

期中实验报告

姓名：徐恒 学号：2050035 日期：2022 年 11 月 23 日

1. 选择问题

问题定义：给定线性序集中 n 个元素和一个整数 k , $1 \leq k \leq n$, 要求找出这 n 个元素中第 k 小的元素, (这里给定的线性集是无序的)。下面三种是可行的方法:

(1) 基于堆的选择: 不需要对全部 n 个元素排序, 只需要维护 k 个元素的最大堆, 即用容量为 k 的最大堆存储最小的 k 个数, 总费时 $O(k + (n-k) * \log k)$

(2) 随机划分线性选择 (教材上的 RandomizedSelect): 在最坏的情况下时间复杂度为 $O(n^2)$, 平均情况下期望时间复杂度为 $O(n)$ 。

(3) 利用中位数的线性时间选择: 选择中位数的中位数作为划分的基准, 在最坏情况下时间复杂度为 $O(n)$ 。

请给出以上三种方法的算法描述, 用你熟悉的编程语言实现上述三种方法。并通过实际用例测试, 给出三种算法的运行时间随 k 和 n 变化情况的对比图 (表)。

1.1. 算法思想

(1) 基于堆的选择:

- 1、创建一个新数组用于存放一个大根堆, 将原数组的前 k 个搞成大根堆存放到新数组中。
- 2、从第 k 个开始遍历到原数组最后, 只要存在元素比堆顶元素小则将堆顶用该元素替换, 然后维护大根堆。
- 3、取出大根堆的堆顶元素就是第 k 小的数了

(2) 随机划分线性选择:

利用随机函数产生划分基准, 将数组 $a[p:r]$ 划分成两个子数组 $a[p:i]$ 和 $a[i+1:r]$, 使 $a[p:i]$ 中的每个元素都不大于 $a[i+1:r]$ 中的每个元素。接着 " $j=i-p+1$ " 计算 $a[p:i]$ 中元素个数 j 。如果 $k \leq j$, 则 $a[p:r]$ 中第 k 小元素在子数组 $a[p:i]$ 中, 如果 $k > j$, 则第 k 小元素在子数组 $a[i+1:r]$ 中。注意: 由于已知子数组 $a[p:i]$ 中的元素均小于要找的第 k 小元素, 因此, 要找的 $a[p:r]$ 中第 k 小元素是 $a[i+1:r]$ 中第 $k-j$ 小元素。

(3) 利用中位数的线性时间选择:

- 1、将所有的数 n 个以每 5 个划分为一组共组, 将不足 5 个的那组忽略, 然后用任意一种排序算法, 因为只对 5 个数进行排序, 所以任取一种排序法就可以了。将每组中的元素排好序再分别取每组的中位数, 得到个中位数。
- 2、取这个中位数的中位数, 如果是偶数, 就找它的 2 个中位数中较大的一个作为划分基准。
- 3、将全部的数划分为两个部分, 小于基准的在左边, 大于等于基准的放右边。

1.2 编程语言及环境

Microsoft Visual Studio Community 2022 (64 位) 版本 17.2.2

1.3 系统输入输出结果

第一组数据:

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
10
3 3 8 2 4 7 1 3 4 4
5
3
程序运行N次所需时间: 55.155毫秒请按任意键继续. . .
```

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
10
3 3 8 2 4 7 1 3 4 4
5
3
程序运行N次所需时间: 26.592毫秒请按任意键继续. . .
```

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
10
3 3 8 2 4 7 1 3 4 4
5
3
程序运行N次所需时间: 14.077毫秒请按任意键继续. . .
```

第二组数据:

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
50
7 3 39 23 39 18 3 6 10 24 2 34 17 37 3 35 15 3 9 1 23 2 11 15 38
33 17 12 26 37 17 5 8 5 2 3 38 9 3 35 28 26 28 13 35 38 30 33 3
22
25
17
程序运行N次所需时间: 2.828毫秒请按任意键继续. . .
```

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
50
7 3 39 23 39 18 3 6 10 24 2 34 17 37 3 35 15 3 9 1 23 2 11 15 38
33 17 12 26 37 17 5 8 5 2 3 38 9 3 35 28 26 28 13 35 38 30 33 3
22
25
17
```

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe

```
50
7 3 39 23 39 18 3 6 10 24 2 34 17 37 3 35 15 3 9 1 23 2 11 15 38
33 17 12 26 37 17 5 8 5 2 3 38 9 3 35 28 26 28 13 35 38 30 33 3
22
25
17
程序运行N次所需时间: 7.902毫秒请按任意键继续. . .
```

第三组数据:

```
C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe
100
24 26 41 91 58 88 36 9 88 63 29 5 13 68 37 52 63 69 39 28 59 74
46 96 83 93 21 32 78 91 17 58 99 86 25 36 62 98 8 43 57 58 56 52
97 38 7 38 66 46 16 78 28 60 25 76 47 98 71 14 57 0 17 53 97 89
30 51 6 44 3 71 44 44 54 35 51 25 48 56 95 32 80 43 37 90 19 82
86 25 72 97 81 18 7 76 37 36 38 88
50
51
程序运行N次所需时间: 3.831毫秒请按任意键继续. . .

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe
100
24 26 41 91 58 88 36 9 88 63 29 5 13 68 37 52 63 69 39 28 59 74
46 96 83 93 21 32 78 91 17 58 99 86 25 36 62 98 8 43 57 58 56 52
97 38 7 38 66 46 16 78 28 60 25 76 47 98 71 14 57 0 17 53 97 89
30 51 6 44 3 71 44 44 54 35 51 25 48 56 95 32 80 43 37 90 19 82
86 25 72 97 81 18 7 76 37 36 38 88
50
51

C:\Users\YuDan\source\repos\Project7\x64\Debug\Project7.exe
100
24 26 41 91 58 88 36 9 88 63 29 5 13 68 37 52 63 69 39 28 59 74
46 96 83 93 21 32 78 91 17 58 99 86 25 36 62 98 8 43 57 58 56 52
97 38 7 38 66 46 16 78 28 60 25 76 47 98 71 14 57 0 17 53 97 89
30 51 6 44 3 71 44 44 54 35 51 25 48 56 95 32 80 43 37 90 19 82
86 25 72 97 81 18 7 76 37 36 38 88
50
53
程序运行N次所需时间: 10.013毫秒请按任意键继续. . .
```

1.4 算法分析

(1) 构建最大堆的时间复杂度为 $O(k)$ ，遍历数组剩余部分的时间复杂度为 $O(n-k)$ ，每次调整堆的时间复杂度为 $O(\log k)$ ，总的时间复杂度为 $O(k + (n-k) * \log k)$ 。

(2) Partition 的时间复杂度为 $O(n)$ ，RandomizedPartition 的时间复杂度为 $O(1)$ ，RandomizedSelect 的时间复杂度在最大元素处划分时为 $O(n^2)$ 。算法整体的时间复杂度最差为 $O(n^2)$ ，但平均复杂度与 n 呈线性关系，为 $O(n)$ 。

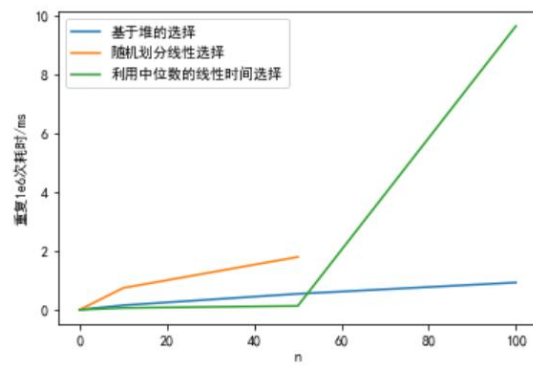
(3) 如果能在线性时间内找到一个划分基准使得按这个基准所划分出的 2 个子数组的长度都至少为原数组长度的常数 C 倍 ($0 < C < 1$)，那么就可以在最坏情况下用 $O(n)$ 时间完成选择任务。例如，当 $C=9/10$ ，算法递归调用所产生的子数组的长度至少缩短 $1/10$ 。所以，在最坏情况下，算法所需的计算时间 $T(n)$ 满足递推式 $T(n) \leq T(9n/10) + O(n)$ 。由此可得 $T(n) = O(n)$ 。即该算法在最坏情况下时间复杂度为 $O(n)$ 。

1.5 调试分析（遇到的问题 and 解决方法）

理论上利用中位数的线性时间选择算法时间复杂度最低，但选取 $n=100$ 时的例子发现，其耗时反而更大。

(1) 考虑 n

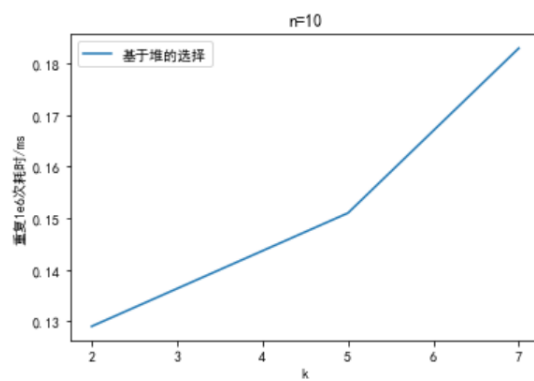
随着 n 的增大，算法的耗时均增大。



(2) 考虑 k

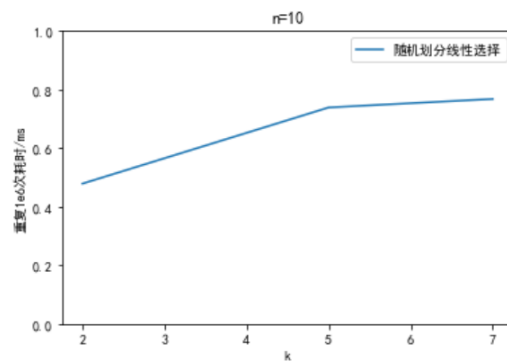
① 基于堆的选择

时间大致随 k 的增大而增大，因为 k 越大，需要建立并维护的堆就越大。

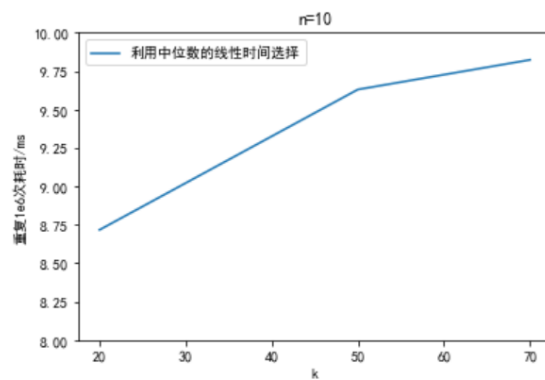


② 随机划分线性选择

由于划分基准随机选取,时间与 k 基本上无关。



③ 利用中位数的线性时间选择



2. 主元素问题

设 A 是含有 n 个元素的数组，如果元素 x 在 A 中出现的次数大于 $n/2$ ，则称 x 是 A 的主元素，

(1) 如果 A 中的元素是可以排序的，设计一个 $O(n\log n)$ 时间的算法，判断 A 中是否存在主元素；

(2) 对于 (1) 中可排序的数组，能否设计一个 $O(n)$ 时间的算法；

(3) 如果 A 中元素只能进行“是否相等”的测试，但是不能排序，设计一个算法判断 A 中是否存在主元素。

2.1 算法思想

(1) 基于堆的选择：

1、快速排序查找出序列的中位数

2、验证其是否为主元素，若为主元素则存在，否则不存在。

(2) 随机划分线性选择：

用 $candidate$ 来储存临时认为的主元素， $count$ 来记录在已遍历序列内主元素与非主元素的个数之差。最初： $count=0$ 。对序列 $a[0:n-1]$ 遍历：

If $count = 0$ ：说明已遍历序列中是偶数个数，并且两两不同元素相消后不会剩余。可以用后面未遍历序列的中位数取代完整序列的主元素。即重新从当前元素开始， $count=1$ ， $candidate$ 为当前遍历的元素 $a[i]$ 。

If else: $candidate == a[i]$ ：已遍历序列内主元素与非主元素的个数之差 + 1 ($count++$)

Else: 已遍历序列内主元素与非主元素的个数之差 - 1 ($count--$)

最终 $candidate$ 内的元素可以暂定为答案，最后进行验证：该答案是否为主元素。

(3) 利用中位数的线性时间选择：

a. 若 T 只含一个元素，则此元素就是主元素，返回此数。

b. 将 T 分为两部分 $T1$ 和 $T2$ (二者元素个数相等或只差一个)，分别递归调用此方法求其主元素 $m1$ 和 $m2$ 。

c. 若 $m1$ 和 $m2$ 都存在且相等，则这个数就是 T 的主元素，返回此数。

d. 若 $m1$ 和 $m2$ 都存在且不等，则分别检查这两个数是否为 T 的主元素，若有则返回此数，若无则返回空值。

e. 若 $m1$ 和 $m2$ 只有一个存在，则检查这个数是否为 T 的主元素，若是则返回此数，若否就返回空值。

f. 若 $m1$ 和 $m2$ 都不存在，则 T 无主元素，返回空值。

2.2 编程语言及环境

Microsoft Visual Studio Community 2022 (64 位) 版本 17.2.2

2.3 系统输入输出结果

第一组数据：

```
C:\Users\YuDan\source\repos\Project7\x64\Del
15
1 3 5 8 15 2 4 8 11 12 7 13 13 2 6
数组A没有主元素请按任意键继续. . .
```

```
C:\Users\YuDan\source\repos\Project7\x64\De
15
1 3 5 8 15 2 4 8 11 12 7 13 13 2 6
数组A没有主元素请按任意键继续. . .
```

```
C:\Users\YuDan\source\repos\Project7\x64\De
15
1 3 5 8 15 2 4 8 11 12 7 13 13 2 6
数组A没有主元素请按任意键继续. . .
```

第二组数据:

```
C:\Users\YuDan\source\repos\Project7\x64\De
15
1 3 5 5 5 5 5 2 8 7 5 5 5 5 2
数组A的主元素为5请按任意键继续. . .
```

```
C:\Users\YuDan\source\repos\Project7\x64\De
15
1 3 5 5 5 5 5 2 8 7 5 5 5 5 2
数组A的主元素为5请按任意键继续. . .
```

```
C:\Users\YuDan\source\repos\Project7\x64\De
15
1 3 5 5 5 5 5 2 8 7 5 5 5 5 2
数组A的主元素为5请按任意键继续. . .
```

2.4 算法分析

(1) 快速排序当中运用到了分治的策略, Partition 产生划分基准并对 $a[]$ 进行划分, QuickSort 分治地对左半段和右半段进行排序。快速排序算法整体在平均情况下的时间复杂度为 $O(n\log n)$; 而在 Judge 中快速排序后的有序数组 $a[]$ 进行判断, 若等于 $a[]$ 中位数的元素个数大于 $n/2$ 则找到了主元素, Judge 算法的时间复杂度为 $O(n)$ 。而快速排序和 Judge 是并列关系, 因此算法整体的时间复杂度为 $O(n\log n)+O(n)=O(n\log n)$ 。

(2) 利用快速排序的思想寻找到数组的中位数的时间复杂度为 $O(n)$, 遍历数组判断该中位数是否为主元素的时间复杂度为 $O(n)$ 。而查找中位数和判断是否为主元素是并列关系, 因此算法整体的时间复杂度为 $O(n)+O(n)=O(n)$ 。

(3) 寻找可能的主元素 candidate 的算法时间复杂度为 $O(n)$, 而通过遍历数组 $a[]$ 判断 candidate 是否真的是数组 $a[]$ 主元素的算法时间复杂度为 $O(n)$ 。上述两个算法是并列关系, 因此算法整体的时间复杂度为 $O(n)+O(n)=O(n)$ 。

3. 博物馆警卫巡逻问题

凸多边形是每个内角小于 180° 的多边形。

博物馆是具有 n 个顶点的凸多边形的形状。博物馆由警卫队通过巡逻来确保馆内物品的安全。博物馆的安全保卫工作遵循以下规则，以尽可能时间经济的方式确保最大的安全性：

(1) 警卫队中每个警卫巡逻都沿着一个三角形的路径；该三角形的每个顶点都必须是多边形的顶点。

(2) 警卫可以观察其巡逻路径三角形内的所有点，并且只能观察到这些点；我们说这些点由该警卫守护并覆盖。

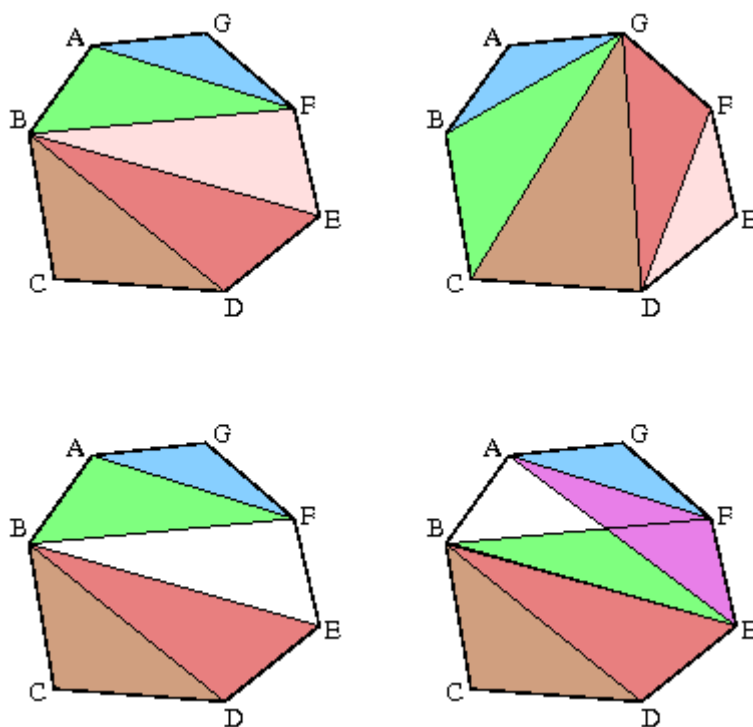
(3) 博物馆内的每一处都必须由警卫人员守护。

(4) 任何两个警卫巡逻所在的三角形在其内部不会重叠，但它们可能具有相同的边。

在这些限制条件下，警卫的成本是警卫巡逻所沿路径的三角形的周长。

我们的目标是找到一组警卫，以使警卫队的总成本（即各个警卫的成本之和）尽可能小。给定博物馆顶点的 x 坐标和 y 坐标以及这些顶点沿博物馆边界的顺序，设计一种算法求解该问题，并给出算法的时间复杂性。

请注意，我们并未试图最小化警卫人数。我们希望使警卫队巡逻的路线的总长度最小化，假定任何线段的长度都是线段端点之间的欧式距离，并且可以在常数时间内计算该长度。



上面是说明本问题的四个图形。博物馆是多边形 $ABCDEFG$ （顶点的逆时针序）。每个彩色（阴影）三角形对应一个警卫。

上面的两个图显示了一组警卫（它们的三角形），它们满足安全保卫规则。在左上方，警卫队巡逻了三角形 AFG （蓝色）， ABF （绿色）， BEF （淡红色）， BDE （浅红色）和 BCD （棕色）的边界。在右上方，守卫巡逻 ABG （蓝色）， BCG （绿色）， CDG （棕色）， DFG （淡红色）和 DEF （浅红色）。

底部的两个图显示了一组不满足这些规则的三角形：在左下图中，博物馆的一部分没

有任何警卫守护（覆盖）（无阴影三角形 BEF），而在右下图，粉色三角形（AEF）和绿色三角形（BEF）相交。

3.1 算法思想

该问题被称为“凸多边形最优三角剖分”问题。对于含有 $(n+1)$ 个顶点的凸多边形，它的顶点集合为 $\{V_0, V_1, \dots, V_n\}$ ，若其最优三角剖分包含三角形 V_0, V_k, V_n ，其中 $1 \leq k \leq n$ ，则该三角剖分的总长度就是三角形 $V_0V_kV_n$ 的周长加上多边形 $\{V_0, V_1, \dots, V_k\}$ 的总长度加上多边形 $\{V_k, V_{k+1}, \dots, V_n\}$ 的总长度三者之和，其中两个小多边形的总长度也一定是对应的最优三角剖分，因此具有最优子结构性质。在程序中令 $m[i][j]$ 表示凸多边形 $\{V_{i-1}, V_i, \dots, V_j\}$ 的最优三角剖分对应的总长度， $C_triangle[i][k][j]$ 表示三角形 ikj 周长，则递推关系如下：

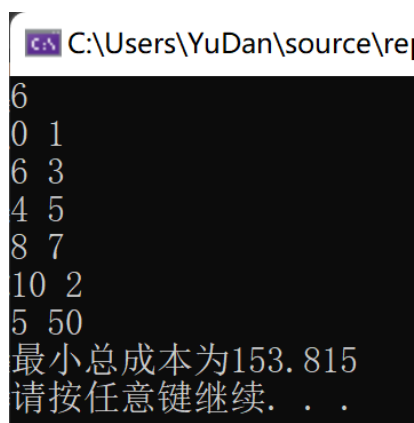
$$m[i][j] = \begin{cases} 0, & i = j \\ \min_{i \leq k < j} \{m[i][k] + m[k+1][j] + C_triangle(V_i V_k V_j)\}, & i < j \end{cases}$$

通过动态规划方法可解决。

3.2 编程语言及环境

Microsoft Visual Studio Community 2022（64 位）版本 17.2.2

3.3 系统输入输出结果



```
C:\Users\YuDan\source\re
6
0 1
6 3
4 5
8 7
10 2
5 50
最小总成本为153.815
请按任意键继续. . .
```

3.4 算法分析

$C_triangle$ 的时间复杂度为 $O(1)$ ，而 $\text{int main}()$ 当中有三个呈包含关系的 for 循环，每个循环的变量都为 $O(n)$ 数量级，因此算法整体时间复杂度为 $O(n^3)$ 。

3.5 调试分析（遇到的问题 and 解决方法）

时间复杂度较高