数据结构与算法期末复习
Homework

尹超

中国科学院大学，北京 100049

Carter Yin

University of Chinese Academy of Sciences, Beijing 100049, China

2024.8 – 2025.1

序言

本文为笔者数据结构与算法的期末复习笔记。
望老师批评指正。

目录

题目 第十三章排序

211

> 迄今为止，我们已经学过许多种排序算法了，请根据描述选择对应的算法(请填入选项大写字母）A, 快速排序 B, 堆排序 C, 归并排序 D, 插入排序 E, 冒泡排序

Solution 1. 反复比较相邻元素，若为逆序则交换，直至有序：***E***

- **分析：**这是冒泡排序的经典定义。它通过重复地遍历待排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。

将原序列以轴点为界分为两部分，递归地对它们分别排序：***A***

- **分析：**这是快速排序的核心思想（“分治法”）。它选取一个“轴点”（*pivot*），将序列分割成小于轴点和大于轴点的两部分，然后对这两部分递归地进行快速排序。

将原序列前半部分和后半部分，分别排序，再将这两部分合并：***C***

- **分析：**这是归并排序的核心思想（也是“分治法”）。它将序列递归地对半分割，直到每个子序列只有一个元素，然后再将这些有序的子序列逐层合并，最终得到完全有序的序列。

不断从原序列中取出最小元素：***B***

- **分析：**这可以看作是选择排序的宏观思想，而堆排序是该思想的一种高效实现。通过构建一个最小堆（*Min-Heap*），根节点总是当前序列中的最小元素。不断地取出根节点（*delMin*），就可以得到一个有序序列。

抓扑克牌时人们常用的排序方法：***D***

- **分析：**这是对插入排序最形象的比喻。当我们整理手中的扑克牌时，通常会拿起一张新牌，然后将它插入到手中已有序的牌的正确位置。

212

> 对于规模为 n 的向量，快速排序在平均情况下得时间复杂度为 A. O(n^2) B. O(nlogn) C. O(n) D. O(n+logn)

Solution 2. 正确答案是 *B*。

详细分析：

(1) **快速排序的核心思想：**快速排序是一种采用“分治法”（*Divide and Conquer*）策略的排序算法。它通过一个“划分”（*Partition*）操作，将一个数组分成两个子数组，然后对这两个子数组进行递归排序。

(2) **时间复杂度分析：**

- **划分操作：**每一次划分操作都需要遍历当前子数组，其时间复杂度与子数组的规模成正比，为 $O(k)$，其中 *‘k‘* 是子数组的长度。
- **平均情况：**在平均情况下，我们可以期望每次选择的“轴点”（*pivot*）能够将数组划分成大致相等（或比例固定）的两个部分。例如，每次都划分为 *‘n/2‘* 和 *‘n/2‘*。这会导致递归的深度为 $O(\log n)$。由于每一层递归的所有划分操作加起来的总时间是 $O(n)$，所以总的平均时间复杂度是 $O(n \log n)$。
- **最坏情况：**当每次选择的轴点都是当前子数组的最大或最小元素时（例如，在一个已经有序的数组中总是选择第一个元素作为轴点），划分会极度不均衡（划分为 *0* 和 *‘k-1‘*）。这会导致递归深度达到 $O(n)$，总时间复杂度退化为 $O(n^2)$。

结论：尽管快速排序的最坏情况时间复杂度是 $O(n^2)$，但通过良好的轴点选择策略（如随机选择、三数取中等），这种情况在实际应用中极少发生。其平均情况下的性能非常出色，时间复杂度为 $O(n \log n)$。

213

> 对序列 A[0, n) 用快速排序算法进行排序，u 和 v 是该序列中的两个元素。
> 在排序过程中，u 和 v 发生过比较，当且仅当（假定所有元素互异）：
> A. u < v
> B. u 在某次被选取为轴点
> C. 对于所有介于 u 和 v 之间的元素（包括 u 和 v 本身），它们之中第一个被选为轴点的是 u 或者 v
> D. 所有比 u 和 v 都小的元素都始终没有被选为轴点

Solution 3. 正确答案是 *C*。

详细分析：

***(1)* 快速排序的比较机制：**在快速排序的任何一步中，元素之间的比较都只发生在**当前子数组的元素与该次划分选出的轴点（*pivot*）之间**。一旦划分完成，位于轴点左侧子数组的元素将永远不会与右侧子数组的元素进行比较。

***(2)* *u* 和 *v* 被比较的条件：**两个元素 *'u'* 和 *'v'* 要想发生比较，它们必须在某一次划分操作中，一个作为轴点，另一个作为普通元素。这意味着，在它们两个被选为轴点之前，它们必须始终位于同一个待排序的子数组中。

***(3)* *u* 和 *v* 被分离的条件：**考虑所有值在 *'u'* 和 *'v'* 之间的元素的集合（包括 *'u'* 和 *'v'*）。假设 *'u < v'*，这个集合就是 *'S = x | u <= x <= v'*。

- 如果从集合 *'S'* 中被选为轴点的第一个元素是 *'u'* 或 *'v'*，那么另一个元素必然还在同一个子数组中，因此它们会发生比较。
- 但是，如果从集合 *'S'* 中被选为轴点的第一个元素是某个 *'p'*，其中 *'u < p < v'*，那么在围绕 *'p'* 进行划分时：
  - *'u'* 会被分到 *'p'* 的左边。
  - *'v'* 会被分到 *'p'* 的右边。

  在此之后，*'u'* 和 *'v'* 就被分到了不同的子数组中，它们再也没有机会进行比较了。

结论：因此，*'u'* 和 *'v'* 能够发生比较的充要条件是：在所有值介于 *'u'* 和 *'v'* 之间的元素中，第一个被选为轴点的必须是 *'u'* 或 *'v'* 本身。这正是选项 *C* 的描述。

其他选项分析：

- *A. u < v:* 这只是一个大小关系，与是否比较无关。
- *B. u* 在某次被选取为轴点*:* 这不充分。如果 *'v'* 在 *'u'* 被选为轴点之前，已经被另一个轴点 *'p' (u < p < v)* 分离出去了，它们就不会比较。
- *D.* 所有比 *u* 和 *v* 都小的元素都始终没有被选为轴点*:* 这不正确。一个比 *'u'* 和 *'v'* 都小的元素 *'p'* 可以被选为轴点，此时 *'u'* 和 *'v'* 会一起被分到右侧子数组，它们仍然有机会在后续的划分中进行比较。

214

> 快速排序算法选取轴点时可以采取不同的策略，本题试图用实例说明“三者取中”的策略比随机选取的策略倾向于得到更平衡的轴点设待排序序列的长度 n 很大，若轴点的选取使得分割后长/短子序列的长度比大于 9:1，则称为不平衡针对不同的轴点选取策略，估计其发生不平衡的概率 (请填十进制小数):

Solution 4. 分析:

***(1)* 定义不平衡:** 设分割后短序列长度为 *'s'*，长序列长度为 *'l'*。*'s + l = n - 1'*。不平衡条件为 *'l/s > 9'*，即 *'l > 9s'*。代入 *'l = n - 1 - s'*，得到 *'n - 1 - s > 9s'*，即 *'n - 1 > 10s'*，或 *'s < (n-1)/10'*。由于 *'n'* 很大，可以近似为 *'s < n/10'*。这意味着，当分割后较短的子序列长度小于总长度的 *10%* 时，我们称之为不平衡。

***(2)* 不平衡的轴点范围:** 为了使短子序列长度小于 *'n/10'*，被选为轴点的元素必须是整个序列中最小的 *10%* 或最大的 *10%*。例如，如果轴点是第 *'k'* 小的元素，分割后的子序列大小为 *'k-1'* 和 *'n-k'*。如果 *'k < n/10'*，则短序列长度为 *'k-1'*，小于 *'n/10'*。如果 *'k > 9n/10'*，则短序列长度为 *'n-k'*，也小于 *'n/10'*。因此，轴点落在排序后序列的前 *10%* 或后 *10%* 区间内，会导致不平衡。这个“不平衡区间”的总大小为 *'0.1n + 0.1n = 0.2n'*。

从 *n* 个元素中等概率随机选取一个作为轴点: *0.2*

- **计算:** 随机选取的轴点落入“不平衡区间”的概率，就是该区间的相对大小。
- 概率 = *(不平衡区间的长度) / (总长度) = '0.2n / n = 0.2'*。

从 *n* 个元素中等概率选取三个元素，以它们的中间元素作为轴点: *0.056*

- **计算:** 设 *'S'* 表示元素落在前 *10%* 的“小”区间，*'M'* 表示落在中间 *80%* 的“中”区间，*'L'* 表示落在后 *10%* 的“大”区间。
- *'P(S) = 0.1', 'P(M) = 0.8', 'P(L) = 0.1'*。
- 三个元素的中间值（中位数）要落入“不平衡区间”(S 或 L)，必须满足以下条件之一:
  - ***(1)*** 至少有两个元素落在 S 区 *(*中位数在 S 区*)*。
  - ***(2)*** 至少有两个元素落在 L 区 *(*中位数在 L 区*)*。
- ***P(*中位数在 S 区*)*** = *P(*两个 *S,* 一个非 *S)* + *P(*三个 *S)*
  - *P(*两个 *S,* 一个非 *S)* = $\binom{3}{2} \times P(S)^2 \times (1 - P(S))^1 = 3 \times 0.1^2 \times 0.9 = 0.027$
  - *P(*三个 *S)* = $P(S)^3 = 0.1^3 = 0.001$
  - *P(*中位数在 S 区*)* = $0.027 + 0.001 = 0.028$
- ***P(*中位数在 L 区*)*** 与 S 区对称，同样为 *'0.028'*。
- **总不平衡概率** = *P(*中位数在 S 区*)* + *P(*中位数在 L 区*)* = $0.028 + 0.028 = 0.056$。

215

> 快速排序基于的思想是: A. 减而治之 B. 分而治之 C. 动态规划 D. 递归跟踪

Solution 5. 正确答案是 B。

详细分析:

- ***A.* 减而治之 *(Decrease and Conquer):*** 这种思想是将问题归约为一个规模更小的子问题。例如，二分查找每次将搜索范围减半，但只处理其中一半。快速排序则是将问题分解为**两个**子问题。

- ***B. 分而治之 (Divide and Conquer):*** 这是快速排序所采用的核心策略。该策略包含三个步骤:
  (1) 分解 (Divide): 选取一个轴点 (*pivot*), 将原序列划分为两个子序列, 使得一个子序列中的所有元素都小于或等于轴点, 另一个子序列中的所有元素都大于或等于轴点。
  (2) 解决 (Conquer): 通过递归调用快速排序算法, 分别对这两个子序列进行排序。
  (3) 合并 (Combine): 因为子序列是原地排序的, 所以当递归返回时, 整个序列就已经有序, 不需要额外的合并步骤。

  这完美地描述了快速排序的工作流程。归并排序也是分而治之的典型例子。
- ***C. 动态规划 (Dynamic Programming):*** 这种思想通常用于解决具有重叠子问题和最优子结构性质的问题, 通过存储子问题的解来避免重复计算。这与快速排序的机制不同。
- ***D. 递归跟踪 (Recursive Tracing):*** 这不是一个标准的算法设计思想或范式。虽然快速排序通常用递归实现, 但"递归"是实现手段, 而"分而治之"是其背后的设计思想。

216

> 对于规模为 n 的向量, 快速排序在平均情况下得时间复杂度为: A. O(n^2) B. O(nlogn) C. O(n) D. O(n+logn)

Solution 6. 正确答案是 B。

详细分析:

(1) **快速排序的核心思想:** 快速排序是一种采用"分治法" (*Divide and Conquer*) 策略的排序算法。它通过一个"划分" (*Partition*) 操作, 将一个数组分成两个子数组, 然后对这两个子数组进行递归排序。

(2) **时间复杂度分析:**

- **划分操作:** 每一次划分操作都需要遍历当前子数组, 其时间复杂度与子数组的规模成正比, 为 $O(k)$, 其中 *'k'* 是子数组的长度。
- **平均情况:** 在平均情况下, 我们可以期望每次选择的"轴点" (*pivot*) 能够将数组划分成大致相等 (或比例固定) 的两个部分。例如, 每次都划分为 *'n/2'* 和 *'n/2'*。这会导致递归的深度为 $O(\log n)$。由于每一层递归的所有划分操作加起来的总时间是 $O(n)$, 所以总的平均时间复杂度是 $O(n \log n)$。
- **最坏情况:** 当每次选择的轴点都是当前子数组的最大或最小元素时 (例如, 在一个已经有序的数组中总是选择第一个元素作为轴点), 划分会极度不均衡 (划分为 *0* 和 *'k-1'*)。这会导致递归深度达到 $O(n)$, 总时间复杂度退化为 $O(n^2)$。

结论: 尽管快速排序的最坏情况时间复杂度是 $O(n^2)$, 但通过良好的轴点选择策略 (如随机选择、三数取中等), 这种情况在实际应用中极少发生。其平均情况下的性能非常出色, 时间复杂度为 $O(n \log n)$。