

Algorithm - Pg5

比较排序概率下界，对于确定的比较排序算法A，其决策树为 T_A ，且 A 的每一种输入情况是等可能的。

决策树 T_A 中有 $n!$ 个叶结点，标记为随机输入到达该结点的概率 $\frac{1}{n!}$

令 $D(T)$ 为决策树 T 中所有叶结点的深度之和

则有 $D(T) = D(LT) + D(RT) + k$ ，其中 k 为决策树 T 的叶结点数

令 $d(k)$ 表示所有具有 $k \geq 1$ 个叶结点的决策树 T 中 $D(T)$ 的最小值

于是有 $d(k) = \min_{1 \leq i \leq k-1} \{d(i) + d(k-i) + k\}$

对于给定的正整数 $k \geq 1$ 和 $1 \leq i \leq k-1$ ，考虑函数 $f_k(i) = i \lg i + (k-i) \lg(k-i)$

则函数 $f_k(i)$ 有导数 $f'_k(i) = \lg i + i \cdot \frac{1}{i \ln 2} - \lg(k-i) - (k-i) \frac{1}{(k-i) \ln 2}$

$$= \lg i - \lg(k-i)$$

$$= \lg \frac{i}{k-i}$$

可知当 $f'_k(i) = 0$ ，即 $i = \frac{k}{2}$ 时， $f_k(i)$ 取得最大值

于是有 $d(k) \geq 2d(k/2) + k$ ，且当 $k=1$ 时， $d(k) \in O(1)$

于是有 $d(k) \in \Omega(k \lg k)$

$(A_j, f_{sk}) \leq f_{sk} = (i_1 \ln 2 + \dots + i_n \ln 2, A_1, d) \frac{1}{A_1} = 10^6$

对于输入 n 个互异元素的决策树 T_A 有 $n!$ 个叶结点

且 $d(k)$ 为具有 k 个叶结点的决策树 T 中 $D(T)$ 的最小值

于是有 $D(T_A) \geq d(n!)$ ，即 $D(T_A) \in \Omega(n! \lg(n!))$

令随机变量 X_i 表示给定 n 个互异元素的决策树中第 i 个叶结点的深度

则可知 X_i 表示到达第 i 个叶结点需要的比较次数

于是排序 n 个互异元素需要的期望比较次数为

$E[X] = \sum_{i=1}^{n!} E[X_i] = \frac{1}{n!} D(T_A)$

即有 $E[X] \in \Omega(\lg(n!)) = \Omega(n \lg n)$

对于随机化的比较排序算法B，考虑其决策树具有两种结点：

普通的比较结点与确定的比较排序算法A的决策树中的结点相同

随机化结点，包含 r 个子结点，用于刻画算法中通过 $RANDOM(1, r)$ 进行随机选择的情形

在算法B的执行过程中，每一个子结点均等概率地被选择

考虑对于任何随机化的比较排序算法B

则考虑确定的比较排序算法A，如果对于所有的A，其期望比较次数均多于B的比较次数

则由于期望运行时间是所有可能结果的平均

于是期望比较次数会产生一个更高的期望，从而产生一个矛盾

于是总会存在一个确定的比较排序算法A，期望比较次数不多于B的比较次数

Algorithm - Pg6

done partitioning

def \leftarrow COUNTING-SORT

线性时间原址排序，对于包含 n 个待排序数据记录的数组，每条记录的 key 值为 0/1

考虑对一组记录进行排序的算法可能具备的特性

① 排序算法的时间复杂度是线性的，即 $O(n)$

② 排序算法是稳定的

③ 排序算法是原地的 (in place)，即除输入数组外算法仅需固定的额外存储，则可知 COUNTING-SORT 满足性质①和②

注意用于 RADIX-SORT 中的基础排序方法需要满足①和②

从而使 RADIX-SORT 在排序有 b 位关键字的 n 条记录时其时间复杂度为 $O(bn)$

可知 INSERTION-SORT 满足性质②和③，但其时间复杂度是 $O(n^2)$ 的

再考虑满足性质①和③的排序算法，即 ■■■ 排序算法可以是不稳定的

由于每条记录的 key 值为 0/1，则排序后以 key=0 和 key=1 划分为两部分

于是可以指定 ■■■ pivot := 0 并应用 QUICKSORT 中的 PARTITION 算法

则可以在 $O(n)$ 的时间复杂度内进行原址排序

对于包含 n 个待排序数据记录的数组，每条记录的 key 值在 1 到 k 的区间内

则考虑修改 COUNTING-SORT 使得在时间复杂度 $O(n+k)$ 内完成对 n 条记录的原址排序

且除 ■■■ 输入数组外，使用 $O(k)$ 的额外存储空间

ALTERNATIVE-COUNTING-SORT (A, k)

let $C[1..k]$ be new array

for $i := 1$ to $A.length$

$C[A[i]] := C[A[i]] + 1$

 for $j := 2$ to k

$C[j] := C[j] + C[j-1]$

$B := 1 : C[-1]$

 for $i := 1$ to $A.length$

 while $C[A[i]] \neq B[A[i]]$

 key := $A[i]$

 swap $A[C[key]]$ with $A[i]$

 while $A[C[key]] \neq key$

$C[key] := C[key] - 1$

return A

Algorithm - Pg7

线性时间原址排序，对于包含 n 个待排序数据记录的数组：每条记录的 key 值为从 1 到 k 的正整数。

考虑修改后的计数排序 ALTERNATIVE-COUNTING-SORT

首先根据输入的 n 个元素数组 A 和 key 值上限 k

生成数组 $C[1..k]$ 表示关键字为 j 的元素可能出现的最后位置，其中 $j=1, 2, \dots, k$
数组 B 的第 1 个元素为 1，其后 $k-1$ 个元素为 $C[1..k-1]$

则数组 B 保存数组 C 在结束时的状态 (end point)

对于三层循环 for - while - while

对于 for 循环满足循环不变量：

在第 i 次迭代之前 $A[1..i-1]$ 是排序好的子数组

且所有 key 值小于等于 $A[i-1]$ 的元素均已完成排序

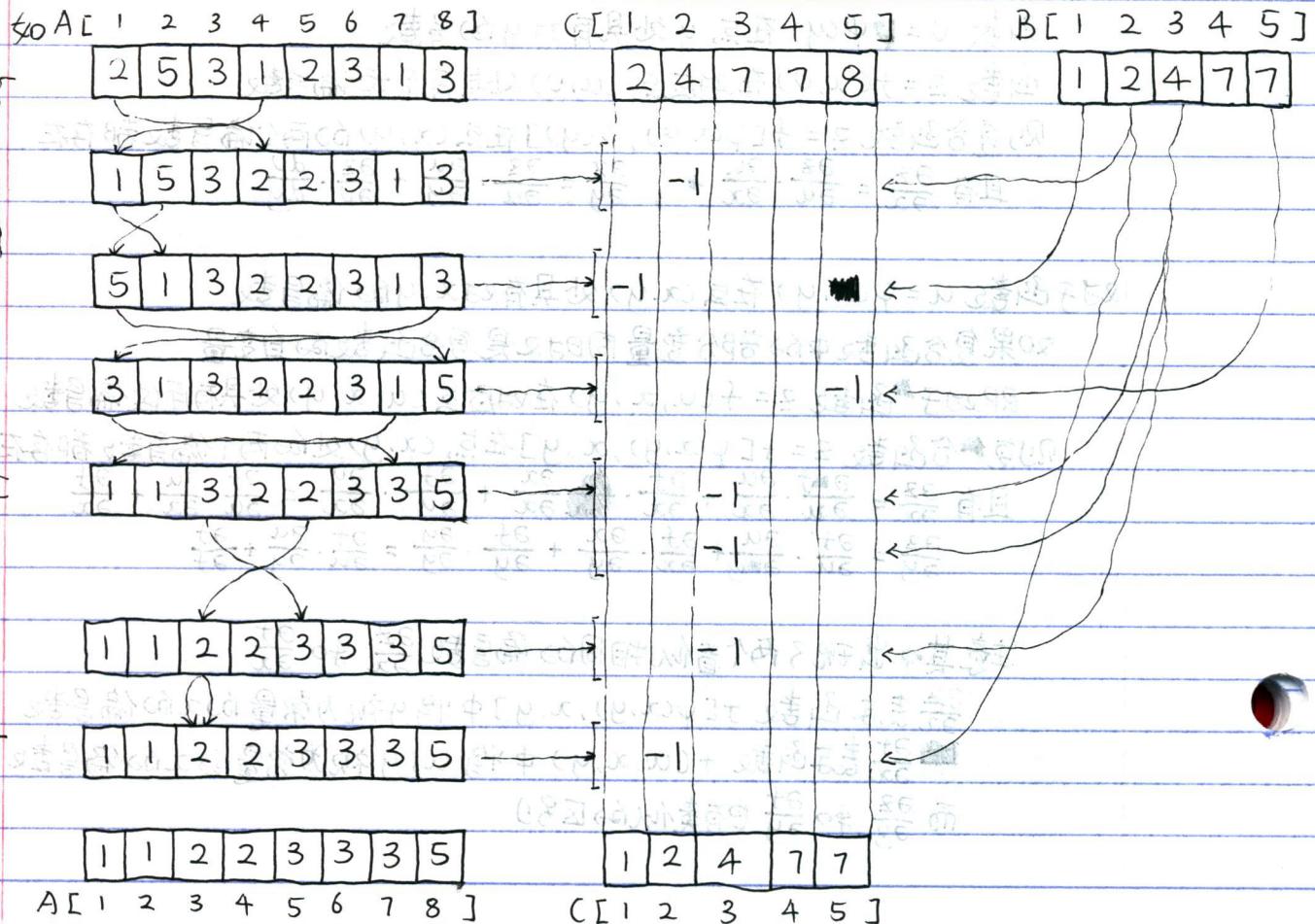
对于第一层 while 循环，包含一次对 key 的赋值和一次交换操作

其终止时 $(A[i]) = B[A[i]]$ ，则表示所有 key 值与 $A[i]$ 相同的元素均已
且其交换操作将 $A[i]$ 交换到最后一个 key 值为 $A[i]$ 的位置

对于第二层 while 循环，不断地减少 [key] 的值。

直到 [key] 指向下一个可以放置关键字为 key 的元素的位置

或者此时有 $(key) = B[key]$



Algorithm - P98

线性时间原地排序，对于包含 n 个待排序数据记录的数组，每条记录的 key 值为从 1 到 k 的正整数
考虑修改后的计数排序 ALTERNATIVE-COUNTING-SORT 的时间复杂度

注意在 ALTERNATIVE-COUNTING-SORT 中

前两个 for 循环的时间复杂度分别为 $\Theta(n)$ 和 $\Theta(k)$

而第三个 for 循环中嵌套了两层 while 循环

注意在 for 循环及嵌套的 while 循环中所有操作均为 $O(1)$ 的
则 for 循环的时间复杂度取决于循环的累计迭代次数

首先在 for 循环与第一层 while 循环之间不存在其他操作

提 for 循环的复杂度依赖于第一层 while 循环的迭代次数

而在第一层 while 循环中包含一次对变量 key 的赋值和一次交换操作

注意到在 swap $A[C[key]]$ with $A[i]$ 操作之后

$A[C[key]]$ 与变量 key 的值必定相同

于是第二层 while 循环必定有一次迭代

于是可知第一层 while 循环的迭代次数等于第二层 while 循环的迭代次数

而对于第二层 while 循环，每次迭代中对数组 C 中的元素减 1

则迭代次数为 $(C[1] - B[1]) + (C[2] - B[2]) + \dots + (C[k] - B[k])$

$= (C[1] - 1) + (C[2] - C[1]) + \dots + (C[k] - C[k-1])$

$= C[k] - 1 = n - 1$

于是可知 ALTERNATIVE-COUNTING-SORT 在时间复杂度 $O(n)$ 中

对包含 n 个待排序数据记录的数组排序，其中每条记录的 key 值为从 1 到 k 的正整数

变长数据项排序：对于给定整数数组，其中不同元素所包含的整数位数可能不同，但所有整数包含的总位数为 n

首先在 $O(n)$ 的时间内计算每个元素包含的整数位数 n_i ，于是有 $\sum n_i = n$

可知存在正整数 $1 \leq k \leq n$ ，使得对任意元素有 $1 \leq n_i \leq k$

则可以将元素根据位数 n_i 放入 k 个桶之一

再对 k 个桶分别进行基数排序，最后顺序地排列 k 个桶中的元素

注意到对于每个桶中的元素进行基数排序的时间复杂度为 $\Theta(d(m+b))$

对于给定的整数数组，每一位数的不同取值 b 是常数

其中 m 表示位数为 $1 \leq d \leq k$ 的整数数量

又总位数为 n

于是有 $\sum_{d=1}^k \Theta(d(m+b)) = \Theta(\sum_{d=1}^k dm_d) = \Theta(n)$

即在 $\Theta(n)$ 的时间复杂度内对总位数为 n 的给定整数数组进行排序

Algorithm - Pg9

Longest common subsequence

Longest common subsequence

变长数据项排序，对于给定整数数组，其中元素所包含的整数位数可能不同，但所有整数包含的总位数为n。

VARIABLE-LENGTH-INTEGER(A, n) // 数组A为b进制整数数组

let B[1..n] be new array // 框，(A, +, 10) 无意义

for j := 1 to n // 建立一个空桶，内向圆柱体，且从右到左
同理建立一个空桶，B[j] := empty array // 同理建立一个空桶，且从左到右

for i := 1 to A.length // 同时下表，不足以支撑对基

let j be length of integer A[i] // 且，且有错

insert A[i] into B[j] // 且，且有错

for j := 1 to n // 且，且有错

RADIX-SORT(B[1], j) // 且，且有错

concatenate list B[1], ..., B[n] together in order // 且，且有错

对于给定字符串数组，其中字符串包含的字符个数可能不同，但数组中所有字符串包含的总字符数为n。

首先创造一个映射，key to value 均为数组中的字符串。

将映射作为算法的输入参数，并假定可能包含的字符串仅有常数个。

根据 value 中的字符串首位字符 装入对应的桶，并删除 value 的首个字符。

顺序地处理每个非空的桶，

将 value 为空字符串的 key 值连接到返回列表中。

对于 value 为非空字符串部分递归地调用排序算法。

返回完成连接的返回列表。

VARIABLE-LENGTH-STRING(M)

let B be new array of empty bucket // 中间圆柱体

for each (key, value) in mapping M // 装 (k, v) = ...

insert (key, value') into bucket of character as first char of value

where value' := value without first character

let ret be new array of string // [k, v]

for each non-empty bucket B[j] in B // 若 -1

let M' be new mapping // if -1

for each (key, value') in B[j] // 装 (k, v)

if value' is empty string then append key to ret

else add (key, value') into M'

append VARIABLE-LENGTH-STRING(M') to ret

return ret

Algorithm - P100

变长数组归顺排序，对于给定字符串数组，其中字符串包含的字符个数可能不同，但数组中所有字符串包含总字符个数为n
考虑 VARIABLE-LENGTH-STRING 的时间复杂度

注意到 VARIABLE-LENGTH-STRING 中包含两个 for 循环

且第二个 for 循环中包含对 VARIABLE-LENGTH-STRING 的递归调用

但是实际上第二个循环中的操作时间复杂度依赖于第一个 for 循环

注意第一个 for 循环中对所有当前非空的字符串操作

在 O(n) 的时间中去掉 ■ 当前 value 中的首个字符

[i , $last - current + 1$] 并在 O(n) 的时间中分配到首个字符对应的桶中

于是算法复杂度依赖于进行删除首个字符的操作次数

又初始输入的字符串数组 ■ 所包含的总字符个数为 n

则 VARIABLE-LENGTH-STRING 在 O(n²) 时间复杂度内对数组进行排序

水壶问题

对于 n 个红色水壶和 n 个蓝色水壶，由于形状和尺寸都各不相同，于是无法直接确定水壶的水量

假设所有红色水壶的水量均不相同，所有蓝色水壶的水量均不相同

但是对于每一个红色水壶，有且只有一个蓝色水壶， ■ 水壶中的水量相同

存在一个对于一个红色水壶和一个蓝色水壶的判断操作

将红色水壶中倒满水，再将水倒入蓝色水壶

通过这个操作可以判断红色水壶和蓝色水壶的水量是少于 / 等于 / 多于

假设这个操作 ■ 需要耗费一个单位的时间

考虑以比较操作为基准的算法，匹配所有水量相同的红色水壶和蓝色水壶

使用尽可能少的比较操作， $O(n^2)$ 中等量级

注意在整个过程中无法直接比较两个红色水壶 或者 两个蓝色水壶

$O(n^2)$ $\approx 100 \times 100 = 10000$

首先考虑一个暴力算法解决问题

RED-BLUE-JUG(R, B)

$n := R.length$

for $i := 1$ to n

 for $j := 1$ to n

 report ■(i, j)

注意 COMPARE(R[i], B[j]) 是常数时间

注意 RED-BLUE-JUG 在时间复杂度 $O(n^2)$ 内解决问题

时间复杂度分析： $n \times n \times n = n^3$ 常数时间复杂度

$$n^3 \cdot \frac{1}{n^2} + \dots + n^3 \cdot \frac{1}{n^2} + n^3 \cdot \frac{1}{n^2} = n^2$$

Algorithm - P101

水壶问题

对于 n 个红色水壶和 n 个蓝色水壶，红色水壶的水量均不相同，蓝色水壶的水量也均不相同

■ 水壶之间无法直接比较水量，但是存在比较一个红色水壶和一个蓝色水壶水量的操作
操作需要一个单位的时间，且无法比较两个红色水壶或两个蓝色水壶

假设对于每个红色水壶，有且仅有一个蓝色水壶与之水量相同
若基于比较操作的匹配算法。

(min) 假设红色水壶以任意顺序排成一列，即 R_1, R_2, \dots, R_n

则目标是对于任意输入的蓝色水壶序列 B_1, B_2, \dots, B_n 进行排序

得到 $B_{\pi(1)}, B_{\pi(2)}, \dots, B_{\pi(n)}$ ，且有 $R_i = B_{\pi(i)}$, $i=1, 2, \dots, n$

则可以构造基于比较操作的排序算法的决策树，

由于每一次比较操作会返回 小于 / 等于 / 大于三种结果之一

于是可知决策树为满二叉树，则对于高度为 h 的满二叉树

其叶结点个数 l 满足关系 $n! \leq l \leq 3^h$

于是有 $h \geq \log_3(n!)$ ，即 $h \in \Omega(n \lg n)$

于是可知对于任意基于比较操作的匹配算法，其比较操作次数的下界为 $\Omega(n \lg n)$

考虑一个类似于 QUICK-SORT 的匹配算法，对于输入的 $R[1..n]$ 和 $B[1..n]$

返回升序排列的 $R[1..n]$ 和 $B[1..n]$ ，则有 $R[i] = B[i]$, $i=1, 2, \dots, n$

RED-BLUE-QUICK-MATCH(R, p_R, r_R, B, p_B, r_B)

if $p_R < r_R$ or $p_B < r_B$:

 pivot := RANDOM(p_R, r_R)

$q_B := \text{RED-BLUE-PARTITION}(B, p_B, r_B, R[pivot])$

$q_R := \text{RED-BLUE-PARTITION}(R, p_R, r_R, B[q_B])$

 RED-BLUE-QUICK-MATCH($R, p_R, q_R-1, B[q_B], p_B, q_B-1$)

 RED-BLUE-QUICK-MATCH($R, q_R+1, r_R, B, q_B+1, r_B$)

RED-BLUE-PARTITION($A, p, r, pivot$)

$i := p + 1 \dots r$

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

 if $\text{COMPARE}(A[i], pivot)$ is LT or EQ

$i := i + 1$

 Swap $A[i]$ with $A[j]$

 if compare result is EQ ($A[i] = A[j]$)

$i := j$

Swap $A[i]$ with $A[p]$ return i

Algorithm - P102

水壶问题

对于 n 个水量不同的红色水壶和 m 个水量不同的蓝色水壶，无法直接比较两个红色水壶或两个蓝色水壶

存在一个在单位时间内比较一个红色水壶和一个蓝色水壶水量的操作

对于每一个红色水壶，有且仅有一个蓝色水壶与其水量相同

考虑基于比较操作的匹配算法 RED-BLUE-QUICK-MATCH

过程类似于随机化的快速排序算法

首先从子数组 $R[P_R..R_R]$ 中随机选取一个红色水壶作为主元 pivot

利用 pivot 对子数组 $B[P_B..R_B]$ 中的蓝色水壶进行划分

返回与 pivot 水量相同的蓝色水壶的下标 q_B

再利用 $B[q_B]$ 对子数组 $R[P_R..R_R]$ 中的红色水壶进行划分

返回与 $B[q_B]$ 水量相同的红色水壶的下标 q_R

注意虽然在过程中使用了两套下标 P_B, q_B, r_B 和 P_R, q_R, r_R

但是在初次调用时 $P_B = P_R = 1$, $r_B = r_R = n$

又由于红色水壶与蓝色水壶是一一对应的

则返回的 q_B 与 q_R 是相等的。由于 $B[q_B] = R[q_R]$

且水量少于/多于 $R[q_R]$ 的红色水壶

与水量少于/多于 $B[q_B]$ 的蓝色水壶的数量是相同的

进一步地对于每次递归调用, $P_B = P_R$ 且 $r_B = r_R$

并有 $B[P_B..r_B]$ 与 $R[P_R..r_R]$ 是一一对应的, 且有 $q_B = q_R$, $B[q_B] = R[q_R]$

于是可知匹配算法 RED-BLUE-QUICK-MATCH ($R, 1..n, B, 1..n$)

实际上是对两个 n 个元素数组 $R[1..n]$ 和 $B[1..n]$ 进行随机化快速排序

而完成排序后, 对于任意 $1 \leq i \leq n$, 有 $R[i] = B[i]$

可知 RED-BLUE-QUICK-MATCH 在时间复杂度 $O(n \lg n)$ 内完成 $B[1..n]$ 和 $R[1..n]$ 的匹配

平均排序 (average sorting), 假设对于输入的 n 个互异元素构成的数组, 不要求严格单调的排序

而是仅要求数组中的元素在平均情况下是升序的 (increase on average)

即对于给定的正整数 k , 对于任意 $i = 1, 2, \dots, n-k$

数组 A 满足 $[A_{j=i}^{i+k-1} A[j]] / k \leq [A_{j=i+1}^{i+k} A[j]] / k$

即 $A[i], \dots, A[i+k-1]$ 的均值 $\leq A[i+1], \dots, A[i+k]$ 的均值

则称包含 n 个元素的数组 A 是 k -排序的 (k -sorted)

注意到当 $k=1$ 时, 1 排序的数组 A 等价于数组 A 是升序排列的

如对于 $k=2$, 数组 A 为整数 $1, 2, \dots, 11$ 的一个排列

则数组 $A = \langle 1, 7, 2, 8, 3, 9, 4, 10, 5, 11, 6 \rangle$ 是 2 排序的

Algorithm - P103 ~~for~~ - ~~end~~

平均排序

对于输入的包含 n 个元素的数组 A 以及给定的正整数 k ，如果对于任意 $i=1, 2, \dots, n-k$ ，有 $(\sum_{j=i}^{i+k-1} A[j])/k \leq (\sum_{j=i+1}^{i+k} A[j])/k$

则称数组 A 是 k 排序的

于是有对于包含 n 个元素的数组 A 是 k 排序的，当且仅当对于任意 $i=1, 2, \dots, n-k$ ，有 $A[i] \leq A[i+k]$

证明过程有，当包含 n 个元素的数组 A 是 k 排序的，其中 k 为正整数

则根据定义有，对于任意 $i=1, 2, \dots, n-k$

$$(\sum_{j=i}^{i+k-1} A[j])/k \leq (\sum_{j=i+1}^{i+k} A[j])/k$$

$$A[i] + \sum_{j=i+1}^{i+k-1} A[j] \leq A[i+k] + \sum_{j=i+1}^{i+k-1} A[j]$$

即有 $A[i] \leq A[i+k]$

当在包含 n 个元素的数组 A 中，对于任意 $i=1, 2, \dots, n-k$ ，有 $A[i] \leq A[i+k]$ 时

$$A[i] + (A[i+1] + \dots + A[i+k-1]) \leq (A[i+1] + \dots + A[i+k-1]) + A[i+k]$$

$$\sum_{j=i}^{i+k-1} A[j] \leq \sum_{j=i+1}^{i+k} A[j]$$

$$(\sum_{j=i}^{i+k-1} A[j])/k \leq (\sum_{j=i+1}^{i+k} A[j])/k$$

则包含 n 个元素的数组 A 是 k 排序的

于是包含 n 个元素的数组 A 是 k 排序的，当且仅当对于任意 $i=1, \dots, n-k$ ，有 $A[i] \leq A[i+k]$

考虑对于输入包含 n 个元素的数组 A 以及给定的正整数 k ，对数组 A 进行 k 排序的算法

注意到数组 A 是 k 排序的，当且仅当对于任意 $i=1, 2, \dots, n-k$ ，有 $A[i] \leq A[i+k]$

则可知对于给定的 $0 \leq d < k$ ，有下标序列 $i_d, i_{d_2}, \dots, i_{d_n}$

有 $1 \leq i_d < i_{d_2} < \dots < i_{d_n} \leq n$ 。且 $i_d \equiv i_{d_2} \equiv \dots \equiv i_{d_n} \equiv d \pmod{k}$

于是可知 $A[i_d] \leq A[i_{d_2}] \leq \dots \leq A[i_{d_n}]$ ，且 $\sum_{d=0}^{k-1} i_d = n$

即 k 排序的数组 A 中包含 k 个 \blacktriangleleft 排序的子序列

$K\text{-SORT}(A, k)$

$n := A.length$

let $B[0..k-1]$ be new array of new array

$O(n)$ [for $i := 1$ to n
 | insert $A[i]$ to $B[i \bmod k]$

 for $d := 0$ to $k-1$ MERGESORT

~~for $i := 0$ to $n-1$ MERGESORT~~ ($B[d], 1, B[d].length$)

$O(n)$ [for $i := 1$ to n
 | $A[i] := B[i \bmod k][i \bmod k + 1]$

注意对于每个子序列 $B[d]$ ，可以近似地看作有 $\lfloor n/k \rfloor$ 个元素

于是 MERGESORT 对 $B[d]$ 排序的时间复杂度为 $O(\frac{n}{k} \lg(n/k))$

则 $K\text{-SORT}$ 在 $O(n \lg(n/k))$ 的时间复杂度内对 n 个元素的数组 A 进行 k 排序

Algorithm - P104

K-SORT

5.6.9 - K-SORT

平均排序

对于输入的包含 n 个元素的数组 A 以及给定的正整数 k

如果对于任意 $i = 1, 2, \dots, n-k$, 有 $(\sum_{j=i}^{i+k-1} A[j]) / k \leq (\sum_{j=i+1}^{i+k} A[j]) / k$

则子数组是 k 对排序的

而对于任意的包含 n 个元素的数组 A 以及正整数 k

K -SORT 在 $O(n \lg(n/k))$ 的时间复杂度内对数组 A 进行 k 排序

由于数组 A 是 k 排序的当且仅当对于任意 $i = 1, 2, \dots, n-k$, 有 $A[i] \leq A[i+k]$

于是 k 对排序的数组 A 可以视为 k 个已排序的子数组交错而成

则可以利用类似于有序链表合并的 MERGE-LINKLIST 算法

对于 k 排序的数组 A 进行全排序

类似地有可以在时间复杂度 $O(n \lg k)$ 内对包含 n 个元素的 k 对排序数组 A 进行全排序

则对于输入的任意包含 n 个元素的数组 A , 以及正整数常数 k

可以在 $O(n \lg(n/k) + O(n \lg k) = O(n \lg n)$ 的时间复杂度内对数组 A 进行全排序

再考虑对输入的任意包含 n 个元素的数组 A 进行 k 排序的比较排序复杂度下界

构造对于任意包含 n 个元素的数组 A 进行 k 排序的比较排序算法的决策树,

可知决策树为满二叉树, 每个叶结点表示一个 k 排序的数组 A

注意 k 对排序的数组 A 中包含 k 个 n/k 元素的子数组, 假设没有 $k \mid n$

每个 n/k 元素的子数组都是输入数组 A 中对应 n/k 个元素的排序

假设决策树的高度为 h , 叶结点数量为 l

则有 $(n/k)!^k \leq l \leq 2^h$

即 $h \geq k \lg(n/k)$

于是有 $h \in k \cdot \Omega(n \lg(n/k)) = \Omega(n \lg(n/k))$

又 k 为正整数常数, 于是有 $h \in \Omega(n \lg n)$

于是可知对任意包含 n 个元素的数组 A 进行 k 排序的比较排序算法都是 $\Omega(n \lg n)$ 的

下界

合并有序列表

对于任意给定的两个包含 n 个元素的有序链表, 考虑合并为一个有序链表的比较次数

注意到对于任意合并两个包含 n 个元素的有序链表的比较算法的决策树

决策树的高度为 h , 叶结点数量为 l

又结点中 $2n$ 个位置中有 n 个位置为第一个有序链表元素的不同选择有 $(2n)^n$ 种

则有 $(2n)^n = \frac{(2n)!}{n! n!} \leq l \leq 2^h$, 即 $h \geq \lg \frac{(2n)!}{n! n!}$

于是有 $h \in \Omega(2n \lg(2n) - 2n \lg n) = \Omega(2n)$

于是可知对于任意两个包含 n 个元素的有序链表合并的比较次数下界为 $\Omega(2n)$

Algorithm - P105

359 - 360

合并有序列表 对于任意给定的两个包含 n 个元素的有序链表，考虑合并为一个有序链表的比较次数下界。

注意在合并形成的包含 $2n$ 个元素的有序链表 A 中

对于任意 $1 \leq i < j \leq n$ ，有 $A[i] \leq A[j]$ 。即 A 中

考虑下标 $i=1, 2, 3, \dots, 2n-1$ 时，元素 $A[i]$ 和 $A[i+1]$ 原本所在的有序列表

当 $A[i]$ 和 $A[i+1]$ 来自同一个有序链表时

由于合并时保留了每一个有序链表内的相对顺序

于是无需比较即有 $A[i] \leq A[i+1]$ 。

而当 $A[i]$ 和 $A[i+1]$ 来自不同的有序链表时，则必须有一次比较用于确定 $A[i]$ 与 $A[i+1]$ 的顺序。

存在排列 π 有 $2n-1$ 对 相邻的元素 $A[i]$ 和 $A[i+1]$ ，使得 $A[i], A[i+1]$ 来自不同有序链表。

提对于任意两个包含 n 个元素的有序链表的合并的比较算法

至少需要 $2n-1$ 次比较。

比较交换操作 (compare-exchange operation)，对于任意包含 n 个元素的数组 A ，以及下标 $1 \leq i < j \leq n$

COMPARE-EXCHANGE (A, i, j)

if $A[i] > A[j]$

将 $A[i]$ 与 $A[j]$ 互换，即 $A[i] = A[j]$, $A[j] = A[i]$ 。

遗忘排序算法 (oblivious compare-exchange algorithm)

即算法仅执行一系列预先定义好的比较交换操作。

operate solely by sequence of prespecified compare-exchange operation

需要比较交换的下标需要事先确定

即算法可能依靠待排序的元素个数

但不能依赖于待排序元素的值

也不能依赖于任何之前的比较排序的结果

index of position compared in sequence must be determined in advance

although can depend on number of element being sorted

can not depend on value being sorted

nor depend on result of any prior compare-exchange operation

如 OBLIVIOUS-INSERTION-SORT (A)

for $j := 2$ to $A.length$

for $i := j-1$ down to 1

COMPARE-EXCHANGE ($A, i, i+1$)

Algorithm - P106

0-1 排序引理 (0-1 sorting lemma), 用于证明遗忘比较交换算法可以产生正确的排序结果

即如果遗忘比较交换算法能正确对所有只包含 0 和 1 的输入序列排序

则该算法可以对包含任意值的输入序列排序

通过证明逆否命题来证明 0-1 排序引理

即如果一个遗忘比较交换算法不能对一个包含任意值的输入序列进行排序

则该算法也不能对某个 0-1 序列进行排序

假设对于某个遗忘比较交换算法 X 未能对数组 $A[1..n]$ 排序

令 $A[p]$ 为算法 X 未能 放到正确位置的最小元素, 其中 $1 \leq p \leq n$

而 $A[q]$ 为算法 X 放在 $A[p]$ 的正确位置上的元素

令 $B[1..n]$ 为一个只包含 0 和 1 的数组

并有 $B[i] = \begin{cases} 0, & A[i] \leq A[p], \\ 1, & A[i] > A[p]. \end{cases}$ 其中 $1 \leq i \leq n$

注意首先有 $1 \leq q < p \leq n$, 由于如果 $p < q$, 则数组 A 中有 $q-1$ 个小于 $A[p]$ 的元素

而如果 $A[p]$ 占据了前 $q-1$ 个位置之一, 则必定存在元素 $A[r]$

有 $A[r]$ 不在正确位置上, 且 $A[r] < A[p]$, 与 $A[p]$ 是最小错误元素矛盾

另外有 $A[q] > A[p]$, 由于如果有 $A[q] < A[p]$

而 $A[q]$ 占用了 $A[p]$ 的正确位置, 则 与 $A[p]$ 是最小错误元素矛盾

于是可知 $B[p] = 0$, $B[q] = 1$ 且 $q < p$, 即 $B[p]$ 与 $B[q]$ 的顺序错误

即该遗忘比较交换算法未能对数组 B 进行正确排序

列排序 (columnsort), 用于对包含 n 个元素的 $r \times s$ 矩形数组的排序

且满足三个限制条件: r 必须是偶数, s 必须是 r 的因子 ($s | r$), $r \geq 2s^2$

当列排序算法完成时, 矩形数但是列优先有序的 (column-major order)

即按照列从上到下, 从左到右, 希望单调递增的 (monotonically increase)

不考虑 n 的值, 列排序包含 8 个步骤, 其中奇数步骤均为对每个单独的列进行排序

① 对每一列单独地进行排序

② 转置矩形数组, 并重新规范化为 $r \times s$ 矩阵

即最左边一列放置于前 r/s 行, 第二列放置于第二组 r/s 行, ...

③ 对每一列单独地进行排序, ④ 执行②的逆操作

⑤ 对每一列单独地进行排序

⑥ 将每一列的上半部分与下半部分交换, 即交换两个 $(r/2) \times s$ 矩阵位置

将上半部分向右移动一列, 形成 $r \times (s+1)$ 矩阵, 其中第一列上半与最后一列下半为空

⑦ 对每一列单独地进行排序, ⑧ 执行⑥的逆操作, 恢复成 $r \times s$ 矩阵

Algorithm - P107

列排序 对于包含 n 个元素的 $r \times s$ 矩形数组的排序
且有 r 是偶数, s 为 r 的因子($s|r$), $r \geq 2s^2$

6	10	14	5	①	4	1	2	②	4	8	10	③	1	3	6	④	1	4	11
8	7	17			8	3	5		12	16	18		2	5	7		3	8	14
12	1	6			10	7	6		1	3	7		4	8	10		6	10	17
16	9	11			12	9	11		9	14	15		9	13	15		2	9	12
4	15	2			16	14	13		2	5	6		11	14	17		5	13	16
18	3	13			18	15	17		11	13	17		12	16	18		7	15	18
⑤	1	4	11	⑥	5	10	16	⑦	4	10	16	⑧	1	7	13				
2	8	12			6	13	17		5	11	17		2	8	14				
3	9	14			7	15	18		6	12	18		3	9	15				
5	10	16			1	4	11		1	7	13		4	10	16				
6	13	17			2	8	12		2	8	14		5	11	17				
7	15	18			3	9	14		3	9	15		6	12	18				

注意在这个例子中, $r=6, s=3$, 并不满足 $r \geq 2s^2$

注意在列排序算法中, 即使不知道奇数步聚采用何种排序算法

也可以假定对于每一列独立地执行 OBLIVIOUS-INSERTION-SORT

由于 OBLIVIOUS-INSERTION-SORT 为遗忘比较交换排序

且列排序的其他操作即不依赖于输入元素的值, 也不依赖于之前排序的结果

于是列排序也可以视为一种遗忘比较交换排序算法

考虑在列排序过程中的 $r \times s$ 矩阵保存的数组

如果数组的某区域全部为0或全部为1, 则称该区域是干净的 (clean)

如果数组的某区域既包含0也包含1, 则称该区域是脏的 (dirty)

则对于任意输入的包含 n 个元素的 $r \times s$ 0-1矩阵进行列排序的过程中

经过 ①-③步: 矩阵顶部为由0组成的干净行, 底部为由1组成的干净行

经过 第④步: 按列优先有序地读取矩阵, 开头为全0序列, 结尾为全1序列

中间为长度至多为 s^2 的脏序列

经过 ⑤-⑧步: 产生一个全排序的0-1输出, 即列排序算法正确地排序0-1矩阵

于是根据0-1引理, 列排序可以正确地排序任意输入包含 n 个元素的 $r \times s$ 矩阵

Algorithm - P108

顺序统计量 (order statistic), 对于包含 n 个元素的数组，~~且数组是有序的~~

集合中的第 i 小的元素为第 i 个顺序统计量

则平凡地有最小值 (minimum) 为集合的第 1 个顺序统计量

最大值 (maximum) 为集合的第 n 个顺序统计量

中位数 (median), 对于包含 n 个元素的数组, 非形式化地描述为集合的“中点元素”(halfway point)

当 n 为奇数时, 数组有唯一的中位数, 即第 $i = (n+1)/2$ 个顺序统计量

当 n 为偶数时, 数组有两个中位数, 即第 $i = n/2$ 和 $j = n/2 + 1$ 个顺序统计量

如果不考虑 n 的奇偶性, 则可以一般地定义中位数

下中位数 (lower median): 第 $\lceil (n+1)/2 \rceil$ 个顺序统计量

上中位数 (upper median): 第 $\lceil (n+2)/2 \rceil$ 个顺序统计量

选择问题 (selection problem), 对于包含 n 个元素的输入数组, 考虑选择第 i 个顺序统计量的问题

输入: 包含 n 个元素的数组 A , 以及正整数 $1 \leq i \leq n$

输出: 元素 $x \in A$, 且数组 A 中恰好存在 $i-1$ 个其他元素小于 x

注意选择问题可在 $O(n \lg n)$ 内解决,

对于输入的包含 n 个元素的输入数组 A 以及正整数 i

可以使用 HEAPSORT 或 MERGESORT 在 $O(n \lg n)$ 的时间复杂度内对数组 A 排序

再查询已排序数组 A 中下标为 i 的元素并输出

对于包含 n 个元素的输入数组 A , 考虑确定其最小元素所需的比较次数

可知需要的比较次数的上界为 $n-1$ 次

MINIMUM(A)

MAXIMUM(A)

$\min := A[1]$ $\max := A[1]$

for $i := 2$ to $A.length$ for $i := 2$ to $A.length$

if $\min > A[i]$ if $\max < A[i]$

$\min := A[i]$ $\max := A[i]$

return \min

return \max

于是有 MINIMUM 通过最多 $n-1$ 次比较次数得到数组 A 的最小值

类似地有 MAXIMUM 通过最多 $n-1$ 次比较得到最大值

注意由于在获得最小值 / 最大值的过程中,

其他 $n-1$ 个元素都至少在一次比较中被确认大于 / 小于另一个元素

于是至少需要 $n-1$ 次比较, 即 MINIMUM 和 MAXIMUM 是最优的