

Algorithm - P73

快速排序

RANDOMIZED-QUICKSORT 的最坏情况分析

对于递归式 $T(n) = \max_{0 \leq q \leq n-1} [T(q) + T(n-q-1)] + \Theta(n)$

其中 q 为 RANDOMIZED-PARTITION 返回值, $\Theta(n)$ 为其时间复杂度
已知 $T(n) \in O(n^2)$, 于是猜测 $T(n) \in \Omega(n^2)$, 即 $T(n) \geq cn^2$, 其中 c 为正实数常数

则 $T(n) = \max_{0 \leq q \leq n-1} [T(q) + T(n-q-1)] + \Theta(n)$

$\geq \max_{0 \leq q \leq n-1} [cq^2 + c \cdot (n-q-1)^2] + \Theta(n)$

$= c \max_{0 \leq q \leq n-1} [q^2 + (n-q-1)^2] + \Theta(n)$

$= c(n-1)^2 + \Theta(n) = cn^2 - \Theta(n) = cn^2 + \Theta(n)$

又存在足够多小的常数 c , 使得 $c(n-1)$ 显著大于 $\Theta(n)$

于是有 $T(n) \in \Omega(n^2)$, 即有 $T(n) \in \Theta(n^2)$

RANDOMIZED-QUICKSORT 的最好情况分析

对于递归式 $T(n) = \min_{0 \leq q \leq n-1} [T(q) + T(n-q-1)] + \Theta(n)$

其中 q 为 RANDOMIZED-PARTITION 返回值, $\Theta(n)$ 为其时间复杂度

于是猜测 $T(n) \in \Omega(cn\lg n)$, 即 $T(n) \geq cn\lg n$, 其中 c 为正实数常数

则 $T(n) = \min_{0 \leq q \leq n-1} [T(q) + T(n-q-1)] + \Theta(n)$

$\geq \min_{0 \leq q \leq n-1} [q\lg^q + (n-q-1)\lg^{(n-q-1)}] + \Theta(n)$

$= c \min_{0 \leq q \leq n-1} [q\lg^q + (n-q-1)\lg^{(n-q-1)}] + \Theta(n)$

当 $q = \lfloor (n-1)/2 \rfloor$ 或 $q = \lceil (n-1)/2 \rceil$ 时, $q\lg^q + (n-q-1)\lg^{(n-q-1)}$ 取得最小值

于是 $T(n) \geq \min_{0 \leq q \leq n-1} [q\lg^q + (n-q-1)\lg^{(n-q-1)}] + \Theta(n)$

$= c\lfloor (n-1)/2 \rfloor \cdot \lg^{\lfloor (n-1)/2 \rfloor} + c\lceil (n-1)/2 \rceil \lg^{\lceil (n-1)/2 \rceil} + \Theta(n)$

又取整是可以忽略的, 即不影响复杂度计算

则 $T(n) \geq (n-1)/2 \cdot \lg^{\lfloor (n-1)/2 \rfloor} + (n-1)/2 \cdot \lg^{\lceil (n-1)/2 \rceil} + \Theta(n)$

$= (n-1) \lg^{\lfloor (n-1)/2 \rfloor} - c(n-1) + \Theta(n)$

于是有 $T(n) \in \Omega(cn\lg n)$

RANDOMIZED-QUICKSORT 的期望运行时间分析

如果在递归的每一层都把任意常数比例的元素划分到一个子数组

则算法的递归树深度为 $\Theta(\lg n)$, 且每一层的工作量都是 $O(n)$

即使在最不平衡的划分下, 会增加新的递归层次

但总运行时间仍然保持为 $O(n\lg n)$

并且假设传入 RANDOMIZED-QUICKSORT 的待排序元素始终是互异的

Algorithm - P74

快速排序

当在包含 n 个元素的数组上运行 QUICKSORT 时，
假设在 PARTITION 中 for 循环中的比较次数为 X ，
则 QUICKSORT 的时间复杂度为 $O(n + X)$ 。
证明过程有，首先注意到每次 PARTITION 调用都会选择一个主元元素，
这个元素在之后的 QUICKSORT 和 PARTITION 中均不再使用。
但是在 QUICKSORT 的整个运行过程中，至多可能调用 PARTITION n 次，
而 PARTITION 的时间取决于其中 for 循环的迭代次数。
每次迭代会比较主元与数组 A 中的另一个元素。
于是 PARTITION 的总时间取决于 for 循环的总比较次数 X 。
又 QUICKSORT 的运行时间取决于 PARTITION 调用的总时间。
于是可知 QUICKSORT 的时间复杂度为 $O(n + X)$ 。

再考虑 QUICKSORT 中比较次数 X 的上界，假定传入数组的元素是互异的。
将数组 A 中的元素重命名为 z_1, z_2, \dots, z_n ，其中 z_i 表示数组 A 中第 i 大的元素。
并定义数组 A 的子集 $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ ，其中 $1 \leq i < j \leq n$ 。
即 Z_{ij} 表示大小在 z_i 和 z_j 之间的所有元素。
注意到对于任意一对元素 z_i, z_j 在 QUICKSORT 的过程中至多比较一次，其中 $1 \leq i < j \leq n$ 。
由于参与比较的两个元素其一必定是当前 PARTITION 中选择的主元。
而这个元素在其后的 PARTITION 中不再用作比较。

取指示器变量 I_{ij} 表示在 QUICKSORT 过程中， z_i 与 z_j 被比较过，其中 $1 \leq i < j \leq n$ 。
则总比较次数 $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n I_{ij}$ 。

则 QUICKSORT 的期望运行时间取决于比较次数 X 的期望。

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n-1} \sum_{j=i+1}^n I_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[I_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr\{z_i \text{ 与 } z_j \text{ 在 QUICKSORT 中被比较过}\} \end{aligned}$$

注意在 Z_{ij} 中的元素被 PARTITION 选为主元之前， Z_{ij} 始终被划分在同一个子数组中。

当 Z_{ij} 中的元素 $z_i < z_k < z_j$ 被选为主元时， z_i 与 z_j 不会被比较。

且 z_i, z_j 在 PARTITION 之后划分入不同子数组，则之后也不再会被比较。

则仅当 z_i 或 z_j 是 Z_{ij} 中第一个被选为主元的元素时， z_i 与 z_j 会被比较。

又 Z_{ij} 中的元素都是等可能地被选为主元。

$$\begin{aligned} \text{于是 } E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \left(\sum_{k=1}^{n-i} \frac{2}{k} \right) \\ &< \sum_{i=1}^{n-1} 2 \sum_{k=1}^n \frac{1}{k} < \sum_{i=1}^{n-1} 2 \ln n \\ &\in \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n) \end{aligned}$$

于是可知 RANDOMIZED-QUICKSORT 的期望运行时间是 $O(n \lg n)$ 的。

Algorithm - P75

几乎有序

注意当输入的序列处于“几乎有序”的状态时，插入排序的速度可能很快
这个特点可以用于提高快速排序的速度

对于输入的包含n个元素的序列 $A[1..n]$ 进行快速排序时

选择适当大小的正整数 $k > 1$

当调用或递归调用 $\text{QUICKSORT}(A, p, r)$ 时

如果子数组 $A[p..r]$ 中包含的元素个数小于 k 时

不进行任何排序操作而直接返回

当所有的 $\text{QUICKSORT}(A, p, r)$ 完成后，对整个序列调用 $\text{INSERTION-SORT}(A)$

如 ~~XXXXXXXXXX~~

$\text{QUICKSORT-HELPER}(A, p, r)$

if $r - p + 1 \geq k$ 注意这里的 k 是预定义的正整数常数

$q := \text{RANDOMIZED-PARTITION}(A, p, r)$

$\text{QUICKSORT-HELPER}(A, p, q-1)$

$\text{QUICKSORT-HELPER}(A, q+1, r)$

$\text{INSERTION-QUICKSORT}(A)$

$\text{QUICKSORT-HELPER}(A, 1, A.length)$

$\text{INSERTION-SORT}(A)$

首先对于 QUICKSORT-HELPER 的部分

由于 QUICKSORT-HELPER 在子数组 $A[p..r]$ 少于 k 个元素时不进行任何操作

所以在 QUICKSORT-HELPER 的递归树中共有 $O(n/k)$ 个叶结点

即递归树的深度是 $O(\lg(n/k))$ 的

又每层的代价和均为 $O(n)$ ，于是 QUICKSORT-HELPER 的时间复杂度是 $O(n \lg(n/k))$

再考虑 INSERTION-SORT 的部分，考虑 A 已划分为 A_1, A_2, \dots, A_{nk} 子数组

注意尽管 A_i 内部是未排序的，但子数组间是已排序的

即对于任意 $a_i \in A_i, a_j \in A_j$ ，如果 $i < j$ ，则有 $a_i < a_j$

则在插入排序时 a_i 只在子数组 A_i 内移动，即可以看作对 A_i 进行独立的插入排序

又 A_i 的大小是 $O(k)$ 的，则 INSERTION-SORT 部分的时间复杂度为 $\frac{n}{k} O(k^2)$

于是 $\text{INSERTION-QUICKSORT}$ 的时间复杂度为 $O(nk + n \lg(n/k))$ 的

考虑 插入排序和快速排序拥有不同的常数参数 C_i, C_q

如果 $C_q n \lg n \geq C_i nk + C_q n \lg(n/k)$

$C_q \lg k \geq C_i k$ ，即 $\lg k \geq \frac{C_i}{C_q} k$

所以对常数 k 的选择需要根据实际运行环境的参数 C_i, C_q 选择

Algorithm - P76

快速排序

最早由 C.R.Hoare 设计的划分算法

```
HOARE-PARTITION(A, p, r) | HOARE-QUICKSORT(A, p, r)
    x := A[p] | if p < r
    i := p-1 | j := r+1
    while TRUE: | q := HOARE-PARTITION(A, p, r)
        do j := j-1 | repeat i := i+1
        while A[j] > x | until A[i] ≤ x
        do i := i+1 | repeat j := j-1
        while A[i] < x | until A[j] ≥ x
        if i < j | swap A[i] with A[j]
        else | return j
```

考虑 HOARE-PARTITION 的正确性，假设 $A[p..r]$ 包含至少两个元素

首先 i 和 j 不会访问子数组 $A[p..r]$ 之外的元素

如果 $i = j$ 在 return 语句执行前未执行过任何 swap 操作

则此时 $i = p$ 且 $j = p$ ，均未出现下标越界

如果至少进行过一次 swap 操作

则可知子数组的两端 $A[p] \leq x$ 且 $A[r] \geq x$ ，可以阻止 i, j 出现下标越界

考虑 j 返回值 j 的取值范围，已知返回值 j 必定满足 $j \geq p$

如果未执行过 swap 操作，则返回值 $j = p$

如果执行过 swap 操作，则至少进入两次 while 循环迭代

由于每次 while 循环迭代中 $j := j-1$ 语句至少执行过一次

于是可知 $j \leq r+1-2$ ，即有 $p \leq j < r$

考虑子数组 $A[p..j]$ 和 $A[j..r]$ 在 while 循环迭代之后的状态

由于 repeat $j := j-1$ until $A[j] \leq x$

则计语句前 $A[j+1..r]$ 中的所有元素满足 $\geq x$ （包含已迭代的部分）

由于 repeat $i := i+1$ until $A[i] \geq x$

则计语句前 $A[p..i-1]$ 中的所有元素满足 $\leq x$ （包含已迭代的部分）

如果选择 swap 操作，则完成后 $A[j..r] \geq x$, $A[p..i] \leq x$

如果选择 return，则此时 $i \geq j$

于是子数组满足 $A[p..j] \leq x \leq A[j+1..r]$

与已有的 PARTITION 方法不同，在 HOARE-PARTITION 返回后

主元 x 可能包含在 $A[p..j]$ 或 $A[j+1..r]$ 中

而由于 $p \leq j < r$ ，于是划分总是非平凡的

Algorithm - P77

快速排序

注意对于 RANDOMIZED - QUICKSORT 的分析依旧假设所有输入元素是互异的
那么考虑存在相同元素的情形，对于极端情况下所有元素相等
如果使用原始的 PARTITION，则时间复杂度为 $\Theta(n^2)$
而如果使用 BALANCED - PARTITION，则时间复杂度为 $\Theta(n \lg n)$

考虑对 PARTITION 进行修改，返回一组下标 q, t $A[P..r]$

TRIPARTITE-PARTITION(A, p, r)

$\text{pivot} := A[r]$

$i := p-1, j := p-1$
for $k := p$ to $r-1$

if $A[k] < \text{pivot}$

$i := i+1, j := j+1$

swap $A[k]$ with $A[j]$

swap $A[j]$ with $A[i]$

if $A[k] == \text{pivot}$

$j := j+1$

swap $A[j]$ with $A[k]$

swap $A[j+1]$ with $A[r]$

return $i+1, j+1$

QUICKSORT-HELPER(A, p, r)

if $r-p+1 \geq 3$ 这里相当于保证 TRIPARTITE-PARTITION 至少输入 3 个元素

$q, t := \text{TRIPARTITE-PARTITION}(A, p, r)$

QUICKSORT-HELPER($A, p, q-1$)

QUICKSORT-HELPER($A, t+1, r$)

INSERTION-QUICKSORT(A)

QUICKSORT-HELPER($A, 1, A.length$)

INSERTION-SORT(A)

对于 TRIPARTITE - PARTITION(A, p, r) 的返回值 q, t 有

子数组 $A[p..q-1]$ 都小于选择的主元 pivot

子数组 $A[q..t]$ 都等于选择的主元 pivot

子数组 $A[t+1..r]$ 都大于选择的主元 pivot

其中子数组 $A[p..q-1]$ 和 $A[t+1..r]$ 可能为空

于是 TRIPARTITE - PARTITION 的时间复杂度仍为 $\Theta(r-p)$

且避免了所有元素互异的假设

Algorithm - P78

快速排序

对于随机化版本的快速排序算法，考虑一种关注于单次递归调用的期望运行时间的分析
 对于输入大小为 n 的子数组 $A[p..r]$ ，任何特定元素被选为主元的概率均为 $\frac{1}{n}$
 令指针变量 X_i 表示子数组 $A[p..r]$ 中第 i 小的元素被选为主元，其中 $i = 1, 2, \dots, n$
 则可知 $E[X_i] = \Pr\{\text{第 } i \text{ 小的元素被选为主元}\} = \frac{1}{n}$

令 $T(n)$ 表示快速排序在输入大小为 n 的子数组上的运行时间的随机变量
 已知 X_1, X_2, \dots, X_n 为互斥的事件，且有且仅有一个发生

当 $X_1 = 1$ 时，划分子数组为大小为 $q-1$ 和 $n-q$ 的两个问题是

而划分的时间复杂度为 $\Theta(n)$

$$\begin{aligned} \text{于是有 } T(n) &= \sum_{q=1}^n X_q [T(q-1) + T(n-q) + \Theta(n)] \\ \text{则 } E[T(n)] &= E[\sum_{q=1}^n X_q [T(q-1) + T(n-q) + \Theta(n)]] \\ &= \sum_{q=1}^n \Pr\{X_q\} \cdot [E[T(q-1)] + E[T(n-q)] + \Theta(n)] \\ &= \frac{1}{n} [\sum_{q=1}^n E[T(q-1)] + \sum_{q=1}^n E[T(n-q)] + n\Theta(n)] \\ &= \frac{1}{n} [\sum_{q=2}^{n-1} E[T(q)] + \sum_{q=2}^{n-1} E[T(n-q)] + n\Theta(n)] \\ &= \frac{2}{n} \sum_{q=2}^{n-1} E[T(q)] + \Theta(n) \end{aligned}$$

考虑对于正整数 $n > 2$ ， $\sum_{k=2}^{n-1} k \lg k$ 的取值情况

$$\text{可知 } \sum_{k=2}^{n-1} k \lg k \leq \int_2^n k \lg k \, dk$$

$$\begin{aligned} \text{又 } \int k \lg k \, dk &= \int k \frac{1}{2} \lg k \, d(k^2) \\ &= \frac{1}{2} k^2 \lg k - \int k^2 d(\frac{1}{2} \lg k) = \frac{1}{2} k^2 \lg k - \frac{1}{2} \int k \, dk \\ &= \frac{1}{2} k^2 \lg k - \frac{1}{4} k^2 \end{aligned}$$

$$\begin{aligned} \text{于是 } \sum_{k=2}^{n-1} k \lg k &\leq \int_2^n k \lg k \, dk = \left[\frac{1}{2} k^2 \lg k - \frac{1}{4} k^2 \right]_2^n \\ &= \frac{1}{2} n^2 \lg n - \frac{1}{4} n^2 - 2 + \frac{1}{4} \\ &\leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \end{aligned}$$

假设对于正常数 a 和足够大的 n ，有 $T(n) \leq an \lg n + \Theta(n)$

$$\text{则有 } E[T(n)] = \frac{2}{n} \sum_{q=2}^{n-1} E[T(q)] + \Theta(n)$$

$$\leq \frac{2}{n} \sum_{q=2}^{n-1} [a q \lg q + \Theta(q)] + \Theta(n)$$

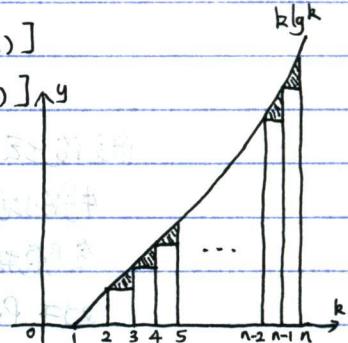
$$= \frac{2a}{n} \sum_{q=2}^{n-1} q \lg q + \frac{2}{n} \Theta(n^2) + \Theta(n)$$

$$\leq \frac{2a}{n} \cdot (\frac{1}{2} n^2 \lg n - \frac{1}{8} n^2) + \Theta(n)$$

$$= a n \lg n - \frac{a}{4} n + \Theta(n)$$

$$= a n \lg n + \Theta(n)$$

于是递归式 $E[T(n)] = \frac{2}{n} \sum_{q=2}^{n-1} E[T(q)] + \Theta(n)$ 有解 $E[T(n)] = \Theta(n \lg n)$



Algorithm - P79

快速排序

在通常的QUICKSORT算法中包含了两个对其自身的递归调用
调用PARTITION后, QUICKSORT分别递归调用左边与右边的子数组
考虑尾递归(tail recursion)版本的快速排序

TAIL-RECURSIVE-QUICKSORT(A, p, r)

while $p < r$

$q := \text{PARTITION}(A, p, r)$

TAIL-RECURSIVE-QUICKSORT(A, p, q-1)

$p := q+1$

注意到 TAIL-RECURSIVE-QUICKSORT 的改动在于循环结构

其中对左子数组的递归调用等价于 QUICKSORT 对左子数组的调用

而 while 循环的下一次迭代等价于 QUICKSORT 对右子数组的调用

于是 TAIL-RECURSIVE-QUICKSORT(A, 1, A.length) 能正确地将数组 A 排序

编译器通常使用栈(stack)来存储递归执行过程的相关信息, 包括递归调用的参数

最新调用的信息存储在栈的顶部, 而最早调用的信息存储在栈的底部

当过程被调用时, 相关信息压入(push)栈中

当过程返回时, 相关信息从栈中弹出(pop)

如果数组参数是通过指针传递 / 引用传递的

则每次过程调用只需要占用 O(1) 的栈空间

栈深度(stack depth)表示在一次计算中使用的栈空间的最大值

注意到当输入数组为升序排列时, 每次递归调用的 while 循环仅有一次迭代

于是要进行 $\Theta(n)$ 次递归调用, 即栈深度为 $\Theta(n)$ 的

于是对 TAIL-RECURSIVE-QUICKSORT 进行调整

TAIL-RECURSIVE-QUICKSORT(A, p, r)

while $p < r$

$q := \text{RANDOMIZED-PARTITION}(A, p, r)$

if $q \leq \lfloor (p+r)/2 \rfloor$

TAIL-RECURSIVE-QUICKSORT(A, p, q-1)

if $q > \lfloor (p+r)/2 \rfloor$

TAIL-RECURSIVE-QUICKSORT(A, q+1, r)

$r := q-1$

调整后最坏情形下栈深度是 $\Theta(\lg n)$ 的, 且算法期望时间复杂度是 $O(n \lg n)$ 的

Algorithm - P80

Oct 19 - 2023

三数取中划分

考虑在快速排序的 RANDOMIZED-PARTITION 进行划分时更细致地选择主元元素

常用的方法是从子数组 $A[P..r]$ 中随机地选出三个元素，并取其中位数作为主元

首先考虑其产生的最坏划分的根概率，

即对于实数 $0 < \alpha \leq \frac{1}{2}$ ，出现最坏划分比例为 $\alpha : (1-\alpha)$ 的近似根概率 $P(\alpha)$

假设子数组 $A[P..r]$ 中包含 n 个元素，为讨论方便假定 n 也是正整数

则考虑最坏划分坏于比例 $\alpha : (1-\alpha)$ 的根概率 $1 - P(\alpha)$

令 a_1, a_2, a_3 为川顺序地从 $A[P..r]$ 中选取的元素

则出现坏于比例 $\alpha : (1-\alpha)$ 的划分当且仅当

a_1, a_2, a_3 中至少有两个元素或者在最小的 αn 个元素中，或者在最大的 $(1-\alpha)n$ 个元素中
又 $0 < \alpha \leq \frac{1}{2}$ ，所以两个事件是互斥的

于是有 $1 - P(\alpha) = \Pr\{\text{在最小的 } \alpha n \text{ 个元素中}\} + \Pr\{\text{在最大的 } (1-\alpha)n \text{ 个元素中}\}$

假定 αn , $\alpha(1-\alpha)n$ 足够大

$$= \binom{3}{2} \alpha^2 (1-\alpha) + \binom{3}{3} \alpha^3 + \binom{3}{2} \alpha^2 (1-\alpha) + \binom{3}{3} \alpha^3$$

使得选择根概率始终近似地为 $\alpha : (1-\alpha)$

$$= 6\alpha^2 - 4\alpha^3$$

于是有对于 $0 < \alpha \leq \frac{1}{2}$ ，出现最坏划分比例 $\alpha : (1-\alpha)$ 的近似根概率为 $1 - 6\alpha^2 + 4\alpha^3$

假设数组 $A[1..n]$ 中的元素是互异的且有 $n \geq 3$ ， $A'[1..n]$ 表示已排序的数组 $A[1..n]$

考虑三数取中划分时选择的主元为 x 。令 $P_i = \Pr\{x = A'[i]\}$ ，其中 $i = 1, 2, \dots, n$

首先注意到由于选取三个不同的元素，于是 $P_{i_1} = P_{i_2} = 0$

再考虑 $i = 2, 3, \dots, n-1$ 的情况，在所有 $\binom{n}{3}$ 种组合中

如果 $A'[i]$ 为主元，则其本身必定被选中，且必定为三个数的中位数

而另外从小于 $A'[i]$ 的 $i-1$ 个元素中选择 1 个，共有 $i-1$ 种方式

再从大于 $A'[i]$ 的 $n-i$ 个元素中选择 1 个，共有 $n-i$ 种方式

于是有 $P_i = (i-1)(n-i)/\binom{n}{3} = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$ ，其中 $i = 1, 2, \dots, n$

考虑与平凡实现相比，选中 $A'[L(n+1)/2]$ 作为主元的根概率的提升

可知平凡实现选中 $A'[L(n+1)/2]$ 的根概率为 $\frac{1}{n}$

三数取中选中 $A'[L(n+1)/2]$ 的根概率为 $6(L(n+1)/2 - 1)(n - L(n+1)/2) / n(n-1)(n-2)$

考虑当 n 趋于无穷大时，两个根概率的关系

则有 $\lim_{n \rightarrow \infty} P(L(n+1)/2) / \frac{1}{n} = 6(L(n+1)/2 - 1)(n - L(n+1)/2) / (n-1)(n-2)$

$$= \lim_{n \rightarrow \infty} 6(L(n+1)/2 - 1)(n - L(n+1)/2) / (n-1)(n-2)$$

$$= 6 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{3}{2}$$

可见三数取中划分提升了选中 $A'[L(n+1)/2]$ 作为主元的根概率

$$6 \cdot \frac{1}{2} \cdot \frac{1}{2} = 0.75 \approx 75\%$$

Algorithm - P81

三数取中划分

Fastest partition

三数取中划分 考虑与平凡实现相比，选中主元 $x = A[i]$ ，其中 $\frac{n}{4} \leq i \leq \frac{3}{4}n$ 的概率提升至 $\frac{3}{4}$ 的概率

注意在这种情形下，平凡实现有 $\frac{1}{2}$ 的概率获得最坏比例为 $\frac{1}{4} : \frac{3}{4}$ 的划分

对于 $i = 1, 2, \dots, n$ ，有 $P_i = \frac{6(i-1)(n-i)}{n(n-1)(n-2)}$

为方便起见，假定 $\frac{n}{4}$ 和 $\frac{3}{4}n$ 均为整数

又选择 $A[i]$ 的事件是互斥的，其中 $i = 1, 2, \dots, n$

于是有 $\Pr\{\frac{n}{4} \leq i \leq \frac{3}{4}n\} = \sum_{i=\frac{n}{4}}^{\frac{3}{4}n} P_i$

注意到当 n 足够大时， $\sum_{i=n/4}^{3n/4} P_i$ 可近似地看作积分形式

$$\begin{aligned}\text{即 } \Pr\{\frac{n}{4} \leq i \leq \frac{3}{4}n\} &= \sum_{i=n/4}^{3n/4} P_i \approx \int_{n/4}^{3n/4} P_i di \\ &= \int_{n/4}^{3n/4} \frac{6(i-1)(n-i)}{n(n-1)(n-2)} di \\ &= \int_{n/4}^{3n/4} \frac{6(-ni^2 + i^2 - n^2)}{n(n-1)(n-2)} di \\ &= \left[-\frac{1}{3}i^3 + \frac{n+1}{2}i^2 - ni \right]_{n/4}^{3n/4} \cdot \frac{6}{n(n-1)(n-2)} \\ &= (\frac{11}{16}n^3 - \frac{3}{2}n^2) / n(n-1)(n-2)\end{aligned}$$

则当 $n \rightarrow +\infty$ 时， $\lim_{n \rightarrow +\infty} \Pr\{\frac{1}{4}n \leq i \leq \frac{3}{4}n\} = \frac{11}{16} > \frac{1}{2}$

可见三数取中划分提升了获得最坏比例为 $\frac{1}{4} : \frac{3}{4}$ 的划分的概率

注意到由于概率的提升并不改变概率的复杂度

即平凡实现中每个元素选为主元的概率均为 $\frac{1}{n}$

而三数取中划分选中每个元素为主元的概率为 $O(\frac{1}{n})$

于是利用三数取中法实现的快速排序依旧是 $\Theta(n \lg n)$ 的。

而三数取中划分只能改善 $\Theta(n \lg n)$ 中的常数因子。

模糊排序 (fuzzy sort)，考虑对区间序列 $\langle [a_1, b_1], [a_2, b_2], \dots, [a_n, b_n] \rangle$ 进行排序，其中 $a_i \leq b_i$

对区间序列进行模糊排序得到 $\langle i_1, i_2, \dots, i_n \rangle$

使得存在 $c_j \in [i_j, a_j, i_j, b_j]$ ，有 $c_1 \leq c_2 \leq \dots \leq c_n$ ，其中 $j = 1, 2, \dots, n$

FUZZYSORT(A, p, r)

if $p+1 = r$ && $A[p].b > A[r].b$

swap $A[p]$ with $A[r]$

if $r-p+1 \geq 3$

$q, t := FUZZYSORT-PARTITION(A, p, r)$

FUZZYSORT(A, p, q-1)

FUZZYSORT(A, t+1, r)

Algorithm - P82

Algorithm - P82

模糊排序

对区间序列进行排序得到 $\langle [a_1, b_1], [a_2, b_2], \dots, [a_n, b_n] \rangle$

使得存在 $c_j \in [a_j, b_j]$, 有 $c_1 \leq c_2 \leq \dots \leq c_n$, 其中 $j = 1, 2, \dots, n$

可以设计用于 n 个区间模糊排序的随机算法

同时利用区间的重叠性质来改善时间性能

即如果对于 $[a_{i_1}, b_{i_1}], [a_{i_2}, b_{i_2}], \dots, [a_{i_k}, b_{i_k}]$, 其交集非空

存在 $x \in [\max\{a_{i_1}, \dots, a_{i_k}\}, \min\{b_{i_1}, \dots, b_{i_k}\}]$

则可知不论这 k 个区间在结果中如何排序, 都可以选择 x 使得排序成立

于是可以参考对相同元素的处理设计模糊排序的 PARTITION 方法

FUZZYSORT - PARTITION(A, p, r)

1. pivot.a := $A[r].a$, pivot.b := $A[r].b$

2. $i := p-1, j := p-1$

for $k := p$ to $r-1$

if $A[k].b < \text{pivot}.a$

$i := i+1, j := j+1$

 swap $A[k]$ with $A[j]$

$i := i+1, j := j+1$

 if $A[k].b \geq \text{pivot}.a$ and $A[k].a \leq \text{pivot}.b$

$j := j+1$

 swap $A[k]$ with $A[j]$

$i := i+1, j := j+1$

 pivot.a := $\max(\text{pivot}.a, A[k].a)$

 pivot.b := $\min(\text{pivot}.b, A[k].b)$

 swap $A[j+1]$ with $A[r]$

return $i+1, j+1$

对于 FUZZYSORT - PARTITION(A, p, r) 的返回值 q, t

可知 $A[q..t]$ 中的区间元素交集非空, 且交集为区间 $[\text{pivot}.a, \text{pivot}.b]$

则从其中选择任意元素均可使 $A[q..t]$ 的排序成立

子数组 $A[p..q-1]$ 中的区间均有 $A[i].b < \text{pivot}.a$

子数组 $A[t+1..r]$ 中的区间均有 $A[i].a > \text{pivot}.b$

一般情况下, FUZZYSORT 具有与 QUICKSORT 相同的时间复杂度

即有 $\Theta(n \lg n)$ 的期望时间复杂度

但是当 $A[p..r]$ 中所有区间均重叠时, 即所有区间的交集非空

此时仅进行一次 FUZZYSORT - PARTITION 调用, 退时间复杂度为 $\Theta(n)$

Algorithm - P83

比较排序 (comparison sort), 指排序的最终结果中各元素的次序依赖于元素间的比较
即只使用元素间的比较来获得输入序列 $\langle a_1, a_2, \dots, a_n \rangle$ 中的元素间次序的信息

对于给定元素 a_i, a_j , 可以通过比较操作来确定 a_i, a_j 之间的相对次序

$a_i < a_j$ (LT), $a_i \leq a_j$ (LE), $a_i = a_j$ (EQ), $a_i \geq a_j$ (GE), $a_i > a_j$ (GT)

可以不失一般性地假设所有输入元素是互异的

则 $a_i = a_j$ 没有意义, 且 $a_i \leq a_j$ 和 $a_i \geq a_j$ 等价于 $a_i < a_j$ 和 $a_i > a_j$

由于 $a_i \leq a_j$ 与 $a_i \geq a_j$ 得到的相对次序信息相同, 于是假设仅采用 $a_i \leq a_j$ 比较

决策树模型 (decision-tree model), 比较排序可以由象为满二叉树 (full binary tree)

用于表示在给定输入规模情况下, 特定排序算法对所有元素的比较操作

其中控制、数据移动等其他操作均被忽略

对于输入的 n 个元素的序列 $\langle a_1, a_2, \dots, a_n \rangle$

决策树的内部结点, 标记为 $i:j$, 其中 $1 \leq i < j \leq n$, 表示元素 a_i 和 a_j 间的比较操作

左子树表示比较结果为 $a_i \leq a_j$ 之后的比较操作

右子树表示比较结果为 $a_i > a_j$ 之后的比较操作

● 决策树的叶结点, 标记为 $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$

表示得到已排序的序列 $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$

由于在有根树中, 对于给定叶结点, 有且仅有唯一从根结点到该叶结点的路径

于是这条路径上的内部结点, 表示了得到该叶结点结果所需的比较操作结果

称这样的叶结点是可达的 (reachable)

由于任何正确的排序算法都能生成输入的每一个排列

所以 n 个元素的 $n!$ 种排列都应该出现在正确的比较排序决策树的叶结点上

决策树中从根结点到任意可达叶结点之间的最长简单路径长度

表示对应比较排序算法中最坏情形下的比较次数

于是比较排序算法中的最坏情形比较次数等于对应决策树的高度

由于决策树中的每种排序均以可达叶结点的形式出现

则该决策树高度的下界即比较排序算法运行时间的下界

如对于输入 3 个元素序列进行插入排序

