Bonus Question ☺

Problem #1: Binary Search

```
Given an integer array nums, return the number of triplets chosen from the array that
can make triangles if we take them as side lengths of a triangle.

Example 1:

Input: nums = [2,2,3,4]
Output: 3
Explanation: Valid combinations are:
2,3,4 (using the first 2)
2,3,4 (using the second 2)
2,2,3

Example 2:

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

Constraints:

1 <= nums.length <= 1000
0 <= nums[i] <= 1000

/**
 * @param {number[]} nums
 * @return {number}
 */
cibst triangleNumber = (nums) => {

};
```

Problem #2: Frequency Counter Pattern

```
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 extra space.

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 <= n <= 105
1 <= nums[i] <= n
Each element in nums appears once or twice.

/**
 * @param {number[]} nums
 * @return {number[]}
 */
const findDuplicates = (nums) => {

};
```