Chapter 2

Random(ized) Search Trees

Keywords: binary search tree, random binary search tree, randomized search tree, quicksort, median search, quickselect, probability distribution, linearity of expectation, indicator variable, conditional probability, conditional expectation, tail estimate, Markov's Inequality, Jensen's Inequality, Chernoff type bounds, solving recurrences, specification by replacement system (grammar), harmonic number.

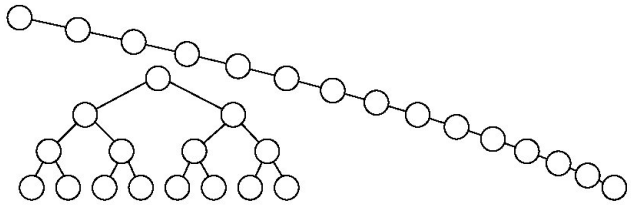


Figure 2.1: The Good and the Bad.

Random binary search trees, and their analysis, provide a good vehicle for recapitulating basic concepts and tools from probability theory, including conditional expectation and linearity of expectation, which will be vital ingredients all over this course. Moreover, search trees are crucial data structures in computer science.[1] We will see how the favorable expected behavior of random search trees (i.e. binary search trees obtained by inserting elements in random order without rebalancing) can be preserved, even if the elements are inserted in any "bad" order. Moreover, there will be a (re-)encounter with quicksort.

[1]Trees often lose against hashing, though, if simple searching, insertion and deletion is all that is needed. Most likely this is true in practice, and in theory it depends on the underlying model of computation. But see quote in beginning of Section 2.7.

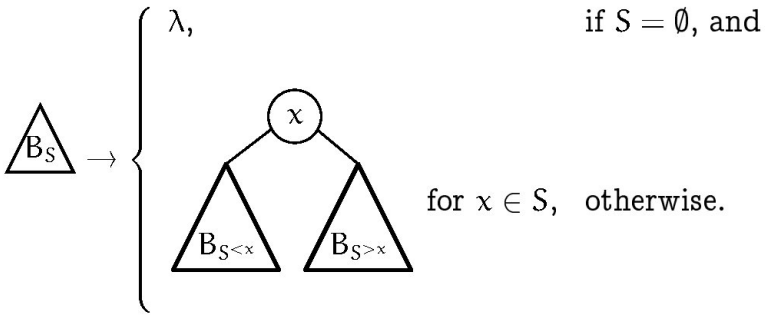
2.1 Definition

Before talking about *random* search trees, let us clarify what we mean (here) by a *search tree* for a given set S of n, $n \in \mathbf{N}_0$, distinct real numbers[2], the *keys*. These are binary ordered rooted trees, i.e. there is a distinguished node, called the root, and every node has at most two children, either a left and a right child, or one left or one right child, or no children at all; nodes without children are called *leaves*. The nodes are labeled by elements in S.

In order to specify this labeling, we define

$$\begin{aligned} S^{<x} &:= \{a \in S \mid a < x\} \text{ and} \\ S^{>x} &:= \{a \in S \mid a > x\}\text{, for } x \in \mathbf{R}. \end{aligned}$$

With λ denoting the empty tree, we let



We have specified here search trees for S by means of a replacement system (or grammar), similar to context free grammars[3], except that we are defining here trees instead of strings. Also, it is important to note that B_S does not stand for just one tree (unless $|S| \leq 1$) but for a whole family of trees, namely all trees that can be derived following the above recipe. We denote this family by $\mathcal{B}_S$. For example,

$$\mathcal{B}_\emptyset = \{\lambda\},\ \mathcal{B}_{\{1\}} = \{\textcircled{1}\},\ \mathcal{B}_{\{1,2\}} = \{\textcircled{1}^{\textcircled{2}},\ \textcircled{1}_{\textcircled{2}}, \ldots\}$$

(for $\mathcal{B}_{\{1,2,3\}}$ see Figure 2.2 below). Note that we skip the empty subtrees λ in the drawings of the trees, i.e. we display ① instead of 

.

[2] For the sake of concreteness we talk here about real numbers. All we need and exploit is that S is a totally ordered set, e.g. strings over some alphabet, or such like.

[3] In fact, we are dealing here with an *attributed grammar*, since the variables of our grammar are associated with an attribute, here the set S.

For searching a key x in such a tree, we traverse the tree starting at the root. If its key is x we are done; if x is less than the key in the root, we continue with the left subtree; otherwise, with the right subtree. If we ever enter an empty subtree, we know that x is not in the tree. The number of nodes traversed in the tree determines the cost of the search, which is 1 plus the depth of the searched for node:

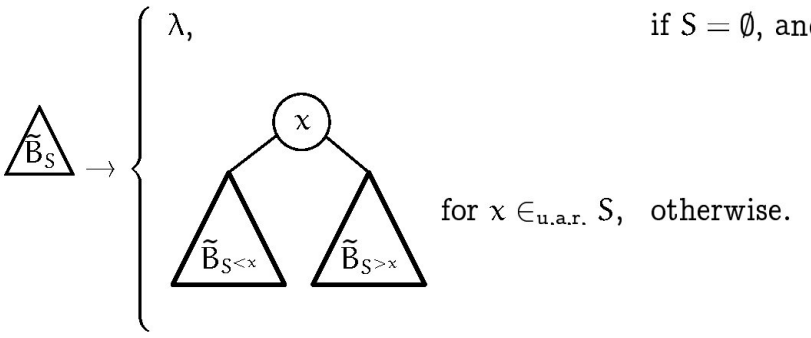
The *depth,* $d(v)$, of a node v is defined by

$$d(v) := \begin{cases} 0, & \text{if } v \text{ is the root, and} \\ 1 + d(u), u \text{ parent of } v, & \text{otherwise.} \end{cases}$$

The *height* of a tree is the maximum depth among its nodes. Figure 2.1 shows two trees with 15 nodes of height 3 and 14, respectively.

If we want to insert a new key in a binary search tree, then we first start a search for it. Either we find the key, when there is nothing to do. Or we end up in an empty subtree, when we replace it by a one-node tree holding the new key.

Random Search Trees. A slight twist in the definition of search trees leads to our specification of random search trees:



In this procedural definition the left and right subtree of the root are assumed to develop independently. The probability of a tree is defined as the probability that it is the result of the above randomized process—we obtain a probability distribution[4] on $\mathcal{B}_S$. For $|S| \leq 2$ this is the uniform distribution, but for $|S| \geq 3$ it is not uniform, see Figure 2.2.

[4]Recall that, given a countable probability space Ω, a *probability distribution* is a mapping $\Pr[]$ that assigns to every element $\omega \in \Omega$ a probability $\Pr[\omega] \in [0,1]$, so that $\sum_{\omega\in\Omega} \Pr[\omega] = 1$. Often, when the probability space is finite, we consider the *uniform distribution* where each element gets equal probability $\frac{1}{|\Omega|}$.

Lemma 2.1. $S \subseteq \mathbf{R}$, *finite. Given a tree in* $\mathcal{B}_S$, *we let* $w(v)$, v *a node, denote the number of nodes in the subtree rooted at* v.

The probability of the tree according to the above distribution is $\prod_v \frac{1}{w(v)}$, *where the product is over all nodes* v *of the tree.*

Proof. For S a finite set, let $\Pr_S[]$ be the probability distribution on $\mathcal{B}_S$ according to our procedural definition above. Then $\Pr_\emptyset[\lambda] = 1$. If the root of a non-empty tree T in $\mathcal{B}_S$ holds key x and has a left subtree T' and a right subtree T'', then $\Pr_S[T] = \frac{1}{|S|} \cdot \Pr_{S<x}[T'] \cdot \Pr_{S>x}[T'']$. The values suggested in the assertion of the observation satisfy this recurrence. ◻

Why do we consider this distribution, rather than any other, the uniform distribution, say? The reason is that this distribution is the same as the one obtained by inserting the keys in S into an initially empty tree in random order drawn u.a.r. from all permutations. For us the above definition comes more handy.

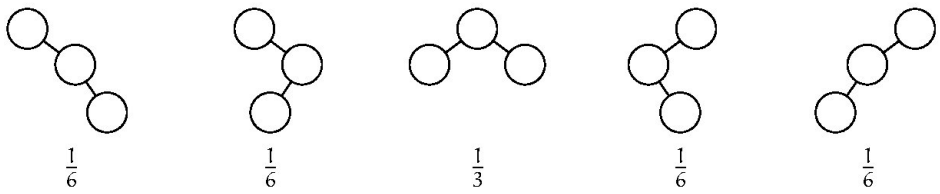


Figure 2.2: The trees in $\mathcal{B}_{\{1,2,3\}}$, keys omitted, with their probabilities according to our definition of random search trees.

Exercise 2.1. As Balanced as it Goes

Let us define a balanced search tree *for keys* S *as a search tree in* $\mathcal{B}_S$ *where for every node in the tree, the number of nodes in the left and right subtree differ by* 1 *at most.*

Provide a replacement system that defines all balanced search trees for a given key set S.

Exercise 2.2. *Consider the following attributed grammars (the only variable is* A *attributed by elements in* $\mathbf{N}_0$, *the only terminal symbol is* a*):*

$$A(n) \to \begin{cases} a, & \text{if } n = 0, \text{ and} \\ A(n-1)\,A(n-1), & \text{otherwise.} \end{cases}$$

Figure 2.3: Grammars all over—an example from architecture, provided by [Havemann, Fellner, 2004].

$$A(n) \to \begin{cases} \lambda \text{ (the empty string)}, & \text{if } n = 0, \\ a, & \text{if } n = 1, \text{ and} \\ A(n-1)\,A(n-2), & \text{otherwise.} \end{cases}$$

For example, in the second grammar, we derive

$$A(3) \Rightarrow A(2)A(1) \Rightarrow A(1)A(0)A(1) \overset{3\,\text{steps}}{\Rightarrow} aa = a^2 \,.$$

For both grammars characterize the word (string) generated by $A(n)$ *for* $n \in \mathbf{N}_0$.

Exercise 2.3. *Consider the attributed grammar with* P *the only variable, terminal symbols* $\langle$ *and* $\rangle$ *and with rules* $P^{(0)} \to \lambda$ *(the empty string)*

and

$$P^{(n)} \to P^{(0)}\langle P^{(n-1)}\rangle \mid P^{(1)}\langle P^{(n-2)}\rangle \mid \ldots \mid P^{(n-1)}\langle P^{(0)}\rangle \qquad \text{for } n \in \mathbf{N};$$

('|' here separates alternative rules in the grammar).

What are the strings that can be derived from $P(n)$, $n \in \mathbf{N}$*? How many such strings are there?*

Exercise 2.4. Leftist Tendency

$n \in \mathbf{N}$, $n \geq 2$. *We choose* $\{A, B\} \in_{\text{u.a.r.}} \binom{[n]}{2}$, *and then set* $C := \min\{A, B\}$.

(1) For $i \in [n]$, *determine* $\Pr[C = i]$.

(2) Determine $\mathbf{E}[C \cdot (n - C + 1)]$ *and compare with* $\mathbf{E}[X \cdot (n - X + 1)]$ *for* $X \in_{\text{u.a.r.}} [n]$.

Exercise 2.5. Trees by Random Insertion

$S \subseteq \mathbf{R}$, *finite. Show that if elements in* S *are inserted in an initially empty binary search tree in random order—u.a.r. from all permutations of* S*—then the resulting distribution on* $\mathcal{B}_S$ *is the same as for random search trees as we defined it.*

Exercise 2.6. Uniformly Centered Triples

$n \in \mathbf{N}, n \geq 3$. *We choose a random triple* ABC *of numbers in* $[n]$ *as follows. First choose* $B \in_{u.a.r.} \{2..n-1\}$ *and then* $A \in_{u.a.r.} [B-1]$ *and* $C \in_{u.a.r.} \{B+1..n\}$. *(That is, we have* $1 \leq A < B < C \leq n$.*)*

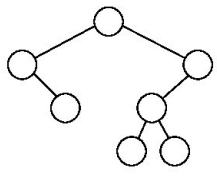
(1) Given integers a, b, *and* c *with* $1 \leq a < b < c \leq n$, *what is the probability* $\Pr[ABC = abc]$*?*

(2) Is $\{A, B, C\}$ *uniformly distributed in* $\binom{[n]}{3}$*?*

(3) Determine $\mathbf{E}[A]$, $\mathbf{E}[B]$, *and* $\mathbf{E}[C]$.

Exercise 2.7. A Random Tree? How random?

Determine the probability of the following tree with 7 nodes.



What is the smallest, what is the largest possible probability of a tree with 7 nodes?

Exercise 2.8. Possible Heights

$n \in \mathbf{N}$. *What are the possible heights of a tree in* $\mathcal{B}_{[n]}$*?*

Exercise 2.9. Very Deep Nodes

$n \in \mathbf{N}$. *Show that the expected number of nodes of depth* $n-1$ *in a random search tree for* n *keys is* $\frac{2^{n-1}}{n!}$. *What is the probability that there is a node of depth* $n-1$*?*

Exercise 2.10. High Trees

$n \in \mathbf{N}$. *Determine the number of trees of height* $n-2$ *in* $\mathcal{B}_{[n]}$.

Exercise 2.11. Catalan Number

$n \in \mathbf{N}$. *Show that for* $|S| = n$,

$$|\mathcal{B}_S| = \frac{1}{n+1}\binom{2n}{n}$$

(the n*th* Catalan number*).*

2.2 Overall (and Average) Depth of Keys

Given some finite set $S \subseteq \mathbf{R}$, the rank of $x \in \mathbf{R}$ in S is

$$\mathrm{rk}(x) = \mathrm{rk}_S(x) := 1 + |\{y \in S \mid y < x\}| \,.$$

For example, the smallest element of S has rank 1. In general, if $x \in S$, then x is the $\mathrm{rk}(x)$-smallest element in S. For a tree in $\mathcal{B}_S$, we abuse notation by writing $\mathrm{rk}(v)$ short for the rank of v's key in S.

For $i, n \in \mathbf{N}$, $i \leq n$, we denote by $D_n^{(i)}$ the random variable for the depth of the key of rank i in a random search tree for n keys. Our goal is to investigate the expected overall depth $\mathbf{E}\left[\sum_{i=1}^{n} D_n^{(i)}\right]$ in a random search tree; one nth of this quantity is the expected average depth of the nodes in a random search tree.

Depth of Smallest Key. For a warm-up we first analyze $\mathbf{E}\left[D_n^{(1)}\right]$. Let us write D_n short for $D_n^{(1)}$. We easily get $\mathbf{E}[D_1] = 0$, $\mathbf{E}[D_2] = \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 0 = \frac{1}{2}$, and inspection of Figure 2.2 yields

$$\mathbf{E}[D_3] = \tfrac{1}{6}\cdot 0 + \tfrac{1}{6}\cdot 0 + \tfrac{1}{3}\cdot 1 + \tfrac{1}{6}\cdot 1 + \tfrac{1}{6}\cdot 2 = \tfrac{5}{6} \,.$$

For the general case $n \in \mathbf{N}$, we can discriminate between the possible ranks of the root key and get

$$\mathbf{E}[D_n] = \sum_{i=1}^{n} \underbrace{\mathbf{E}[D_n \,|\, \mathrm{rk}(\mathrm{root}) = i]}_{= \begin{cases} 0, & \text{if } i = 1, \text{ and} \\ 1 + \mathbf{E}[D_{i-1}], & \text{otherwise.} \end{cases}} \cdot \underbrace{\Pr[\mathrm{rk}(\mathrm{root}) = i]}_{=1/n} \,.$$

Here we use the observation, that the depth of the smallest key is 0 if it sits in the root, or it is 1 plus the depth of the smallest key within the left subtree.

Writing d_n for $\mathbf{E}[D_n]$, $n \in \mathbf{N}$, we get the recurrence

$$d_n = \begin{cases} 0, & \text{if } n = 1, \text{ and} \\ \frac{1}{n}\sum_{i=2}^{n}(1 + d_{i-1}), & \text{otherwise.} \end{cases}$$

For $n \in \mathbf{N}$, $n \geq 3$, we have

$$\begin{aligned} n d_n &= (n-1) + d_1 + d_2 + \ldots + d_{n-2} + d_{n-1}, \quad \text{and} \\ (n-1) d_{n-1} &= (n-2) + d_1 + d_2 + \ldots + d_{n-2} \end{aligned}$$

and thus by subtracting the two identities

$$\begin{aligned} & n d_n - (n-1) d_{n-1} = 1 + d_{n-1} \\ \Leftrightarrow \quad & n d_n = 1 + n d_{n-1} \\ \Leftrightarrow \quad & d_n = \frac{1}{n} + d_{n-1} \quad \text{for } n \geq 3. \end{aligned} \tag{2.1}$$

Together with $d_1 = 0$ and $d_2 = \frac{1}{2}$, successive invocation of (2.1) yields

$$\begin{aligned} d_n &= \frac{1}{n} + d_{n-1} = \frac{1}{n} + \frac{1}{n-1} + d_{n-2} = \cdots \\ &= \frac{1}{n} + \frac{1}{n-1} + \ldots + \frac{1}{3} + \underbrace{d_2}_{1/2} = H_n - 1 \end{aligned}$$

where

$$H_n := \sum_{i=1}^{n} \frac{1}{i}, \text{ for } n \in \mathbf{N}_0, \text{ is the } n\textit{th harmonic number.}$$

The harmonic numbers are considered the discrete analogue of the natural logarithm and they are omnipresent in discrete probabilistic analysis. We have an upper bound of

$$H_n \leq 1 + \int_1^n \frac{1}{x}\,dx = 1 + (\ln n - \ln 1) = 1 + \ln n \quad \text{for } n \geq 1.$$

Also $\ln(n+1) \leq H_n$ for all $n \in \mathbf{N}_0$, and $\lim_{n\to\infty}(H_n - \ln n) =: \gamma = 0.57721\ldots$, Euler's constant.

Knowing the expected depth of the smallest key, we might wonder next what the chances are that this depth is large (so-called *tail estimates*). We can always apply Markov's Inequality[5]: Here we have

$$\Pr[D_n \geq \lambda \ln n] \leq \Pr[D_n \geq \lambda(H_n - 1)] \leq \frac{1}{\lambda} \text{ for } \lambda \in \mathbb{R}^+.$$

Better tail estimates will follow.

Overall Depth. For $n \in \mathbb{N}_0$, let $X_n := \sum_{i=1}^{n} D_n^{(i)}$, the random variable for the *overall depth* of all nodes in a random search tree for n keys. We have $X_0 = 0$, $X_1 = 0$, and $X_2 = 1$, always. The complete list of search trees for 3 keys lets us calculate $\mathbf{E}[X_3] = 4 \cdot \frac{1}{6} \cdot (0+1+2) + \frac{1}{3} \cdot (0+1+1) = \frac{8}{3}$.

For general $n \geq 1$,

$$\begin{aligned} \mathbf{E}[X_n] &= \sum_{i=1}^{n} \underbrace{\mathbf{E}[X_n \,|\, \mathrm{rk}(\mathrm{root}) = i]}_{n-1+\mathbf{E}[X_{i-1}]+\mathbf{E}[X_{n-i}]} \cdot \underbrace{\Pr[\mathrm{rk}(\mathrm{root}) = i]}_{=1/n} \\ &= n - 1 + \frac{1}{n} \cdot 2 \cdot \sum_{i=1}^{n} \mathbf{E}[X_{i-1}] \,, \end{aligned}$$

since every node in the two subtrees attached to the root—altogether there are $n-1$ such nodes—increases its depth by 1, while there is no contribution from the root. With $x_n := \mathbf{E}[X_n]$, $n \in \mathbb{N}_0$,

$$x_n = \begin{cases} 0, & \text{if } n = 0, \text{ and} \\ n - 1 + \frac{2}{n} \sum_{i=0}^{n-1} x_i, & \text{otherwise.} \end{cases}$$

For $n \geq 2$, we get

$$\begin{aligned} n x_n &= n(n-1) + 2(x_0 + x_1 + \ldots + x_{n-2} + x_{n-1}), \quad \text{and} \\ (n-1)x_{n-1} &= (n-1)(n-2) + 2(x_0 + x_1 + \ldots + x_{n-2}) \end{aligned}$$

[5] Markov's Inequality: If X is a non-negative real random variable with finite expectation, then $\Pr[X \geq \lambda \mathbf{E}[X]] \leq \frac{1}{\lambda}$ for all $\lambda \in \mathbb{R}^+$. Or, equivalently, $\Pr[X \geq t] \leq \frac{\mathbf{E}[X]}{t}$ for all $t \in \mathbb{R}^+$.

and by subtraction

$$\begin{aligned}
& nx_n - (n-1)x_{n-1} = 2(n-1) + 2x_{n-1} \\
\Leftrightarrow\quad & nx_n = 2(n-1) + (n+1)x_{n-1} \qquad \text{now multiply by } \tfrac{1}{2n(n+1)} \\
\Leftrightarrow\quad & \underbrace{\frac{x_n}{2(n+1)}}_{=:f_n} = \frac{(n-1)}{n(n+1)} + \underbrace{\frac{x_{n-1}}{2n}}_{=f_{n-1}} \\
\Leftrightarrow\quad & f_n = \frac{1}{n+1} - \frac{1}{n(n+1)} + f_{n-1}, \quad \text{for } n \geq 2.
\end{aligned}$$

Therefore

$$f_n = \left(\frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{3}\right) - \left(\frac{1}{n(n+1)} + \frac{1}{(n-1)n} + \ldots + \frac{1}{2\cdot 3}\right) + \underbrace{f_1}_{=0}$$

where the first sum is $H_{n+1} - \frac{3}{2}$, and the second[6] sum can be turned into a telescoping sum (via the expansion $\frac{1}{n(n+1)} = \frac{1}{n} - \frac{1}{n+1}$ into partial fractions):

$$\sum_{i=2}^{n} \frac{1}{i(i+1)} = \sum_{i=2}^{n} \left(\frac{1}{i} - \frac{1}{i+1}\right) = \frac{1}{2} - \frac{1}{n+1} .$$

Now, for $n \geq 2$, $f_n = H_{n+1} - 2 + \frac{1}{n+1} = H_n - 2 + \frac{2}{n+1}$ and

$$x_n = 2(n+1)f_n = 2(n+1)H_n - 4n ,$$

and we see that the formula extends also to $n = 0, 1$.

Theorem 2.2. $n \in \mathbb{N}_0$. *The expected overall depth of a random search tree for n keys is $2(n+1)H_n - 4n = 2n \ln n + O(n)$.*

Exercise 2.12. Harmonic Lower Bound

Show that $\ln(n+1) \leq H_n$ for all $n \in \mathbb{N}_0$.

Exercise 2.13. $n \in \mathbb{N}_0$. *Determine the sum $\sum_{i=1}^{n} \frac{1}{i(i+2)}$.*

[6] A quick estimate observes that this second sum is upper bounded by $\sum_{i=1}^{\infty} \frac{1}{i^2} = O(1)$. Recall that $\sum_{i=1}^{\infty} \frac{1}{i^\alpha} = O(1)$ for each $\alpha \in \mathbb{R}$, $\alpha > 1$, while $\sum_{i=1}^{\infty} \frac{1}{i} = \infty$.

Exercise 2.14. Solving Recurrences

Determine closed forms for the following recursively defined series:

(1) *For* $n \in \mathbf{N}$,

$$a_n = \begin{cases} 1, & \text{if } n = 1, \text{ and} \\ 1 + \frac{1}{n}\sum_{i=1}^{n-1} a_i, & \text{otherwise.} \end{cases}$$

(2) *For* $n \in \mathbf{N}$,

$$b_n = \begin{cases} 1, & \text{if } n = 1, \text{ and} \\ 2 + \sum_{i=1}^{n-1} b_i, & \text{otherwise.} \end{cases}$$

(3) *For* $n \in \mathbf{N}_0$,

$$c_n = \begin{cases} 0, & \text{if } n = 0, \text{ and} \\ n - 1 + \sum_{i=1}^{n} \frac{c_{i-1}+c_{n-i}}{2}, & \text{otherwise.} \end{cases}$$

(4) *For* $n \in \mathbf{N}_0$,

$$d_n = \begin{cases} 0, & \text{if } n = 0, \text{ and} \\ 1 + 2\sum_{i=0}^{n-1} (-1)^{n-i} d_i, & \text{otherwise.} \end{cases}$$

(5) *For* $n \in \mathbf{N}_0$,

$$e_n = \begin{cases} 1, & \text{if } n = 0, \text{ and} \\ 1 + n\, e_{n-1}, & \text{otherwise.} \end{cases}$$

Exercise 2.15. Number of Leaves

$n \in \mathbf{N}$. *Determine the expected number of leaves in a random search tree for* n *keys.*

Exercise 2.16. Depths of Leaves

$n \in \mathbf{N}_0$. *Determine the expected overall depth of all leaves in a random search tree for* n *keys, i.e. the expectation of the random variable* $Y_n := \sum_{i=1,\, i \text{ is leaf}}^{n} D_n^{(i)}$.

Exercise 2.17. Random Decline

$n \in \mathbf{N}$. *We consider the following random process: First we choose a number* $k_1 \in_{\text{u.a.r.}} [n]$, *then a number* $k_2 \in_{\text{u.a.r.}} [k_1 - 1]$, ... *In general, we choose* $k_{i+1} \in_{\text{u.a.r.}} [k_i - 1]$ *until we have reached* $k_N = 1$.

(1) *Determine* $\mathbf{E}[N]$ *(in terms of* n*), i.e. the expected number of numbers chosen altogether.*

(2) *Determine* $\mathbf{E}[k_1 + k_2 + \ldots + k_N]$.

Exercise 2.18. Biased Random Walk to a Leaf

$n \in \mathbf{N}$. *In a random search tree for n keys we start in the root and perform a random walk to a leaf as follows: If the current node is a leaf, we stop. Otherwise, if the left subtree and right subtree of the current node contain i and j nodes, respectively, then we proceed to the left child with probability $\frac{i}{i+j}$, to the right with probability $\frac{j}{i+j}$.*

Determine the expected depth of the leaf reached (which is the length of the random walk taken).

Exercise 2.19. Leftist Root, Rightist Tree

Here is an alternative probability distribution for $\mathcal{B}_S$, $S \subseteq \mathbf{R}$, finite. If $|S| \leq 1$, we choose the unique tree in $\mathcal{B}_S$. Otherwise we choose a root with key x, where $x = \min\{a, b\}$ for $\{a, b\} \in_{\text{u.a.r.}} \binom{S}{2}$ and produce recursively a left subtree for $S^{<x}$ and a right subtree for $S^{>x}$. We call a tree drawn from this distribution a rightist random search tree for S.

(1) *Determine the resulting distribution for $\mathcal{B}_{\{1,2\}}$ and $\mathcal{B}_{\{1,2,3\}}$ (i.e. calculate for each tree its probability).*

(2) *Determine the expected depth of the largest key in a rightist random search tree for n keys.*

(3) *Determine the expected overall depth of a rightist random search tree for n keys.*

Exercise 2.20. Rightist Search Tree Distribution

Analogous to Lemma 2.1, determine a formula for the probability of a tree in $\mathcal{B}_S$, $S \subseteq \mathbf{R}$ finite, to occur as a rightist random search tree.

Exercise 2.21. Rightist Trees via Random Insertions

Recall the definition of rightist random search trees. Now specify an insertion procedure for binary search trees, so that the search trees obtained from insertions in random order (u.a.r. from all permutations) yields the same distribution as the one for rightist random search trees. (Still insertion of an element should be doable in time linear in the depth of its final position.)

Exercise 2.22. Shifted Markov

$B \in \mathbf{R}$. *Show that if X is a real random variable with $X \geq B$ and $\mathbf{E}[X]$ finite, then*

$$\Pr[X \geq t] \leq \frac{\mathbf{E}[X] - B}{t - B} \quad \text{for } t \in \mathbf{R},\ t > B.$$

HINT: Apply Markov's Inequality to $X - B$.

2.3 Expected Height

We wish to analyze the height of a random search tree for n keys. In other words, we study the random variable $X_n := \max_{i=1}^{n} D_n^{(i)}$; recall that $D_n^{(i)}$ is the depth of the key of rank i. We recall that every binary tree with n nodes has height at least $\lfloor \log n \rfloor$, so this is definitely also a lower bound for the expected height. Note, on the one hand, that the expected depth of $\ln n + O(1)$ we calculated for the node of rank 1 is smaller than $\log n$ ($\ln n = \frac{\log n}{\log e} = 0.693\ldots \log n$). On the other hand, we know that the expected average depth is $2 \ln n + O(1) = 1.386\ldots \log n$ which is a better lower bound for the expected height (the maximum of numbers is at least their average). When it gets to bound $\mathsf{E}[X_n]$ from above, knowing the expected average depth is of no help (see Exercise 2.24).

Determining $\mathsf{E}[X_n]$ seems to be hard because of the max-operator involved. Instead, we use

$$\mathsf{E}[X_n] \le \log \mathsf{E}\left[2^{X_n}\right] = \log \mathsf{E}\left[2^{\max_{i=1}^{n} D_n^{(i)}}\right] \le \log \mathsf{E}\left[\sum_{i=1,\, i \text{ is leaf}}^{n} 2^{D_n^{(i)}}\right], \qquad (2.2)$$

(the first inequality follows from Jensen's Inequality[7]) and analyze $Z_n := \sum_{i=1,\, i \text{ is leaf}}^{n} 2^{D_n^{(i)}}$ instead.

$Z_0 = 0$, $Z_1 = 1$, and $Z_2 = 2$, always. Once more we consult Figure 2.2 for $\mathsf{E}[Z_3] = 4$. Enough of fiddling around with small values. For $n \ge 2$,

$$\mathsf{E}[Z_n] = \sum_{i=1}^{n} \underbrace{\mathsf{E}[Z_n \,|\, \mathrm{rk}(\mathrm{root}) = i]}_{2(\mathsf{E}Z_{i-1} + \mathsf{E}Z_{n-i})} \cdot \underbrace{\Pr[\mathrm{rk}(\mathrm{root}) = i]}_{=1/n}.$$

Put $z_n := \mathsf{E}[Z_n]$.

$$z_n = \begin{cases} 0, & \text{if } n = 0, \\ 1, & \text{if } n = 1, \text{ and} \\ \frac{4}{n} \sum_{i=1}^{n} z_{i-1}, & \text{otherwise.} \end{cases} \qquad (2.3)$$

[7] Jensen's Inequality: If $f : \mathsf{R} \to \mathsf{R}$ is a convex function, then $f(\mathsf{E}[X]) \le \mathsf{E}[f(X)]$. A function f is convex, if for any x and y and $0 \le \lambda \le 1$, it holds that $f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y)$ (like this: ⌒). Examples are $f(x) = x^2$ and $f(x) = 2^x$. Applying Jensen's Inequality to the first shows $\mathsf{E}[X]^2 \le \mathsf{E}[X^2]$ (which is equivalent to the non-negativity of the variance of a random variable).

Consequently, for $n \geq 3$, $nz_n - (n-1)z_{n-1} = 4z_{n-1}$, and so $nz_n = (n+3)z_{n-1}$ or (multiply by $\frac{1}{n(n+1)(n+2)(n+3)}$)

$$\frac{z_n}{(n+3)(n+2)(n+1)} = \frac{z_{n-1}}{(n+2)(n+1)n} = \cdots = \frac{z_2}{5 \cdot 4 \cdot 3} = \frac{1}{30} .$$

If we plug $\mathbf{E}[Z_n] = \frac{(n+3)(n+2)(n+1)}{30}$ into (2.2), then we get an upper bound of $3 \log n + O(1) = 4.328\ldots \ln n + O(1)$ for the expected height of a random search tree. So an asymptotic logarithmic upper bound is established. But what about the leading constant?

If we go back to (2.2) we see that there is still the base '2' to play with.

Hunting for the Constant. For $C \in \mathbf{R}$, $C > 1$, we redefine $Z_n := \sum_{i=1,\, i \text{ is leaf}}^{n} C^{D_n^{(i)}}$. Similar to (2.2), we have $\mathbf{E}[X_n] \leq \log_C \mathbf{E}[Z_n]$. The recurrence for $z_n := \mathbf{E}[Z_n]$ is the same as the one given in (2.3), except that '4' gets replaced by $2C$, which eventually leads to $nz_n = (n + 2C - 1)z_{n-1}$ for $n \geq 3$. This yields

$$z_n = (1 + \frac{2C-1}{n})z_{n-1} = (1 + \frac{2C-1}{n})(1 + \frac{2C-1}{n-1}) \cdots (1 + \frac{2C-1}{3})z_2 .$$

Since $z_2 = C < (1 + \frac{2C-1}{2})$, and $1 + x \leq e^x$ for $x \in \mathbf{R}$, we obtain

$$z_n < e^{(2C-1)\sum_{i=2}^{n} \frac{1}{i}} = e^{(2C-1)(H_n - 1)} < e^{(2C-1)\ln n} = n^{2C-1} \qquad (2.4)$$

for $n \geq 3$. Invoking (2.2) (in its adopted version with 'C' instead of '2'), this gives

$$\mathbf{E}[X_n] < \frac{2C-1}{\ln C} \ln n \quad \text{for } n \geq 3 \text{ and any real } C > 1.$$

The bound attains its local extrema if $2 \ln C - 2 + \frac{1}{C} = 0$ (obtained by setting the first derivative to 0), or, equivalently, if $(\frac{e}{C})^{2C} = e$. Note that for these values of C, we have $\frac{2C-1}{\ln C} = 2C$. We write c for $2C$ to get the following theorem.

Theorem 2.3. *The expected height of a random search tree for n keys is upper bounded by $c \ln n$, where $c = 4.311\ldots$ is the unique value greater than 2 which satisfies $(\frac{2e}{c})^c = e$.*

Surprisingly, the constant in the leading term is already tight, see note below.

Tail Estimates. Let us conclude by pointing out that knowing a good estimate for $\mathbf{E}\left[C^{X_n}\right]$ immediately gives a good tail estimate for X_n via Markov's inequality, namely

$$\Pr[X_n \geq \tau \ln n] = \Pr\left[C^{X_n} \geq C^{\tau \ln n}\right] \leq \frac{\mathbf{E}\left[C^{X_n}\right]}{C^{\tau \ln n}} \leq n^{2C-1-\tau \ln C} .$$

$2C - 1 - \tau \ln C$ is minimized for $C = \tau/2$.

Theorem 2.4. $n \in \mathbf{N}$, $\tau \in \mathbf{R}^+$.

$$\Pr[X_n \geq \tau \ln n] \leq n^{\tau(1-\ln(\tau/2))-1} ;$$

in particular $\Pr[X_n \geq 2e \ln n] \leq \frac{1}{n}$, *(*$2e = 5.435\ldots$*).*

NOTE 2.1 The upper bound in Theorem 2.3 was first established in [Robson, 1982]. The leading constant was shown to be tight in [Devroye, 1986] by an argument that is considerably more involved than the upper bound proof we have just seen.

Meanwhile, the expected height of a random binary search tree for n keys is known to an amazing extent, [Reed, 2003], [Drmota, 2003]: It is

$$c \ln n - c' \ln \ln n + O(1)$$

for constants $c = 4.311\ldots$ and $c' = \frac{3c}{2(c-1)} = 1.953\ldots$. The variance is $O(1)$.

Exercise 2.23. Depth of Smallest Key Revisited

$n \in \mathbf{N}$. *Determine* $\mathbf{E}\left[2^{D_n^{(1)}}\right]$. *Use the result for a tail estimate for* $D_n^{(1)}$.

Exercise 2.24. Maximum Expectation vs. Expected Maximum

$n \in \mathbf{N}$.

(1) *Define random variables* X_i *with* $\mathbf{E}[X_i] = O(1)$ *for* $i \in [n]$ *and* $\mathbf{E}[\max_{i=1}^n X_i] \geq n$.

(2) *Define* n *mutually independent random variables* X_i *with* $\mathbf{E}[X_i] = O(1)$ *for* $i \in [n]$ *and* $\mathbf{E}[\max_{i=1}^n X_i] \geq n$.

2.4 Expected Depth of Individual Keys

We want to analyze $d_{i,n} := \mathbf{E}\left[D_n^{(i)}\right]$, the expected depth of the node holding the key of rank i in a random search tree for n keys. We could go along the by now usual path—setting up a two parameter recurrence for the $d_{i,n}$'s and solve it—, but that turns out to be tedious. So we change tools and employ indicator variables and linearity of expectation[8].

$n \in \mathbf{N}$. For $i \in [n]$ let us use 'node i' short for the node holding the key of rank i in a random search tree for n keys. Moreover, let us recall that a node u is an *ancestor* of node v in a rooted tree if u lies on the unique path from v to the root on the tree (including v). For $i, j \in [n]$, we introduce the indicator variable

$$A_i^j := [\text{node } j \text{ is ancestor of node } i]$$

(in a random search tree for n keys), i.e.

$$A_i^j = \begin{cases} 1, & \text{if node } j \text{ is ancestor of node } i, \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

We have $D_n^{(i)} = \sum_{j=1,\, j\neq i}^{n} A_i^j$ and, by linearity of expectation,

$$\mathbf{E}\left[D_n^{(i)}\right] = \sum_{j=1,\, j\neq i}^{n} \mathbf{E}\left[A_i^j\right] \tag{2.5}$$

Since

$$\mathbf{E}\left[A_i^j\right] = \Pr\left[A_i^j = 1\right] = \Pr\left[\text{node } j \text{ is ancestor of node } i\right],$$

all that is left to do is to determine these probabilities for the ancestor relations.

Lemma 2.5. $i, j \in \mathbf{N}$. *In a random search tree for* $n \geq \max\{i, j\}$ *keys*

$$\Pr\left[\textit{node } j \textit{ is ancestor of node } i\right] = \frac{1}{|i-j|+1}.$$

[8] Linearity of Expectation: If X and Y are random variables then $\mathbf{E}[\lambda X] = \lambda\,\mathbf{E}[X]$ for all $\lambda \in \mathbf{R}$ (provided $\mathbf{E}[X]$ is finite, and $\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y]$, without any assumption about independence of X and Y).

Proof. If $i = j$, the claim is trivially true. We assume $i < j$, the alternative being symmetric.

Note that if $n = j - i + 1$, then $i = 1$ and $j = n$. As we choose the key for the root of the tree, we see that the only chance for the key of rank j (now the largest key) to be an ancestor of the key of rank i (now the smallest key) is that the key of rank j is selected for the root. This happens with probability $\frac{1}{n}$ which is indeed $\frac{1}{j-i+1}$.

If $n > j - i + 1$ there are two possibilities. Either we choose for the root a key with rank in $\{i..j\}$; then—conditioned on this—we have to choose the key of rank j for our event to occur, which happens with probability $\frac{1}{j-i+1}$. Otherwise, the keys of rank i and j, respectively, land in a common subtree; the keys get possibly new ranks inside the subtree (if it is the right subtree), but the difference between their respective ranks stays $j - i$. That is, by induction, again the probability for our event to occur is $\frac{1}{j-i+1}$, so the claim is established.

A more formal set-up for this reasoning builds on the law of total probability[9] and observes that

$$\begin{aligned}\Pr\left[A_i^j = 1\right] &= \Pr\left[A_i^j = 1 \,|\, \mathrm{rk}(\mathrm{root}) \in \{i..j\}\right] \cdot \Pr\left[\mathrm{rk}(\mathrm{root}) \in \{i..j\}\right] \\ &\quad + \Pr\left[A_i^j = 1 \,|\, \mathrm{rk}(\mathrm{root}) \notin \{i..j\}\right] \cdot \Pr\left[\mathrm{rk}(\mathrm{root}) \notin \{i..j\}\right]\end{aligned}$$

We have argued that

$$\begin{aligned}\Pr\left[A_i^j = 1 \,|\, \mathrm{rk}(\mathrm{root}) \in \{i..j\}\right] &= \frac{1}{j-i+1} \quad \text{(directly), and} \\ \Pr\left[A_i^j = 1 \,|\, \mathrm{rk}(\mathrm{root}) \notin \{i..j\}\right] &= \frac{1}{j-i+1} \quad \text{(by induction).}\end{aligned}$$

Finally, $\Pr[\mathrm{rk}(\mathrm{root}) \in \{i..j\}] + \Pr[\mathrm{rk}(\mathrm{root}) \notin \{i..j\}] = 1$. ◻

It follows that for $1 \le i \le n$, $\sum_{j=1}^{i-1} \mathbf{E}\left[A_i^j\right] = H_i - 1$ and $\sum_{j=i+1}^{n} \mathbf{E}\left[A_i^j\right] = H_{n-i+1} - 1$ and thus, by (2.5) we have the desired expectations.

Theorem 2.6. $i, n \in \mathbf{N}$, $i \le n$. *Then* $\mathbf{E}\left[D_n^{(i)}\right] = H_i + H_{n-i+1} - 2 \le 2 \ln n$.

This is consistent with our findings for the expected depth of the smallest key in Section 2.2. Moreover, linearity of expectation lets us reestablish

[9] Law of Total Probability: If A is an event, and $B_1, B_2, \ldots, B_k$ are events that partition the probability space then $\Pr[A] = \sum_{i=1}^{k} \Pr[A \,|\, B_i] \cdot \Pr[B_i]$.

Theorem 2.2 for the expectation of the overall depth:

$$\begin{aligned}\mathsf{E}\left[\sum_{i=1}^{n} D_n^{(i)}\right] &= \sum_{i=1}^{n} \mathsf{E}\left[D_n^{(i)}\right] \\ &= 2\sum_{i=1}^{n} H_i - 2n \\ &= 2\left((n+1)H_n - n\right) - 2n \\ &= 2(n+1)H_n - 4n\ .\end{aligned}$$

The third equality is left as an exercise.

Exercise 2.25. Sum of Harmonic Numbers
$n \in \mathbb{N}$. *Show* $\sum_{i=1}^{n} H_i = (n+1)H_n - n = (n+1)(H_{n+1} - 1)$.

Exercise 2.26. Size of Subtrees
$i \in \mathbb{N}$, $n \in \mathbb{N}_0$, $i \leq n$. *For a random search tree for* n *keys, let* $W_n^{(i)}$ *be the random variable for the number of nodes in the subtree rooted at the node of rank* i *(including the node itself).*

(1) Show that $\mathsf{E}\left[\sum_{i=1}^{n} W_n^{(i)}\right] = n + \mathsf{E}\left[\sum_{i=1}^{n} D_n^{(i)}\right]$, *where* $D_n^{(i)}$ *is the random variable for the depth of the node of rank* i.

(2) Show that $\mathsf{E}\left[W_n^{(i)}\right] = 1 + \mathsf{E}\left[D_n^{(i)}\right]$ *for all* $i \in [n]$.

(3) Determine $\mathsf{E}\left[\max\{W_n^{(i)} \mid i \in [n]\}\right]$.

HINT: Think before calculate!

Exercise 2.27. Product of Subtree Sizes
$n \in \mathbb{N}$, $W_n^{(i)}$ *as in Exercise 2.26. Determine* $\mathsf{E}\left[\prod_{i=1}^{n} W_n^{(i)}\right]$.

Exercise 2.28. $n \in \mathbb{N}$.

(1) What is the probability that the key of rank 1 *ends up as a leaf in a random search tree for* n *keys.*

(2) For $i \in [n]$, *what is the probability that the key of rank* i *ends up as a leaf in a random search tree for* n *keys.*

Exercise 2.29. Left to Right Minima
Given a permutation $(a_1, a_2, \ldots, a_n)$ *u.a.r. from all permutations of* $[n]$, *we call* a_i *a* left-to-right minimum *if* $a_i = \min_{j=1}^{i} a_j$. *Calculate the expected number of left-to-right minima in two ways.*

(1) *For* $i \in [n]$*, let* X_i *be the indicator variable for the event that* a_i *(i.e. the* i*th number in the sequence) is a left-to-right minimum. And then consider* $\sum_{i=1}^{n} \mathbf{E}[X_i]$*.*

HINT: Generate the random sequence backwards, i.e. first choose $a_n \in_{\text{u.a.r.}} [n]$, then $a_{n-1} \in_{\text{u.a.r.}} [n] \setminus \{a_n\}$, etc.

(2) *For* $k \in [n]$*, let* Y_k *be the indicator variable for the event that the number* k *is a left-to-right minimum. And then consider* $\sum_{k=1}^{n} \mathbf{E}[Y_k]$*.*

Exercise 2.30. *In a graph* $G = (V, E)$*, the vertices are randomly colored red and blue, each vertex obtains each color with equal probability* $\frac{1}{2}$*, independently from the coloring of the other vertices.*

What is the expected number of edges whose endpoints have distinct colors? Express the answer in terms of $m := |E|$*.*

HINT: For each edge $e \in E$, introduce an indicator variable X_e for the event that e receives two distinct colors ...

2.5 Quicksort

Quicksort is a sorting algorithm, introduced in [Hoare, 1962], that needs $O(n \log n)$ time on the average, performs well in practice, can be done in-place and is easy to implement. In this discussion we concentrate on the number of comparisons[10] made by the algorithm[11].

Recall that for sorting a set S, quicksort selects one $x \in S$, called the *pivot*, and partitions S into $S^{<x}$, $\{x\}$, and $S^{>x}$. It then sorts $S^{<x}$ and $S^{>x}$ recursively, and composes the sorted sequences in the right order. The choice of the pivot is crucial, and bad choices can cause the algorithm to do $\Theta(n^2)$ comparisons. Here we consider *randomized quicksort*, where the pivot is chosen u.a.r. among all keys; see procedure[12] quicksort()[13].

The comparisons occur in the splitting step, $|S| - 1$ of them. We let t_n, $n \in \mathbf{N}_0$, stand for the expected number of comparisons made when sorting

[10] We mean here only comparisons between the elements we want to sort, not those inferred by while-loops and such like.

[11] Quicksort's favorable behavior in practice relies also on the small number of data movements (swaps) inferred; see also the note on median-of-three quicksort below.

[12] The description doesn't do full justice to quicksort, since the exact way of splitting S is crucial for its efficiency.

[13] $\circ$ is used for concatenation of sequences.

	function quicksort(S)
$S \subseteq \mathbf{R}$, finite.	if $S = \emptyset$ then return $()$;
POSTCONDITION: returns S as a sequence sorted in increasing order.	else $x \leftarrow_{\text{u.a.r.}} S$; split S into $S^{<x}$, $\{x\}$, $S^{>x}$; return quicksort$(S^{<x}) \circ (x) \circ$ quicksort$(S^{>x})$;

a set of n keys. Then $t_0 = 0$ and, for $n \geq 1$,

$$t_n \;=\; n-1+\sum_{i=1}^{n}(t_{i-1}+t_{n-i})\,\frac{1}{n} \;=\; n-1+\frac{2}{n}\sum_{i=1}^{n} t_{i-1}$$

This is exactly the recurrence we obtained for the expected overall depth of random search trees (x_n in Section 2.2), and we get the solution for t_n for free.

Theorem 2.7. $n \in \mathbf{N}_0$. *The expected number of comparisons performed by* quicksort() *for sorting a set of* n *numbers is* $2(n+1)H_n - 4n$.

It is important to note the difference between (i) the average behavior of deterministic quicksort (that e.g. chooses the first element in the list as pivot element) and (ii) the expected behavior of randomized quicksort. The former, (i), assumes that the input is given in random order (according to the uniform distribution)—hence, there can be bad inputs (an almost sorted sequence is very bad for the aforementioned pivot choice). In contrast, for the randomized version (ii) there is nothing like a 'bad' input, the randomness is introduced internally by the algorithm without having to rely on a nice input distribution.

We shouldn't shrug off the fact that the expected number of comparisons of quicksort() equals the overall depth of a random search tree. Given the structure of the procedure quicksort() and our definition of random search trees, the analogy becomes intuitively evident. In fact, during an execution of quicksort() we can build a binary search tree on the side, always making the pivot the root of a new subtree. In this way, every computation of quicksort(S) maps to a search tree B in $\mathcal{B}_S$, and it is to show that the resulting distribution is the same as for random search trees.

How can we read the number of comparisons from the search tree? Whenever a pivot element is chosen, it is compared to all the elements that

eventually end up in its subtree (excluding itself). If $w(v)$ is the number of nodes in the subtree rooted at node v, then the number of comparisons can be written as $\sum_v (w(v) - 1)$ (sum over all nodes v in the tree). Now we redistribute this sum as follows: For every node we open an 'account', and every node pays 1 to each account in its subtree (itself excluded), so node v pays $w(v) - 1$ altogether. The final balance on an account is the depth $d(v)$ of its node v, as we can easily see (a node gets 1 from every node on the path to the root, excluding itself). Thus $\sum_v d(v) = \sum_v (w(v) - 1)$, and these sums have to equal in expectation (to be precise, the random variables $X = \sum_v d(v)$ and $Y = \sum_v (w(v) - 1)$ are the same). In this way many properties of random search trees can be translated to properties of randomized quicksort (see, e.g., Exercises 2.31 and 2.33).

NOTE 2.2 Median-of-three quicksort chooses the pivot in S as follows, assuming $|S| \geq 3$. First we sample a set of three numbers $\{a, b, c\}$ u.a.r. in $\binom{S}{3}$, and then we select the pivot as the median among the three (the element of rank 2 in $\{a, b, c\}$). In this way the expected number of comparisons goes down to $\frac{12}{7}(n+1)H_n + O(n)$, [Sedgewick, Flajolet, 1996]. On the one hand, this looks like a speed-up by 14%. On the other hand, there is some extra overhead, and the expected number of data movements goes up (from roughly[14] $\frac{2}{3} n \ln n$ to $\frac{24}{35} n \ln n$). Hence, whether this variant results in a speed-up in terms of CPU seconds, depends on implementation details, hardware, and the cost ratio between comparisons[15] and data swaps. The variants median-of-$(2t+1)$ for $t \geq 2$ are usually inferior.

Exercise 2.31. How Many (Truly) Random Choices?

If we call quicksort() *with a set of n keys, how many executions of the statement '$x \leftarrow_{\text{u.a.r.}} S$;' occur altogether? What is the expected number of executions of this statement with a set S of size at least 2?*

Exercise 2.32. Misplaced

Quicksort's efficiency relies not only on the number of comparisons (which is actually larger than for merge sort), but also on the small number of keys that have to be moved. To that end analyze the expected number of keys moved in a partitioning step, where the elements

[14] Often the number of exchanges (swaps) of misplaced elements are counted; then these numbers have to be divided by two (see also Exercise 2.32).

[15] For example there is a number type REAL in the LEDA package, [Mehlhorn, Näher, 1999] which holds—believe it or not—'real' real numbers (to some extent). For such numbers comparisons can be very expensive.

are given in u.a.r. order and the first element is chosen for the pivot (average case for classical unrandomized quicksort):

For a permutation $\pi = (a_1, a_2, \ldots, a_n)$ *of a set* $S \subseteq \mathbf{R}$ *of cardinality* n*, let* $\mathrm{mispl}(\pi)$ *be the number of elements that have to be moved if* a_1 *goes to position* $\mathrm{rk}(a_1)$ *and elements preceding this position have to be smaller than* a_1*, and elements succeeding have to be larger. Formally,*

$$\begin{aligned}\mathrm{mispl}(\pi) \;:=\; & \overbrace{[\mathrm{rk}(a_1) \neq 1]}^{a_1 \text{ misplaced}} + \overbrace{|\{j \in \{2..(\mathrm{rk}(a_1)-1)\} \mid a_j > a_1\}|}^{a_j\text{'s},\, 1<j<\mathrm{rk}(a_1),\, \text{misplaced}} \\ & + \underbrace{[\mathrm{rk}(a_1) \neq 1]}_{a_{\mathrm{rk}(a_1)} \text{ misplaced}} + \underbrace{|\{j \in \{(\mathrm{rk}(a_1)+1)..n\} \mid a_j < a_1\}|}_{a_j\text{'s},\, j>\mathrm{rk}(a_1),\, \text{misplaced}} \;,\end{aligned}$$

Elements that have to be moved we call misplaced*. For example, here are all permutations of* $\{1,2,3\}$ *with misplaced elements underlined:*

1 2 3	$\underline{2}$ $\underline{1}$ 3	$\underline{3}$ 1 $\underline{2}$
1 3 2	$\underline{2}$ $\underline{3}$ $\underline{1}$	$\underline{3}$ 2 $\underline{1}$

$$\mathbf{E}[M_3] = \tfrac{1}{6}(0+0+2+3+2+2) = \tfrac{3}{2}\;.$$

Determine $\mathbf{E}[M_n]$*,* M_n *the random variable for the number of misplaced elements in a u.a.r. permutation of* n *elements.*

Exercise 2.33. Leftist Pivot

Consider the following variant of randomized quicksort. Whenever we have to choose a pivot element for a set S*,* $|S| \geq 2$*, we select a pair* $\{A, B\} \in_{\text{u.a.r.}} \binom{S}{2}$*, and let* $\min\{A, B\}$ *be this pivot element. What is the expected number of comparisons of this variant? Note that the "other" element in* $\{A, B\}$ *does not need to be compared with the pivot element once more, so still altogether* $|S| - 1$ *comparisons suffice for the splitting.*

Exercise 2.34. Leftist Pivot and Stack Size

For those who know about the issue of the stack used in resolving recursion (basically, whenever we make a call, we have to store the state of the procedure in order to have this information ready as we return from the call). Argue why the leftist-pivot variant of quicksort (as described in Exercise 2.33) is favorable in the sense that the size of the stack tends to be smaller there.

Exercise 2.35. Depth and Subtree Size Again

$n \in \mathbf{N}_0$*. Consider a set* S *of* n *keys, say* $S = [n]$ *and consider a binary search tree that evolves from inserting the elements in* S *in order*

$a_1, a_2, \ldots, a_n$, *u.a.r. from all* $n!$ *orderings of* S *(which, we know, generates the distribution of random search trees as defined in Section 2.1). For* $i, j \in [n]$, *let* $A^{\langle i,j\rangle}$ *be the indicator variable for the event that the node holding the* j*th key,* a_j, *in insertion order is an ancestor of the node holding the* i*th key,* a_i, *inserted.*

(1) Determine $\mathsf{E}\left[A^{\langle i,j\rangle}\right]$ *for all* $i, j \in [n]$ *and use it to determine* $\mathsf{E}\left[W^{\langle j\rangle}\right]$, $W^{\langle j\rangle}$ *the size of the subtree of the* j*th key inserted.*

(2) Determine $\mathsf{E}\left[D^{\langle i\rangle}\right]$, $D^{\langle i\rangle}$ *the depth of the* i*th key inserted,* $i \in [n]$.

HINT: First determine $\mathsf{E}\left[A^{\langle j+1,j\rangle}\right]$ and then show that $\mathsf{E}\left[A^{\langle i,j\rangle}\right] = \mathsf{E}\left[A^{\langle j+1,j\rangle}\right]$ for all $i > j$.

REMARK: $\sum_{j=1}^{n}\left(\mathsf{E}\left[W^{\langle j\rangle}\right] - 1\right) = \sum_{i=1}^{n}\mathsf{E}\left[D^{\langle i\rangle}\right]$ is just another way of calculating the expected overall depth of a random search tree. Also compare $\mathsf{E}\left[D^{\langle n\rangle}\right]$ to the answers to Exercises 2.18 and 2.19 (2).

Exercise 2.36. $i, j, n \in \mathbf{N}$, $i < j \leq n$. *What is the probability that the randomized procedure* quicksort() *applied to a set of* n *numbers compares the element of rank* i *with the element of rank* j*?*

2.6 Quickselect

Given a set S of keys, suppose we want to know the middle element (median), or more generally, the k-smallest element in S (i.e. the element of rank k in S). Nothing easier than that: We sort S and access the kth element in the sorted sequence. That works in expected $O(n \log n)$ time, $n := |S|$, with randomized quicksort, say. But can we do it faster?—after all that can obviously be done in linear time for the smallest or the largest element.

In fact, a moment of reflection shows that if all we care about is the element of a given rank k, in each call to quicksort() one of the two recursive calls is irrelevant (eventually, when the pivot is the element we are looking for, even both)—an insight that immediately suggests the procedure quickselect(,).

For the analysis we let $t(k, n)$, $1 \leq k \leq n$, denote the expected number of comparisons among elements in S inferred by a call quickselect(k, S) with $|S| = n$. We are by now experienced enough to quickly derive $t(1, 1) = 0$

$S \subseteq \mathbf{R}$, finite. $k \in \mathbf{Z}$.
PRECONDITION:
$1 \le k \le |S|$.
POSTCONDITION:
returns the element of rank k in S.

```
function quickselect(k, S)
x ←_u.a.r. S;
split S into S^<x, x, S^>x;
ℓ ← |S^<x| + 1;   (i.e. ℓ = rk_S(x))
if k < ℓ then
    return quickselect(k, S^<x);
elseif k = ℓ then
    return x;
else   (i.e. k > ℓ)
    return quickselect(k − ℓ, S^>x);
```

and, for $n \ge 2$,

$$t(k,n) = n - 1 + \frac{1}{n}\left(\sum_{\ell=1}^{k-1} t(k-\ell, n-\ell) + \sum_{\ell=k+1}^{n} t(k,\ell-1)\right) ,$$

but solving this recurrence looks scary, if not hopeless.

So, in order to get at least an idea of the asymptotics, we retreat to the following estimate, where we omit the parameter k, set $t_n := \max_{k=1}^{n} t(k,n)$, and assume for the recursive call always the size of the larger of the two sets[16] $S^{<x}$ and $S^{>x}$. That is, $t_1 = 0$ and, for $n \ge 2$,

$$t_n \le n - 1 + \frac{1}{n}\sum_{\ell=1}^{n} t_{\max\{\ell-1,n-\ell\}} .$$

In a fit of optimism we conjecture that $t_n \le cn$ for some positive real constant c. $t_1 = 0$ serves as a safe basis for induction, so all we need to ensure is that $n - 1 + \frac{1}{n}\sum_{\ell=1}^{n} c \cdot \max\{\ell-1, n-\ell\} \le cn$ or, equivalently,

$$\sum_{\ell=1}^{n} \max\{\ell-1, n-\ell\} \le \frac{c-1}{c}n^2 + \frac{1}{c}n .$$

The sum can be computed to be

$$\sum_{\ell=1}^{n} \max\{\ell-1, n-\ell\} = \begin{cases} \frac{3}{4}n^2 - \frac{1}{2}n, & \text{if } n \text{ even, and} \\ \frac{3}{4}n^2 - \frac{1}{2}n - \frac{1}{4}, & \text{otherwise.} \end{cases}$$

Therefore, the proof by induction indeed goes through with $c = 4$.

[16] Here we assume, without proof, that t_n is monotone in n.

Theorem 2.8. $k, n \in \mathbf{N}_0$, $1 \leq k \leq n$. *The expected number of comparisons (among input numbers) performed by* quickselect(,) *for determining the element of rank* k *in a set of* n *numbers is at most* $4n$.

Exercise 2.37. Max und ...

$n \in \mathbf{N}_0$. *Prove*

$$\sum_{i=1}^{n} \max\{i-1, n-i\} = \begin{cases} \frac{3}{4}n^2 - \frac{1}{2}n, & \text{if } n \text{ even, and} \\ \frac{3}{4}n^2 - \frac{1}{2}n - \frac{1}{4}, & \text{otherwise.} \end{cases}$$

and determine $\sum_{i=1}^{n} \min\{i-1, n-i\}$.

Exercise 2.38. "Quickselect" Smallest Element

Determine the expected number of comparisons performed by quickselect for finding the smallest element (case $k = 1$*) in a set of* n *numbers. (Not*[17] *that we recommend quickselect for finding the smallest number.)*

Exercise 2.39. Quickselect vs. Random Search Trees

Let $X_{k,n}$ *be the random variable for the number of comparisons made by quickselect when searching for the element of rank* k *in a set of* n *numbers. Define a random variable on random search trees with the same distribution.*

Exercise 2.40. Number of Recursive Calls

What is the expected number of recursive calls made by quickselect when searching for the k*th in* n *numbers?*

Exercise 2.41. Comparisons with Object of Desire

When we use quickselect to search for the element of rank k *in a set of* n *numbers, what is the expected number of comparisons the number of rank* k *is involved in? (Note that, in fact, no number takes part in more comparisons than the* k*th number does.)*

Exercise 2.42. Expected Amount of Work Left

$k, n \in \mathbf{N}$, $k \leq n$. *In search for the* k*th element in a set* S *of* n *numbers, quickselect chooses a random number in* S *and either luckily holds the result or recurses on a subset* S' *of* S. *Determine the expectation of* $|S'|$ *in terms of* k *and* n *(where we define* $S' := \emptyset$*, if the pivot has rank* k*). For which* k*, in terms of* n*, is this expectation the smallest?*

[17] Note: "Not", not "Note"!

Exercise 2.43. Decreasing Process
Given $n \in \mathbf{N}$*, a random process generates a sequence of integers* $n = X_0, X_1, \ldots, X_{N-1}, X_N = 0$*. All we know is that there is a constant* $0 < c < 1$ *such that if* $X_i \geq 1$ *then* $X_{i+1} < X_i$ *and* $\mathbf{E}[X_{i+1} \mid X_i = x] < c \cdot x$*, and if* $X_i = 0$ *then the process stops (i.e.* $N = i$*). What can you say about* $\mathbf{E}[N]$*, the expected length of the process, and* $\mathbf{E}\left[\sum_{i=0}^{N} X_i\right]$*? In particular, can you prove that the later expectation is at most linear in* n*?*

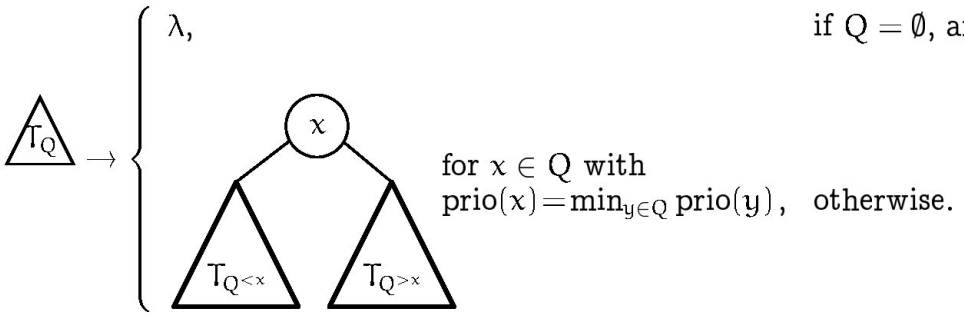
2.7 Randomized Search Trees

"If you could use only one data structure, which one would you choose? A hash table? While it supports the basic insert, find, and remove operations, it doesn't keep the elements in sorted order. Therefore, it can't efficiently perform some tasks that are frequently encountered, such as finding the minimum element or producing an ordered list of all elements. ..." Stefan Nilsson writes in his article "Treaps in Java" in *Dr. Dobb's Journal*, **267** (July 1997) 40-44, and he continues: "What would you require of this ideal, sole structure? It should be easy to use (and preferably easy to implement); it should be able to hold an object of any class (as long as we provide a method for comparing objects); it should be thread safe.

The randomized search tree ('treap'), devised by C.R. Aragon and R. Seidel and described in 'Randomized Search Trees' (*Algorithmica*, 16(4/5):464–497, 1996), fulfills all of these requirements, and it offers the functionality of a general-purpose sorting routine, priority queue, hash table, stack, or standard queue. ..."

Treaps. 'Treap' is a coined word that stands for a symbiosis of a binary search *tree* and a h*eap*. It is defined for sets $Q \subseteq \mathbf{R} \times \mathbf{R}$, with the elements in Q called *items*. The first component of an item x is its *key*, $\text{key}(x)$, and the second component is its *priority*, $\text{prio}(x)$. Suppose that no two keys in Q are the same, nor are two priorities the same. Then a *treap* on Q is a binary tree with nodes labeled by Q which is a search tree with respect to the keys, and a min-heap with respect to the priorities (i.e. if x is parent of y then $\text{prio}(x) \leq \text{prio}(y)$). The fact that every set of items Q allows a treap, and this treap is actually unique, becomes evident from the alternative grammar-style definition ($Q^{<x} := \{y \in Q \mid \text{key}(y) < \text{key}(x)\}$

and analogously for $Q^{>x}$):



This view also reveals that if, for a set of keys, the priorities are chosen independently and u.a.r. from the interval $[0, 1)$ then the resulting treap is a random search tree[18] for the keys of its items. Therefore, if we maintain for a set of keys a treap, where for every newly inserted key a random priority is chosen, then this treap is a random search tree for the keys, independently from the insertion order. And we can assume all the wonderful properties like expected (and high probability) logarithmic height of the tree. Remains the issue of maintenance of a treap under insertions and deletions.

Insertions, Rotations. To insert a new item x in a treap, we first proceed as in a standard binary search tree and insert the new item as a leaf according to its key $\mathrm{key}(x)$. Next we have to reinstall the heap property: As long as x has a smaller priority than its parent item y we perform a rotation at y which makes[19] y a child of x (see Figure 2.5; you know rotations well as rebalancing tool for AVL-tree and such like). This will let x eventually end up at its proper place in the treap.

The running time is proportional to the depth of the new item as a leaf before reinstalling the heap property, plus the number of rotations necessary to get the new item to its place. Both is bounded by the height of the tree and thus expected $O(\log n)$.

However, since rotations are much more costly than just walking down a tree in a search, let us take a closer look at them right away. How many

[18] Once more, what do we mean by a statement like "this *is* a random search tree"? We mean that in this way the same distribution on the binary search trees for the given set of keys is defined.

[19] If x is left child, rotate right at y, if x is right child, rotate left.

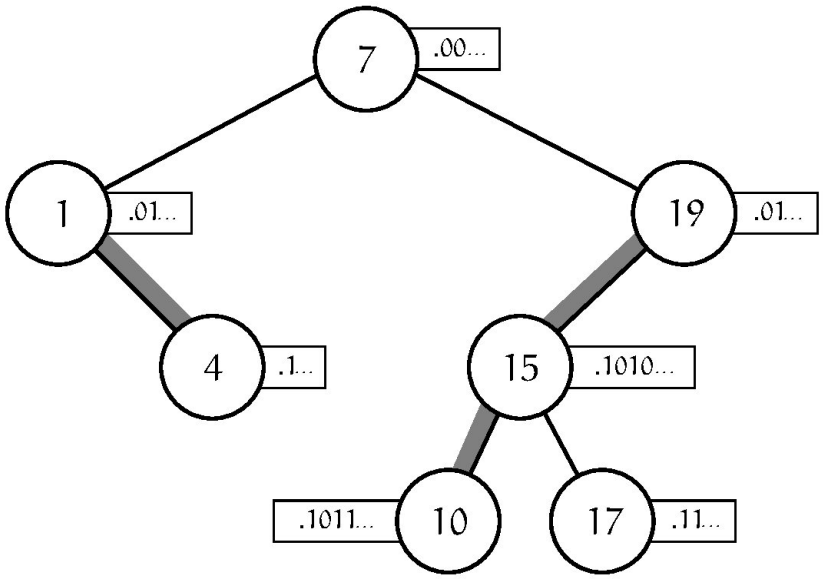


Figure 2.4: Treap for items with real priorities from $[0, 1)$ in binary representation: $(1, .0100...)$, $(4, .1010...)$, $(7, .0001...)$, $(10, .1011...)$, $(15, .1010...)$, $(17, .1101...)$, $(19, .0111...)$. In the treap the real priorities are displayed only to the extent necessary for its specification. The shaded parts indicate the root's left subtree's right spine and the root's right subtree's left spine.

rotations do we need? We can read the number of such rotations from the repaired treap without knowing the priorities (and thus without knowing which rotations were indeed performed). In order to quantify this, we define the *left (right) spine* of a subtree rooted at node v as the sequence of nodes on the path from v to the smallest (largest, respectively) key in the subtree (for example, in Figure 2.4, the subtree rooted at the node holding key 19 has a left spine of length 3 and a right spine of length 1). Now associate with a node—hold your breath, but see Figure 2.5—the sum of the lengths of the right spine in the left child's subtree and of the left spine in the right child's subtree. This number increases by exactly one with every rotation for the node with the new item, and thus this quantity at its final position specifies the necessary number of rotations. In Figure 2.4 for the root holding key 7, the right spine of the left subtree has length 2 (with keys 1 and 4), the left spine of the right subtree has length 3 (with keys 19, 15, and 10). Consequently, if this was the last node inserted it

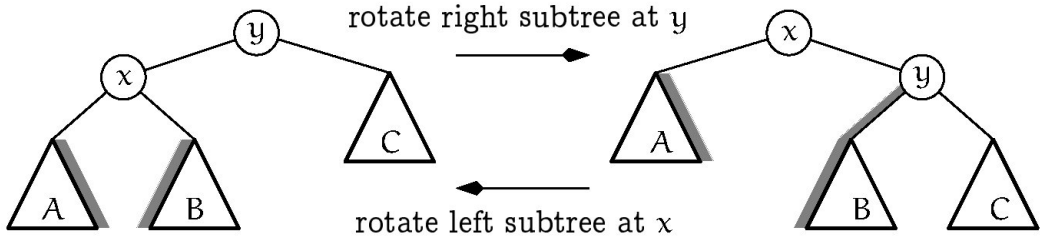


Figure 2.5: Rotations in a tree. (The shaded parts indicate x's respective right spine in its left subtree and left spine in its right subtree. Note that the change is restricted to gain/loss of a single node, namely y, in these spines.)

took $2+3=5$ rotations to move it to its place.

The expected number of rotations is less than 2, no matter how large the tree is. This is, at least at first glance, quite surprising, so let us see a proof of this.

Lemma 2.9. $n \in \mathbf{N}$, $j \in [n]$. *In a random binary search tree for n keys, the right spine of the left subtree of the node of rank j has expected length $1-\frac{1}{j}$, and the left spine of the right subtree has expected length $1-\frac{1}{n-j+1}$.*

Proof. Consider the nodes on the right spine of the left subtree of node j. We observe that all keys on this spine have rank smaller than j. The nodes are characterized as those nodes which are ancestors of node $j-1$ but not common ancestors of j and $j-1$. Hence, if we define

$$C^k_{i,j} := [\text{node } k \text{ is ancestor of nodes } i \text{ and } j]$$

then the number of nodes on this spine is $\sum_{k=1}^{j-1}\left(A^k_{j-1}-C^k_{j-1,j}\right)$. But for $k \le i \le j$, $C^k_{i,j} = A^k_j$, since if node k has node $j \ge k$ in its subtree then it has all nodes $i \in \{k..j\}$ in its subtree (see Exercise 2.46). We conclude

$$\begin{aligned}\mathsf{E}\left[\sum_{k=1}^{j-1}\left(A^k_{j-1}-C^k_{j-1,j}\right)\right] &= \mathsf{E}\left[\sum_{k=1}^{j-1}\left(A^k_{j-1}-A^k_j\right)\right] = \sum_{k=1}^{j-1}\left(\mathsf{E}\left[A^k_{j-1}\right]-\mathsf{E}\left[A^k_j\right]\right)\\ &= \sum_{k=1}^{j-1}\left(\frac{1}{j-k}-\frac{1}{j-k+1}\right) = 1-\frac{1}{j}\,.\end{aligned}$$

The claim for the left spine of the right subtree follows by symmetry. ◻

Deletions, Splits, Joins. A deletion in a treap is an inverse insertion. First we rotate the item to be removed down the tree until it is a leaf, then we remove it. When we push the item down the tree, we always have a choice of a right or left rotation—which one to select is decided by the priorities. If the left child has the smallest priority of the children, then this child should become the new root of the subtree and we rotate right. Otherwise, we rotate left. Analysis is the same as for the insertion.

For a given pivot value s, a *split* of a treap for items Q generates two treaps for items $Q^{<s}$ and items $Q^{>s}$ (we assume that s is not among the keys of the treap). An implementation is easy to describe. We first insert an item with key s and priority $-\infty$, this item ends up as the root. Then we remove the root. Its left subtree is a treap for $Q^{<s}$, the right subtree for $Q^{>s}$. The time is again bounded by the height of the tree—the expected number of rotations is not constant in this case, though.

The *join* operation takes two treaps with one holding keys all of which are smaller than that of the other one. This can be seen as an inverse split. That is, we generate a new root with an in-between key s and priority $-\infty$, attach the treap with the smaller keys as a left subtree and the other as a right subtree, and then we delete the root item.

Theorem 2.10. *In a* randomized search tree *(a treap with priorities independently and u.a.r. from* $[0,1)$*) operations find, insert, delete, split and join can be performed in expected time* $O(\log n)$*,* n *the number of keys currently stored. The expected number of rotations necessary for an insertion or a deletion is always less than* 2*.*

Random Real Numbers? You may be worried that the priorities are real numbers. On the one hand, this was very convenient since in this way the probability of getting the same priority twice is 0. For all practical purposes we can choose the priorities from a sufficiently large ordered domain. $\{0..2^{32}-1\}$ will be by far enough for most practical purposes, even more so since most common random number generators, once initiated, generate a permutation of their domain without repetition.

However, we can simulate the (for us) necessary functionality of random real numbers on a digital computer (an exercise in the spirit of object-oriented programming). After all, all we need is to compare these numbers. For that we recall that a real number in $[0,1)$ in binary can be

encoded by a sequence $(b_i)_{i\in\mathbb{N}} \in \{0,1\}^{\mathbb{N}} \setminus \{0,1\}^*\{1\}^{\mathbb{N}}$, representing[20] the number $\sum_{i=1}^{\infty} b_i 2^{-i}$. A random real number is internally represented by a finite (random) sequence $(b_i)_{i=1}^{\ell}$, ℓ the current length of the representation. If two such numbers have to be compared, we check whether the available bits suffice[21] to take such a decision. Otherwise we produce extra random bits for the two numbers which extend these sequences until a decision is possible.

In fact, Figure 2.4 indicates such a representation. For example the items for keys 1 and 19 have the same representation .01... for their priorities. If these priorities had to be compared[22] (for example, if we remove key 7 we have to know which priority is smaller for deciding which way to rotate), then we produce one extra random bit for each and append it to the sequence. If the two bits are the same, we repeat. So, for example after the comparison, the representations may be .0101... for 1 and .0100... for 19.

If we want to compare two such partially represented random real numbers and we need extra bits, an expected number of $O(1)$ of them suffice (the exact expected number depends on the difference of their lengths; it is never more than 4, see Exercise 2.45). Now note that for an insertion and deletion, we need the random bits only to decide whether we have to rotate or which way to rotate, and in expectation there are only a constant number of such decisions to make, hence altogether only a constant number of bits suffice for these operations (the find-operation needs none).

NOTE 2.3 The original source for randomized search trees is [Aragon, Seidel, 1996], where we read about the underlying treaps that "... [Vuillemin, 1980] introduced the same data structure and called it 'Cartesian trees'. The term 'treap' was first used for a different data structure by [McCreight, 1985], who later abandoned it for the more mundane 'priority search tree'."

[Pugh, 1990] had proposed earlier another randomized scheme for the dictio-

[20] In order to make this representation unique, we have to exclude the sequences that eventually finish in an infinite sequence of 1s. For that recall that e.g. binary $0.0111111\ldots = 0.100000000\ldots$. The set $\{0,1\}^*\{1\}^{\mathbb{N}}$ stands for these sequences we exclude. However, such sequences occur with probability 0 in random $\{0,1\}$-sequences, so we justifiably ignore them in the further discussion.

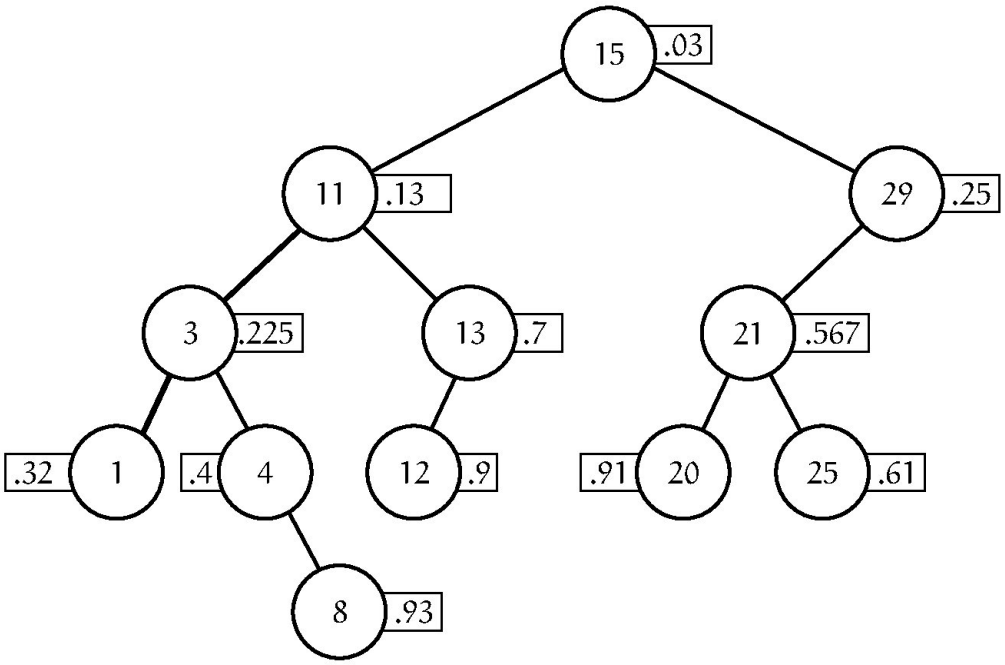
[21] Two sequences σ and τ of lengths i and j, respectively, $i \le j$, allow a decision, if the prefix of τ of length i differs from σ.

[22] Admittedly, the example is special since even different length representations may force extra bits until a decision can be made; .11... and .1101..., say.

nary problem called *skip lists*. A different kind of randomized search tree based on random rebalancing was later described by [Martinéz, Roura, 1998] with basically identical performance characteristics.

Exercise 2.44. Rotations

How many rotations are necessary for deleting the item with key 11 *in the following treap?*



Exercise 2.45. Extra Bits for Comparison

If two partial representations σ *and* τ *of random real numbers in* $[0,1)$ *with* i *and* j *bits, respectively,* $i \leq j$*, do not allow a comparison, what is the expected number of extra random bits that have to be generated until such a comparison is possible?*

Exercise 2.46. Ranks in Subtrees

Consider the set of the ranks of the keys in a subtree of a binary search tree. Show that this set is of the form $\{a..b\}$*, where* a *is the smallest rank in the subtree and* b *is the largest rank in the subtree.*

Exercise 2.47. Common Ancestors

For a random search tree on n *nodes and* $i, j, k \in [n]$*,* $i \leq j$*, define the*

random indicator variable

$$C^k_{i,j} := [\textit{node } k \textit{ is common ancestor of nodes } i \textit{ and } j]$$

(where "node k*" stands for "node holding key of rank* k*", etc.).*

Determine $\Pr\left[C^k_{i,j} = 1\right]$ *for* $i, j, k \in [n]$, $i \leq j$.

HINT: You will have to discriminate the cases $k < i \leq j$, $i \leq k \leq j$, and $i \leq j < k$.

Exercise 2.48. Paths between Nodes

In a tree there is always a unique path between any two nodes.

$n \in \mathbf{N}$, $i, j \in [n]$. *Determine the expected length of the path between the nodes holding the keys of rank* i *and* j *in a random search tree for* n *keys.*

Exercise 2.49. Random Bits

Show that in the proceeding described above for successively generating the bits of random priorities, the expected number of random bits to be generated for an insertion is at most 12*, and for a deletion it is at most* 4*.*