searching for a value in the tree is one plus the number of nodes examined when the value was first inserted into the tree.

13.3-3
We can sort a given set of n numbers by first building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the numbers by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

13.3-4
Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

13.3-5
Suppose that another data structure contains a pointer to a node y in a binary search tree, and suppose that y's predecessor z is deleted from the tree by the procedure TREE-DELETE. What problem can arise? How can TREE-DELETE be rewritten to solve this problem?

13.3-6
Is the operation of deletion "commutative" in the sense that deleting x and then y from a binary search tree leaves the same tree as deleting y and then x? Argue why it is or give a counterexample.

13.3-7
When node z in TREE-DELETE has two children, we could splice out its predecessor rather than its successor. Some have argued that a fair strategy, giving equal priority to predecessor and successor, yields better empirical performance. How might TREE-DELETE be changed to implement such a fair strategy?

★ 13.4 Randomly built binary search trees

We have shown that all the basic operations on a binary search tree run in $O(h)$ time, where h is the height of the tree. The height of a binary search tree varies, however, as items are inserted and deleted. In order to analyze the behavior of binary search trees in practice, it is reasonable to make statistical assumptions about the distribution of keys and the sequence of insertions and deletions.

Unfortunately, little is known about the average height of a binary search tree when both insertion and deletion are used to create it. When the tree is created by insertion alone, the analysis becomes more tractable. Let us therefore define a ***randomly built binary search tree*** on n distinct keys as

one that arises from inserting the keys in random order into an initially empty tree, where each of the $n!$ permutations of the input keys is equally likely. (Exercise 13.4-2 asks you to show that this notion is different from assuming that every binary search tree on n keys is equally likely.) The goal of this section is to show that the expected height of a randomly built binary search tree on n keys is $O(\lg n)$.

We begin by investigating the structure of binary search trees that are built by insertion alone.

Lemma 13.3

Let T be the tree that results from inserting n distinct keys $k_1, k_2, \ldots, k_n$ (in order) into an initially empty binary search tree. Then k_i is an ancestor of k_j in T, for $1 \leq i < j \leq n$, if and only if

$$k_i = \min\{k_l : 1 \leq l \leq i \text{ and } k_l > k_j\}$$

or

$$k_i = \max\{k_l : 1 \leq l \leq i \text{ and } k_l < k_j\} .$$

Proof $\Rightarrow$: Suppose that k_i is an ancestor of k_j. Consider the tree T_i that results after the keys $k_1, k_2, \ldots, k_i$ have been inserted. The path in T_i from the root to k_i is the same as the path in T from the root to k_i. Thus, if k_j were inserted into T_i, it would become either the left or the right child of k_i. Consequently (see Exercise 13.2-6), k_i is either the smallest key among $k_1, k_2, \ldots, k_i$ that is larger than k_j or the largest key among $k_1, k_2, \ldots, k_i$ that is smaller than k_j.

$\Leftarrow$: Suppose that k_i is the smallest key among $k_1, k_2, \ldots, k_i$ that is larger than k_j. (The case in which k_i is the largest key among $k_1, k_2, \ldots, k_i$ that is smaller than k_j is handled symmetrically.) Comparing k_j to any of the keys on the path in T from the root to k_i yields the same results as comparing k_i to the keys. Hence, when k_j is inserted, it follows a path through k_i and is inserted as a descendant of k_i. ■

As a corollary of Lemma 13.3, we can precisely characterize the depth of a key based on the input permutation.

Corollary 13.4

Let T be the tree that results from inserting n distinct keys $k_1, k_2, \ldots, k_n$ (in order) into an initially empty binary search tree. For a given key k_j, where $1 \leq j \leq n$, define

$$G_j = \{k_i : 1 \leq i < j \text{ and } k_l > k_i > k_j \text{ for all } l < i \text{ such that } k_l > k_j\}$$

and

$$L_j = \{k_i : 1 \leq i < j \text{ and } k_l < k_i < k_j \text{ for all } l < i \text{ such that } k_l < k_j\} .$$

Then the keys on the path from the root to k_j are exactly the keys in $G_j \cup L_j$, and the depth in T of any key k_j is

$$d(k_j, T) = |G_j| + |L_j| \ . \qquad \blacksquare$$

Figure 13.5 illustrates the two sets G_j and L_j. The set G_j contains any key k_i inserted before k_j such that k_i is the smallest key among $k_1, k_2, \ldots, k_i$ that is larger than k_j. (The structure of L_j is symmetric.) To better understand the set G_j, let us explore a method by which we can enumerate its elements. Among the keys $k_1, k_2, \ldots, k_{j-1}$, consider in order those that are larger than k_j. These keys are shown as G'_j in the figure. As each key is considered in turn, keep a running account of the minimum. The set G_j consists of those elements that update the running minimum.

Let us simplify this scenario somewhat for the purpose of analysis. Suppose that n distinct numbers are inserted one at a time into a dynamic set. If all permutations of the numbers are equally likely, how many times on average does the minimum of the set change? To answer this question, suppose that the ith number inserted is k_i, for $i = 1, 2, \ldots, n$. The probability is $1/i$ that k_i is the minimum of the first i numbers, since the rank of k_i among the first i numbers is equally likely to be any of the i possible ranks. Consequently, the expected number of changes to the minimum of the set is

$$\sum_{i=1}^{n} \frac{1}{i} = H_n \ ,$$

where $H_n = \ln n + O(1)$ is the nth harmonic number (see equation (3.5) and Problem 6-2).

We therefore expect the number of changes to the minimum to be approximately $\ln n$, and the following lemma shows that the probability that it is much greater is very small.

Lemma 13.5

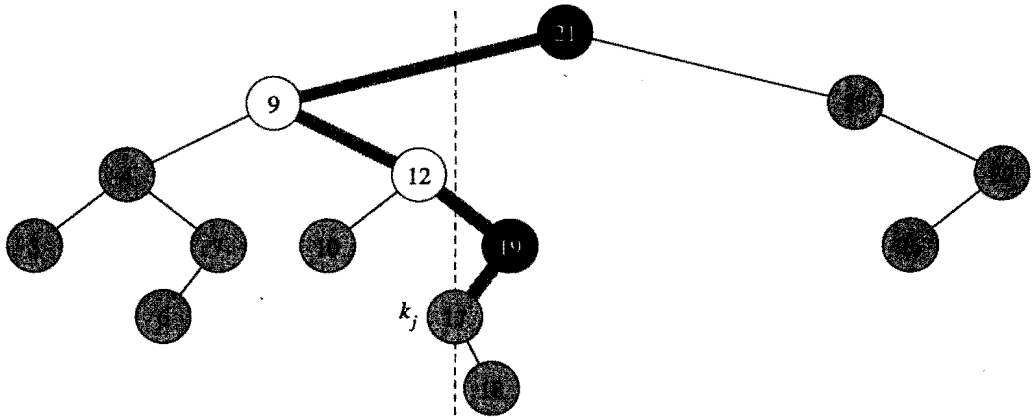
Let $k_1, k_2, \ldots, k_n$ be a random permutation of n distinct numbers, and let $|S|$ be the random variable that is the cardinality of the set

$$S = \{k_i : 1 \le i \le n \text{ and } k_l > k_i \text{ for all } l < i\} \ . \tag{13.1}$$

Then $\Pr\{|S| \ge (\beta + 1)H_n\} \le 1/n^2$, where H_n is the nth harmonic number and $\beta \approx 4.32$ satisfies the equation $(\ln \beta - 1)\beta = 2$.

Proof We can view the cardinality of the set S as being determined by n Bernoulli trials, where a success occurs in the ith trial when k_i is smaller than the elements $k_1, k_2, \ldots, k_{i-1}$. Success in the ith trial occurs with probability $1/i$. The trials are independent, since the probability that k_i is the minimum of $k_1, k_2, \ldots, k_i$ is independent of the relative ordering of $k_1, k_2, \ldots, k_{i-1}$.

We can use Theorem 6.6 to bound the probability that $|S| \ge (\beta + 1)H_n$. The expectation of $|S|$ is $\mu = H_n \ge \ln n$. Since $\beta > 1$, Theorem 6.6 yields



(a)

keys	21	9	4	25	7	12	3	10	19	29	17	6	26	18
G'_j	21			25					19	29				
G_j	21								19					
L'_j		9	4		7	12	3	10						
L_j		9				12								

(b)

Figure 13.5 Illustrating the two sets G_j and L_j that comprise the keys on a path from the root of a binary search tree to a key $k_j = 17$. **(a)** The nodes with keys in G_j are black, and the nodes with keys in L_j are white. All other nodes are shaded. The path from the root down to the node with key k_j is shaded. Keys to the left of the dashed line are less than k_j, and keys to the right are greater. The tree is constructed by inserting the keys shown in the topmost list in **(b)**. The set $G'_j = \{21, 25, 19, 29\}$ consists of those elements that are inserted before 17 and are greater than 17. The set $G_j = \{21, 19\}$ consists of those elements that update a running minimum of the elements in G'_j. Thus, the key 21 is in G_j, since it is the first element. The key 25 is not in G_j, since it is larger than the running minimum 21. The key 19 is in G_j, since it is smaller than the running minimum 21. The key 29 is not in G_j, since it is larger than the running minimum 19. The structures of L'_j and L_j are symmetric.

$$\begin{aligned}
\Pr\{|S| \geq (\beta+1)H_n\} &= \Pr\{|S| - \mu \geq \beta H_n\} \\
&\leq \left(\frac{eH_n}{\beta H_n}\right)^{\beta H_n} \\
&= e^{(1-\ln\beta)\beta H_n} \\
&\leq e^{-(\ln\beta-1)\beta \ln n} \\
&= n^{-(\ln\beta-1)\beta} \\
&= 1/n^2 ,
\end{aligned}$$

which follows from the definition of β. ■

We now have the tools to bound the height of a randomly built binary search tree.

Theorem 13.6
The average height of a randomly built binary search tree on n distinct keys is $O(\lg n)$.

Proof Let $k_1, k_2, \ldots, k_n$ be a random permutation on the n keys, and let T be the binary search tree that results from inserting the keys in order into an initially empty tree. We first consider the probability that the depth $d(k_j, T)$ of a given key k_j is at least t, for an arbitrary value t. By the characterization of $d(k_j, T)$ in Corollary 13.4, if the depth of k_j is at least t, then the cardinality of one of the two sets G_j and L_j must be at least $t/2$. Thus,

$$\Pr\{d(k_j, T) \geq t\} \leq \Pr\{|G_j| \geq t/2\} + \Pr\{|L_j| \geq t/2\} . \tag{13.2}$$

Let us examine $\Pr\{|G_j| \geq t/2\}$ first. We have

$$\begin{aligned}
&\Pr\{|G_j| \geq t/2\} \\
&\quad = \Pr\{|\{k_i : 1 \leq i < j \text{ and } k_l > k_i > k_j \text{ for all } l < i\}| \geq t/2\} \\
&\quad \leq \Pr\{|\{k_i : i \leq n \text{ and } k_l > k_i \text{ for all } l < i\}| \geq t/2\} \\
&\quad = \Pr\{|S| \geq t/2\} ,
\end{aligned}$$

where S is defined as in equation (13.1). To justify this argument, note that the probability does not decrease if we extend the range of i from $i < j$ to $i \leq n$, since more elements are added to the set. Likewise, the probability does not decrease if we remove the condition that $k_i > k_j$, since we are substituting a random permutation on possibly fewer than n elements (those k_i that are greater than k_j) for a random permutation on n elements.

Using a symmetric argument, we can prove that

$$\Pr\{|L_j| \geq t/2\} \leq \Pr\{|S| \geq t/2\} ,$$

and thus, by inequality (13.2), we obtain

$$\Pr\{d(k_j, T) \geq t\} \leq 2\Pr\{|S| \geq t/2\} .$$

If we choose $t = 2(\beta + 1)H_n$, where H_n is the nth harmonic number and $\beta \approx 4.32$ satisfies $(\ln \beta - 1)\beta = 2$, we can apply Lemma 13.5 to conclude that

$$\begin{aligned} \Pr\{d(k_j, T) \geq 2(\beta + 1)H_n\} &\leq 2\Pr\{|S| \geq (\beta + 1)H_n\} \\ &\leq 2/n^2 . \end{aligned}$$

Since there are at most n nodes in a randomly built binary search tree, the probability that *any* node's depth is at least $2(\beta + 1)H_n$ is therefore, by Boole's inequality (6.22), at most $n(2/n^2) = 2/n$. Thus, at least $1 - 2/n$ of the time, the height of a randomly built binary search tree is less than $2(\beta + 1)H_n$, and at most $2/n$ of the time, it is at most n. The expected height is therefore at most $(2(\beta + 1)H_n)(1 - 2/n) + n(2/n) = O(\lg n)$. ■

Exercises

13.4-1
Describe a binary search tree on n nodes such that the average depth of a node in the tree is $\Theta(\lg n)$ but the height of the tree is $\omega(\lg n)$. How large can the height of an n-node binary search tree be if the average depth of a node is $\Theta(\lg n)$?

13.4-2
Show that the notion of a randomly chosen binary search tree on n keys, where each binary search tree of n keys is equally likely to be chosen, is different from the notion of a randomly built binary search tree given in this section. (*Hint:* List the possibilities when $n = 3$.)

13.4-3 ⋆
Given a constant $r \geq 1$, determine t such that the probability is less than $1/n^r$ that the height of a randomly built binary search tree is at least tH_n.

13.4-4 ⋆
Consider RANDOMIZED-QUICKSORT operating on a sequence of n input numbers. Prove that for any constant $k > 0$, all but $O(1/n^k)$ of the $n!$ input permutations yield an $O(n \lg n)$ running time.

Problems

13-1 Binary search trees with equal keys
Equal keys pose a problem for the implementation of binary search trees.