Roll Number : 238W1A1266

Week – 1

Post Labs :

1. 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.

Ans :

class Solution {

    static int singleNumber(int[] nums) {

        int xor = 0;

        for(int i = 0;i<nums.length;i++){

            xor = xor ^ nums[i];

        }

        return xor;

    }

}

Output :



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

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

* 0 <= j <= nums[i] and
* i + j < n

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

 Ans : public int jump(int[] nums) {

        int n = nums.length;

        if (n == 1) return 0;

        int jumps = 0;

        int farthest = 0;

        int currentEnd = 0;

        for (int i = 0; i < n - 1; i++) {

            farthest = Math.max(farthest, i + nums[i]);

            if (i == currentEnd) {

                jumps++;

                currentEnd = farthest;

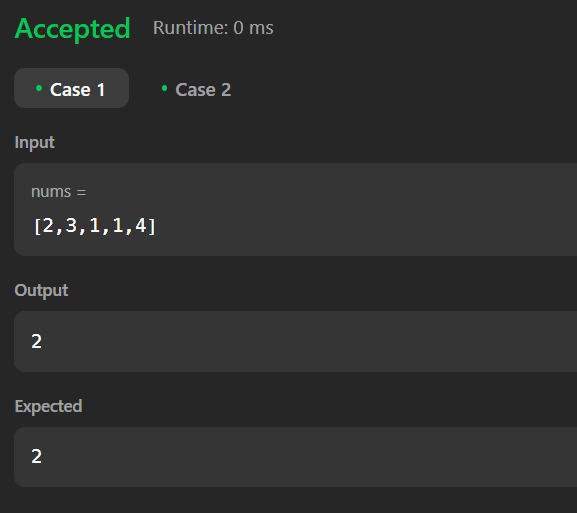
                if (currentEnd >= n - 1) break;

            }

        }

        return jumps;

    }



3. Given an integer array nums sorted in **non-decreasing order**, remove some duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each unique element appears **at most twice**. The **relative order** of the elements should be kept the **same**.

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

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

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

Ans :

public int removeDuplicates(int[] nums) {

        int n = nums.length;

        if (n <= 2) return n;

        int index = 2;

        for (int i = 2; i < n; i++) {

            if (nums[i] != nums[index - 2]) {

                nums[index] = nums[i];

                index++;

            }

        }

        return index;

    }

