## ЗАДАЧИ НА АЛГОРИМНИЗАЦИЮ от ALEXGLEKOV

## Find an area of the intersection of two rectangles

class Rect {

double x, y, width, height;

// you can use x1, y1, x2, y2 properties, too

}

**double intersectionArea(Rect r1, Rect r2)**

**JS:**

class Rectangle{

    constructor(x1,y1,x2,y2){

        this.x1 = x1

        this.x2 = x2

        this.y1 = y1

        this.y2 = y2

    }

}

const intersection = (rec1, rec2) =>{

    let left = Math.max(rec1.x1, rec2.x1)

    let top = Math.min(rec1.y2,rec2.y2)

    let rigth = Math.min(rec1.x2, rec2.x2)

    let bottom = Math.max(rec1.y1, rec2.y1)

    let width = rigth - left

    let heigth = top - bottom

    if(width < 0 || heigth < 0){

        return 0

    }

    return width \* heigth

}

console.log(intersection(new Rectangle(5,2,10,10), new Rectangle(2,7,7,15)))

## Palindrome check

Write an **efficient** function that checks whether **any permutation** of an input string is a palindrome. Note that the function is not a palindrome check (see example 2).

Examples:

1. "civic" should return True
2. "ivicc" should return True (because “civic” is a permutation of “ivicc”)
3. "civil" should return False
4. "livci" should return False

**bool isPalindromable(string s)**

**JS:**

const palindromeCheck = (string) =>{

    let temp = string

    let temp2 = null

    let mass = []

    for(i = string.length-1; i > -1; i--){

        mass.push(string[i])

    }

    temp2 = mass.join().split(',').join("")

    return temp === temp2

}

## Duplicates

In an array of integers of length n + 1 (n > 1), every number in the range 1...n appears once except for one number which appears twice (so the array’s length is n+1). Write an efficient function that finds the number that appears twice.

*Examples:*

1,2,2,3 => 2

1,2,3,3 => 3

2,1,4,3,5,4 => 4

**int findDuplicate(int[] values)**

**JS:**

const findDuplicate = (arr) =>{

    for(let i = 0; i < arr.length; i++){

        let temp = arr[i]

        delete arr[i]

        for(let j = 0; j < arr.length; j++){

            if(temp === arr[j]){

                return temp

            }

            else{

                continue

            }

        }

        arr[i] = temp

    }

}

## Intersection

Given two arrays of numbers where each one contains unique values and is already sorted in ascending order, find the number of elements that belong to both arrays.

**int intersectionCount(int[] a, int[] b)**

**JS:**

const intersectionCount = (a,b) =>{

    let counter = 0

    if(a.length > b.length){

        for(let i = 0; i < a.length; i++){

            for(let j = 0; j < b.length; j++){

                if(a[i] === b[j]){

                    counter++

                }else{

                    continue

                }

            }

        }

    }else{

        for(let i = 0; i < b.length; i++){

            for(let j = 0; j < a.length; j++){

                if(b[i] === a[j]){

                    counter++

                }else{

                    continue

                }

            }

        }

    }

    return counter

}

## RLE encoder

Given a string of letters (without numbers), create a string encoding it by the rules where the first character is char itself, followed by a number indicating the number of letter repeats.

*Examples:*

ABBA => A1B2A1

ATTTGC => A1T3G1C1

**string rle(string s)**

**JS:**

const encoder = (string) =>{

    let arrCounter = []

    let arrLetters = []

    let temp = ''

    for(let i = 0; i < string.length; i++){

        let counter = 0

        for(let j = 0; j < string.length; j++){

            if(string[i] === string[j]){

                counter++

            }

        }

        arrCounter.push(counter)

        arrLetters.push(string[i])

    }

    for(let i = 0; i < arrLetters.length; i++){

        temp = temp + arrLetters[i] + arrCounter[i]

    }

    return temp

}

## Search insert position

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

*Examples:*

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

Input: values = [1], target = 0. Output: 0

**int findIndex(int[] values, int target)**

**JS:**

const findIndex = (arr, target) =>{

    for(let i = 0; i < arr.length; i++){

        if(arr[i] === target){

            return arr.indexOf(target)

        }

    }

    if(target > arr[arr.length-1]){

        return arr.indexOf(arr[arr.length-1])+1

    }

    for(let i = 0; i < arr.length; i++){

        if(target > arr[i]){

                //continue

        }else{

             return arr.indexOf(arr[i])

        }

    }

}