Single Number (easy)

Problem Statement

In a non-empty array of integers, every number appears twice except for one, find that single number.

Example 1:

```
Input: 1, 4, 2, 1, 3, 2, 3
Output: 4
```

Example 2:

```
Input: 7, 9, 7
Output: 9
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
using namespace std;

#include <iostream>
#include <vector>

class SingleNumber {
  public:
    static int findSingleNumber(const vector<int>& arr) {
      // TODO: Write your code here
      return -1;
    }
};

int main(int argc, char* argv[]) {
  cout << SingleNumber::findSingleNumber(vector<int>{1, 4, 2, 1, 3, 2, 3}) << endl;
}
```

RUN SAVE RESET

Solution

One straight forward solution can be to use a **HashMap** kind of data structure and iterate through the input:

- If number is already present in **HashMap**, remove it.
- If number is not present in **HashMap**, add it.
- In the end, only number left in the **HashMap** is our required single number.

Time and space complexity Time Complexity of the above solution will be $O(n)$ and space complexity will also be $O(n)$.

Can we do better than this using the **XOR Pattern**?

Solution with XOR

Recall the following two properties of XOR:

- It returns zero if we take XOR of two same numbers.
- It returns the same number if we XOR with zero.

So we can XOR all the numbers in the input; duplicate numbers will zero out each other and we will be left with the single number.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
using namespace std;

#include <iostream>
#include <vector>
```

```cpp

class SingleNumber {
  public:
    static int findSingleNumber(const vector<int>& arr) {
      int num = 0;
      for (int i=0; i < arr.size(); i++) {
        num = num ^ arr[i];
      }
      return num;
    }
};

int main(int argc, char* argv[]) {
  cout << SingleNumber::findSingleNumber(vector<int>{1, 4, 2, 1, 3, 2, 3}) << endl;
}
```

RUN SAVE RESET

Close

Output 0.758s

```
4
```

Time Complexity: Time complexity of this solution is $O(n)$ as we iterate through all numbers of the input once.

Space Complexity: The algorithm runs in constant space O(1).

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. See how

Back
Introduction

Next
Two Single Numbers (medium)

Mark as Completed

Report an Issue Ask a Question