Selected Solutions for Chapter 8: Sorting in Linear Time

Solution to Exercise 8.1-3

If the sort runs in linear time for m input permutations, then the height h of the portion of the decision tree consisting of the m corresponding leaves and their ancestors is linear.

Use the same argument as in the proof of Theorem 8.1 to show that this is impossible for $m = n!/2$, $n!/n$, or $n!/2^n$.

We have $2^h \geq m$, which gives us $h \geq \lg m$. For all the possible m's given here, $\lg m = \Omega(n \lg n)$, hence $h = \Omega(n \lg n)$.

In particular,

$$\begin{aligned}
\lg \frac{n!}{2} &= \lg n! - 1 \geq n \lg n - n \lg e - 1 \ , \\
\lg \frac{n!}{n} &= \lg n! - \lg n \geq n \lg n - n \lg e - \lg n \ , \\
\lg \frac{n!}{2^n} &= \lg n! - n \geq n \lg n - n \lg e - n \ .
\end{aligned}$$

Solution to Exercise 8.2-3

The following solution also answers Exercise 8.2-2.

Notice that the correctness argument in the text does not depend on the order in which A is processed. The algorithm is correct no matter what order is used!

But the modified algorithm is not stable. As before, in the final **for** loop an element equal to one taken from A earlier is placed before the earlier one (i.e., at a lower index position) in the output arrray B. The original algorithm was stable because an element taken from A later started out with a lower index than one taken earlier. But in the modified algorithm, an element taken from A later started out with a higher index than one taken earlier.

In particular, the algorithm still places the elements with value k in positions $C[k-1]+1$ through $C[k]$, but in the reverse order of their appearance in A.

Solution to Exercise 8.3-3

Basis: If $d = 1$, there's only one digit, so sorting on that digit sorts the array.

Inductive step: Assuming that radix sort works for $d - 1$ digits, we'll show that it works for d digits.

Radix sort sorts separately on each digit, starting from digit 1. Thus, radix sort of d digits, which sorts on digits $1, \ldots, d$ is equivalent to radix sort of the low-order $d - 1$ digits followed by a sort on digit d. By our induction hypothesis, the sort of the low-order $d - 1$ digits works, so just before the sort on digit d, the elements are in order according to their low-order $d - 1$ digits.

The sort on digit d will order the elements by their dth digit. Consider two elements, a and b, with dth digits a_d and b_d respectively.

- If $a_d < b_d$, the sort will put a before b, which is correct, since $a < b$ regardless of the low-order digits.
- If $a_d > b_d$, the sort will put a after b, which is correct, since $a > b$ regardless of the low-order digits.
- If $a_d = b_d$, the sort will leave a and b in the same order they were in, because it is stable. But that order is already correct, since the correct order of a and b is determined by the low-order $d - 1$ digits when their dth digits are equal, and the elements are already sorted by their low-order $d - 1$ digits.

If the intermediate sort were not stable, it might rearrange elements whose dth digits were equal—elements that *were* in the right order after the sort on their lower-order digits.

Solution to Exercise 8.3-4

Treat the numbers as 3-digit numbers in radix n. Each digit ranges from 0 to $n - 1$. Sort these 3-digit numbers with radix sort.

There are 3 calls to counting sort, each taking $\Theta(n + n) = \Theta(n)$ time, so that the total time is $\Theta(n)$.

Solution to Problem 8-1

a. For a comparison algorithm A to sort, no two input permutations can reach the same leaf of the decision tree, so there must be at least $n!$ leaves reached in T_A, one for each possible input permutation. Since A is a deterministic algorithm, it must always reach the same leaf when given a particular permutation as input, so at most $n!$ leaves are reached (one for each permutation). Therefore exactly $n!$ leaves are reached, one for each input permutation.

These $n!$ leaves will each have probability $1/n!$, since each of the $n!$ possible permutations is the input with the probability $1/n!$. Any remaining leaves will have probability 0, since they are not reached for any input.

Without loss of generality, we can assume for the rest of this problem that paths leading only to 0-probability leaves aren't in the tree, since they cannot affect the running time of the sort. That is, we can assume that T_A consists of only the $n!$ leaves labeled $1/n!$ and their ancestors.

b. If $k > 1$, then the root of T is not a leaf. This implies that all of T's leaves are leaves in LT and RT. Since every leaf at depth h in LT or RT has depth $h+1$ in T, $D(T)$ must be the sum of $D(LT)$, $D(RT)$, and k, the total number of leaves. To prove this last assertion, let $d_T(x) =$ depth of node x in tree T. Then,

$$\begin{aligned}
D(T) &= \sum_{x \in \text{leaves}(T)} d_T(x) \\
&= \sum_{x \in \text{leaves}(LT)} d_T(x) + \sum_{x \in \text{leaves}(RT)} d_T(x) \\
&= \sum_{x \in \text{leaves}(LT)} (d_{LT}(x) + 1) + \sum_{x \in \text{leaves}(RT)} (d_{RT}(x) + 1) \\
&= \sum_{x \in \text{leaves}(LT)} d_{LT}(x) + \sum_{x \in \text{leaves}(RT)} d_{RT}(x) + \sum_{x \in \text{leaves}(T)} 1 \\
&= D(LT) + D(RT) + k \ .
\end{aligned}$$

c. To show that $d(k) = \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$ we will show separately that

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

and

$$d(k) \ge \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\} \ .$$

- To show that $d(k) \le \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$, we need only show that $d(k) \le d(i) + d(k-i) + k$, for $i = 1, 2, \ldots, k-1$. For any i from 1 to $k-1$ we can find trees RT with i leaves and LT with $k-i$ leaves such that $D(RT) = d(i)$ and $D(LT) = d(k-i)$. Construct T such that RT and LT are the right and left subtrees of T's root respectively. Then

  $$\begin{aligned}
  d(k) &\le D(T) && \text{(by definition of } d \text{ as min } D(T) \text{ value)} \\
  &= D(RT) + D(LT) + k && \text{(by part (b))} \\
  &= d(i) + d(k-i) + k && \text{(by choice of } RT \text{ and } LT) \ .
  \end{aligned}$$

- To show that $d(k) \ge \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\}$, we need only show that $d(k) \ge d(i) + d(k-i) + k$, for some i in $\{1, 2, \ldots, k-1\}$. Take the tree T with k leaves such that $D(T) = d(k)$, let RT and LT be T's right and left subtree, respecitvely, and let i be the number of leaves in RT. Then $k-i$ is the number of leaves in LT and

  $$\begin{aligned}
  d(k) &= D(T) && \text{(by choice of } T) \\
  &= D(RT) + D(LT) + k && \text{(by part (b))} \\
  &\ge d(i) + d(k-i) + k && \text{(by defintion of } d \text{ as min } D(T) \text{ value)} \ .
  \end{aligned}$$

Neither i nor $k-i$ can be 0 (and hence $1 \le i \le k-1$), since if one of these were 0, either RT or LT would contain all k leaves of T, and that k-leaf subtree would have a D equal to $D(T)-k$ (by part (b)), contradicting the choice of T as the k-leaf tree with the minimum D.

d. Let $f_k(i) = i \lg i + (k-i)\lg(k-i)$. To find the value of i that minimizes f_k, find the i for which the derivative of f_k with respect to i is 0:

$$\begin{aligned} f_k'(i) &= \frac{d}{di}\left(\frac{i \ln i + (k-i)\ln(k-i)}{\ln 2}\right) \\ &= \frac{\ln i + 1 - \ln(k-i) - 1}{\ln 2} \\ &= \frac{\ln i - \ln(k-i)}{\ln 2} \end{aligned}$$

is 0 at $i = k/2$. To verify this is indeed a minimum (not a maximum), check that the second derivative of f_k is positive at $i = k/2$:

$$\begin{aligned} f_k''(i) &= \frac{d}{di}\left(\frac{\ln i - \ln(k-i)}{\ln 2}\right) \\ &= \frac{1}{\ln 2}\left(\frac{1}{i} + \frac{1}{k-i}\right) . \end{aligned}$$

$$\begin{aligned} f_k''(k/2) &= \frac{1}{\ln 2}\left(\frac{2}{k} + \frac{2}{k}\right) \\ &= \frac{1}{\ln 2} \cdot \frac{4}{k} \\ &> 0 \qquad\qquad \text{since } k > 1 . \end{aligned}$$

Now we use substitution to prove $d(k) = \Omega(k \lg k)$. The base case of the induction is satisfied because $d(1) \ge 0 = c \cdot 1 \cdot \lg 1$ for any constant c. For the inductive step we assume that $d(i) \ge ci \lg i$ for $1 \le i \le k-1$, where c is some constant to be determined.

$$\begin{aligned} d(k) &= \min_{1 \le i \le k-1} \{d(i) + d(k-i) + k\} \\ &\ge \min_{1 \le i \le k-1} \{c(i \lg i + (k-i)\lg(k-i)) + k\} \\ &= \min_{1 \le i \le k-1} \{c f_k(i) + k\} \\ &= c\left(\frac{k}{2}\lg\frac{k}{2}\left(k - \frac{k}{2}\right)\lg\left(k - \frac{k}{2}\right)\right) + k \\ &= ck \lg\left(\frac{k}{2}\right) + k \\ &= c(k \lg k - k) + k \\ &= ck \lg k + (k - ck) \\ &\ge ck \lg k \qquad \text{if } c \le 1 , \end{aligned}$$

and so $d(k) = \Omega(k \lg k)$.

e. Using the result of part (d) and the fact that T_A (as modified in our solution to part (a)) has $n!$ leaves, we can conclude that

$$D(T_A) \ge d(n!) = \Omega(n! \lg(n!)) .$$

$D(T_A)$ is the sum of the decision-tree path lengths for sorting all input permutations, and the path lengths are proportional to the run time. Since the $n!$ permutations have equal probability $1/n!$, the expected time to sort n random elements (1 input permutation) is the total time for all permutations divided by $n!$:

$$\frac{\Omega(n!\lg(n!))}{n!} = \Omega(\lg(n!)) = \Omega(n \lg n) \ .$$

f. We will show how to modify a randomized decision tree (algorithm) to define a deterministic decision tree (algorithm) that is at least as good as the randomized one in terms of the average number of comparisons.

At each randomized node, pick the child with the smallest subtree (the subtree with the smallest average number of comparisons on a path to a leaf). Delete all the other children of the randomized node and splice out the randomized node itself.

The deterministic algorithm corresponding to this modified tree still works, because the randomized algorithm worked no matter which path was taken from each randomized node.

The average number of comparisons for the modified algorithm is no larger than the average number for the original randomized tree, since we discarded the higher-average subtrees in each case. In particular, each time we splice out a randomized node, we leave the overall average less than or equal to what it was, because

- the same set of input permutations reaches the modified subtree as before, but those inputs are handled in less than or equal to average time than before, and
- the rest of the tree is unmodified.

The randomized algorithm thus takes at least as much time on average as the corresponding deterministic one. (We've shown that the expected running time for a deterministic comparison sort is $\Omega(n \lg n)$, hence the expected time for a randomized comparison sort is also $\Omega(n \lg n)$.)