Leet code: 136.Single Number

Problem statement

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]

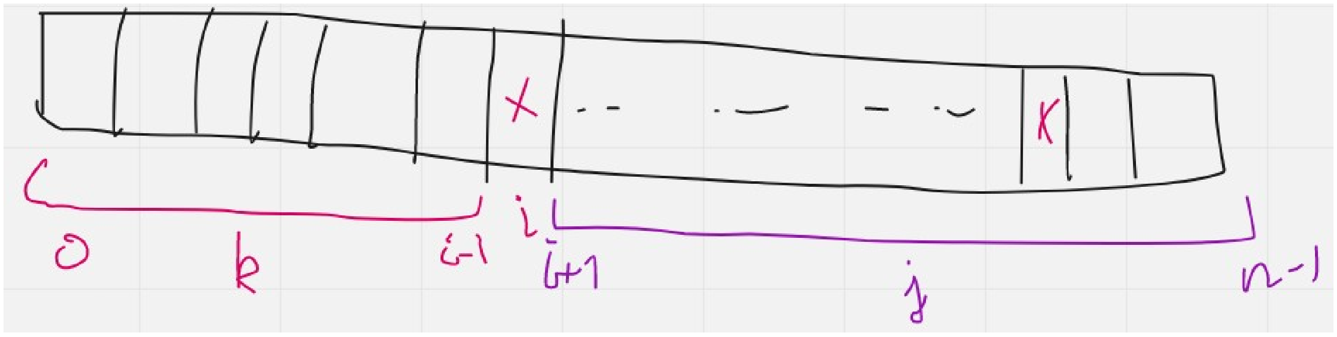
Output: 1

Constraints:

- `1 <= nums.length <= 3 * 10`$_4$
- `-3 * 10`$_4$ `<= nums[i] <= 3 * 10`$_4$
- Each element in the array appears twice except for one element which appears only once.

Brute force

for every number in the array, check if it exists ***in the right side of the array*** **and** ***in the left side of the array.***



C++ >= 11
```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0 ; i < n ; ++i){
            int j = i+1;
            while (j < n && nums[i] != nums[j]) j++;
            int k = i-1;
            while (k >= 0 && nums[i] != nums[k]) k--;

            if (j == n && k == -1) return nums[i];
        }

        return -1;
    }
};
```

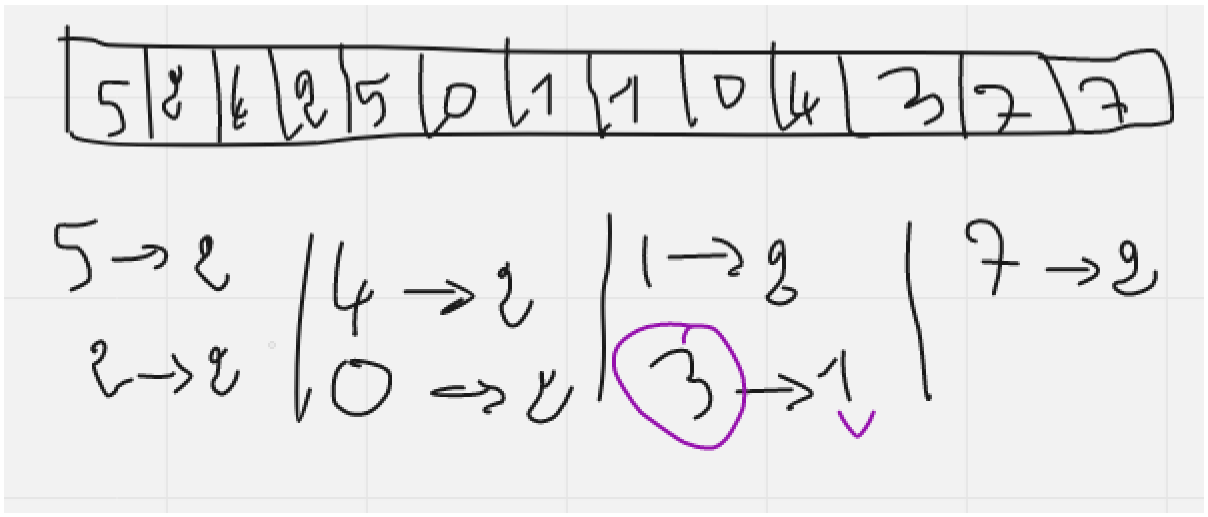
Time complexity: $O(n^2)$

Auxiliary space complexity: $O(1)$

Frequency array

Using a frequency array to **count the occurrence of each number**, then **iterate over that frequency array, if we find a number that appears once, then take it**.

Example:



C++ >= 11

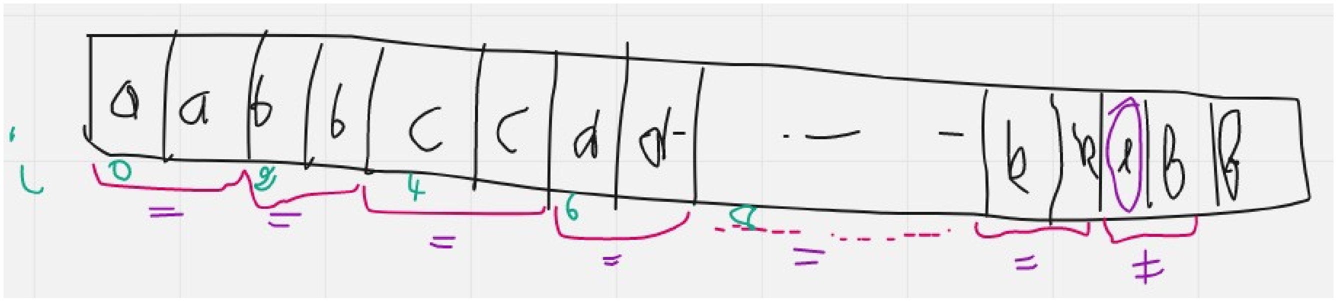
```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        unordered_map<int,int> f;
        for(auto x: nums){
            f[x]++;
        }
        for (auto x: f){
            if (x.second == 1) return x.first;
        }
        return -1;
    }
};
```

Time complexity: O(n)

Auxiliary space complexity: O(n)

Sorting

Sort the array, and **check the numbers every two adjacent numbers, if we find two numbers not equal we take the first one**.



C++ >= 11
```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(),nums.end());
        int i = 0;
        while (i < n-1 && nums[i] == nums[i+1]) i += 2;
        if (i <= n-1) return nums[i];
        return -1;
    }
};
```

Time complexity: O(n log n) + O(n)

Auxiliary space complexity: O(1)

Xor

Properties

https://en.wikipedia.org/wiki/Exclusive_or#Properties

0 ⊕ 0 = 0

a ⊕ a = 0

a ⊕ 0 = a

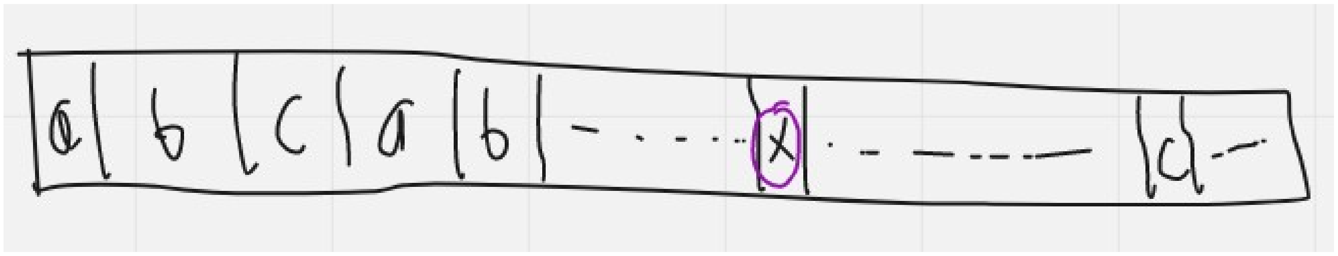
a ⊕ b = b ⊕ a

a ⊕ (b ⊕ c) = (a ⊕ b) ⊕ c

a ⊕ (b ⊕ a) = a ⊕ (a ⊕ b) = (a ⊕ a) ⊕ b = 0 ⊕ b = b

Main idea of the solution

Every element in the array appears twice except one.



We can write:

(a ⊕ a) ⊕ (b ⊕ b) ⊕ (c ⊕ c) ⊕ ….. ⊕ x = 0 ⊕ 0 ⊕ 0 ⊕...⊕ x = 0 ⊕ x = x

C++ >= 11
```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = 0;
        for(auto x: nums){
            ans ^= x;
        }
        return ans;
    }
};
```
Time complexity: O(n)

Space complexity: O(1)