2.Medium\08.Next_permutation.cpp

```
/*
QUESTION:-

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]
*/

/*
APPROACH:-

To find the next permutation of an array, we can follow these steps:

1. Find the first index `i` from the right such that `nums[i] < nums[i+1]`. This is the first element that needs to be swapped.
2. Find the first index `j` from the right such that `nums[j] > nums[i]`. This is the element that will replace `nums[i]`.
3. Swap `nums[i]` and `nums[j]`.
4. Reverse the subarray starting from `i+1` till the end of the array.
5. If step 1 does not find any index `i`, it means the array is in descending order. In that case, reverse the entire array to get the lowest possible order.

*/

// CODE:

void nextPermutation(vector<int> &nums)
{

    int bp = -1;
    // finding the break point
    for (int i = nums.size() - 2; i >= 0; i--)
```

```cpp
    {
        if (nums[i] < nums[i + 1])
        {
            bp = i;
            break;
        }
    }
    // first greater element from back
    if (bp != -1)
    {
        for (int i = nums.size() - 1; i >= 0; i--)
        {
            if (nums[i] > nums[bp])
            {
                swap(nums[i], nums[bp]);
                break;
            }
        }
    }
    // reverse the array from bp+1 to end
    reverse(nums.begin() + bp + 1, nums.end());
}

// TIME COMPLEXITY: O(n), where n is the size of the input array.
// SPACE COMPLEXITY: O(1)

```