# 题目

设计一个找到数据流中第K大元素的类（class）。注意是排序后的第K大元素，不是第K个不同的元素。

你的KthLargest类需要一个同时接收整数k和整数数组nums的构造器，它包含数据流中的初始元素。每次调用KthLargest.add，返回当前数据流中第K大的元素。

示例:

int k = 3; // 初始化数组大小为3

int[] arr = [4,5,8,2];

KthLargest kthLargest = new KthLargest(3, arr);

kthLargest.add(3); // returns 4

kthLargest.add(5); // returns 5

kthLargest.add(10); // returns 5

kthLargest.add(9); // returns 8

kthLargest.add(4); // returns 8

说明:

你可以假设nums的长度≥k-1且k≥1。

**说明：**类似剑指offer 40

# 分析

**注：对于第K元素问题，推荐使用排序和优先队列！**

## 方法一：排序

## 方法二：数组/堆

**思路：**

维护一个大小为k的小根堆，则根节点就是第k大元素。但是这种方法效率不高。

**代码：**

class KthLargest {

public:

int len=0;

vector<int> a;

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

len=k;

if(nums.size()>0)

{

if(len>nums.size())

{

a.assign(nums.begin(),nums.end());

}

else

{

sort(nums.begin(),nums.end(),ismax);

a.assign(nums.begin(),nums.begin()+k);

}

}

}

int add(int val) {

make\_heap(a.begin(),a.end(),greater<int>());//创建小根堆

a.push\_back(val);

push\_heap(a.begin(),a.end(),greater<int>());

if(len<a.size())//将堆大小控制在k

{

pop\_heap(a.begin(),a.end(),greater<int>());

a.pop\_back();

}

return a[0];

}

static bool ismax(int a,int b)

{

return a>b;

}

};

## 方法三：优先队列（推荐）

**思路：**

C++中的“优先队列（priority\_queue）"，包含在头文件queue中。优先队列具有队列的所有特性，包括基本操作，只是在这基础上添加了内部的一个排序，它本质是一个堆实现的。

**如果不写后两个参数，那么容器默认用的是vector，比较方式默认用operator<，也就是优先队列是大顶堆，队头元素最大，本题为小顶堆**。

定义：priority\_queue<Type, Container, Functional>

Type就是数据类型；

Container就是容器类型（Container必须是用数组实现的容器，比如vector,deque等等，但不能用list。STL里面默认用的是vector）；

Functional就是比较的方式，当需要用自定义的数据类型时才需要传入这三个参数，使用基本数据类型时，只需要传入数据类型，默认是大根堆。

//降序队列（大根堆，默认）

priority\_queue <int,vector<int>,less<int> >q;

//升序队列（小根堆）

priority\_queue <int,vector<int>,greater<int> > q;

我们可以使用一个大小为k 的优先队列来存储前k 大的元素，其中优先队列的队头为队列中最小的元素，也就是第k 大的元素。

在单次插入的操作中，我们首先将元素val 加入到优先队列中。如果此时优先队列的大小大于k，我们需要将优先队列的队头元素弹出，以保证优先队列的大小为 k。

**代码：**

class KthLargest {

int K;

priority\_queue<int, vector<int>, greater<int>> pri\_queue;

public:

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

for (int n : nums) {

pri\_queue.push(n); //入堆的是n不是nums.at(n)

if (pri\_queue.size() > k) pri\_queue.pop(); // 这里是k不是K

//如果全部入堆，则无法获取第k大元素，而是最小的k个数（剑指offer 40），因此这里只维护k大小的堆（适用于第k大/小元素问题）

}

K = k; //标识堆的大小，add的时候使用

}

//堆中增加元素

int add(int val) {

pri\_queue.push(val);

if (pri\_queue.size() > K) pri\_queue.pop();

return pri\_queue.top();

}

};

**另一种写法：**

class KthLargest {

public:

    int heapSize = 0;

    priority\_queue<int,vector<int>,greater<int>> pri\_que;

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

        for(int num : nums)

        {

            pri\_que.push(num);

            if(pri\_que.size()>k)

                pri\_que.pop();

        }

        heapSize = k;

    }

    int add(int val) {

        pri\_que.push(val);

        if(pri\_que.size()>heapSize)

            pri\_que.pop();

        return pri\_que.top();

    }

};

/\*\*

 \* Your KthLargest object will be instantiated and called as such:

 \* KthLargest\* obj = new KthLargest(k, nums);

 \* int param\_1 = obj->add(val);

 \*/

**复杂度分析：**

**时间复杂度：**

初始化时间复杂度为：O(nlogk) ，其中n为初始化时nums的长度；

单次插入时间复杂度为：O(logk)。

**空间复杂度：**O(k)。需要使用优先队列存储前k大的元素。

## 方法四：multiset

**思路：**

利用set自动排序。

**代码：**

class KthLargest {

int K;

multiset<int> st; //从前往后分别是第k大~最大值

public:

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

for (int n : nums) {

st.insert(n);

if (st.size() > k) st.erase(st.begin());

}

K = k;

}

int add(int val) {

st.insert(val);

if (st.size() > K) st.erase(st.begin());

return \*st.begin();

}

};

**复杂度：**