LeetCode 41

https://leetcode.com/problems/first-missing-positive/description/

Yifeng Zeng

Description

41. First Missing Positive

Given an unsorted integer array, find the first missing positive integer.

For example,

Given [1,2,0] return 3,

and [3,4,-1,1] return 2.

Your algorithm should run in O(n) time and uses constant space.

Idea Report

The primitive idea is to sort this array and find the smallest missing positive. Take [3,4,-1,1] for example, after we sort [-1,1,3,4], we go from left to right, skip all the non-positive numbers, found 1 and missing 2 so we find out 2 is the first missing positive number. How to speed up O(nlogn) sort? We can think of the basic manipulation which is swap. Starting from the basic input and handle the corner cases later. If we have [2,1,4,3] as input, the sorted array is [1,2,3,4], so each number should have its "correct" location, which number n should be at location n - 1. So for input [2,1,4,3] we have a for loop to traverse each number. When a number x is not in its "correct" location x - 1, we use swap to make it be at the correct location. But what if a number is negative or a very large number? We just leave it and this negative number's location is the correct location of another number, it will be swapped later when we traversed to that number. After swapping, we do the traverse again from left to right, the first number at location x - 1 not equal to its value x, is our first missing positive, given x is from 1 to nums.length. If all the numbers are in correct locaiton, we return nume.length + 1 because the first missing the postive is the number after the

last number (e.g. return 5 if [1,2,3,4]).

Code

```
class Solution {
    public int firstMissingPositive(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 1;
        }

        for (int i = 0; i < nums.length; i++) {
            while (0 < nums[i] && nums[i] <= nums.length
                   && nums[i] != i + 1) {
                if (nums[i] == nums[nums[i] - 1]) {
                    // for corner case [1,1]
                    break;
                }
                swap(nums, nums[i] - 1, i);
            }
        }

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != i + 1) {
                return i + 1;
            }
        }
        return nums.length + 1;
    }

    private void swap(int[] nums, int i, int j) {
        int t = nums[i];
        nums[i] = nums[j];
        nums[j] = t;
    }
}
```

Summary

- Basic manipulation of inplace swap in array can be fast as O(n) time.
- Modulize the code using swap().