CS 3201 Algorithm Design 1/2021

6128110 Nguyen Huy Tung

Term Project

Content

1. Introduction

This term project, after filtering questions from many pages, I decide to pick question 503. Next Greater Element II from leetcode.com. This question's difficulty is medium, meet requirement.
Description:

Given a circular integer array `nums` (i.e., the next element of `nums[nums.length - 1]` is `nums[0]`), return *the **next greater number** for every element in* `nums`.

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, return `-1` for this number.

Example 1:

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

Output: [2,-1,2]

Explanation: The first 1's next greater number is 2;

The number 2 can't find next greater number.

The second 1's next greater number needs to search circularly, which is
also 2.
```

Example 2:

```
Input: nums = [1,2,3,4,3]

Output: [2,3,4,-1,4]
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^9 <= nums[i] <= 10^9`

2. Problem

The problem is very straightforward, also short.
The output is a new array which has the same length with input.
This is a data structure problem.
For each of item of the output, it is the next greater element of the original item, we have to search for the next item larger than the original one. If search failed, we could go back to the beginning and search again. If could not, output number would be -1. For short, search next greater element in circular with traversing order from left to right.

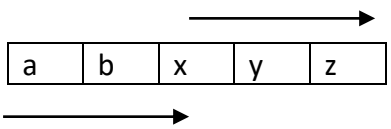
3. Solving Technique

First of all, to deal with problem that need to process the whole bucket, brute-force come up as the first choice.

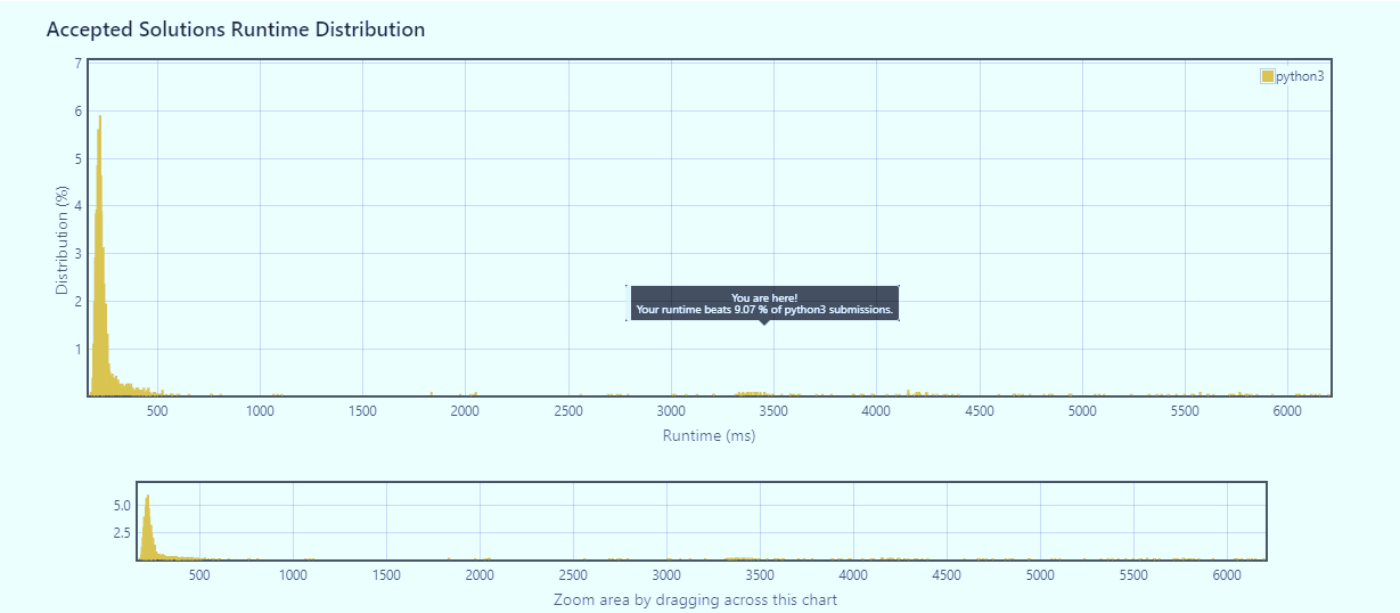
```python
def nextElement(nums):
    result = []
    lenss = len(nums)
    for i in range(lenss):
        ok = False
        for j in range(i,lenss):
            if nums[j] > nums[i]:
                result.append(nums[j])
                ok = True
                break

        if ok == False:
            for k in range(0,i+1):
                if nums[k] > nums[i]:
                    result.append(nums[k])
                    break
                elif k == i:
                    result.append(-1)

    return result
```

For this approach, let call the origin number is x

If I could find the larger number after x, that is the output number, else find again from the beginning.



This is it performance



09/23/2021 09:01	Accepted	3444 ms	15.6 MB	python3

It got accepted, but the running time is very large if compare to other submissions, analyze this I could say, the time complexity is O(n^2) which is not ideal at all.

So for this, I decide to use stack structure for this problem, which suggest by leetcode.

Stack is a LIFO structure.
Example [5,4,3,2,1]
Output [-1,5,5,5,5]
I will start process in reverse way, right to left.
Let call processing number is x
I use this stack to store temp number, if stack is blank push x in to compare with the original next number. The result number will be temporary is -1 as define of biggest number so far
If the stack is not blank, I will start pop each number from the stack to see whether it is the next biggest or not if yes also output as -1
If x not bigger than item y from the stack, output would y
After one time loop, there are many -1 items which is not correct and the stack still having the remaining items, so I with start using the same loop again.
After 1 loop [-1,-1,-1,-1,-1]
Stack remaining 5 5 > 1,2,3,4
So result would be [-1,5,5,5,5]
If the stack is blank and the loop done, operation complete.

```
def nextElement(nums):
    temp = []
    n = len(nums)
    result = [0]*n
    for i in range((n-1)*2,-1,-1):
        if temp == []:
            temp.append(nums[i%n])
            result[i%n] = -1
        else:
            while temp != []:
                if temp[-1] > nums[i%n]:
                    result[i%n] = temp[-1]
                    temp.append(nums[i%n])
                    break
                else:
                    temp.pop()
                if temp == []:
                    temp.append(nums[i%n])
                    result[i%n] = -1

    return(result)
```

The loop is * 2 so I could loop twice the array

For this solution I could analaze it has time complexity O(2n) which is O(n)

Submission Detail

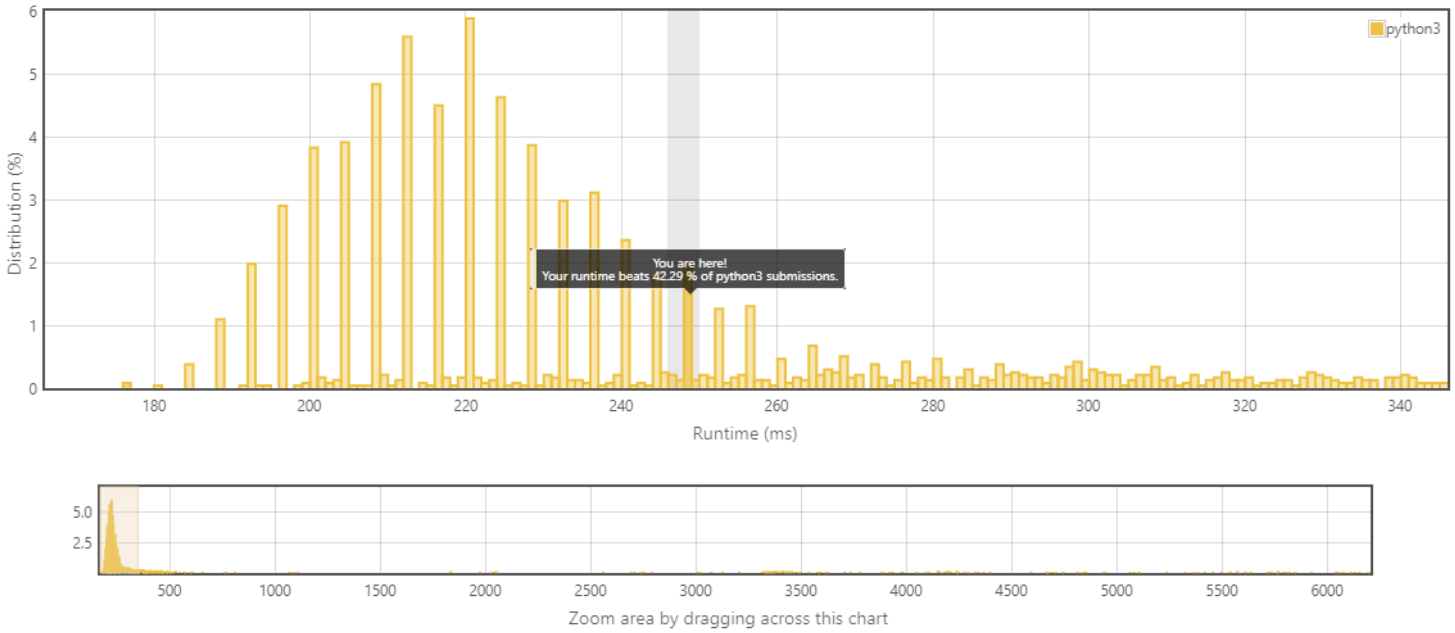
223 / 223 test cases passed.

Runtime: **248 ms**
Memory Usage: **15.6 MB**

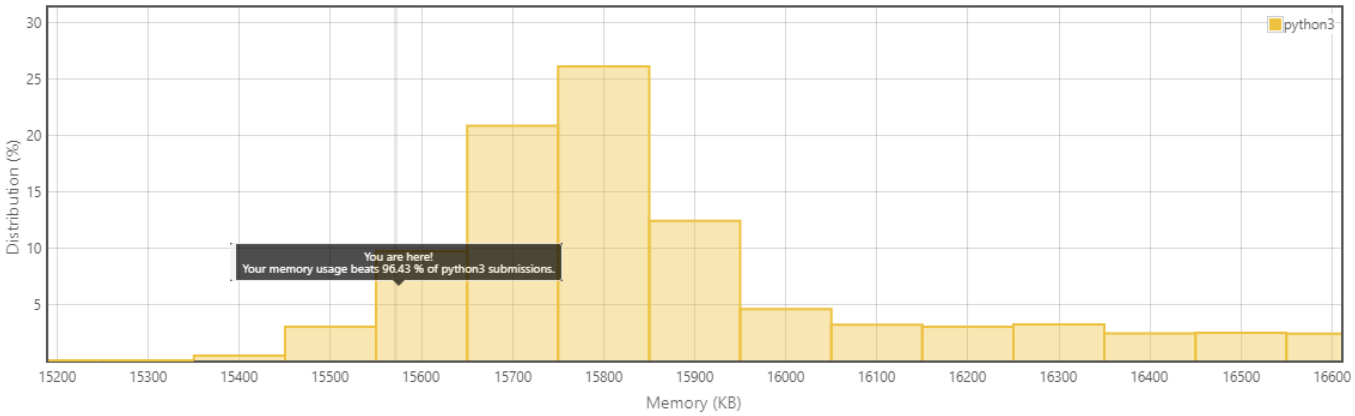
Status: **Accepted**

Submitted: **0 minutes ago**

Accepted Solutions Runtime Distribution



Accepted Solutions Memory Distribution



09/23/2021 09:42	Accepted	244 ms	15.8 MB	python3

4. Conclusion

The first solution I do on my own
The second once I got idea from leetcode.com.
The second solution that I show is not very straight forward, although it is the best solution with incredible runtime with various editions. The stack is just the way to store temporary biggest number then comparing with other number

5. References

https://leetcode.com/problems/next-greater-element-ii/submissions/