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

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 O(1) auxiliary space.

**Example 1:**

Input: nums = [1,2,0]

Output: 3

Explanation: The numbers in the range [1,2] are all in the array.

**Example 2:**

Input: nums = [3,4,-1,1]

Output: 2

Explanation: 1 is in the array but 2 is missing.

**Example 3:**

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

Output: 1

Explanation: The smallest positive integer 1 is missing.

**Constraints:**

* 1 <= nums.length <= 105
* -231 <= nums[i] <= 231 - 1

**Attempt 1: 2023-09-17**

**Solution 1:  Hash Table (10 min)**

class Solution {

public int firstMissingPositive(int[] nums) {

Set<Integer> set = new HashSet<>();

for(int num : nums) {

set.add(num);

}

int i = 1;

while(true) {

if(!set.contains(i)) {

return i;

}

i++;

}

//return -1; --> not reachable

}

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://grandyang.com/leetcode/41/>

这道题让我们找缺失的首个正数，由于限定了 O(n) 的时间，所以一般的排序方法都不能用，最开始博主没有看到还限制了空间复杂度，所以想到了用 HashSet 来解，这个思路很简单，把所有的数都存入 HashSet 中，然后循环从1开始递增找数字，哪个数字找不到就返回哪个数字，如果一直找到了最大的数字（这里是 nums 数组的长度），则加1后返回结果 res，参见代码如下：

// NOT constant space

class Solution {

public:

int firstMissingPositive(vector<int>& nums) {

unordered\_set<int> st(nums.begin(), nums.end());

int res = 1, n = nums.size();

while (res <= n) {

if (!st.count(res)) return res;

++res;

}

return res;

}

};

**Solution 2: Swap and make sure nums[i] at nums[nums[i] - 1] (60 min)**

class Solution {

public int firstMissingPositive(int[] nums) {

int n = nums.length;

for(int i = 0; i < n; i++) {

while(nums[i] > 0 && nums[i] <= n && nums[i] != nums[nums[i] - 1]) {

swap(nums, i, nums[i] - 1);

}

}

for(int i = 0; i < n; i++) {

if(nums[i] != i + 1) {

return i + 1;

}

}

return n + 1;

}

private void swap(int[] nums, int i, int j) {

int tmp = nums[i];

nums[i] = nums[j];

nums[j] = tmp;

}

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://grandyang.com/leetcode/41/>

但是上面的解法不是 O(1) 的空间复杂度，所以需要另想一种解法，既然不能建立新的数组，那么只能覆盖原有数组，思路是把1放在数组第一个位置 nums[0]，2放在第二个位置 nums[1]，**即需要把 nums[i] 放在 nums[nums[i] - 1]上**，遍历整个数组，如果 nums[i] != i + 1, 而 nums[i] 为整数且不大于n，另外 nums[i] 不等于 nums[nums[i] - 1] 的话，将两者位置调换，如果不满足上述条件直接跳过，最后再遍历一遍数组，如果对应位置上的数不正确则返回正确的数，参见代码如下：

class Solution {

public:

int firstMissingPositive(vector<int>& nums) {

int n = nums.size();

for (int i = 0; i < n; ++i) {

while (nums[i] > 0 && nums[i] <= n && nums[nums[i] - 1] != nums[i]) {

swap(nums[i], nums[nums[i] - 1]);

}

}

for (int i = 0; i < n; ++i) {

if (nums[i] != i + 1) return i + 1;

}

return n + 1;

}

};

**Refer to**

<https://leetcode.wang/leetCode-41-First-Missing-Positive.html>

# **解法一 交换**

参考[这里](https://leetcode.com/problems/first-missing-positive/discuss/17071/My-short-c++-solution-O(1)-space-and-O(n)-time?orderBy=most\_votes)。

如果没限制空间复杂度，我们可以这样想。用一个等大的数组去顺序保存这些数字。

比如说，数组 nums [ 3 4 -1 1 8]，它的大小是 5。然后再创建一个等大的数组 a，初始化为 [ - 1，- 1，- 1，- 1，-1] 。然后我们遍历 nums，把数字分别存到对应的位置。1 就存到数组 a 的第 1 个位置（a [ 0 ]），2 就存到数组 a 的第 2 个位置（a [ 1 ]），3 就存到数组 a 的第 3 个位置（a [ 2 ]）...

nums [ 0 ] 等于 3，更新 a [ - 1，- 1，3，- 1，-1] 。

nums [ 1 ] 等于 4，更新 a [ - 1，- 1，3，4，-1 ] 。

nums [ 2 ] 等于 - 1，不是正数，忽略。

nums [ 3 ] 等于 1，更新 a [ 1，- 1，3，4，-1 ] 。

nums [ 4 ] 等于 8，我们的 a 数组只能存 1 到 5，所以同样忽略。

最后，我们只需要遍历 a 数组，遇到第一次 a [ i ] ！= i + 1，就说明缺失了 i + 1。因为我们的 a 数组每个位置都存着比下标大 1 的数。

当然，上边都是基于有一个额外空间讲的。如果没有额外空间，怎么办呢？

我们直接把原数组当成 a 数组去用。 这样的话，会出现的问题就是之前的数就会被覆盖掉。覆盖之前我们把它放回到当前数字的位置， 换句话说就是交换一下位置。然后把交换回来的数字放到应该在的位置，又交换回来的新数字继续判断，直到交换回来的数字小于 0，或者大于了数组的大小，或者它就是当前位置放的数字了。接着遍历 nums 的下一个数。具体看一下。

nums = [ 3 4 -1 1 8 ]

nums [ 0 ] 等于 3，把 3 放到第 3 个位置，并且把之前第 3 个位置的 -1 放回来，更新 nums [ **-1**， 4， **3**， 1， 8 ]。

然后继续判断交换回来的数字，nums [ 0 ] 等于 -1，不是正数，忽略。

nums [ 1 ] 等于 4，把 4 放到第 4 个位置，并且把之前第 4个位置的 1 放回来，更新 nums [ -1， **1**， 3， **4**， 8 ]。

然后继续判断交换回来的数字，nums [ 1 ] 等于 1，把 1 放到第 1 个位置，并且把之前第 1 个位置的 -1 放回来，更新 nums [ **1**， **-1**， 3， 4， 8 ]。

然后继续判断交换回来的数字，nums [ 1 ] 等于 -1，不是正数，忽略。

nums [ 2 ] 等于 3，刚好在第 3 个位置，不用管。

nums [ 3 ] 等于 4，刚好在第 4 个位置，不用管。

nums [ 4 ] 等于 8，我们的 nums 数组只能存 1 到 5，所以同样忽略。

最后，我们只需要遍历 nums 数组，遇到第一次 nums [ i ] ！= i + 1，就说明缺失了 i + 1。因为我们的 nums 数组每个位置都存着比下标大 1 的数。

看下代码吧，一个 for 循环，里边再 while 循环。

public int firstMissingPositive(int[] nums) {

int n = nums.length;

//遍历每个数字

for (int i = 0; i < n; i++) {

//判断交换回来的数字

while (nums[i] > 0 && nums[i] <= n && nums[i] != nums[nums[i] - 1]) {

//第 nums[i] 个位置的下标是 nums[i] - 1

swap(nums, i, nums[i] - 1);

}

}

//找出第一个 nums[i] != i + 1 的位置

for (int i = 0; i < n; i++) {

if (nums[i] != i + 1) {

return i + 1;

}

}

//如果之前的数都满足就返回 n + 1

return n + 1;

}

private void swap(int[] nums, int i, int j) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

时间复杂度：for 循环里边套了个 while 循环，如果粗略的讲，那时间复杂度就是 O（n²）了。我们再从算法的逻辑上分析一下。因为每交换一次，就有一个数字放到了应该在的位置，只有 n 个数字，所以 while 里边的交换函数，最多执行 n 次。所以时间复杂度更精确的说，应该是 O（n）。

空间复杂度：O（1）。