Previous Next

31. Next Permutation

July 23, 2016 | 268.3K views

★★★★★
Average Rating: 4.56 (227 votes)

Implement **next permutation**, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be **in-place** and use only constant extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

`1,2,3` → `1,3,2`
`3,2,1` → `1,2,3`
`1,1,5` → `1,5,1`

Summary

We need to find the next lexicographic permutation of the given list of numbers than the number formed by the given array.

Solution

Approach 1: Brute Force

Algorithm

In this approach, we find out every possible permutation of list formed by the elements of the given array and find out the permutation which is just larger than the given one. But this one will be a very naive approach, since it requires us to find out every possible permutation which will take really long time and the implementation is complex. Thus, this approach is not acceptable at all. Hence, we move on directly to the correct approach.

Complexity Analysis

- Time complexity : $O(n!)$. Total possible permutations is $n!$.
- Space complexity : $O(n)$. Since an array will be used to store the permutations.

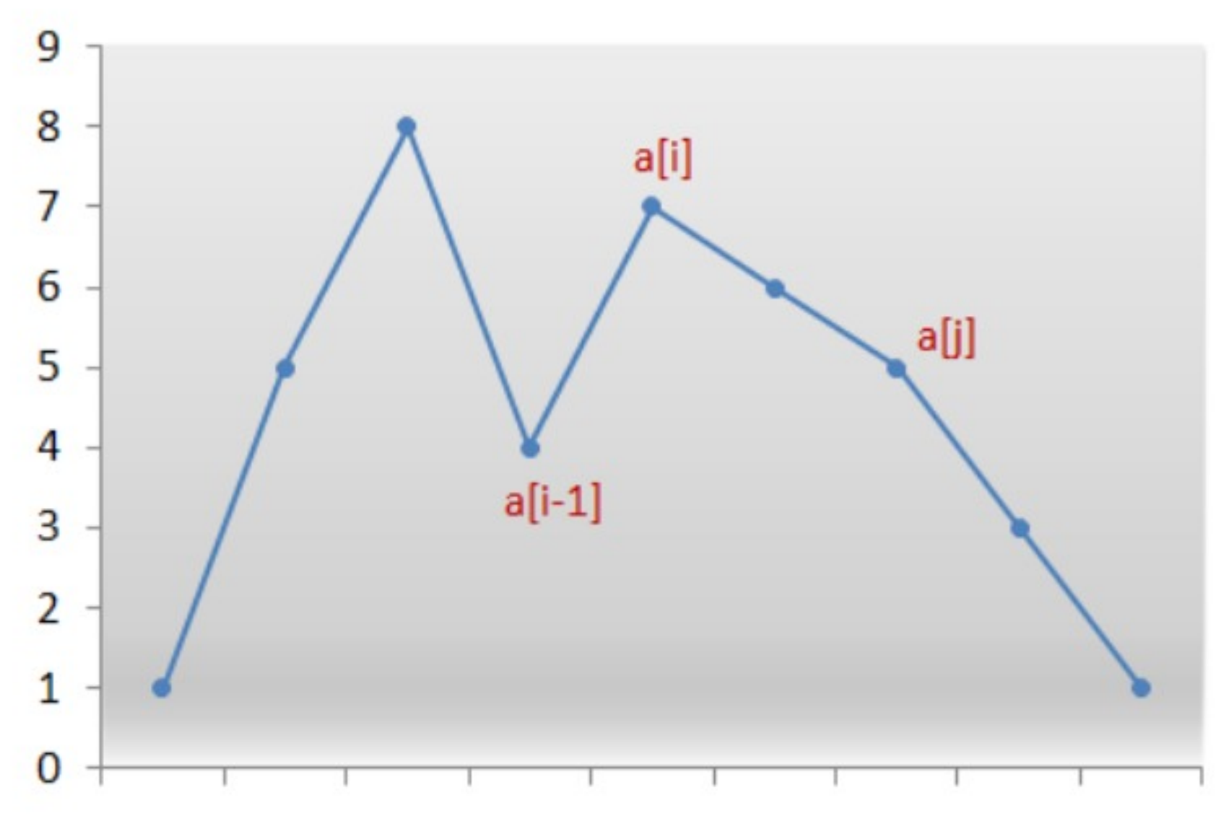
Approach 2: Single Pass Approach

Algorithm

First, we observe that for any given sequence that is in descending order, no next larger permutation is possible. For example, no next permutation is possible for the following array: `[9, 5, 4, 3, 1]`

We need to find the first pair of two successive numbers $a[i]$ and $a[i-1]$, from the right, which satisfy $a[i] > a[i-1]$. Now, no rearrangements to the right of $a[i-1]$ can create a larger permutation since that subarray consists of numbers in descending order. Thus, we need to rearrange the numbers to the right of $a[i-1]$ including itself.

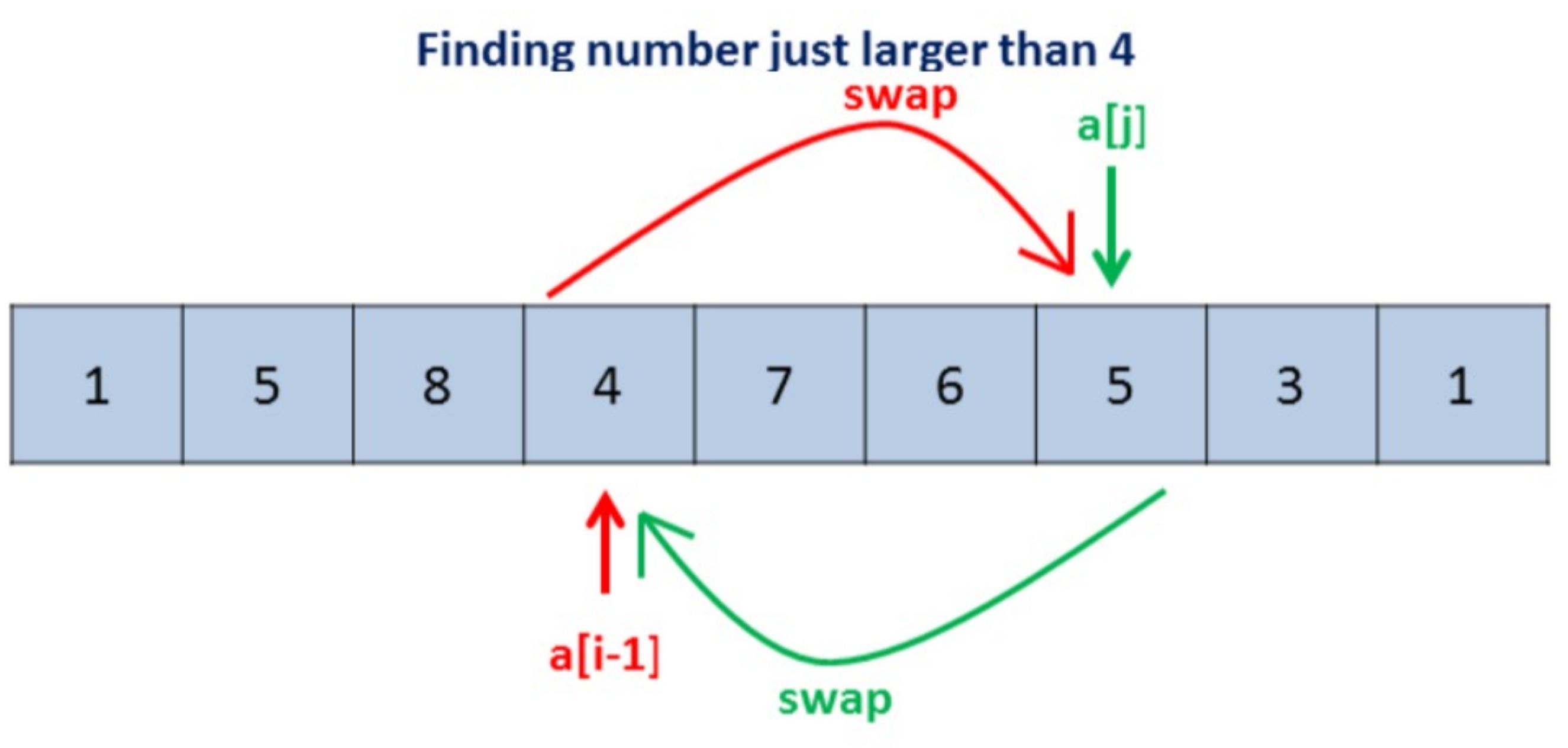
Now, what kind of rearrangement will produce the next larger number? We want to create the permutation just larger than the current one. Therefore, we need to replace the number $a[i-1]$ with the number which is just larger than itself among the numbers lying to its right section, say $a[j]$.



We swap the numbers $a[i-1]$ and $a[j]$. We now have the correct number at index $i-1$. But still the current permutation isn't the permutation that we are looking for. We need the smallest permutation that can be formed by using the numbers only to the right of $a[i-1]$. Therefore, we need to place those numbers in ascending order to get their smallest permutation.

But, recall that while scanning the numbers from the right, we simply kept decrementing the index until we found the pair $a[i]$ and $a[i-1]$ where, $a[i] > a[i-1]$. Thus, all numbers to the right of $a[i-1]$ were already sorted in descending order. Furthermore, swapping $a[i-1]$ and $a[j]$ didn't change that order. Therefore, we simply need to reverse the numbers following $a[i-1]$ to get the next smallest lexicographic permutation.

The following animation will make things clearer:



Java | Copy

```java
1   public class Solution {
2       public void nextPermutation(int[] nums) {
3           int i = nums.length - 2;
4           while (i >= 0 && nums[i + 1] <= nums[i]) {
5               i--;
6           }
7           if (i >= 0) {
8               int j = nums.length - 1;
9               while (j >= 0 && nums[j] <= nums[i]) {
10                  j--;
11              }
12              swap(nums, i, j);
13          }
14          reverse(nums, i + 1);
15      }
16
17      private void reverse(int[] nums, int start) {
18          int i = start, j = nums.length - 1;
19          while (i < j) {
20              swap(nums, i, j);
21              i++;
22              j--;
23          }
24      }
25
26      private void swap(int[] nums, int i, int j) {
27          int temp = nums[i];
```

Complexity Analysis

- Time complexity : $O(n)$. In worst case, only two scans of the whole array are needed.
- Space complexity : $O(1)$. No extra space is used. In place replacements are done.

Rate this article: ★★★★★

Previous Next

Comments: 103 Sort By

Type comment here... (Markdown is supported)

Preview Post

rexonestudio ★ 653 November 23, 2018 6:06 AM

Am I the only ones that find the category marked as 'medium' strange? I think if you know the solution already, it's medium. But if you don't know this problem is much "harder" than some designated "hard" problem.

648 Share Reply

SHOW 9 REPLIES

lijeffrey ★ 173 July 30, 2018 8:55 AM

what's the point of this question testing for during interview, the skills to analyse problems?

143 Share Reply

SHOW 3 REPLIES

custodian ★ 202 October 2, 2018 10:29 AM Report

I'm an interviewer at a tech company and would never consider asking a candidate this question. It's a type of question where you either know the trick that gets you to the solution or not.

147 Share Reply

SHOW 15 REPLIES

bjwu ★ 215 December 14, 2018 8:06 AM

hate such questions. first, the pattern is not easy to observe. second, the most important part is even if you observe the pattern, how to prove it is correct? usually if you cannot prove its correctness, the algorithm may not work for the corner cases.....

138 Share Reply

SHOW 3 REPLIES

willye ★ 861 August 16, 2019 10:16 PM Report

Very difficult problem and *entirely* unreasonable to ask in an interview without heavy hints, but not completely impossible to solve on your own. First of all, the requirements of **in-place replacement and constant space** should immediately indicate **swapping** (this goes for other questions too). Secondly, it should be obvious that **if the elements are increasing from the right, they are currently at their largest possible permutation**, so nothing can be done. I think the tricky part is simply knowing where

Read More

29 Share Reply

SHOW 5 REPLIES

ankghost0912 ★ 44 August 23, 2018 7:58 PM

This question has a broad scope. You don't just go and define the "next lexicographically greater" by some random ordering. If input `[1,2,3]` then a valid "next" greater permutation is also `[2,1,3]`. This question is encouraging nothing but coding to the input. Nor is it defined in what case a permutation is not possible. I would suggest making the problem statement clearer in a way that does not confuse people and encourages coding to the input.

Read More

40 Share Reply

SHOW 3 REPLIES

nanier ★ 26 September 16, 2018 6:21 PM

Be careful with the equal numbers. Such as the case [2,3,1,3,3] . The condition "nums[j] <= nums[i]" is the key point in line 9. If there miss equal cases, you will find the first smallest larger one, not the last smallest larger one. Then when you swap, you get the wrong answer.

26 Share Reply

SHOW 2 REPLIES

ChrisTrompf ★ 835 July 26, 2018 4:26 AM Report

The c++ algorithm libraries make this really easy to solve. There is a ready made function for each of these steps;

- Find first unsorted element using std::is_sorted_until
- Find first larger element using either std::upper_bound or std::find_if

Read More

19 Share Reply

SHOW 2 REPLIES

appenthused0418 ★ 66 October 4, 2018 6:36 AM

I have been super lazy to touch leetcode until lately.. This issue I definitely have met somewhere before and I couldn't solve back then. This time I solved it on my own but it took me quite some time.

16 Share Reply

sli9871230 ★ 92 March 28, 2019 5:51 AM Report

I see two sides in the comments. Some people say that this question is just one of those tricky ones that you either know the trick or you don't. Some people say that this tests your analytical skills under a time limit.

My personal opinion? Both sides have valid points. However, if there was a middle point in between the

Read More

23 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 5 6 ... 10 11 ›