# 题目

给定一个循环数组（最后一个元素的下一个元素是数组的第一个元素），输出每个元素的下一个更大元素。数字 x 的下一个更大的元素是按数组遍历顺序，这个数字之后的第一个比它更大的数，这意味着你应该循环地搜索它的下一个更大的数。如果不存在，则输出 -1。

**示例 1:**

输入: [1,2,1]

输出: [2,-1,2]

**解释：**

第一个 1 的下一个更大的数是 2；

数字 2 找不到下一个更大的数；

第二个 1 的下一个最大的数需要循环搜索，结果也是 2。

**注意：**

输入数组的长度不会超过 10000。

# 分析

## 方法一：暴力破解

## 方法二：单调栈

**思路：**

我们可以使用单调栈解决本题。单调栈中保存的是下标，从栈底到栈顶的下标在数组nums中对应的值是单调不升的。

每次我们移动到数组中的一个新的位置i，我们就将当前单调栈中所有对应值小于 nums[i]的下标弹出单调栈，这些值的下一个更大元素即为nums[i]（证明很简单：如果有更靠前的更大元素，那么这些位置将被提前弹出栈）。随后我们将位置i入栈。

但是注意到只遍历一次序列是不够的，例如序列[2,3,1]，最后单调栈中将剩余[3,1]，其中元素[1]的下一个更大元素还是不知道的。

一个朴素的思想是，我们可以把这个循环数组「拉直」，即复制该序列的前n−1个元素拼接在原序列的后面。这样我们就可以将这个新序列当作普通序列，用上文的方法来处理。

而在本题中，我们不需要显性地将该循环数组「拉直」，而只需要在处理时对下标取模即可。

**代码：**

class Solution {

public:

vector<int> nextGreaterElements(vector<int>& nums) {

int n = nums.size();

vector<int> ret(n, -1);

stack<int> stk;

for (int i = 0; i < n \* 2 - 1; i++) {

while (!stk.empty() && nums[stk.top()] < nums[i % n]) {

ret[stk.top()] = nums[i % n];

stk.pop();

}

stk.push(i % n);

}

return ret;

}

};

**复杂度：**

时间复杂度: O(n)，其中n是序列的长度。我们需要遍历该数组中每个元素最多2次，每个元素出栈与入栈的总次数也不超过4次。

空间复杂度: O(n)，其中n是序列的长度。空间复杂度主要取决于栈的大小，栈的大小至多为2n−1。

或：

class Solution {

public:

vector<int> nextGreaterElements(vector<int>& nums) {

int sLen = nums.size();

for (int i = 0; i < sLen; i++) {

nums.push\_back(nums[i]);

}

vector<int> rtAll(nums.size());

stack<int> u;

for (int i = nums.size() - 1; i >= 0; i--) {

while (!u.empty() && u.top() <= nums[i]) {

u.pop();

}

rtAll[i] = u.empty() ? -1 : u.top();

u.push(nums[i]);

}

vector<int> rtPart(rtAll.begin(), rtAll.begin() + nums.size() / 2);

return rtPart;

}

};

或：

vector<int> nextGreaterElements(vector<int>& nums)

{

vector<int> ans(nums.size(), -1);

stack<int> st;

for (int i = 0; i < nums.size() \* 2; i++)

{

int j = i % nums.size();

while (!st.empty() && nums[st.top()] < nums[j])

{

ans[st.top()] = j;

st.pop();

}

st.push(j);

}

for (auto& n : ans)

{

n = (n == -1) ? -1 : nums[n];

}

return ans;

}