Solution: Missing Number

Let's solve the Missing Number problem using the Cyclic Sort pattern.

We'll cover the following

Statement

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.

Constraints:

- $n =$ `nums.length`
- $1 \leq n \leq 10^3$
- $0 \leq$ `nums[i]` $\leq n$
- There are no duplicates in the array.

Solution

So far, you've probably brainstormed some approaches and have an idea of how to solve this problem. Let's explore some of these approaches and figure out which one to follow based on considerations such as time complexity and any implementation constraints.

Naive approach

A naive approach would be to first sort the array using quick sort and then traverse the array with two adjacent pointers. Since the integers are sorted, the difference between two adjacent array elements should be 1 if there is no missing integer. We can start by having the first pointer at index `0` and the second pointer at index `1`, moving both 1 step forward each time. If the difference is greater than 1, our missing value would be the value of the first pointer $+ 1$.

The time complexity for this approach becomes $O(n \log n) + O(n)$. The space complexity is $O(n)$.

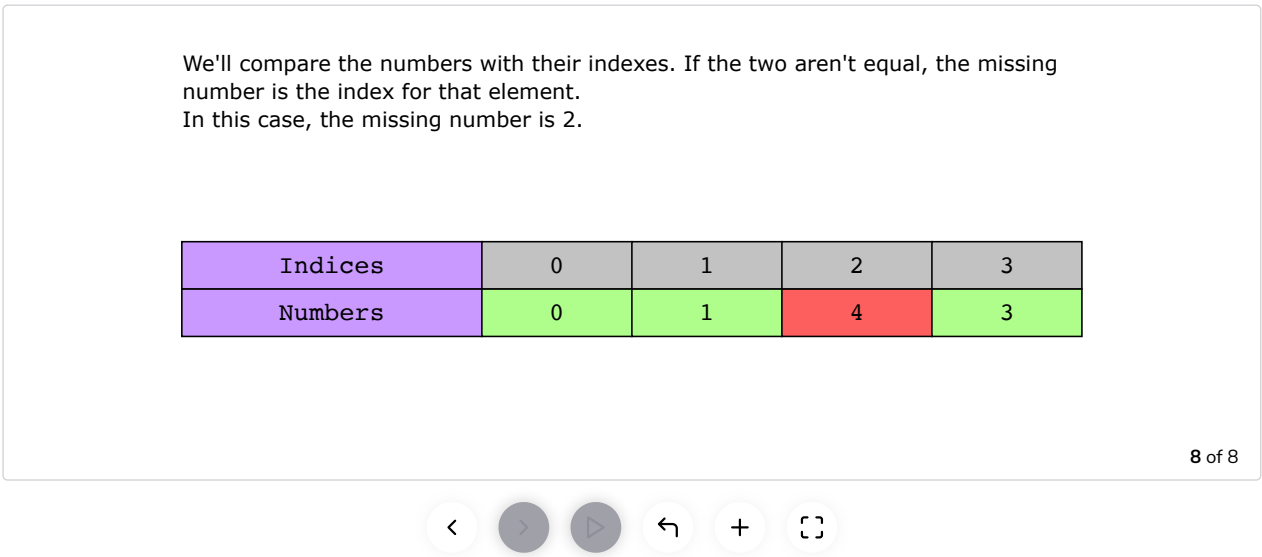
Optimized approach using cyclic sort

Since our input contains unique numbers in the range 0 to n, we can try placing them at their correct index. For example, we'll place the number 1 on index 1, 2 on index 2, and so on. The first index with the incorrect value will be our missing number. If all numbers from 0 to $n - 1$ are present, n is the missing number.

An efficient approach would be to sort the elements in-place. We can iterate over the array elements one at a time. If the current number is not at its correct index, we swap it with the number at its correct index. This is a classic example of cyclic sort. Hence, the problem fits naturally under the cyclic sort pattern.

We'll iterate over the array elements to place them in the correct positions. Once the array is sorted, we'll compare the elements with their indices. If the two are not equal, the missing number will be the index of that element.

Visualize the above algorithm below:



> **Note**: In the following section, we will gradually build the solution. Alternatively, you can skip straight to just the code.

Step-by-step solution construction

The first step is to sort the array using cyclic sort. We'll start the traversal from the first element of the array. If it's not equal to its index, we swap the element with the element on its correct index. For example, if `3` is on index `0`, swap it with the element on index `3`.

Java

```
class MissingNumber {

  public static String printArraywithMarkers(int[] arr, List < Integer > iValue) {
    String out = "[";
    for (int i = 0; i < arr.length - 1; i++) {
      if (i == iValue.get(0) || i == iValue.get(1)) {
        out += "«";
        out += String.valueOf(arr[i]) + "»" + ", ";
      }
      else {
        out += String.valueOf(arr[i]) + ", ";
      }
    }
    if (iValue.get(0) == arr.length - 1 || iValue.get(1) == arr.length - 1) {
      out += "«" + String.valueOf(arr[arr.length - 1]) + "»";
      out += "]";
    }
    else {
      out += arr[arr.length - 1];
      out += "]";
    }
    return out;
  }

```

```
  public static void findMissingNumber(int[] nums) {
    List < Integer > parms;
    int lenNums = nums.length;
    int index = 0;
```

Missing Number

However, the above code throws an error. This happens when we encounter a value that's greater than the array length. To cater to this, we'll add a condition that ensures that the number we swap is less than the length of our array. Else, we'll skip it and move one step forward.

Java

```
class MissingNumber {

  public static String printArraywithMarkers(int[] arr, List < Integer > iValue) {
    String out = "[";
    for (int i = 0; i < arr.length - 1; i++) {
      if (i == iValue.get(0) || i == iValue.get(1)) {
        out += "«";
        out += String.valueOf(arr[i]) + "»" + ", ";
      }
      else {
        out += String.valueOf(arr[i]) + ", ";
      }
    }
    if (iValue.get(0) == arr.length - 1 || iValue.get(1) == arr.length - 1) {
      out += "«" + String.valueOf(arr[arr.length - 1]) + "»";
      out += "]";
    }
    else {
      out += arr[arr.length - 1];
      out += "]";
    }

    return out;
  }

  public static void findMissingNumber(int[] nums) {
    List < Integer > parms;
    int lenNums = nums.length;
```

Missing Number

Now that our array is sorted, we'll compare each element with its index. The first element that's not equal to its index isn't in the correct position. This index will be equal to the missing element.

Java

```
class MissingNumber {

  public static String printArraywithMarkers(int[] arr, List < Integer > iValue) {
    String out = "[";
    for (int i = 0; i < arr.length - 1; i++) {
      if (i == iValue.get(0) || i == iValue.get(1)) {
        out += "«";
        out += String.valueOf(arr[i]) + "»" + ", ";
      }
      else {
        out += String.valueOf(arr[i]) + ", ";
      }
    }
    if (iValue.get(0) == arr.length - 1 || iValue.get(1) == arr.length - 1) {
      out += "«" + String.valueOf(arr[arr.length - 1]) + "»";
      out += "]";
    }
```

```
    else {
      out += arr[arr.length - 1];
      out += "]";
    }

    return out;
  }

  public static int findMissingNumber(int[] nums) {
    List < Integer > parms;
```

Missing Number

Just the code

Here's the complete solution to this problem:

Java

```
class MissingNumber {

  public static int findMissingNumber(int[] nums) {
    List < Integer > parms;
    int lenNums = nums.length;
    int index = 0;
    int value = 0;

    while (index < lenNums) {
      parms = new ArrayList < Integer > (Collections.nCopies(2, -88));
      parms.set(0, index);
      value = nums[index];

      if (value < lenNums && value != nums[value]) {
        parms.set(1, value);
        int temp = nums[index];
        nums[index] = nums[value];
        nums[value] = temp;
      }

      else if (value >= lenNums) {
        index += 1;
      }

      else {
```

Missing Number

Solution summary

To recap, the solution to this problem can be divided into the following parts:

1. Sort the elements in-place.
2. Compare the array elements and their indices.
3. The index of the first mismatch is the missing number.

Time complexity

The time complexity of the above algorithm is $O(n)$, where n is the number of elements in the input array.

Space complexity

The algorithm runs in constant space $O(1)$.

Back
Missing Number

Next
First Missing Positive

Mark as Completed