# 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：**

**代码：**

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;

}

};

**复杂度：**

时间复杂度：O(N)。

空间复杂度：O(N)。

也可以采用如下编码：

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

int n = nums.size();

vector<int> res(n); // 存放结果

stack<int> s;

// 假装这个数组长度翻倍了

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

while (!s.empty() && s.top() <= nums[i % n])

s.pop();

res[i % n] = s.empty() ? -1 : s.top();

s.push(nums[i % n]);

}

return res;

}

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;

}