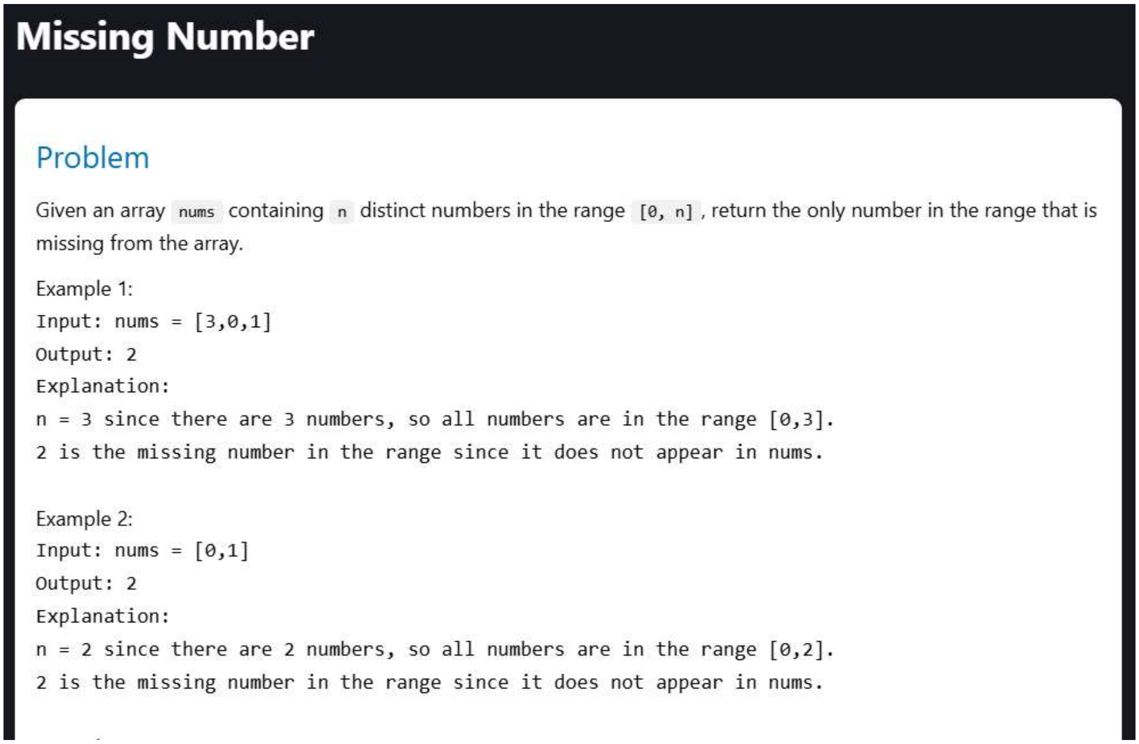


Missing Number

Problem

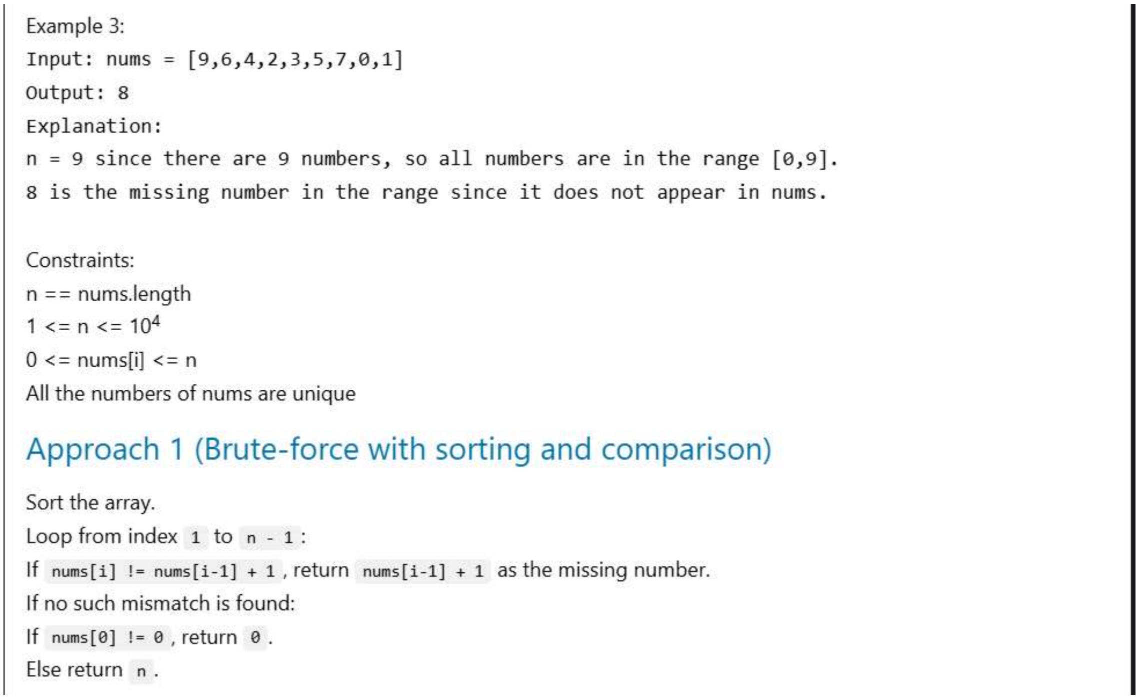
Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return the only number in the range that is missing from the array.

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation:
n = 3 since there are 3 numbers, so all numbers are in the range [0,3].
2 is the missing number in the range since it does not appear in nums.
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation:
n = 2 since there are 2 numbers, so all numbers are in the range [0,2].
2 is the missing number in the range since it does not appear in nums.
```



Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation:
n = 9 since there are 9 numbers, so all numbers are in the range [0,9].
8 is the missing number in the range since it does not appear in nums.
```

Constraints:

n == nums.length

1 <= n <= 10^4

0 <= nums[i] <= n

All the numbers of nums are unique

Approach 1 (Brute-force with sorting and comparison)

Sort the array.

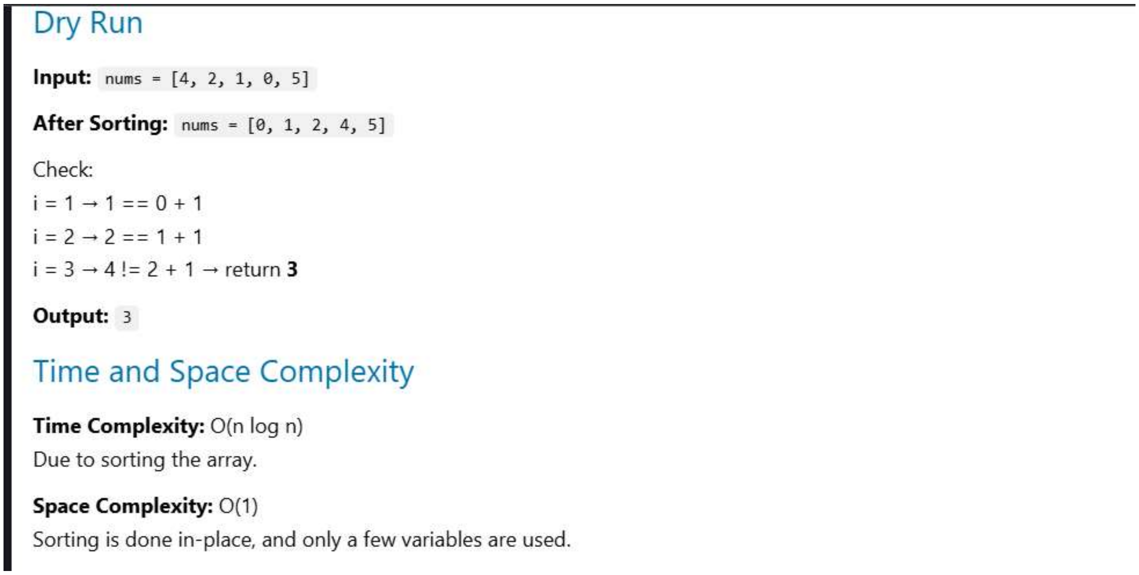
Loop from index `1` to `n - 1`:

If `nums[i] != nums[i-1] + 1`, return `nums[i-1] + 1` as the missing number.

If no such mismatch is found:

If `nums[0] != 0`, return `0`.

Else return `n`.



Dry Run

Input: `nums = [4, 2, 1, 0, 5]`

After Sorting: `nums = [0, 1, 2, 4, 5]`

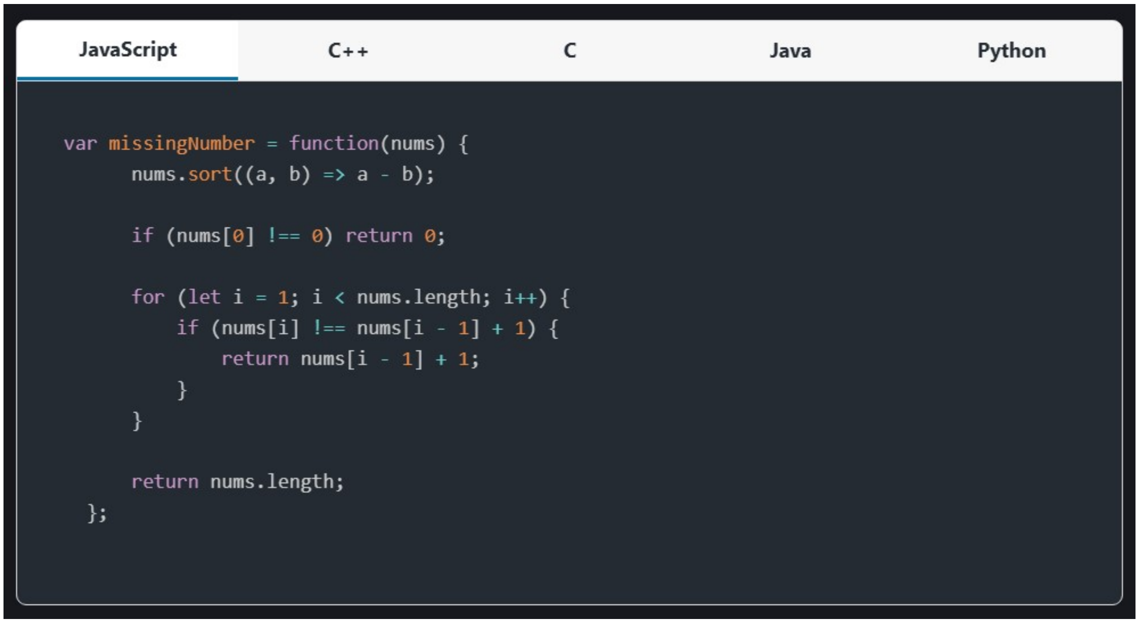
Check:
i = 1 → 1 == 0 + 1
i = 2 → 2 == 1 + 1
i = 3 → 4 != 2 + 1 → return **3**

Output: `3`

Time and Space Complexity

Time Complexity: O(n log n)
Due to sorting the array.

Space Complexity: O(1)
Sorting is done in-place, and only a few variables are used.

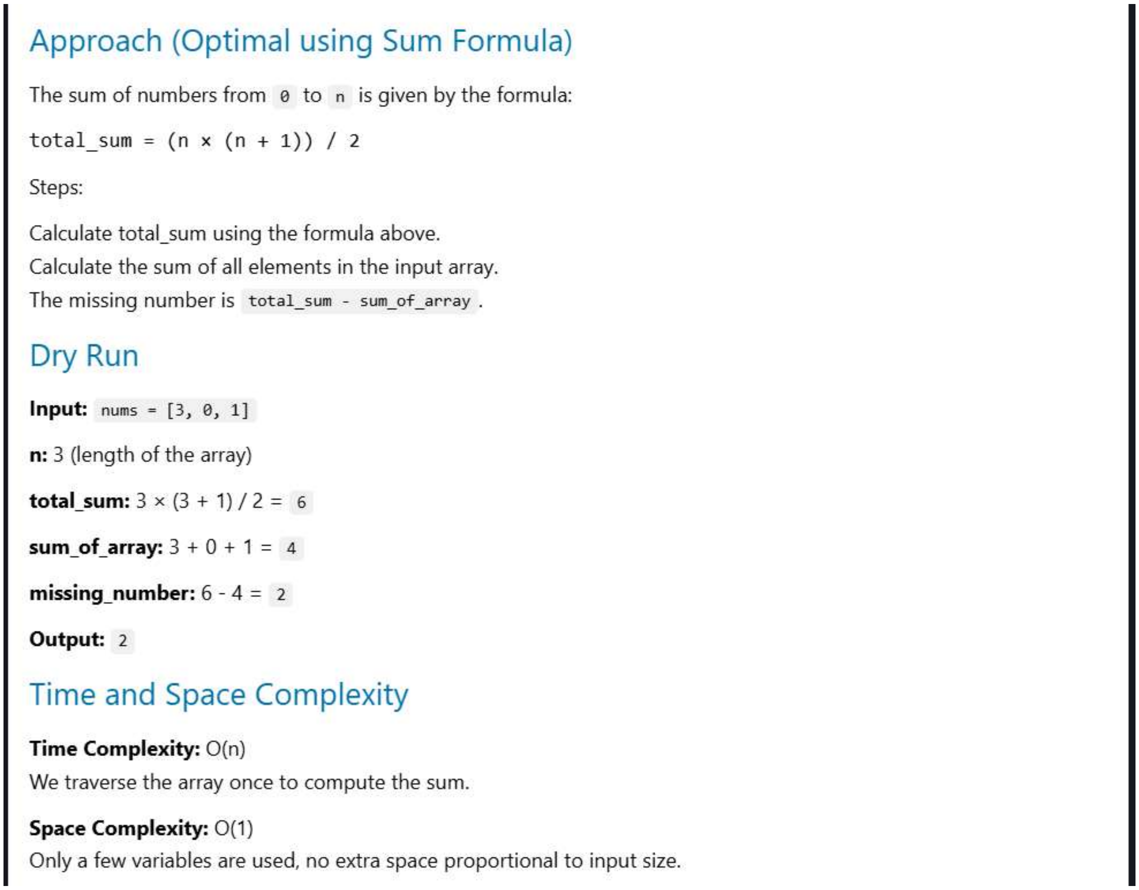


```javascript
var missingNumber = function(nums) {
      nums.sort((a, b) => a - b);

      if (nums[0] !== 0) return 0;

      for (let i = 1; i < nums.length; i++) {
          if (nums[i] !== nums[i - 1] + 1) {
              return nums[i - 1] + 1;
          }
      }

      return nums.length;
  };
```



Approach (Optimal using Sum Formula)

The sum of numbers from `0` to `n` is given by the formula:

```
total_sum = (n × (n + 1)) / 2
```

Steps:

Calculate total_sum using the formula above.
Calculate the sum of all elements in the input array.
The missing number is `total_sum - sum_of_array`.

Dry Run

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

n: 3 (length of the array)

total_sum: 3 × (3 + 1) / 2 = `6`

sum_of_array: 3 + 0 + 1 = `4`

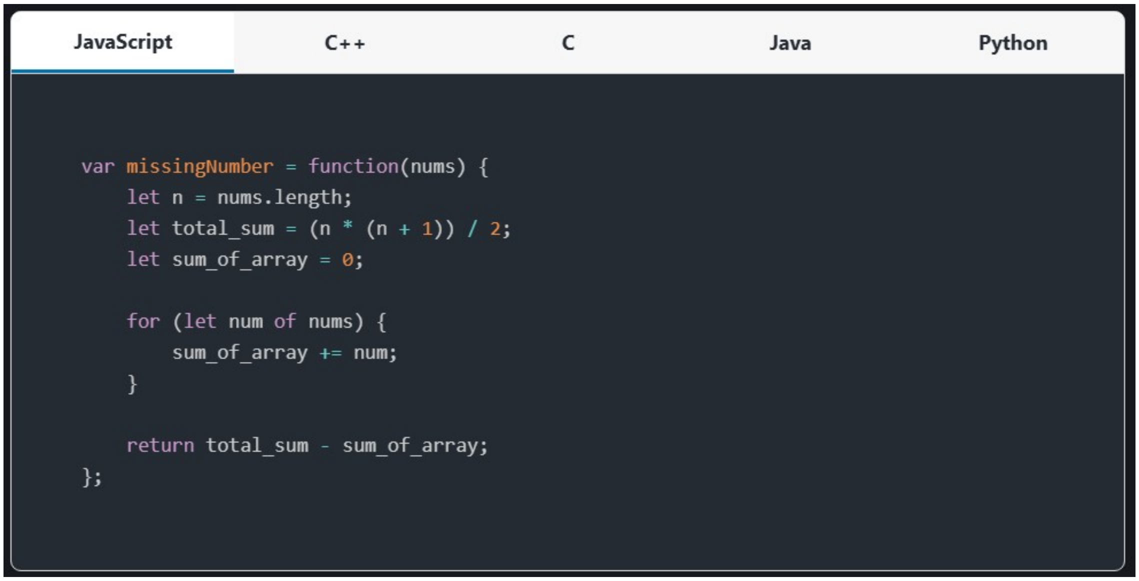
missing_number: 6 - 4 = `2`

Output: `2`

Time and Space Complexity

Time Complexity: O(n)
We traverse the array once to compute the sum.

Space Complexity: O(1)
Only a few variables are used, no extra space proportional to input size.



```
var missingNumber = function(nums) {
    let n = nums.length;
    let total_sum = (n * (n + 1)) / 2;
    let sum_of_array = 0;

    for (let num of nums) {
        sum_of_array += num;
    }

    return total_sum - sum_of_array;
};
```