25. Next Permutation

31. Next Permutation

Medium 14.7K 4.1K

Companies

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3]`, `[1,3,2]`, `[2, 1, 3]`, `[2, 3, 1]`, `[3,1,2]`, `[3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`.
- Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`.
- While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a lexicographical larger rearrangement.

Given an array of integers `nums`, *find the next permutation of* `nums`.

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

Example 1:

```
Input: nums = [1,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: nums = [3,2,1]
Output: [1,2,3]
```

Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```

Next Greater Permutation

Contributed by Ratnesh

Medium 0/80 Share 0 upvotes

Problem Statement Suggest Edit

You are given an array 'A' of 'N' integers.
You have to return the lexicographically next to greater permutation.
Note:

```
If such a sequence is impossible, it must be rearranged in the lowest
possible order.
```

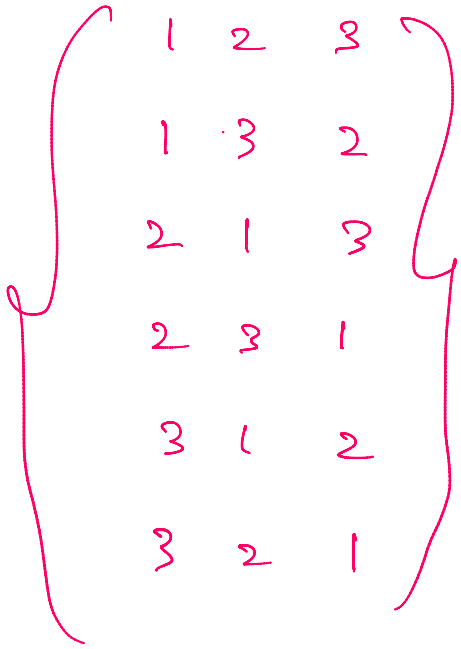
Example:

```
Input:
A = [1, 3, 2]
Output:
2 1 3

Explanation: All the permutations of [1, 2, 3] are [[1, 2, 3], [1, 3, 2],
[2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1], ]. Hence the next greater
permutation of [1, 3, 2] is [2, 1, 3].
```

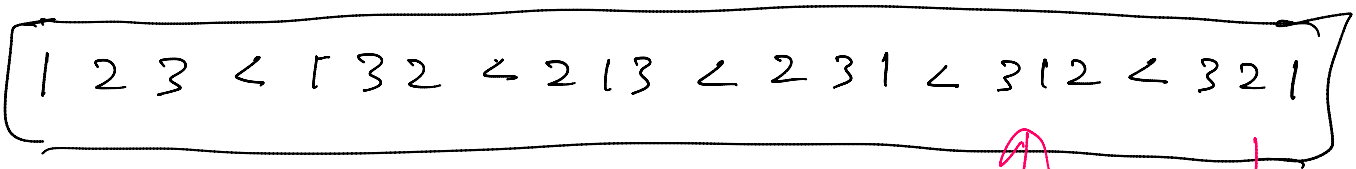
Next Permutation:

arr[] = [3 1 2]



$3! = 3 \times 2 \times 1$

$= 6$ ways



→ In-Order

3(2

return this

If arr[] = {3 2 1} → arr[] ⇒ {1 2 3}

ans

1) Brute force :

- Generate all permutation (sorted order) → Recursion
- Linear Search find the array and

→ return the next array.

T.C ⇒ $O(N! \times N)$

2) Optimal Solution :

Only for C++ Using STL.

```cpp
vector<int> nextGreaterPermutation(vector<int> &A) {
    next_permutation(A.begin(), A.end());
    return A;
}
```

To implement this STL:

arr[] = {2 1 5 4 3 0 0}

one prefix match:

2 5 1 4 3 0 0

5 1 4 3 0 0

5 3 4 1 0 0

English Dictionary Order:

raj

rax

rbx

1. longer prefix match

Watch this Video [NP for the observation

longest prefix match

1. a[i] < a[i+1]

2. find >1, but the smallest one so that you stay close

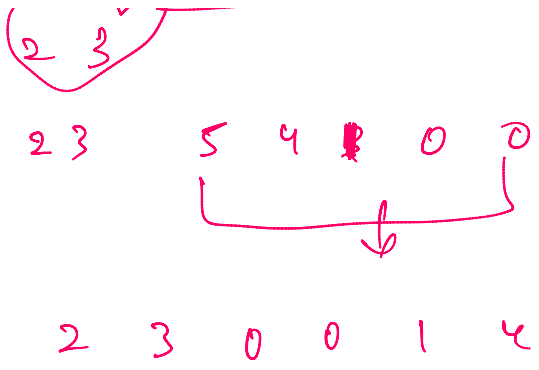
2 3

2 5

slight greater than 1

2 4

2 3



3. try to place remaining in sorted order.

```cpp
vector<int> nextGreaterPermutation(vector<int> &A) {
    int ind = -1;
    int n = A.size();
    for(int i = n - 2;i>=0;i--) {
        if(A[i] < A[i+1]) {
            ind = i;
            break;
        }
    }

    if(ind == -1) {
        reverse(A.begin(), A.end());
        return A;
    }

    for(int i = n-1;i>ind;i--) {
        if(A[i] > A[ind]) {
            swap(A[i], A[ind]);
            break;
        }
    }

    reverse(A.begin() + ind + 1, A.end());
    return A;
}
```

```cpp
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size(), k, l;
        for (k = n - 2; k >= 0; k--) {
            if (nums[k] < nums[k + 1]) {
                break;
            }
        }
        if (k < 0) {
            reverse(nums.begin(), nums.end());
        } else {
            for (l = n - 1; l > k; l--) {
                if (nums[l] > nums[k]) {
                    break;
                }
            }
            swap(nums[k], nums[l]);
            reverse(nums.begin() + k + 1, nums.end());
        }
    }
};
```