## 41. First Missing Positive

Hard 8998 1283 Add to List Share

Given an unsorted integer array `nums`, return the smallest missing positive integer.

You must implement an algorithm that runs in `O(n)` time and uses constant extra space.

**Example 1:**

```
Input: nums = [1,2,0]
Output: 3
```

**Example 2:**

```
Input: nums = [3,4,-1,1]
Output: 2
```

**Example 3:**

```
Input: nums = [7,8,9,11,12]
Output: 1
```

**Constraints:**

- $1 \le nums.length \le 5 * 10^5$
- $-2^{31} \le nums[i] \le 2^{31} - 1$

# 41. Find First Missing Positive

OutPlace Algorithm

99% the value is between [1,n] where n is the length of the array.

## Observations on problem Statement

On edge case [1%], If all the [1,n] values present in array then we should return n+1

Least possible missing positive value is always 1

input[]

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 0 | 15 |

n=4

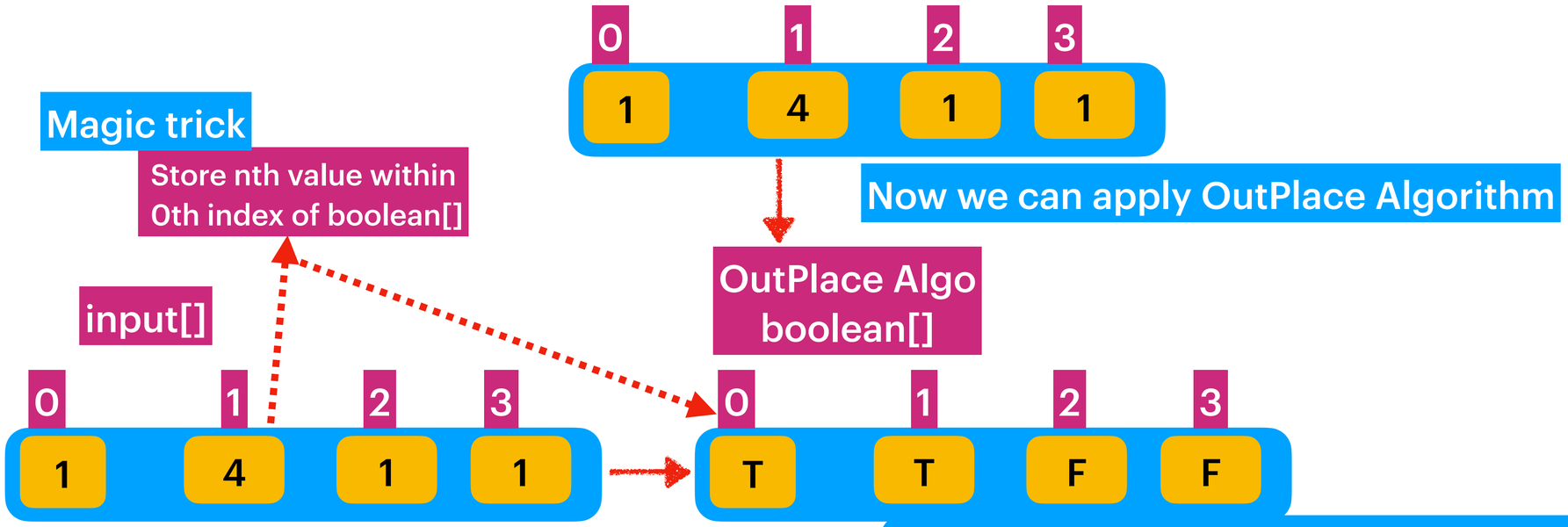
Check if one is present

No → Then return 1

YES → We don't need to worry about Value <= 0 || value > n as the number expected number lies between [1,n]
Replace them with with '1'

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 1 | 1 |

Now we can apply OutPlace Algorithm




Space Complexity : O(n)

Time Complexity : O(n)

## OutPlace Algorithm

-> Check if the value 1 is presented if not return 1.

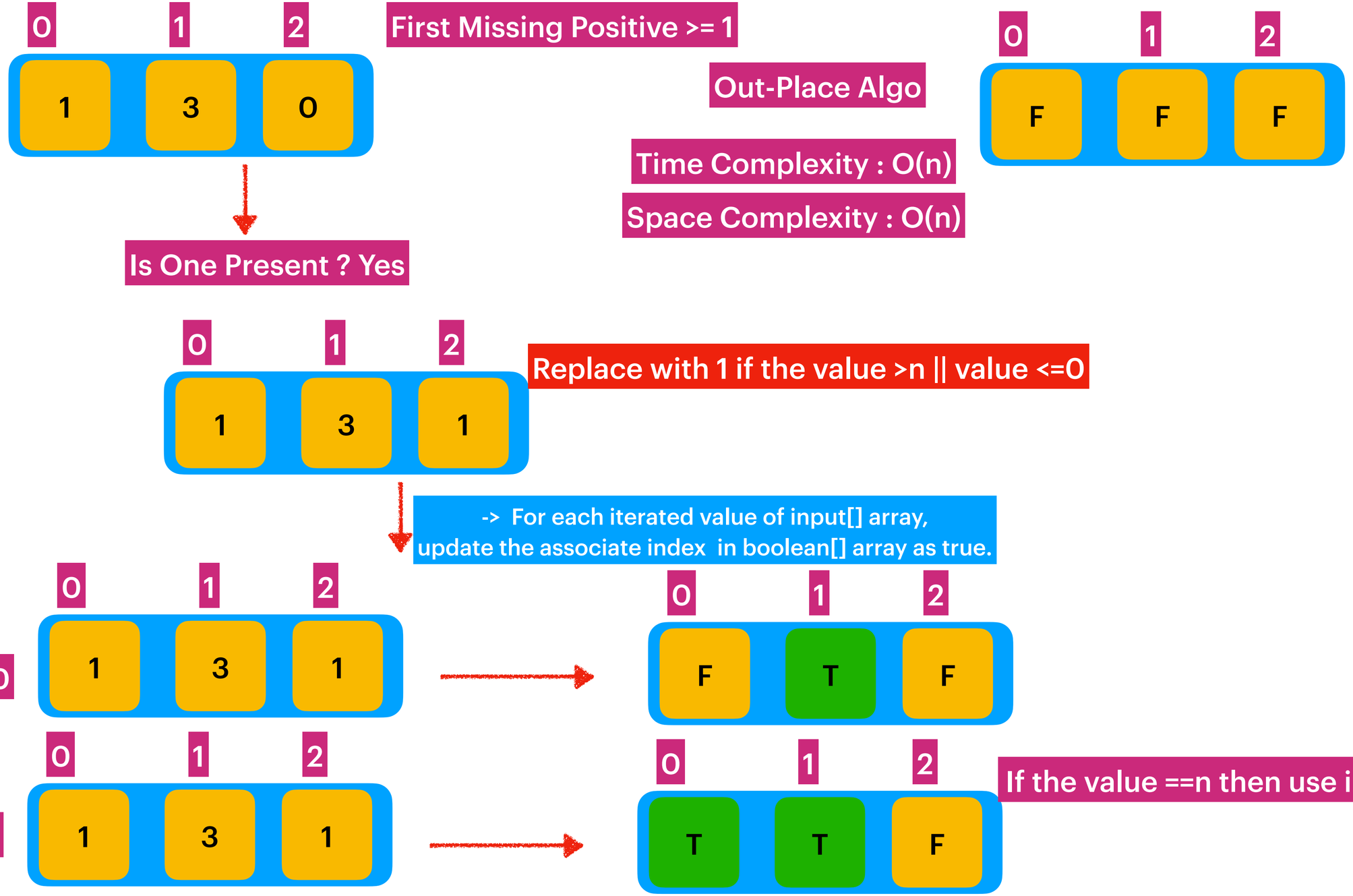
->For each iterated value check value is
( value > n || value <= 0 ) then replace with 1.

-> Take a boolean[] with length n .

-> For each iterated value of input[] array, update the associate index in boolean[] array as true.

As we don't find the index for n^th value in boolean[], we use 0^th index to represent nth value.

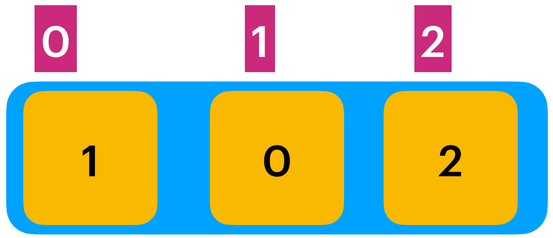
-> In boolean[] array iterate from index=1 till n to find any value is missing, the first index having False value is the missing value.
-> edge case 1: What if n is missing then index[0] would be False.
-> edge case2 : What if all the elements present then return n+1

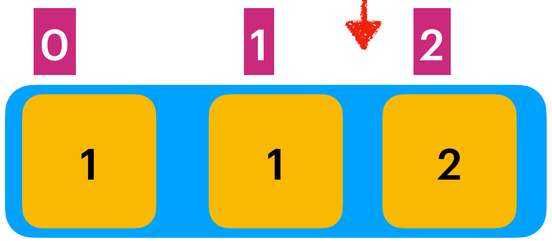

0
1
2
First Missing Positive >= 1
1
3
0
Out-Place Algo
0
1
2
F
F
F
Time Complexity : O(n)
Space Complexity : O(n)
Is One Present ? Yes
0
1
2
1
3
1
Replace with 1 if the value >n || value <=0
-> For each iterated value of input[] array,
update the associate index in boolean[] array as true.
0
1
2
i : 0
1
3
1
0
1
2
F
T
F
0
1
2
i : 1
1
3
1
0
1
2
T
T
F
If the value ==n then use index 0

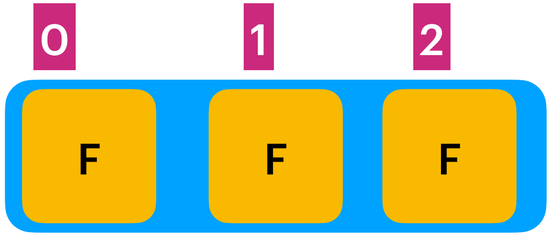

0
1
2
i : 2
1
3
1
0
1
2
T
T
F

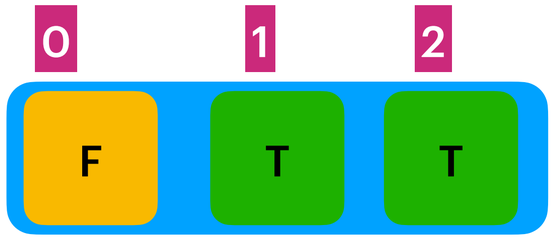

0
1
2
T
T
F
In boolean[] array iterate from index=1 till n to find any value is missing, the first found index having False value is the missing value.

edge case 1: What if n is missing then index[0]  would be False.



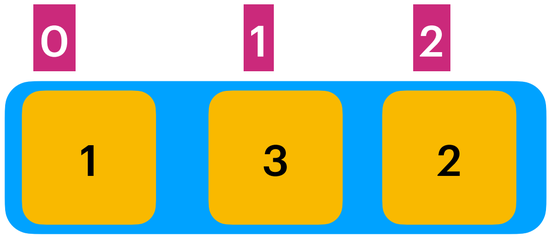



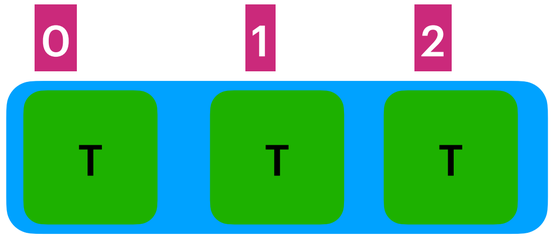




```
for(int index = 1 ;  index < n ; I++)
{
if(arr[index] == false)
{
return false;
}
}
```

```
If(arr[0] == false]
{
return n;
}
```

edge case2 : What if all the elements present then return n+1




```
for(int index = 1 ;  index < n ; I++)
{
if(arr[index] == false)
{
return false;
}
}
```

```
If(arr[0] == false]
{
return n;
}
```

return n+1;

# 41. Find First Missing Positive

InPlace Algorithm

99% the value is between [1,n] where n is the length of the array.

## Observations on problem Statement

On edge case 1%, If all the [1,n] values present in array then we should return n+1

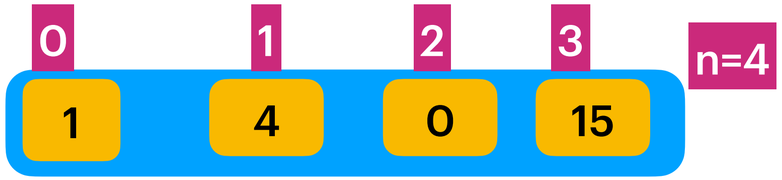
Least possible missing positive value is always 1

Check if one is present

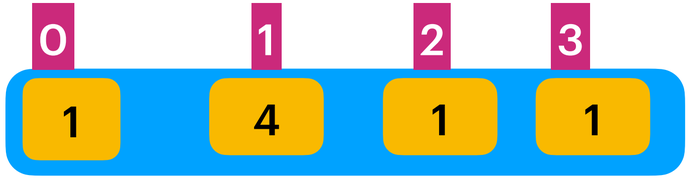
No

Then return 1

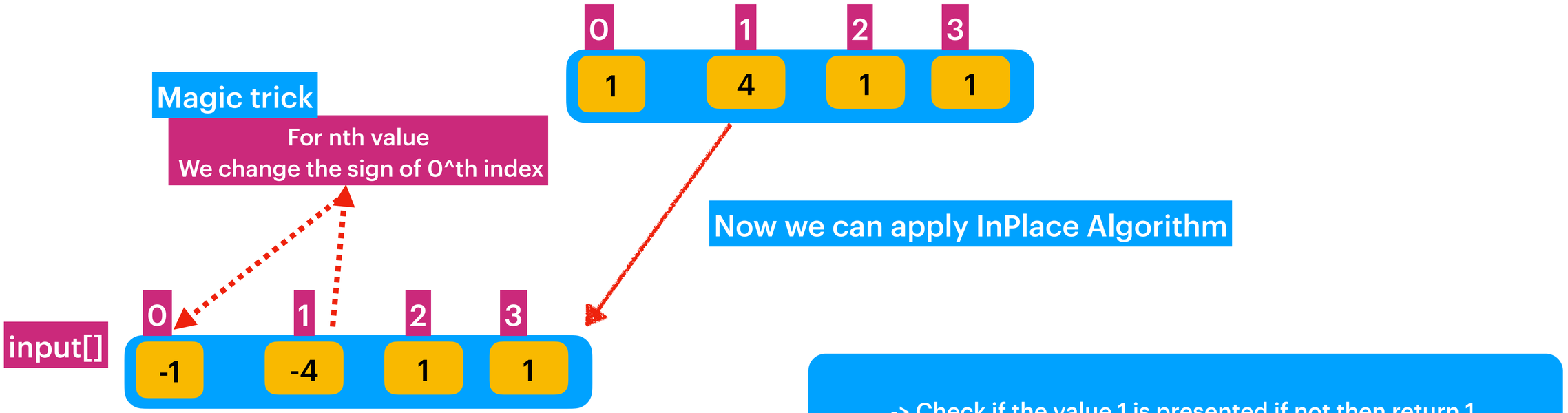
YES




We don't need to worry about
Value <= 0 || value > n as the number expected number lies between [1,n]
Replace them with with '1'




Now we can apply InPlace Algorithm




Space Complexity : O(1)

InPlace Algorithm

Time Complexity : O(n)

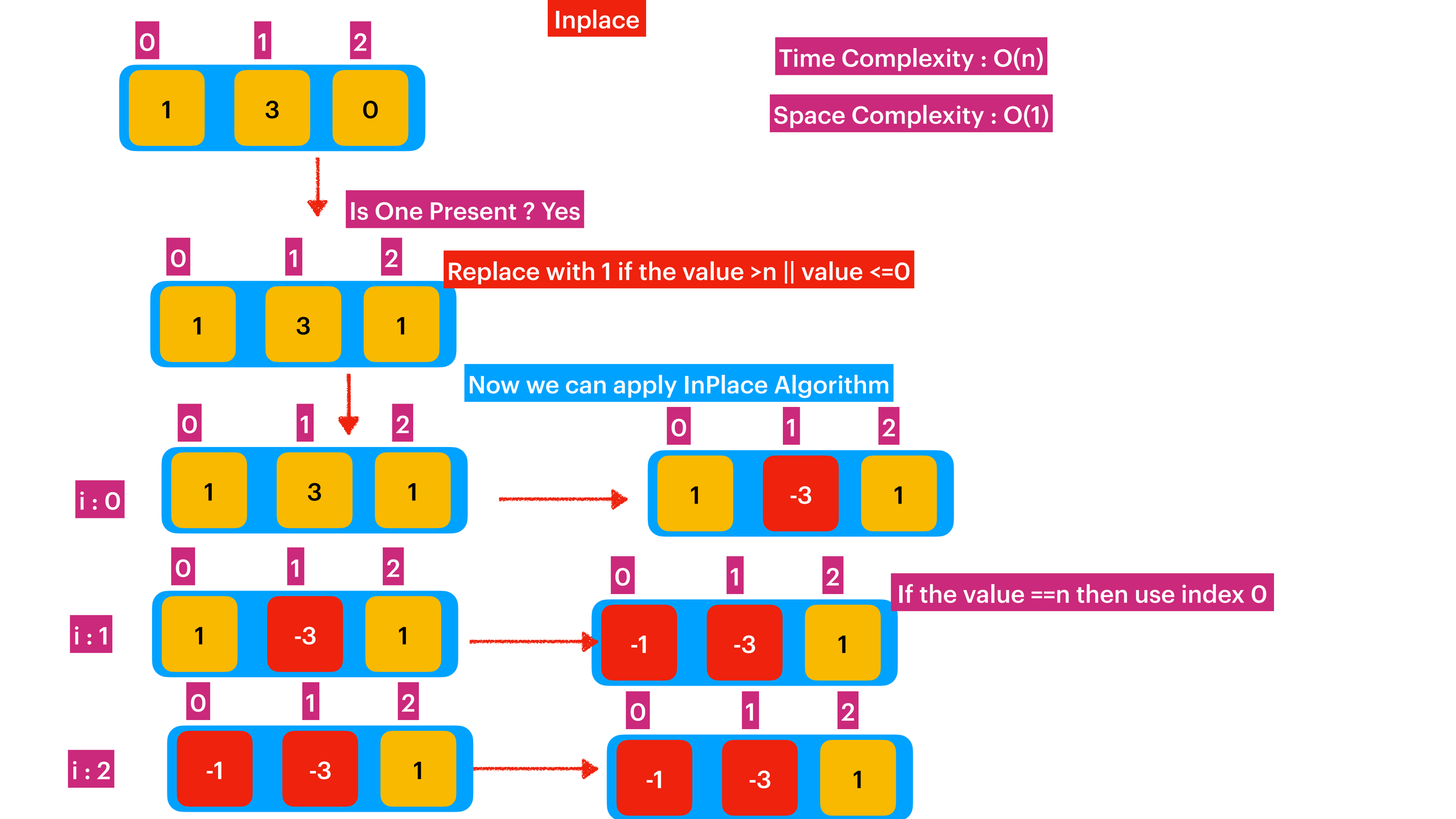
-> Check if the value 1 is presented if not then return 1.

->If value 1 present then for each iterated value, having value is ( value > n || value <= 0 ) then replace with 1.

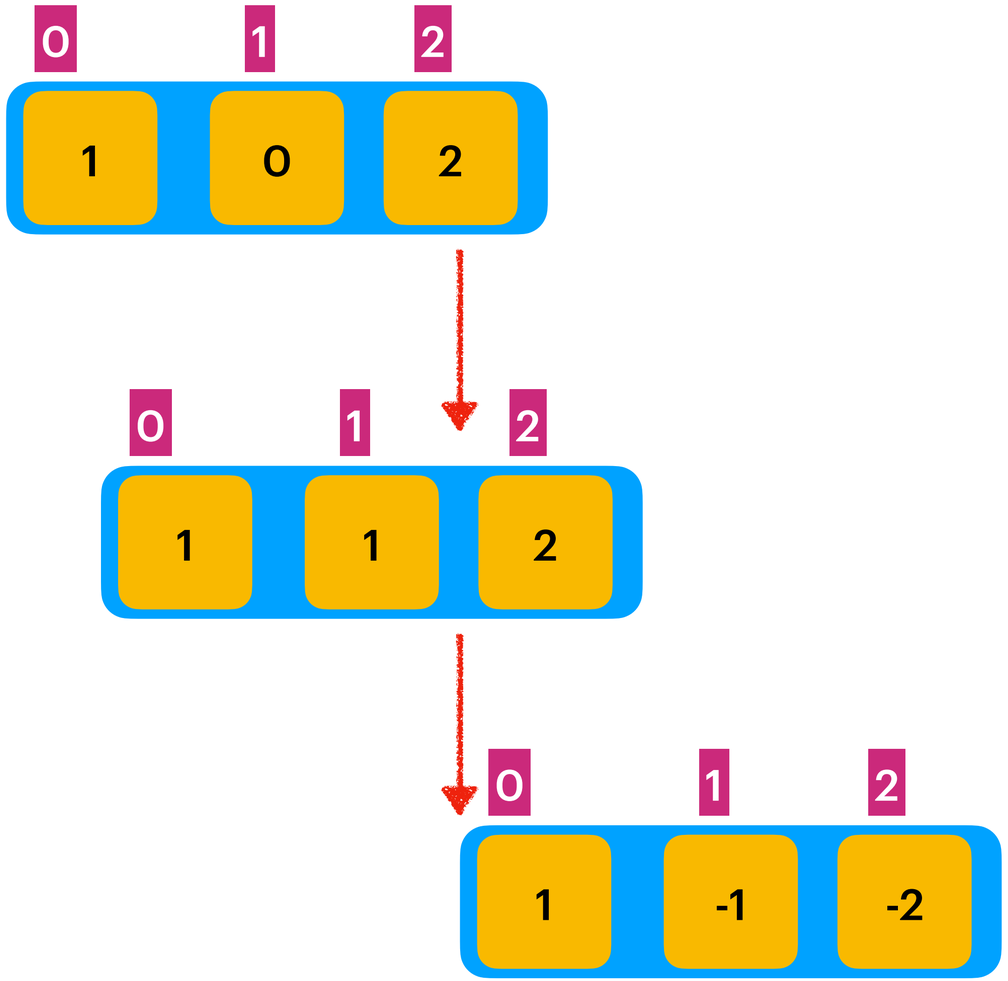
-> For each iterated value of input[] array, change the associate index sign. [ As the value repeats always consider absolute value from the index]

As we don't find the index for n^th value, we use 0^th index to represent nth value.

-> Iterate from index=1 till n, the first index having value > 0 is the missing value.
-> edge case 1: What if n is missing then index[0] would be > 0.
-> edge case2 : What if all the elements present then return n+1


Inplace
Time Complexity : O(n)
Space Complexity : O(1)
0 1 2
1 3 0
Is One Present ? Yes
0 1 2
1 3 1
Replace with 1 if the value >n || value <=0
Now we can apply InPlace Algorithm
i : 0
0 1 2
1 3 1
0 1 2
1 -3 1
i : 1
0 1 2
1 -3 1
0 1 2
-1 -3 1
If the value ==n then use index 0
i : 2
0 1 2
-1 -3 1
0 1 2
-1 -3 1

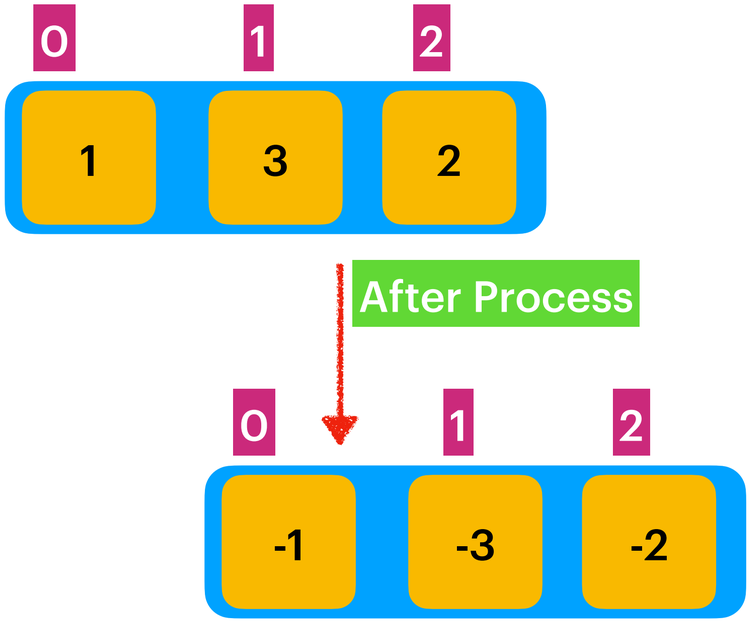
edge case 1: What if n is missing then index[0]  would be > 0.




```
for(int index = 1 ;  index < n ; I++)
{
if(arr[index] > 0)
{
return false;
}
}
```

```
If(arr[0] > 0]
{
return n;
}
```

## edge case2 : What if all the elements present then return n+1




```
for(int index = 1 ;  index < n ; I++)
{
if(arr[index] > 0)
{
return false;
}
}
```

```
If(arr[0] > 0]
{
return n;
}
```

```
return n+1;
```

# Exercise Problem : On Hash

## 442. Find All Duplicates in an Array

Medium 5916 245 Add to List Share

Given an integer array `nums` of length `n` where all the integers of `nums` are in the range `[1, n]` and each integer appears **once** or **twice**, return *an array of all the integers that appears **twice***.

You must write an algorithm that runs in `O(n)` time and uses only constant extra space.

**Example 1:**

```
Input: nums = [4,3,2,7,8,2,3,1]
Output: [2,3]
```

**Example 2:**

```
Input: nums = [1,1,2]
Output: [1]
```

**Example 3:**

```
Input: nums = [1]
Output: []
```

**Constraints:**

- `n == nums.length`
- `1 <= n <= 10^5`
- `1 <= nums[i] <= n`
- Each element in `nums` appears **once** or **twice**.