# 题目

给定一个数组nums，有一个大小为k的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的k个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

**进阶：**

你能在线性时间复杂度内解决此题吗？

**示例:**

输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3

输出: [3,3,5,5,6,7]

解释:

滑动窗口的位置 最大值

--------------- -----

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6] 7 6

1 3 -1 -3 5 [3 6 7] 7

**提示：**

1 <= nums.length <= 10^5

-10^4<= nums[i]<= 10^4

1 <= k<= nums.length

# 分析

## 方法一：暴力破解

## 方法二：双向队列

**思路：**

由题意可知，这个完全符合双端队列的特性，即不断向右侧移动，左侧不断移除。

**代码：**

class Solution {

public:

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

if(0==nums.size())return {};

vector<int> ret;

deque<int> deq; //存储最大值的下标

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

{

while(!deq.empty() && nums.at(i) > nums.at(deq.back()))

deq.pop\_back();

if(!deq.empty() && deq.front()<i-k+1)deq.pop\_front();

deq.push\_back(i);

if(i>=k-1)ret.push\_back(nums.at(deq.front()));

}

return ret;

}

};

## 方法三：动态规划

## 方法四：堆/优先队列

**思路：**

每次读入一个新的数字，就把不在区间范围内的数字移除堆里，然后堆的顶部就是此次循环的结果。

一个堆其实非常容易实现。我们甚至可以使用一个C++ STL中的 priority\_queue。难点就在于，如何把这个不在区间范围内的数字移除。

**代码：**

class Solution {

public:

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

int n = nums.size();

priority\_queue<pair<int, int>> q;

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

q.emplace(nums[i], i);

}

vector<int> ans = {q.top().first};

for (int i = k; i < n; ++i) {

q.emplace(nums[i], i);

while (q.top().second <= i - k) {

q.pop();

}

ans.push\_back(q.top().first);

}

return ans;

}

};

**复杂度分析：**

时间复杂度：O(nlogn)，其中n是数组nums的长度。在最坏情况下，数组nums中的元素单调递增，那么最终优先队列中包含了所有元素，没有元素被移除。由于将一个元素放入优先队列的时间复杂度为O(logn)，因此总时间复杂度为O(nlogn)。

空间复杂度：O(n)，即为优先队列需要使用的空间。这里所有的空间复杂度分析都不考虑返回的答案需要的O(n)空间，只计算额外的空间使用。