

Algorithm - P84

决策树

在最坏情形下，任何比较排序算法都需要做 $\Omega(n \lg n)$ 次比较

证明过程有，对于每个排列都是一个可达叶结点的决策树，高度是可确定的
考虑高度为 h ，具有 l 个可达叶结点的决策树

对应于一个对 n 个输入元素所进行的比较排序

由于 $n!$ 个可能排列均出现在决策树的可达叶结点中
于是有 $n! \leq l$ ，即某些排列可能出现在多于 1 个叶结点中

而对于高度为 h 的满二叉树，叶结点数 $l \leq 2^h$
于是有 $n! \leq l \leq 2^h$

$h \geq \lg(n!)$ ，即有 $h \in \Omega(n \lg n)$

由于决策树高度的下界即比较排序算法运行时间的下界
于是最坏情形下，任何比较排序算法都需要 $\Omega(n \lg n)$ 次比较

在最好情形下，任何比较排序算法都需要做 $n-1$ 次比较

证明过程有，考虑比较排序算法的决策树中，叶结点可能的最小深度
注意对于输入的 n 个元素进行比较排序

其输出序列 $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ 中存在 $n-1$ 个相对次序信息

于是至少需要经过 $n-1$ 个内结点，才能达到对应的叶结点

即决策树中的叶结点，可能的最小深度为 $n-1$

这是最好情形下，任何比较排序算法都需要做 $n-1$ 次比较

考虑对 $n!$ 个长度为 n 的输入序列中的一部分，达到线性运行时间的比较排序算法的可能性

如果对于 $n!/2$ ，则有 $n!/2 \leq l \leq 2^{h+1}-1$

则 $h \geq \lg(n!/2+1)-1 \geq \lg(n!/2)-1 \geq \lg(n!) - 2$ ，即有 $h \in \Omega(n \lg n)$

如果对于 $n!/n$ ，则有 $n!/n \leq l \leq 2^{h+1}-1$

则 $h \geq \lg(n!/n+1)-1 = \lg[(n-1)!+1]-1$ ，即有 $h \in \Omega(n \lg n)$

如果对于 $n!/2^n$ ，则有 $n!/2^n \leq l \leq 2^{h+1}-1$

则 $h \geq \lg(n!/2^n+1)-1 \geq \lg(n!/2^n)-1 = \lg(n!) - n - 1$ ，即有 $h \in \Omega(n \lg n)$

堆排序 (heap sort) 和归并排序 (merge sort)

都是渐近最优 (asymptotically optimal) 的比较排序算法

由于堆排序和归并排序的运行时间上界为 $O(n \lg n)$

而最坏情况下的下界是 $\Omega(n \lg n)$ 的

Algorithm - P85

决策树

对于一个包含 n 个元素的待排列序列，该序列由 n/k 个子序列组成，每个子序列包含 k 个元素。
给定子序列中每个元素均小于其后继子序列中的所有元素
且大于其前驱子序列中的所有元素

于是对长度 n 的序列排序转化为对 n/k 个子序列中的 k 个元素的排序

考虑对应比较排序算法对应的决策树，其高度为 h ，可达叶结点数量为 l

$$\text{于是 } (k!)^{n/k} \leq l \leq 2^{h+1}$$

$$h \geq \lg [(k!)^{n/k} + 1] - 1 \geq \lg [(k!)^{n/k}] - 1$$

$$= \frac{n}{k} (\lg k!) - 1 \geq \frac{n}{k} \cdot \frac{k \ln k - k}{\ln k} = \frac{1}{\ln k} (n \ln k - n)$$

于是排序问题所需比较次数的下界为 $\Omega(n \lg k)$

但是注意如果直接将 $\frac{n}{k}$ 个子序列所需的比较次数的下界合并

则有比较次数的下界为 $\frac{n}{k} \Omega(k \lg k)$ ，也即 $\Omega(n \lg k)$ 的

计数排序 (Counting Sort)，如果 n 个输入元素均为 $[0, k]$ 区间内的一个整数，其中 $k \in \mathbb{N}$

则当 $k \in O(n)$ 时，计数排序的时间复杂度为 $O(n)$

基本思想是对每一个输入元素，确定小于 x 的元素个数

利用这个信息可以直接将 x 放在输出数组的正确位置

另外特别地处理当存在相同元素的情形

COUNTING-SORT(A, B, k)

① let C[0..k] be new array

for i := 0 to k

 C[i] := 0

② for j := 1 to A.length

 C[A[j]] := C[A[j]] + 1

③ for i := 1 to k

 C[i] := C[i] + C[i-1]

④ for j := A.length to 1

 B[C[A[j]]] := A[j]

 C[A[j]] := C[A[j]] - 1

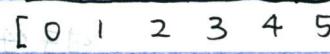
→ 初始化临时数组 C[0..k]

→ C[i] 保存 A[1..n] 中元素 i 的出现次数

→ C[i] 保存 A[1..n] 中不大于 i 的元素个数，即元素 i 最后可以排入已排序数组的位置

→ 将 A[i] 放入 B[1..n] 中的正确位置。

		B	
		0	1
		0	2
		2	3
		2	4
		3	5
		3	6
		3	7
		5	8



Algorithm - P86

计数排序

在 COUNTING-SORT 实现中，~~时间复杂度为 $O(n+k)$~~ ，~~空间复杂度为 $O(k)$~~ 。

第①③个循环的时间复杂度是 $O(k)$ 的常数倍，即 $O(k \cdot n)$ 。

第②④个循环的时间复杂度是 $O(n)$ 的常数倍，即 $O(n^2)$ 。

于是 COUNTING-SORT 在时间复杂度 $O(n+k)$ 对输入的 n 个元素进行排序。

在实际中，如果选择 $k = O(n)$ ，则 COUNTING-SORT 时间复杂度为 $O(n)$ 的常数倍，即 $O(n^2)$ 。

注意 COUNTING-SORT 的复杂度低于 $O(n \lg n)$ 。

这是由于 $O(n \lg n)$ 是比较排序算法的下界。

但 COUNTING-SORT 并不是基于元素间的比较操作，即不是比较排序算法。

于是可以突破 $O(n \lg n)$ 的下界。

COUNTING-SORT 是稳定的 (stable)。

证明过程有，考虑在输入序列中两个相同的元素 $A[i]$ 和 $A[j]$ ， $1 \leq i < j \leq n$ 。

则在第④个循环将 $A[i]$ 和 $A[j]$ 装入 $B[\pi(i)]$ 和 $B[\pi(j)]$ 的过程， $1 \leq \pi(i), \pi(j) \leq n$ 。

由于第④个循环中 $j := A.length - 1$

所以 $A[j]$ 会先于 $A[i]$ 放入输出序列 B 中。

又放入索引 $[A[i]]$ 在循环④中是递减的。

于是有 $\pi(j) > \pi(i)$ ，即 $A[i]$ 与 $A[j]$ 在输出序列中保持相对应位置顺序不变。

注意如果把第④个循环改为 $j := 1 \text{ to } A.length$

虽然输出序列依旧是非好序的，但不再是稳定的。

且对于输入序列中 $A[i_1] = A[i_2] = \dots = A[i_k]$ ，其中 $1 \leq i_1 < i_2 < \dots < i_k \leq n$

则在输出序列中有 $\pi(i_1) > \pi(i_2) > \dots > \pi(i_k)$ 。

考虑对于任意给定的 n 个在 0 和 k 之间的整数 $A[1..n]$ 进行预处理。

并可以以 $O(1)$ 的时间复杂度内完成查询在区间 $[a, b]$ 内的元素个数， $0 \leq a \leq b \leq k$ 。

PREPROCESSING (A , B)

let C be new array

for $i := 0$ to k

$C[i] := 0$

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

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

for $i := 1$ to k

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

QUERY (a, b)

return $B[b] - B[a] + C[a]$

于是可知 PREPROCESSING 时间复杂度为 $O(n+k)$ 。

QUERY 时间复杂度为 $O(1)$ 。

Algorithm - P87

基数排序 (radix sort) 应用于卡片排序机上的算法

通常一张卡片有 80 列，每列可以在 12 个位置中选择一处穿孔

可以对排序机“编程”以检查每张卡片的指定列

并根据穿孔的位置将卡片放置于 12 个容器之一

操作员可以依次从各个容器收集卡片

并保持第一个位置穿孔 → 第二个位置穿孔 → … → 第 12 个位置穿孔

注意如果按最高有效位进行排序，再对每个容器中的卡片递归地排序

则为了排序一个容器的卡片需要临时地保存其余容器的卡片

假设需要排序 n 张 d 位 b 进制的卡片

则最坏情形下，每次排序都使得 b 个容器中都有卡片

现在递归地执行下，需要 $\Theta(bd)$ 次的排序操作

而由于在每一轮都需要临时地保存容器中的卡片

会产生 $\Theta(bd)$ 的临时卡片容器

基数排序是按最低有效位 (least significant digit first) 进行排序的

在每一轮排序结束之后，算法将卡片合并为一堆

其中按容器顺序编号最小在上 编号最大在下 合并容器内的卡片

合并后再按照次低有效位进行排序，重复直到所有 d 位数字都被排序

假设在数组 A 中存放了 n 个 d 位元素，其中第 1 位为最低位，第 d 位为最高位

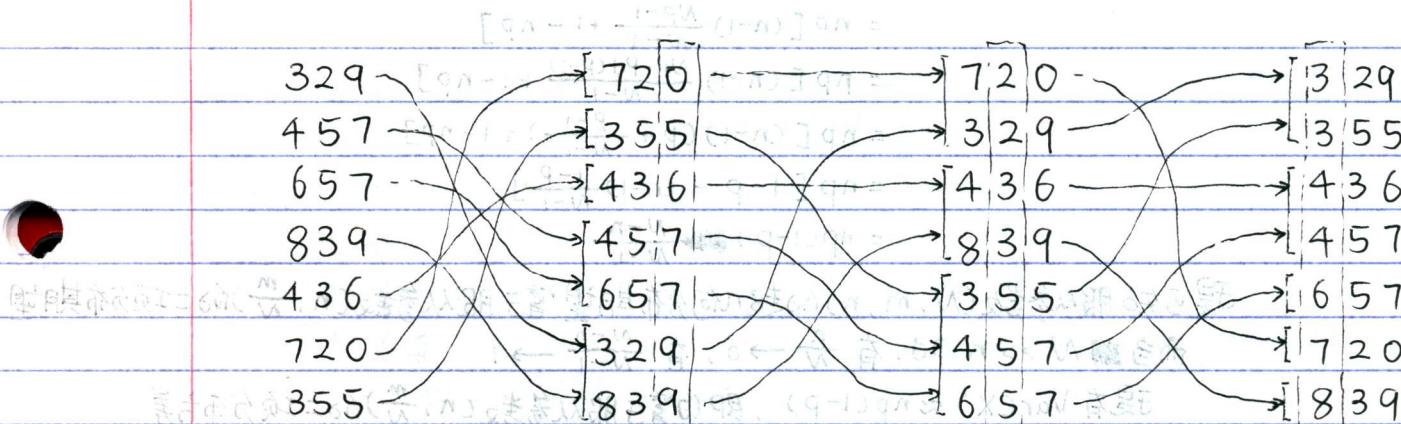
RADIX-SORT(A, d)

for $i := 1$ to d

use stable sort to sort array A on digit i

为确保基数排序的正确性，基数排序算法必须是稳定的 (计数/插入/归并)

且操作员必须确保卡片从容器中取出时不改变顺序



Algorithm - P88

基数排序

循环不变量：在循环的第*i*次迭代开始前，数组A是按最低*i*-1位排序的。

初始化：在for循环的第一次迭代开始前。

数组A是按最低0个有效位排序平凡地为真。

保持：假设对于正整数*i*，在for循环的第*i*次迭代开始前，数组A是按最低*i*-1个有效位排序。

则在第*i*次迭代中对数组A按第*i*位最低有效位排序。

对于数组中任意两个元素 a_j, a_k ，其中 $1 \leq j, k \leq n$

如果 a_j 的第*i*位小于 a_k 的第*i*位，则第*i*次迭代结束时 a_j 排在 a_k 之前。

如果 a_j 的第*i*位与 a_k 的第*i*位相同，则在第*i*次迭代结束时

a_j 与 a_k 的相对顺序取决于第*i*次迭代开始前的顺序。

又由于循环迭代中使用的是对有效位的稳定排序。

则 a_j 排在 a_k 之前且仅当 a_j 的低*i*-1位小于 a_k 的低*i*-1位。

于是可知在第*d*次迭代前，数组A是按低*d*位排序的。

终止：在循环结束后，即在第*d*+1次迭代前，数组A是按低*d*位排序的。

$c_{\text{cost}} + c_{\text{loop}} + c_{\text{out}} = c_{\text{cost}} + c_{\text{loop}} + c_{\text{out}} = c_{\text{cost}}[c_{\text{out}} + c_{\text{loop}}]$

对于给定*n*个*d*位数，其中每一位数有*k*个可能取值。

如果RADIX-SORT使用复杂度为 $\Theta(n+k)$ 的稳定排序算法，则可以在 $\Theta(d(n+k))$ 内完成排序。

证明过程有，注意对于RADIX-SORT的时间复杂度分析依赖于所使用的稳定排序算法。

假定每位数字都取值于 $\{0, 1, \dots, k-1\}$

则当*k*值不太大时，可以选择计数排序。

又对于*n*个*d*位数，可以在 $\Theta(n)$ 内提取待排序的指定位数。

并可以在 $\Theta(n)$ 时间内根据排序的指定位数排序*n*个*d*位数。

于是每个循环迭代的时间复杂度为 $\Theta(n+k)$ 。

则RADIX-SORT在时间复杂度 $\Theta(d(n+k))$ 内完成排序。

特别地当*d*为常数且 $k \in O(n)$ 时，RADIX-SORT时间复杂度为 $\Theta(n)$ 。

对于给定*n*个*b*位数和任意正整数 $r \leq b$ ，其中为*b*位二进制数。

如果RADIX-SORT使用时间复杂度为 $\Theta(n+k)$ 对数值取值区间 $[0, k]$ 排序的稳定排序算法。

则可以在 $\Theta(\frac{b}{r}(n+2^r))$ 时间内完成排序。

证明过程有，对于给定 $r \leq b$ ，每个关键字可以看作 $d = \lceil b/r \rceil$ 个*r*位数。

其中每个数都是 $[0, 2^r-1]$ 的整数，则可以采用计数排序，且 $k = 2^r$ 。

则每一轮计数排序的时间复杂度为 $\Theta(n+k) = \Theta(n+2^r)$ 。

又总共有 $d = \lceil b/r \rceil$ 次循环迭代。

于是排序的时间复杂度为 $\Theta(\frac{b}{r}(n+2^r))$ 。

Algorithm - P89

基数排序

对于给定 n 个 b 位二进制数以及任意正整数 $r \leq b$

如果 RADIX-SORT 使用时间复杂度为 $\Theta(n+k)$ 对数值取值于区间 $[0, k]$ 的排序的稳定排序算法

则可以在 $\Theta\left(\frac{b}{r}(n+2^r)\right)$ 时间复杂度内完成排序

考虑对于给定的正整数 n 和 b , 选择 $r \leq b$ 的值以最小化 $\frac{b}{r}(n+2^r)$ 的值

注意如果 $b < \lfloor L \lg^n \rfloor$, 则对于任意 $r \leq b$, 均有 $(n+2^r) \in \Theta(n)$

选择 $r = b$, 则基数排序的时间代价为 $\frac{b}{b}(n+2^b) \in \Theta(n)$

这个时间复杂度是渐近意义上最优的 (asymptotically optimal)

如果 $b \geq \lfloor L \lg^n \rfloor$, 则考虑选择 $r = \lfloor L \lg^n \rfloor$

此时基数排序的时间代价为 $\frac{b}{\lfloor L \lg^n \rfloor}(n+2^{\lfloor L \lg^n \rfloor}) \in \Theta(bn/\lg^n)$

而当 r 的选择逐步增加而大于 $\lfloor L \lg^n \rfloor$ 之后, 2^r 的增加更快

得到时间复杂度为 $\Omega(bn/\lg^n)$

而当 r 的选择逐步减少而小于 $\lfloor L \lg^n \rfloor$ 之后

此时 b/r 项增大而 $(n+2^r)$ 则保持 $\Theta(n)$

于是选择 $r = \lfloor L \lg^n \rfloor$ 可以得到偏差不超过常数系数范围内的时间代价

考虑基数排序与基于比较的排序算法 (如快速排序) 的时间代价比较

在通常情况下, 如果有 $b = O(\lg^n)$, 则选择 $r \approx \lg^n$

此时基数排序的运行时间代价为 $\Theta(n)$

从结果上看是优于期望时间复杂度为 $\Theta(n \lg^n)$ 的快速排序

但是隐藏在算法中的常数因子是不同的

对于给定的 n 个元素进行排序时, 基数排序执行的循环轮数比快速排序要少

但是每一轮基数排序需要耗费的时间要长得多

于是排序算法的选择依赖于具体实现和底层硬件的特性以及输入数据的特征

快速排序通常比基数排序更有效地使用硬件缓存

当主存容量相对宝贵时, 更倾向于使用原地排序的算法 (如快速排序)

而利用计数排序作为中间稳定排序的基数排序不是原地排序 (in-place sort)

而时间复杂度为 $\Theta(n \lg^n)$ 的比较排序包含很多原地排序的算法

对于给定 n 个在区间 $[0, n^3-1]$ 内的整数进行排序

可以在 n 个 $O(1)$ 时间内将 n 个整数转换为 n 进制整数, 且均不超过 3 位

于是进行 3 轮时间复杂度为 $\Theta(n+n)$ 的计数排序进行排序

于是可以在时间复杂度 $O(n)$ 内完成对 n 个在区间 $[0, n^3-1]$ 内的整数进行排序

注意对于 n 个在区间 $[0, n^d-1]$ 内的整数进行排序时

时间复杂度为 $nO(d) + d\Theta(n+n) \in \Theta(nd)$ 内完成排序

Algorithm - P90 1P9 - multimap

桶排序

(bucket sort). 与计数排序类似, 对输入数据做出某种假设以提高速度

桶排序假设输入数据服从均匀分布, 平均情况下时间代价为 $O(n)$

即输入由一个随机过程产生, 将输入元素均匀独立地分布于 $[0, 1]$ 区间上

桶排序将 $[0, 1]$ 区间划分为 n 个相同大小的子区间, 称为桶(bucket)

将 n 个输入元素分别放入各个桶中

由于输入数据是均匀, 独立地分布在 $[0, 1]$ 区间上

所以通常不会出现过多元素落在同一个桶中的情况

先对每个桶中的元素进行排序, 再遍历每个桶按照次序将各个桶中的元素排列出来

对于一个包含 n 个元素的数组 A, 且对每个元素有 $0 \leq A[i] < 1$

同时需要一个临时数组 B[0..n-1] 用于存放链表(linked list)

BUCKET-SORT(A)

$n := A.length$

let B[0..n-1] be new array

for $i := 0$ to $n-1$

 let B[i] be empty linked list

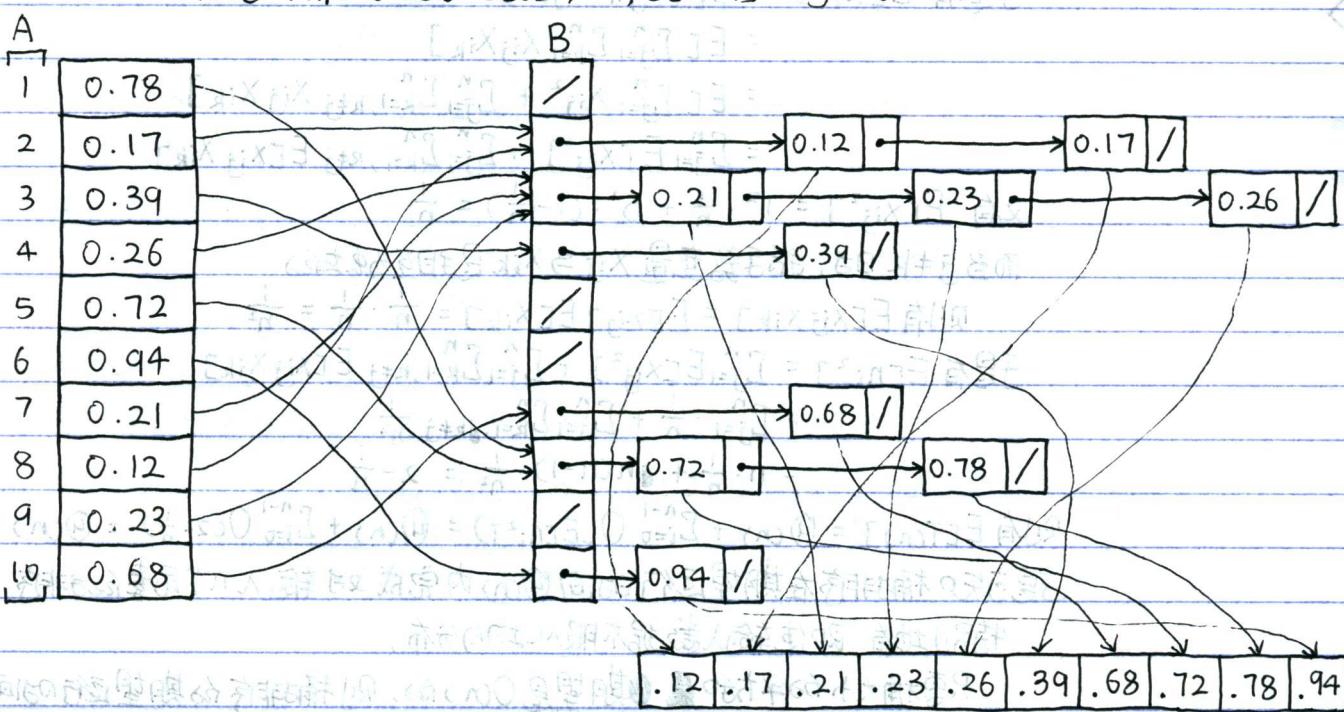
 for $j := 1$ to n

 insert $A[j]$ into linked list B[L $n * A[j]$]

 for $i := 0$ to $n-1$

 sort linked list B[i] with INSERTION-SORT

concatenate linked list B[0], ..., B[n-1] together in order



Algorithm - Pg | 例题 - 证明桶排序

桶排序 对于输入数组 A 中的两个元素 $A[i], A[j]$, 其中 $1 \leq i, j \leq n$

不失一般性地假设 $A[i] \leq A[j]$, 则有 $\ln A[i] \leq \ln A[j]$

于是 $A[i]$ 或者放入 $A[j]$ 所在的桶中, 则插入排序会将 $A[i]$ 和 $A[j]$ 正确地排序
 或者放入下标更大的桶中, 则顺序地选择桶时 $A[i]$ 和 $A[j]$ 正确地排序

于是可知桶排序是正确的

在桶排序的过程中, 将元素放入桶中和顺序地排列桶都是 $O(n)$ 的

则考虑对每个桶进行插入排序的时间复杂度

令随机变量 n_i 表示放入桶 $B[i]$ 的元素个数, 其中 $i = 0, 1, \dots, n-1$

则可知桶排序的时间复杂度为 $T(n) = O(n) + \sum_{i=0}^{n-1} O(n_i^2)$

于是考虑桶排序的期望时间复杂度

$$\begin{aligned} E[T(n)] &= E[O(n) + \sum_{i=0}^{n-1} O(n_i^2)] \\ &= O(n) + E[\sum_{i=0}^{n-1} O(n_i^2)] \\ &= O(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \\ &= O(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \end{aligned}$$

注意对于 $i = 0, 1, \dots, n-1$, 由于元素均匀独立地随机落入一个桶中

于是随机变量 n_i 是同分布的, 即 $E[n_i^2]$ 都是相同的

再令指针变量 X_{ij} 表示元素 $A[j]$ 落入了桶 $B[i]$, 其中 $i = 0, 1, \dots, n-1, j = 1, 2, \dots, n$

则有随机变量 $n_i = \sum_{j=1}^n X_{ij}$

$$\begin{aligned} \text{于是有 } E[n_i^2] &= E[(\sum_{j=1}^n X_{ij})^2] \\ &= E[\sum_{j=1}^n \sum_{k=1}^n X_{ij} X_{ik}] \\ &= E[\sum_{j=1}^n X_{ij}^2 + \sum_{j=1}^n \sum_{k=1, k \neq j}^n X_{ij} X_{ik}] \\ &= \sum_{j=1}^n E[X_{ij}^2] + \sum_{j=1}^n \sum_{k=1, k \neq j}^n E[X_{ij} X_{ik}] \end{aligned}$$

$$\text{又有 } E[X_{ij}^2] = 1^2 \cdot \frac{1}{n} + 0^2 \cdot (1 - \frac{1}{n}) = \frac{1}{n}$$

而当 $j \neq k$ 时, 指针变量 X_{ij} 与 X_{ik} 是相互独立的

$$\text{则有 } E[X_{ij} X_{ik}] = E[X_{ij}] E[X_{ik}] = \frac{1}{n} \cdot \frac{1}{n} = \frac{1}{n^2}$$

$$\begin{aligned} \text{于是有 } E[n_i^2] &= \sum_{j=1}^n E[X_{ij}^2] + \sum_{j=1}^n \sum_{k=1, k \neq j}^n E[X_{ij} X_{ik}] \\ &= \sum_{j=1}^n \frac{1}{n} + \sum_{j=1}^n \sum_{k=1, k \neq j}^n \frac{1}{n^2} \\ &= n \cdot \frac{1}{n} + n(n-1) \cdot \frac{1}{n^2} = 2 - \frac{1}{n} \end{aligned}$$

$$\text{则有 } E[T(n)] = O(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) = O(n) + \sum_{i=0}^{n-1} O(2 - \frac{1}{n}) = O(n)$$

于是可知桶排序在期望运行时间 $O(n)$ 内完成对输入 n 个元素的排序

特别地有, 即使输入数据不服从均匀分布,

只要桶大小的平方和 期望是 $O(n)$ 的, 则桶排序的期望运行时间是 $O(n)$ 的

Algorithm - P92

桶排序

注意桶排序的最坏情形为所有元素被装入同一个桶中

则对其余桶的排序都是 $O(n)$, 而对这个桶进行插入排序的时间复杂度是 $\Theta(n^2)$ 的

于是为了避免这种情形, 可以将对单个桶排序使用归并排序

则此时保持平均情况为线性时间代价 $\Theta(n)$ 和同时通过归并排序

最坏情况下时间代价 $O(n \lg n)$

考虑单位圆内的给定 n 个点, $P_i = (x_i, y_i)$, 其中 $0 < \sqrt{x_i^2 + y_i^2} \leq 1$, $i = 1, 2, \dots, n$

假设所有的点在单位圆上服从均匀分布,

即在单位圆的任意区域内找到给定点的概率与该区域的面积成正比

则考虑对给定的 n 个点, 按到原点之间距离 $d_i = \sqrt{x_i^2 + y_i^2}$ 对 n 个点, 进行排序

首先对单位圆进行划分为 n 个面积相等的区域,

则此时每个点落入 n 个区域之一的概率是相同的

而为了使得桶与桶之间具有相对顺序, 则考虑将单位圆划分为同心圆环.

于是考虑当 $\frac{j-1}{n} < \sqrt{x_i^2 + y_i^2} \leq \frac{j}{n}$ 时, 将点 P_i 放入桶 j

则此时对于桶 j , 其面积为 $\pi [(\frac{j}{n})^2 - (\frac{j-1}{n})^2] = \frac{1}{n} \pi$

可知对于任意点 P_i 放入桶 j 的概率均为 $\frac{1}{n}$

于是有 $j-1 < n(x_i^2 + y_i^2) \leq j$, 即 $j = \lceil n(x_i^2 + y_i^2) \rceil$

BUCKET-SORT-CIRCLE(P)

$h := P.length$

let $B[1..n]$ be new array

for $j := 1$ to n

let $B[j]$ be empty list

for $i := 1$ to n

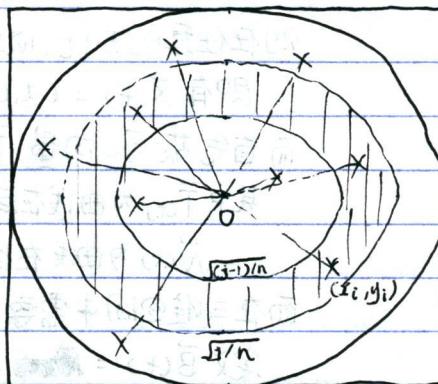
$k := \lceil n * (P[i].x^2 + P[i].y^2) \rceil$

insert $P[i]$ into list $B[k]$

for $j := 1$ to n

sort list $B[j]$ with MERGE-SORT with key as $x^2 + y^2$

concatenate list $B[1], \dots, B[n]$ together in order



注意对点的 key 值的计算是 $O(1)$ 的, 则总时间复杂度与桶排序一样

可知 BUCKET-SORT-CIRCLE 在期望时间复杂度 $\Theta(n)$ 内

对给定的 n 个点, 按到原点距离进行排序

Algorithm - Pg3

桶排序

对于随机变量 X 服从累计分布函数 (cumulative distribution function)

$$P(x) = \Pr\{X \leq x\}, \text{ 其中 } x \in \mathbb{R}$$

对于给定的独立同分布的 n 个随机变量 X_1, X_2, \dots, X_n

假设其累计分布函数 $P(x)$ 可以在 $O(1)$ 的时间内计算得到

则考虑对于一次试验的观测值 x_1, x_2, \dots, x_n 进行排序

注意到对于任意 $x \in \mathbb{R}$, 其累计分布函数值 $P(x) = \Pr\{X \leq x\}$

有 $0 \leq P(x) \leq 1$, 且对于任意 $x \leq y$, 有 $P(x) \leq P(y)$

于是可以将一组观测值 x_1, x_2, \dots, x_n 转换为对应的累计概率

有 $P_i = P(x_i)$, 其中 $i = 1, 2, \dots, n$

且对于 P_1, P_2, \dots, P_n 的排序等价于对 x_1, x_2, \dots, x_n 的排序

则将 $[0, 1]$ 的概率区间划分为 n 个子区间,

$$\text{即 } [0, \frac{1}{n}], [\frac{1}{n}, \frac{2}{n}], \dots, [\frac{n-1}{n}, 1]$$

从而将观测值 x_i 按照 P_i 的值放入对应的桶, 从而应用桶排序

注意在这个过程中实际上并不需要计算实际的分位点的值

对于 n 个服从独立同分布的随机变量 X_1, X_2, \dots, X_n , 以及累计分布函数 $P(x)$

输入一组观测值 x_1, x_2, \dots, x_n

BUCKET-SORT-PROBABILITY ($\langle x_1, x_2, \dots, x_n \rangle, P$)

let $B[1..n]$ be new array

for $j := 1$ to n

 let $B[j]$ be empty list

 for $i := 1$ to n

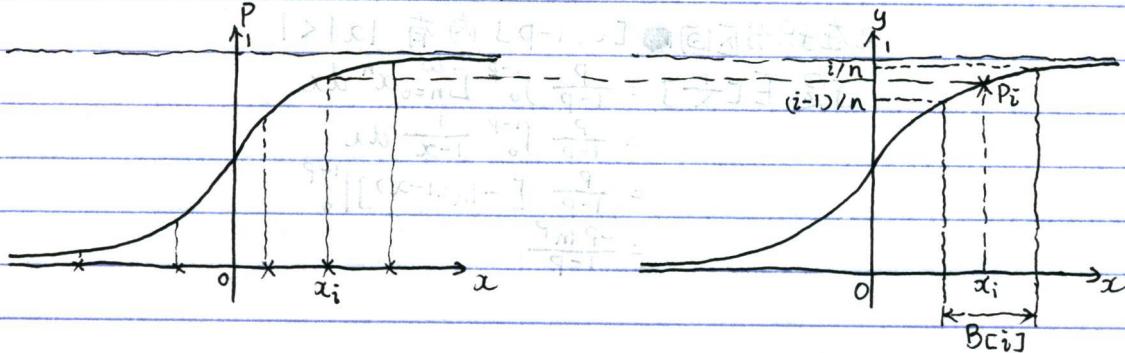
$k := \max\{1, \lceil n * P(x_i) \rceil\}$

 insert x_i into list $B[k]$

 for $j := 1$ to n

 sort list $B[j]$ with MERGE-SORT

 concatenate list $B[1], \dots, B[n]$ in order



Algorithm - Pg 4 例 9 - Comparison

比较排序概率下界，对于一个确定的比较排序算法 A，其决策树为 T_A ，且 A 的输入的每一种情况是等可能的。如果 T_A 的每个叶结点都标记有给定，在随机输入的情况下到达该结点的概率，则对于给定的 n 个互异的输入元素，有 $n!$ 种可能的排列输出。又所有排列是等可能出现的，于是出现的概率均为 $\frac{1}{n!}$ 。而每种排列都能被给定的比较排序算法 A 输出。于是决策树 T_A 中有 $n!$ 个叶结点，标记为 $\frac{1}{n!}$ ，而其他叶结点，标记为 0。

令 $D(T)$ 表示给定决策树 T 的外部路径长度 (external path length)

即 $D(T)$ 是 T 中所有叶结点深度之和

假设决策树 T 有 $k > 1$ 个叶结点， L_T 和 R_T 分别表示决策树的左子树和右子树

则有 $D(T) = D(L_T) + D(R_T) + k$

证明过程有，令 $L(T)$ 表示决策树 T 的叶结点集合

$D_T(l)$ 表示决策树 T 中的叶结点 l 的深度

又 $L(L_T)$ 和 $L(R_T)$ 是集合 $L(T)$ 的一个划分， $|L(t)| > 1$

即决策树 T 的叶结点，或者属于左子树 L_T ，或者属于右子树 R_T

于是有 $D(T) = \sum_{l \in L(T)} D_T(l)$

$$= \sum_{l \in L(L_T)} D_T(l) + \sum_{l \in L(R_T)} D_T(l)$$

$$= \sum_{l \in L(L_T)} (D_{L_T}(l) + 1) + \sum_{l \in L(R_T)} (D_{R_T}(l) + 1)$$

$$= \sum_{l \in L(L_T)} D_{L_T}(l) + \sum_{l \in L(R_T)} D_{R_T}(l) + k$$

$$= D(L_T) + D(R_T) + k = D(T)$$

令 $d(k)$ 表示具有 $k > 1$ 个叶结点的决策树 T 中， $D(T)$ 的最小值

即有 $d(k) = \min \{ D(T) \mid |L(T)| = k > 1 \}$

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

证明过程有，对于任意具有 $k > 1$ 个叶结点的决策树 T

有 $D(T) = D(L_T) + D(R_T) + k$

且如果左子树 L_T 有 $1 \leq i \leq k-1$ 个叶结点，则

右子树 R_T 有 $k-i$ 个叶结点

再令对于具有 $k=1$ 个叶结点的决策树 T

即决策树 T 仅有根结点，于是有 $D(T) = 0$ ， $d(1) = 0$

于是 $d(k) = \min \{ D(T) \mid |L(T)| = k > 1 \}$

$= \min \{ D(L_T) + D(R_T) + k \mid |L(T)| = k > 1, L_T, R_T \text{ 为 } T \text{ 的左子树和右子树} \}$

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