# 2104

# [Problem](2-problem/2104%20--%20K-th%20Number.htm)

## 相似题目

* [编程之美2.5](../../../5-编程之美/2-2.5/2.5.docx)
* [july 伴随数组](../../../4-summary/3-伴随数组/2-problems/002/002.docx)

# 注意

# 分析

# 解法1[朴素法—超时]

## 时间复杂度

近似O(plgp)，p为取值区间的大小。

## 空间复杂度

O(p)， p为取值区间的大小。

## 分析

设选取2-6中的第5个数，则i=2,j=6,k=2。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 6 | 3 | 7 | 4 |

* 将原数组中的第2-6个元素取出，O(j-i+1)=O(5)常数。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 2 | 6 | 3 | 7 |

* 从小到大排列，O(plgp)其中p=j-i+1，表示临时区间的大小。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 7 |

* 取出第K个数，1步取出。

|  |
| --- |
| 3 |

所以每个测试用例的时间复杂度为O(p) + O(plgp) + 1 = O(p(1+lgp))

题目中给出了m个测试用例，所以

近似为O(m\*plgp)。

因为1 <= n <= 100 000, 1 <= m <= 5 000，所以最坏情况下p=n的最大值=100000，m=5000，O(plgp)，O(m\*plgp)。

总的时间要求是小于20s，每个测试用例要求小于1s，计算机每秒大概是百万次运算；80亿/100万/秒=8000秒，轻轻松松超20s，超时。

## [源码](4-solution/solution_1.cc)

# 解法2[维护数组array[k]—超时]

## 时间复杂度

O(k\*(n-k)\*lgk)。

## 空间复杂度

O(k)。

## 分析

使用数组array存储最初的k个元素，对array从小到大进行排序，遍历第k+1个元素一直到最后，如果元素的值小于array[k-1]，则array[k-1]=该元素的值。如此，遍历完取值空间之后，array[k-1]就是我们要找的第k个元素了。

实例如下：

设选取2-6中的第5个数，则i=2,j=6,k=2。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 5 | 2 | 6 | 3 | 7 | 4 |

* int array[2]={5,2}。

|  |  |
| --- | --- |
| 5 | 2 |

* 对array进行快排。

|  |  |
| --- | --- |
| 2 | 5 |

* 依次遍历6,3,7。

3小于5，则array[1]=3，并对array重新快排。

|  |  |
| --- | --- |
| 2 | 3 |

所以第2个元素为3。

* 创建array的时间：O(k)

array排序的时间：O(klgk) ，其实使用**插入排序**和**选择排序**是O(K)的时间复杂度。

因为对取值区间遍历n-k次，所以时间复杂度为：O(k + (n-k)klgk)=O(k\*(n-k)\*lgk)

结果还是超时了。

我使用1000000数组测试，当i=1,j=1000000,k=500000时，许久都没有计算出来。

## [源码](4-solution/solution_2.cc)

# 解法3[维护数组array[k]—超时]

## 时间复杂度

O( (n-k)\*lgk)。

## 空间复杂度

O(k)。

## 分析

与解法2相似，不同的是解法2是利用快排对array排序，本解法使用堆，使用的是最大堆，如果n-k中的元素小于堆顶元素，那么将堆顶置为该元素，然后重新建立堆，时间复杂度是O(lgk)。

## 注意

stl中的make\_heap是O(k)的时间复杂度，需要自己实现一个max\_heapify

## [源码](4-solution/solution_3.cc)

# [解法4[类似快排]](../../../5-编程之美/2-2.5/2.5.docx)

[源码在这](4-solution/solution_4.cc)，poj结果为超时了。

# 解法5[建堆]

## 时间复杂度

O(p+k\*lgp)

## 空间复杂度

O(p)

## 分析

设区间内的数组为array，在array上建立小顶堆，用时O(p)，然后移出k次小顶堆，最后一次移出来的就是第K个数。时间复杂度为O(p+k\*lgp)。

该方法比前面4中解法都快很多，特别是对于测试用例，p=1000000，k=500000时。

但是POJ还是提示Runtime Error。还没有找到原因。

## [源码](4-solution/solution_5.cc)

# 解法6[计数排序]

该方法有待学习

# 解法7[伴随数组]

## 时间复杂度

第一次排序：O(NlgN)

之后每次测试用例：O(N)

## 空间复杂度

O(N)

## 分析

使用伴随数组，参考July：<http://blog.csdn.net/v_JULY_v/article/details/6452100>

例如：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| value | 1 | 5 | 2 | 6 | 3 | 7 | 4 |

对value从小到大排序

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 3 | 5 | 7 | 2 | 4 | 6 |
| value | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

现在要求2,5之间的第3个数。如下所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| value | 1 | 5 | 2 | 6 | 3 | 7 | 4 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 3 | 5 | 7 | 2 | 4 | 6 |
| value | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

我们令k = 3，从遍历数组，如果index在2,5之间那么，k--，当0==k的时候，对应的value就是我们要找的。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| index | 1 | 3 | 5 | 7 | 2 | 4 | 6 |
| value | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| k=3 | 3 | 2 | 1 |  | 0 |  |  |

## [源码](4-solution/solution_7.cc)

# 对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **描述** | **时间复杂度** | **空间复杂度** | **结果** |
| 解法1 | 朴素法，每次都先排序，再直接取值 | 近似O(plgp)  超时 | O(p) |  |
| 解法2 | 使用数组array[k]存储最小的k个元素，对剩余的n-k个元素遍历，array的排序使用快排 | O( (n-k) \*k\*lgk)  超时 | O(k) |  |
| 解法3 | 与解法2相似，不同的是array使用堆来确定array中的最大值 | O( (n-k)\*lgk)  超时 | O(k) |  |
| 解法4 | 快排选择 | O(p) |  |  |
| 解法5 | 建小顶堆，然后取K次堆顶 | O(p+k\*lgp) | O(p) |  |
| 解法6 |  |  |  |  |
| 解法7 | 伴随数组 | 第一次排序O(NlgN)，之后每次都是O(N) | O(N) | AC，不过耗时还是比较长 |