Two Single Numbers (medium)

We'll cover the following

Problem Statement

In a non-empty array of numbers, every number appears exactly twice except two numbers that appear only once. Find the two numbers that appear only once.

Example 1:

```
Input: [1, 4, 2, 1, 3, 5, 6, 2, 3, 5]
Output: [4, 6]
```

Example 2:

```
Input: [2, 1, 3, 2]
Output: [1, 3]
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```python
def find_single_numbers(nums):
  # TODO: Write your code here
  return [-1, -1]


def main():
  print('Single numbers are:' +
        str(find_single_numbers([1, 4, 2, 1, 3, 5, 6, 2, 3, 5])))
  print('Single numbers are:' + str(find_single_numbers([2, 1, 3, 2])))

main()
```

Run | Save | Reset

Solution

This problem is quite similar to Single Number, the only difference is that, in this problem, we have two single numbers instead of one. Can we still use XOR to solve this problem?

Let's assume `num1` and `num2` are the two single numbers. If we do XOR of all elements of the given array, we will be left with XOR of `num1` and `num2` as all other numbers will cancel each other because all of them appeared twice. Let's call this XOR `n1xn2`. Now that we have XOR of `num1` and `num2`, how can we find these two single numbers?

As we know that `num1` and `num2` are two different numbers, therefore, they should have at least one bit different between them. If a bit in `n1xn2` is '1', this means that `num1` and `num2` have different bits in that place, as we know that we can get '1' only when we do XOR of two different bits, i.e.,

```
1 XOR 0 = 0 XOR 1 = 1
```

We can take any bit which is '1' in `n1xn2` and partition all numbers in the given array into two groups based on that bit. One group will have all those numbers with that bit set to '0' and the other with the bit set to '1'. This will ensure that `num1` will be in one group and `num2` will be in the other. We can take XOR of all numbers in each group separately to get `num1` and `num2`, as all other numbers in each group will cancel each other. Here are the steps of our algorithm:

1. Taking XOR of all numbers in the given array will give us XOR of `num1` and `num2`, calling this XOR as `n1xn2`.
2. Find any bit which is set in `n1xn2`. We can take the rightmost bit which is '1'. Let's call this `rightmostSetBit`.
3. Iterate through all numbers of the input array to partition them into two groups based on `rightmostSetBit`. Take XOR of all numbers in both the groups separately. Both these XORs are our required numbers.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```python
def find_single_numbers(nums):
    # get the XOR of the all the numbers
    n1xn2 = 0
    for num in nums:
        n1xn2 ^= num

    # get the rightmost bit that is '1'
    rightmost_set_bit = 1
    while (rightmost_set_bit & n1xn2) == 0:
        rightmost_set_bit = rightmost_set_bit << 1
    num1, num2 = 0, 0

    for num in nums:
        if (num & rightmost_set_bit) != 0:  # the bit is set
            num1 ^= num
        else:  # the bit is not set
            num2 ^= num

    return [num1, num2]


def main():
    print('Single numbers are:' +
          str(find_single_numbers([1, 4, 2, 1, 3, 5, 6, 2, 3, 5])))
    print('Single numbers are:' + str(find_single_numbers([2, 1, 3, 2])))


main()
```

Run | Save | Reset

Time Complexity

The time complexity of this solution is $O(n)$ where 'n' is the number of elements in the input array.

Space Complexity

The algorithm runs in constant space O(1).