Next Greater Element II

Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element. The Next Greater Number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, output -1 for this number.

Example 1:

```
Input: [1,2,1]
Output: [2,-1,2]
Explanation: The first 1's next greater number is 2; The number 2 can't find next g
reater number; The second 1's next greater number needs to search circularly, which
 is also 2.
```

Note: The length of given array won't exceed 10000.

Solution 1

The approach is same as *Next Greater Element I*
See explanation in my solution to the previous problem
The only difference here is that we use `stack` to keep the **indexes** of the **decreasing** subsequence

Java

```
    public int[] nextGreaterElements(int[] nums) {
        int n = nums.length, next[] = new int[n];
        Arrays.fill(next, -1);
        Stack<Integer> stack = new Stack<>(); // index stack
        for (int i = 0; i < n * 2; i++) {
            int num = nums[i % n];
            while (!stack.isEmpty() && nums[stack.peek()] < num)
                next[stack.pop()] = num;
            if (i < n) stack.push(i);
        }
        return next;
    }
```

C++

```
    vector<int> nextGreaterElements(vector<int>& nums) {
        int n = nums.size();
        vector<int> next(n, -1);
        stack<int> s; // index stack
        for (int i = 0; i < n * 2; i++) {
            int num = nums[i % n];
            while (!s.empty() && nums[s.top()] < num) {
                next[s.top()] = num;
                s.pop();
            }
            if (i < n) s.push(i);
        }
        return next;
    }
```

written by yuxiangmusic original link here

Solution 2

The first typical way to solve circular array problems is to extend the original array to twice length, 2nd half has the same element as first half. Then everything become simple.
Naive by simple solution, just look for the next greater element directly. Time complexity: O(n^2).

```
public class Solution {
    public int[] nextGreaterElements(int[] nums) {
        int max = Integer.MIN_VALUE;
        for (int num : nums) {
            max = Math.max(max, num);
        }

        int n = nums.length;
        int[] result = new int[n];
        int[] temp = new int[n * 2];

        for (int i = 0; i < n * 2; i++) {
            temp[i] = nums[i % n];
        }

        for (int i = 0; i < n; i++) {
            result[i] = -1;
            if (nums[i] == max) continue;

            for (int j = i + 1; j < n * 2; j++) {
                if (temp[j] > nums[i]) {
                    result[i] = temp[j];
                    break;
                }
            }
        }

        return result;
    }
}
```

The second way is to use a `stack` to facilitate the look up. First we put all indexes into the stack, `smaller` index on the `top`. Then we start from `end` of the array look for the first element (index) in the stack which is greater than the current one. That one is guaranteed to be the `Next Greater Element`. Then put the current element (index) into the stack.
Time complexity: O(n).

```java
public class Solution {
    public int[] nextGreaterElements(int[] nums) {
        int n = nums.length;
        int[] result = new int[n];

        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            stack.push(i);
        }

        for (int i = n - 1; i >= 0; i--) {
            result[i] = -1;
            while (!stack.isEmpty() && nums[stack.peek()] <= nums[i]) {
                stack.pop();
            }
            if (!stack.isEmpty()){
                result[i] = nums[stack.peek()];
            }
            stack.add(i);
        }

        return result;
    }
}
```

written by shawngao original link here

Solution 3

From viewing other solutions I think in some other languages O(n * n) are accepted. Can my code be improved for O(n * n) or do I have to use another language or approach?

```
class Solution(object):
    def nextGreaterElements(self, nums):

        if not nums:
            return []

        n = len(nums)
        result = [-1 for _ in range(len(nums))]
        max_num = max(nums)

        for i, num in enumerate(nums):

            if num == max_num:
                continue

            j = i
            while nums[j] <= num:
                j += 1
                j %= n
            result[i] = nums[j]

        return result
```

written by yorkshire original link here