself, hence the answer is 1.
```

Example 2:

```
Input: num = 121
Output: 2
Explanation: 121 is divisible by 1, but not 2. Since 1 occurs twice as a digit, we return 2.
```

Example 3:

```
Input: num = 1248
Output: 4
Explanation: 1248 is divisible by all of its digits, hence the answer is 4.
```

Constraints:

- $1 \leq \text{num} \leq 10^9$
- `num` does not contain 0 as one of its digits.

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} num  
 * @return {number}  
 */  
  
var countDigits = function (num) {  
    let val = num;  
    let numStr = "" + num;  
    let count = 0;  
    for (let i = 0; i < numStr.length; i++) {  
        if (num % numStr[i] === 0) {  
            count++;  
        }  
    }  
    return count;  
};  
// countDigits(num);
```

70 Climbing Stairs (link)

Description

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

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

Example 1:

Input: $n = 2$

Output: 2

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

1. 1 step + 1 step
2. 2 steps

Example 2:

Input: $n = 3$

Output: 3

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

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

Constraints:

- $1 \leq n \leq 45$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var climbStairs = function (n) {  
    var step = [0, 1, 2];  
    for (let i = 3; i <= n; i++) {  
        step[i] = step[i - 1] + step[i - 2];  
    }  
    return step[n];  
};  
climbStairs(5);
```

[69 Sqrt\(x\)_\(link\)](#)

Description

Given a non-negative integer x , return *the square root of x rounded down to the nearest integer*. The returned integer should be **non-negative** as well.

You **must not use** any built-in exponent function or operator.

- For example, do not use `pow(x, 0.5)` in c++ or `x ** 0.5` in python.

Example 1:

```
Input: x = 4
Output: 2
Explanation: The square root of 4 is 2, so we return 2.
```

Example 2:

```
Input: x = 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.
```

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number} x  
 * @return {number}  
 */  
var mySqrt = function(x) {  
    let result = 1;  
    while (result * result <= x) {  
        result++;  
    }  
    return result - 1;  
//    console.log(result - 1);  
};
```

[66 Plus One \(link\)](#)

Description

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

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} digits  
 * @return {number[]}()  
 */  
  
const plusOne = (digits) => {  
  
    digits[digits.length - 1]++;  
  
    for (let i = digits.length - 1; i > 0 && digits[i] === 10; i--) {  
        digits[i] = 0;  
        digits[i - 1]++;  
    }  
    if (digits[0] === 10) {  
        digits[0] = 0;  
        digits.unshift(1);  
    }  
    return digits;  
//    console.log(digits);  
};  
  
plusOne([9, 9, 9]);
```

[35 Search Insert Position \(link\)](#)

Description

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

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

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} target  
 * @return {number}  
 */  
var searchInsert = function(nums, target) {  
    for (let i = 0; i < nums.length; i++) {  
  
        if (nums[i] == target) {  
            return i;  
        }  
        if (nums[i] > target) {  
            return i;  
        } else if (i == nums.length - 1) {  
            return nums.length;  
        }  
    }  
};  
// searchInsert([1,3,5,6],5)
```

[27 Remove Element \(link\)](#)

Description

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in `nums` which are not equal to `val`*.

Consider the number of elements in `nums` which are not equal to `val` 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 elements which are not equal to `val`. 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 val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
                           // It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

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

Example 1:

```
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_,_]
```

Explanation: Your function should return $k = 2$, with the first two elements of `nums` being 2. It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

Input: `nums` = [0,1,2,2,3,0,4,2], `val` = 2
Output: 5, `nums` = [0,1,4,0,3,_,_,_]

Explanation: Your function should return $k = 5$, with the first five elements of `nums` containing 0, 0, 1, 3, and 4. Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/**  
 * @param {number[]} nums  
 * @param {number} val  
 * @return {number}  
 */  
var removeElement = function(nums, val) {  
  
    // for (let i = 0; i < nums.length; i++) {  
    for (let i = nums.length - 1; i >= 0; i--) {  
        if (nums[i] == val) {  
            nums.splice(i, 1);  
        }  
    }  
    console.log(nums);  
};  
removeElement([4, 4, 4, 4], 4);
```

[13 Roman to Integer \(link\)](#)

Description

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

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

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply x + II. The number 27 is written as XXVII, which is xx + V + II.

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

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

Given a roman numeral, convert it to an integer.

Example 1:

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

Example 2:

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

Example 3:

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

Constraints:

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

(scroll down for solution)

Solution

Language: javascript

Status: Accepted

```
/*
 * @param {string} s
 * @return {number}
 */
var roman = {
    "I": 1,
    "V": 5,
    "X": 10,
    "L": 50,
    "C": 100,
    "D": 500,
    "M": 1000
}

var romanToInt = function(s) {
    var result = 0;
    for(let i=0; i<s.length; i+=1){
        roman[s[i]] < roman[s[i+1]] ? result -= roman[s[i]]: result += roman[s[i]]
    }
    return result;
};
```

