关于“数组中第k小元素问题”的一些思路分享与讨论

怎么求 kth_min ?

题目与说明

问题描述：

给定线性序列中n个元素和一个整数k，1≤k≤n,要求找出这n个元素中第k小的元素。

界定和说明：

1. 第k小是从第1小开始成立的，即第1小、第2小、第3小……第k小；
2. 给定的线性序列无序，且每个元素的key均不同；
3. 给定的线性序列，下以“**原序列**”代指；
4. 如无特殊声明，均使用升序排序（组内/组间）。

思路一

方案：

1) 对原序列先排序，得到有序序列；
2) 访问有序序列下标[k-1]元素；
3) 得到原序列的第 k 小元素 kth_min。

分析：

原问题 ⟹ **排序**

- 程序和程序员有一个能跑就行？

- 也许还可以再进一步思考！ 😀

思路二推导

大家如何求第 1 小元素（最小元素）？

求序列中的最小元素（C++实现）：

```
int min = arr[0];
for (int i = 1; i < n; i++) {
    if (arr[i] < min) {
        min = arr[i];
    }
}
```

思路二推导

大家如何求第 1 小元素（最小元素）？

求序列中的最小元素（C++实现）：

```
int min = arr[0];
for (int i = 1; i < n; i++) {
    if (arr[i] < min) {
        min = arr[i];
    }
}
```

那如何求第 2 小元素？
（基于同样的思路）

思路二推导

如何求第 2 小元素？

伪代码：

遍历一遍序列得第 1 小元素
删除之，得到新序列
遍历一遍新序列得到其最小元素，即原序列第 2 小元素

思路二 解法1

如何求第 k 小元素？

伪代码：

```
kth_min = 0  // 初始化第k小元素变量kth_min
For i = 1 to k
        遍历一遍序列得第 i 小元素 e
        置 kth_min = e
        将e从原序列中删除，得到新序列
Endfor
```

1 找最小值
2 删除
3 循环至 i == k

思路二 解法1

如何求第 k 小元素？

把刚才的思路反过来？

伪代码：

```
kth_min = 0  // 初始化第k小元素变量kth_min
For i = 1 to k
        遍历一遍序列得第 i 小元素 e
        置 kth_min = e
        将e从原序列中删除，得到新序列
Endfor
```

1 找最小值
2 删除
3 循环至 i == k

思路二 解法2

如何求第 k 小元素？递归思路：

求第k小元素，先把前k-1个最小元素删除再遍历
→ 要想求第k-1小元素，先把前k-2个最小元素删除再遍历
→ …
→ 要想求第2小元素，先把前1个最小元素删除再遍历
→ 第 1 最小元素，可求已知

递归三要素：

1) 大问题可以被分解为有限个子问题;
2) 所有子问题求解方式均相同;
3) 具有已知的最小子问题。

思路二 解法1

如何求第 k 小元素？

不满意

伪代码：

```
kth_min = 0  // 初始化第k小元素变量kth_min
For i = 1 to k
        遍历一遍序列得第 i 小元素 e
        置 kth_min = e
        将e从原序列中删除，得到新序列
Endfor
```

1 找最小值
2 删除
3 循环至 i == k

思路二 解法3

如何求第 k 小元素？

伪代码：

```
kth_min = 0
For i = 1 to k
        MinHeapify(序列)
        置 kth_min = 堆顶
        从原序列中移除堆顶
Endfor
// 循环结束后，kth_min中元素即目标元素
```

- 小根堆！！

- 只是减少求最小值的时间，不影响算法主体

1 找最小值
2 删除
3 循环至 i == k

回顾思路二

求第 1 小元素算法，它的底层逻辑是什么？

求序列中的最小元素：

```
int min = arr[0];      // 先假设首元素是最小元素
for (int i = 1; i < n; i++) {
    if (arr[i] < min) {
        min = arr[i];    // 再更新min
    }
}
```

// 从变量 min 的角度看，它一定逐步减少！(暂定值不变也是减少)

思路三推导

如何求解?

求原序列**第 k 小元素**

原序列的**前 k 小元素序列 S_min**

序列 S_min 中的最大元素就是 “第 k 小元素 ”，即目标元素

思路三推导

问题转化：

原问题：
求原序列第 k 小元素

求原序列的<u>前 k 小元素序列 S_min</u>

求序列 S_min 中的最大元素，即目标

思路三推导：举个栗子

- 原序列：6 1 5 0 4 3 2
- 求前 3 小元素序列（k = 3）

1. 1 3 5 0 4 6 2 // **假设**前 k 个元素是 S_min
2. 1 3 5 0 4 6 2 // 从下标 k 开始遍历
3. 1 3 5 **0** 4 6 2 // 当前元素是 0，小于 S_min 中最大值 5
4. 1 3 0 **5** 4 6 2 // **较大的 5 换成较小的 0**，继续遍历
5. 1 3 0 5 **4** 6 2 // 4, 大于 S_min 中新的最大值 3，跳过
6. 1 3 0 5 4 **6** 2 // 6, 也大于 3，跳过
7. 1 3 0 5 4 6 **2** // 2, 小于 S_min 中最大值 3！换！
8. **1 2 0** 5 4 6 3 // 循环结束，S_min 中是原序列的前 k 小元素

思路三推导

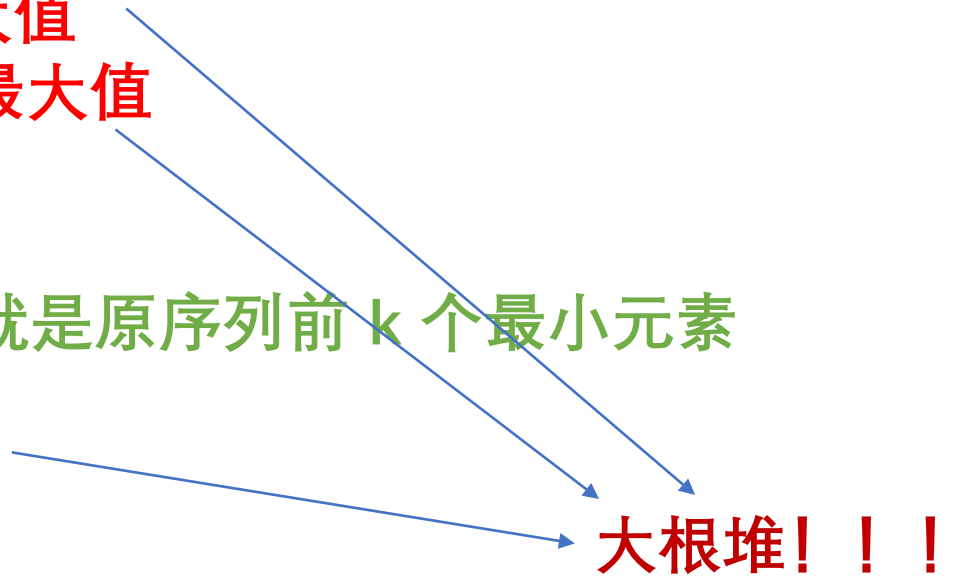
伪代码：

```
序列S_min = 原序列前 k 个元素    // 假设它们是前 k 小元素
For i = k to n
        if arr[i] < S_min中最大值
                arr[i]替换那个最大值
        Endif
Endfor
// For循环结束后，S_min中就是原序列前 k 个最小元素
kth_min = S_min中最大值
```

思路三推导

伪代码：

```
序列S_min = 原序列前 k 个元素    // 假设它们是前 k 小元素
For i = k to n
        if arr[i] < S_min中最大值
                arr[i]替换那个最大值
        Endif
Endfor
// For循环结束后，S_min中就是原序列前 k 个最小元素
kth_min = S_min中最大值
```



思路三实现

// Talk is cheap， show me your code.

伪代码：

```
序列S_min = 原序列前 k 个元素    // 假设它们是前 k 小元素
MaxHeapify(S_min)
For i = k to n
        if arr[i] < S_min堆顶
                将 arr[i] 替换为堆顶
                MaxHeapify(S_min) // 重新堆化，求新的最大值
        Endif
Endfor
// For循环结束后，S_min中就是原序列前 k 个最小元素
kth_min = S_min堆顶
```

当 k = 1时，算法退化为遍历找最小元素，大根堆中仅1元素
当 k = n 时，算法退化为对原序列化大根堆，再取堆顶

时间复杂度：O(nlogk)，1 <= k <= n
多项式： $T(n) <= 4k + c(n-k)\log_2 k$

思路三总结

①得到最小与②得到前k小元素对比表格：

	得到最小（前1小）	得到前 k 小
思想	假设 + 遍历更新	假设 + 遍历更新
中间量	min变量（或min_index)	前k小元素序列S_min
中间量变化	逐步减小	总体逐步减小

思路四 基于partition()划分函数

伪代码：

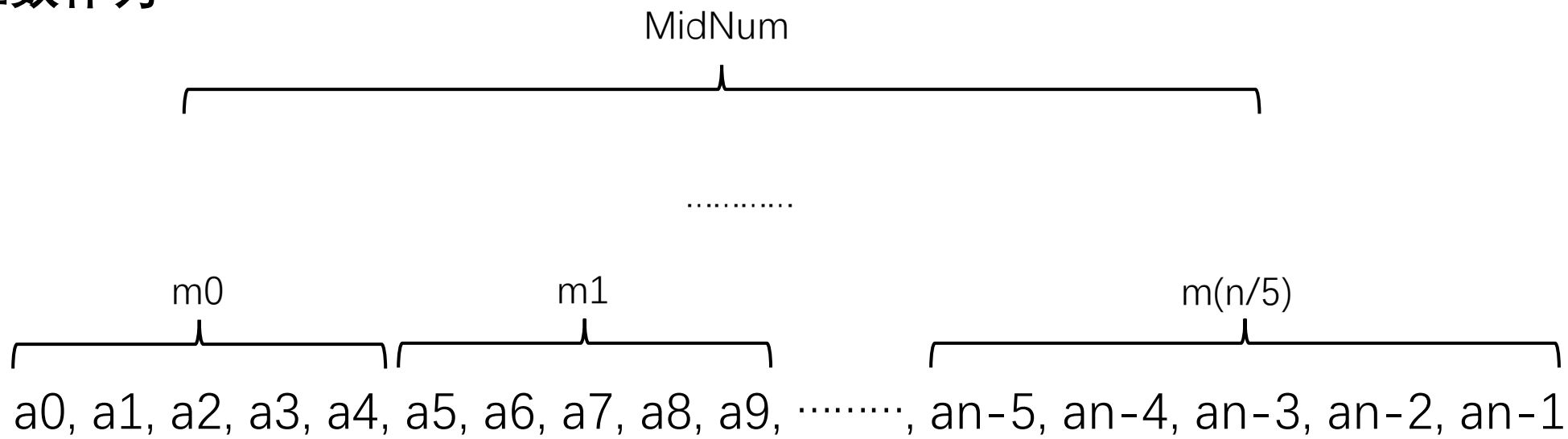
```
low = 0, high = len-1, pos = 0, k = k
while (true) :
        pos = partition(原序列, low, high)
        j = pos – low + 1     // 转换为 pos 的相对位置
        if j == k : break
        elseif k < j : high = pos – 1
        else :
                low = pos + 1
                k -= j

kth_min = 原序列[pos]
```

时间复杂度： $T(n) = 1 * T(n/2) + cn^1 = O(n)$

* 但在极端情况下表现不佳

思路五 4plus

取中位数作为枢轴：



将得到的原序列中位数MidNum作为partition()的枢轴

优点：避免思路④的极端情况，稳定
缺点：多了很多操作，耗费更多时间与空间

运行结果对比

n = 1,0000,0000

	k = 1	k = n/2	k = n
思路一 : Sort()	47.102s	47.122s	47.154s
思路三 : Heap()	0.468s	42.870s	2.885s
思路四 : partition()	1.095s	2.895s	1.262s

稳定的慢
两端快, 中间慢
快!

（无耻偷袭）

1 百万个 同key元素

k = 1	k = n/2	k = n
0.424s	0.425s	0.424s
0.004s	0.005s	0.006s
? ? ?	? ? ?	? ? ?

// 运行过久，被kill了

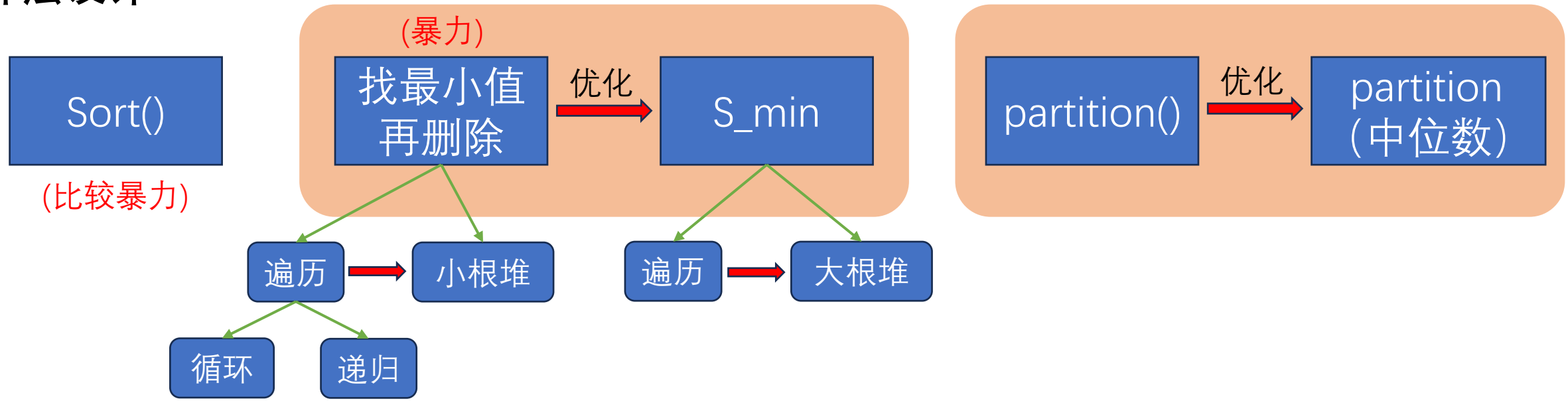
一般情况下:

partition()　优于　Heap()　优于　Sort()

* 具体算法的选择，要根据实际应用场景的需求

总结回顾

算法设计：



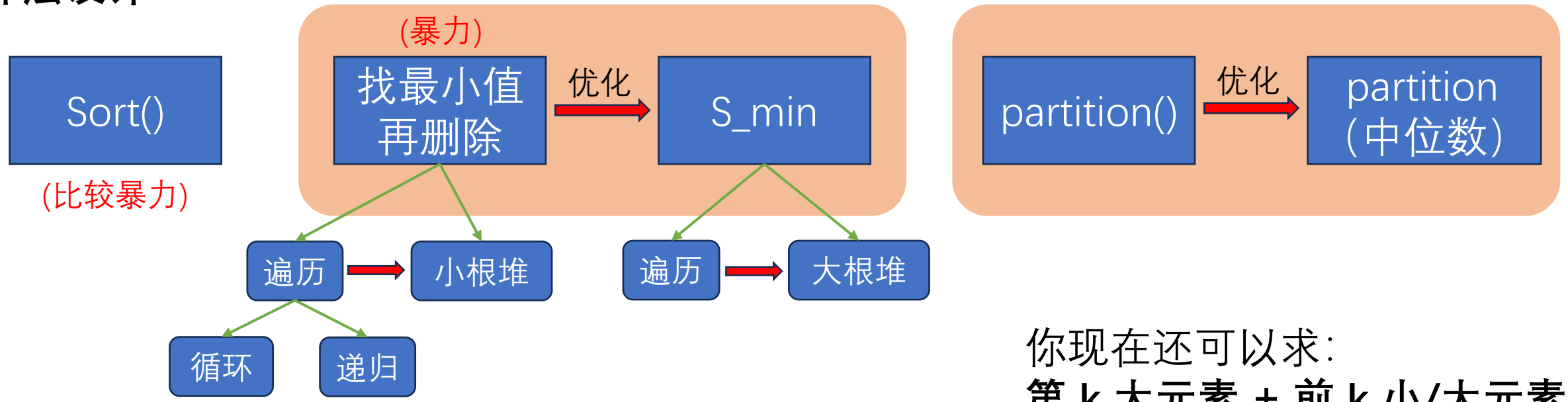
算法分析标准：

算法有效性：如果一个算法与同问题的暴力算法比较，能在最坏情况下达到比暴力算法更好的性能，就说它是有效的。

我们试图找到比思路1 2更高性能的算法，我们找到了！

总结回顾

算法设计：



你现在还可以求：
第 k 大元素 + 前 k 小/大元素

算法分析标准：

算法有效性：如果一个算法与同问题的暴力算法比较，能在最坏情况下达到比暴力算法更好的性能，就说它是有效的。

我们试图找到比思路1 2更高性能的算法，我们找到了！

结束语

“凡事预则立，不预则废”

邮箱：487882183@qq.com

https://github.com/hk416hasu/Kth_min_speech