Solution: Two Single Numbers

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

We'll cover the following

Statement

Given a non-empty array `arr`, in which exactly two elements appear once and all the other elements appear twice, return the two elements that come only once.

> **Note:** The result can be returned in any order. The solution should use constant space.

Constraints:

- $2 \leq$ `arr.length` $\leq 3 * 10^4$
- $-2^{31} \leq$ `arr[i]` $\leq 2^{31} - 1$

Solution

We use the bitwise XOR to find the two single values in the given array:

1. Initialise a variable `bitwise` with 0. Traverse the given array and perform bitwise XOR of `bitwise` with all the elements, updating it each time. After the traversal is complete, the bits in `bitwise` will correspond to the XOR of our two single numbers.
2. Next, we need the rightmost 1-bit that is different between the two numbers. We can get this by retaining the rightmost 1-bit in `bitwise` and setting the rest to 0. For example, for `10101`, we'll get `00001`. This implies that the rightmost bit with 1 is where the first difference between the two single numbers occured. Consider the `bitwise = 010`. Since this value represents the XOR of the two single numbers, the `1` bit locates the difference between the numbers. In this case, we can see that the two numbers have different bits at position 2.

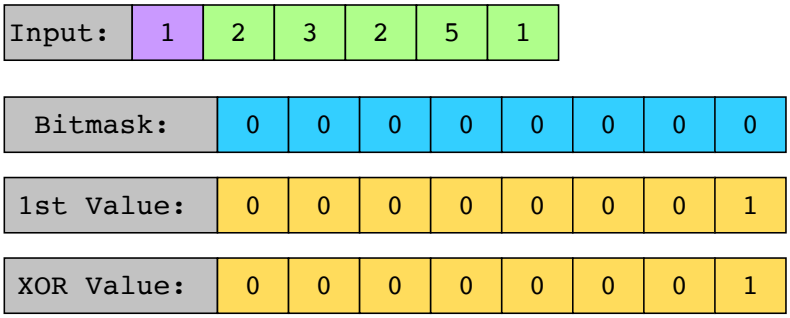
   To achieve this, we'll take 2's complement of `bitwise`, and perform bitwise AND of the 2's complement with the `bitwise` and store it in a variable `bitmask`.
3. Initialise a new variable `result` with 0. Now, in a loop, divide array elements into two groups of numbers to get the group with their bit set as `1`. We'll perform bitwise AND of all the elements of the array with `bitmask`.
   - If the result of the above bitwise AND operation is greater than zero, we'll XOR the array elements with `result` and update its value every time.

- Once we finish the iteration, `result` contains one non-repetitive number. The second non-repetitive number can be obtained by performing XOR operation between `result` and `bitwise`.

To understand this a little better, consider an array `[1, 2, 2, 1, 3, 5]` with the `bitwise = 110`. Only retaining the rightmost 1-bit, we get `bitmask = 010`. Now, we'll traverse the input array and partition our elements into two groups: numbers with their second bit from the right set to `1` and numbers with their second bit from the right set to `0`. If we XOR the elements from the first group, we get `2^2^3 = 3`. Similarly, for the second group, we get `1^1^5 = 5`, which are the two single numbers in the input array.

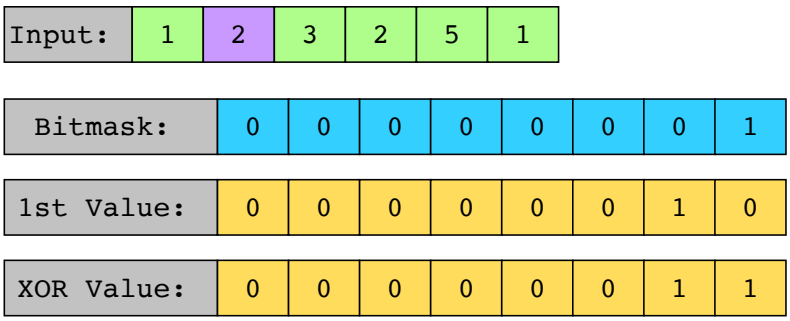
- Return two single numbers in an array.

First of all, find the **XOR** of numbers with each other:



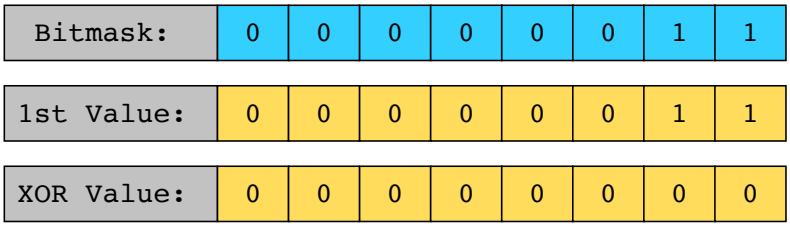
First bitmask is **0**, so **XOR** of 1st value will be equal to value. So, find next **XOR** with 1st value.

First of all, find the **XOR** of numbers with each other:



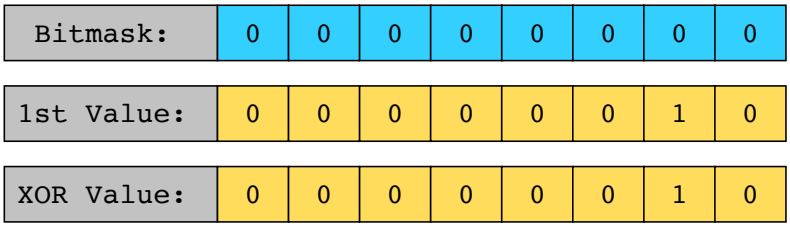
Our updated bitmask is equal to the **XOR** of number with previous bitmask.

First of all, find the **XOR** of numbers with each other:



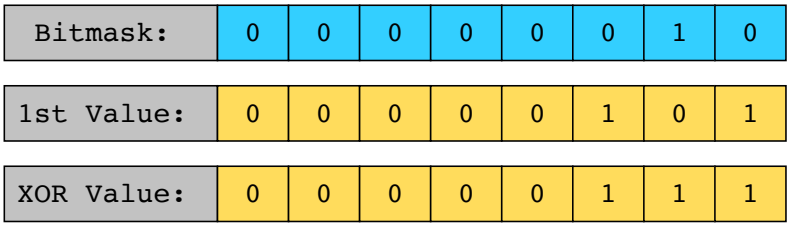
Our updated bitmask is equal to the **XOR** of number with previous bitmask.

First of all, find the **XOR** of numbers with each other:



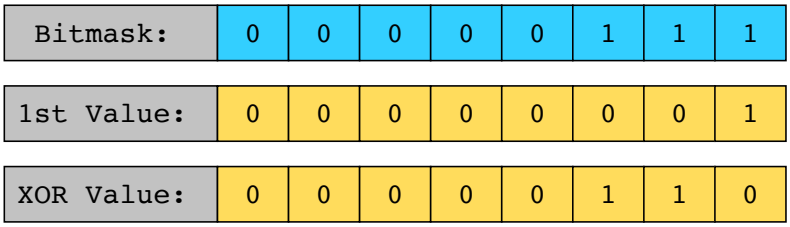
Our updated bitmask is equal to the **XOR** of number with previous bitmask.

First of all, find the **XOR** of numbers with each other:

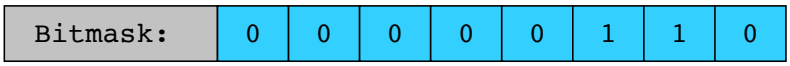


Our updated bitmask is equal to the **XOR** of number with previous bitmask.

First of all, find the **XOR** of numbers with each other:



Our final bitmask is equal to:



Rightmost one-bit difference between:
bitmask & (-bitmask)

Input:	1	2	3	2	5	1

Bitmask:	0	0	0	0	0	1	1	0
~ Bitmask:	1	1	1	1	1	0	0	1
- Bitmask	1	1	1	1	1	0	1	0

Now perform the **AND** operation to find the difference:

Difference:	0	0	0	0	0	0	1	0

Now we need to find the first unique element from an array:

Input:	1	2	3	2	5	1

Difference:	0	0	0	0	0	0	1	0
Value:	0	0	0	0	0	0	0	1

At start we have **x** value equal to **0**.

x	0	0	0	0	0	0	0	0

Now we need to find the first unique element from an array:

Input:	1	2	3	2	5	1

Difference:	0	0	0	0	0	0	1	0

Value:	0	0	0	0	0	0	1	0

Now next value we have **2** which is equal to the difference value so it returns the same value. Now we continuously perform this task till end of the array.

Updated x	0	0	0	0	0	0	0	0

Now we need to find the first unique element from an array:

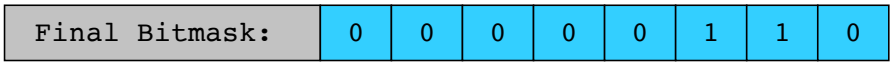
Input:	1	2	3	2	5	1

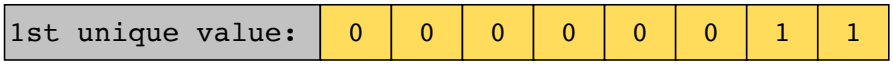
Difference:	0	0	0	0	0	0	1	0

Value:	0	0	0	0	0	0	0	1

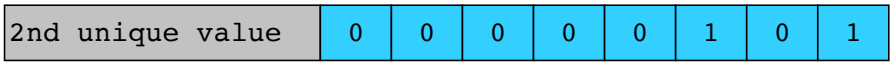
After taking one by one value of array, we find the first unique number which is equal to **3**.

Now to find the next unique value we take a **XOR** of **x** value that is first unique number with the final **bitmask** value.





Value and difference value is difference so it does not fulfill the condition so our x value will be same.



Now to find the next unique value we take a **XOR** of **x** value that is first unique number with the final **bitmask** value.