Suppose we have an input array `nums` containing the following elements: [2, 2, 3, 2].

Initially, both `ones` and `twos` are set to 0.

1. `ones = (ones ^ nums[i]) & ~twos;`

- This line calculates the new value for the `ones` variable.

- The XOR operation `(ones ^ nums[i])` toggles the bits in `ones` for the positions where `nums[i]` has a 1.

- Then, the bitwise AND operation `& ~twos` ensures that any bits that are already present in `twos` are excluded from `ones`.

- This means that if a bit has appeared twice (present in `twos`), it will be removed from `ones`.

- Let's go through the example:

  - Initially, `ones = 0` (binary: 00), and `nums[i] = 2` (binary: 10).

  - After the XOR operation, `ones ^ nums[i]` gives us 10.

  - After the bitwise AND operation with `~twos` (which is 00 initially), the result is 10.

  - So, the updated `ones` becomes 10 (binary representation).

2. `twos = (twos ^ nums[i]) & ~ones;`

- This line calculates the new value for the `twos` variable.

- Similarly to the previous line, the XOR operation `(twos ^ nums[i])` toggles the bits in `twos` for the positions where `nums[i]` has a 1.

- The bitwise AND operation `& ~ones` ensures that any bits that are already present in `ones` are excluded from `twos`.

- This means that if a bit has appeared once (present in `ones`), it will be removed from `twos`.

- Let's continue with our example:

  - Initially, `twos = 0` (binary: 00), and `nums[i] = 2` (binary: 10).

  - After the XOR operation, `twos ^ nums[i]` gives us 10.

  - After the bitwise AND operation with `~ones` (which is 01), the result is 00.

  - So, the updated `twos` becomes 00 (binary representation).

After the first iteration, we have:

- `ones = 10` (binary representation) - This represents the bits that have appeared once so far.

- `twos = 00` (binary representation) - This represents the bits that have appeared twice so far.

In the second iteration, we encounter the element `3`:

1. `ones = (ones ^ nums[i]) & ~twos;`
   - `ones ^ nums[i]` gives us 10 ^ 11, which results in 01.
   - `~twos` is 11 (complement of 00).
   - Performing the bitwise AND operation, we get 01 & 11, resulting in 01.
   - So, `ones` is updated to 01 (binary representation).
2. `twos = (twos ^ nums[i]) & ~ones;`
   - `twos ^ nums[i]` gives us 00 ^ 11, which results in 11.
   - `~ones` is 10 (complement of 01).
   - Performing the bitwise AND operation, we get 11 & 10.