Solution: Single Number

Let's solve the Single Number problem using the Bitwise Manipulation pattern.

We'll cover the following

- Statement
- Solution
  - Time complexity
  - Space complexity

Statement

Given an array of integers, where every element appears twice except for one, find the element that occurs only once.

> **Note:** The solution must have linear runtime and constant space complexity.

Constraints:

- $1 \leq$ `nums.length` $\leq 3 \times 10^3$
- $-3 \times 10^3 \leq$ `nums[i]` $\leq 3 \times 10^3$

Solution

We can use the bitwise XOR operator to find the single number in the array efficiently. XOR has the following two properties:

- Performing the XOR of a number with itself returns zero.
- Performing the XOR of zero with a number returns the same number.

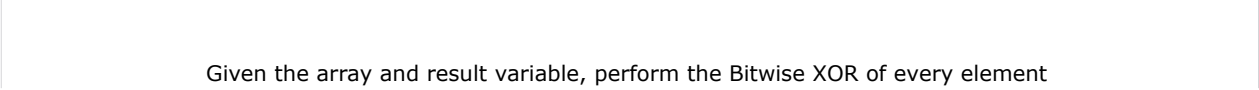
Using the above XOR properties, we can perform the XOR of all the numbers in the input array. The duplicate numbers will cancel each other out, and the single number will be left.

We take the recurring bitwise XOR of the element with the result from the previous iteration until the end of the array.

The algorithm is as follows:

1. Initialize a `result` variable with 0.
2. Iterate over each element of the array, take the bitwise XOR of the element with the `result` variable, and store the resultant value in the `result` variable.
3. After completing the traversal, the `result` variable will have the element that appeared once.
4. Return the `result` variable.

The slides below illustrate how we would like the algorithm to run.



with the result variable to find the single number.

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]					
result	0	0	0	0	0

result					

Perform the Bitwise XOR of the current element 1 with the result.
The resulting number is 0001(1).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	0	1	1
result	0	0	0	0	0

result	0	0	0	1	1

Perform the Bitwise XOR of the current element 2 with the result.
The resulting number is 0011(3).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	1	0	2
result	0	0	0	1	1

result	0	0	1	1	3

Perform the Bitwise XOR of the current element 2 with the result.
The resulting number is 0001(1).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	1	0	2
result	0	0	1	1	3

result	0	0	0	1	1

Perform the Bitwise XOR of the current element 3 with the result.
The resulting number is 0010(2).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	1	1	3
result	0	0	0	1	1

result	0	0	1	0	2

Perform the Bitwise XOR of the current element 3 with the result.
The resulting number is 0001(1).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	1	1	3
result	0	0	1	0	2

result	0	0	0	1	1

Perform the Bitwise XOR of the current element 1 with the result.
The resulting number is 0000(0).

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	0	0	1	1
result	0	0	0	1	1

result	0	0	0	0	0

The result is 0100, which is the binary of 4. So, the single number in the array is 4.

nums	1	2	2	3	3	1	4

Representation	Binary				Decimal
nums[i]	0	1	0	0	4
result	0	0	0	0	0

result	0	1	0	0	4

Java

```java
        for (int num: nums) {
            // Take bitwise xor of all elements with the result
            result ^= num;
        }
        // return the result that contains single number
        return result;
    }

    public static void main(String[] args) {
        int[][] inputs = {
            {1, 2, 2, 3, 3, 1, 4},
            {2, 2, 1},
            {4, 1, 2, 1, 2},
            {-4, -1, -2, -1, -2},
            {25}
        };
        for (int i = 0; i < inputs.length; i++) {
            System.out.print(i + 1);
            System.out.print(".\tFinding the element which appeared once in the list ");
            System.out.println(Arrays.toString(inputs[i]));
            System.out.print("\tResult: ");
            System.out.println(singleNumber(inputs[i]));
            System.out.println(new String(new char[100]).replace('\0', '-'));
```

```
        }

    }

}
```

Single Number

Time complexity

The time complexity of this solution is $O(n)$, because we are iterating the array once, where n is the total number of elements in the array.

Space complexity