# 2529. Maximum Count of Positive Integer and Negative Integer

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 <= nums.length <= 2000`
- `-2000 <= nums[i] <= 2000`
- `nums` is sorted in a **non-decreasing order**.

**Follow up:** Can you solve the problem in `O(log(n))` time complexity?

# 2529. Maximum Count of Positive Integer and Negative Integer

```
/*
   Two passes Binary search
   Time complexity: O(2logn)
   Space complexity: O(1)
*/
class Solution {
   public:
      int maximumCount(vector<int>& nums){
         int n=nums.size();

         // pos_1st_0: will contains
         // - the position of the 1st zero, if zero exists
         // - the position of the 1st positive intger, if zero does not exist
         int pos_1st_0=std::lower_bound(nums.begin(),nums.end(),0)-nums.begin();

         // pos_1st_0==n, means zero does not exists or all integer are negative: max(n,n-n)=n
         // nums[pos_1st_0]!=0, means zero does not exist, but we have the position of the first
         // positive integer:
         // pos_1st_0: is the number of negative integers
         // n-pos_1st_0: is the number of positive integers.
         if(pos_1st_0 == n || nums[pos_1st_0]!=0) return std::max(pos_1st_0,n-pos_1st_0);

         // If the arrays contains at least a zero, in this case:
         // search the position of the last zero
         int pos_last_0=std::upper_bound(nums.begin(),nums.end(),0)-nums.begin();

         // pos_1st_0: is the number of negative integers
         // n-pos_last_0: is the number of positive integers.
         return  std::max(pos_1st_0,n-pos_last_0);
      }
};
```