Solution 6.1  Median-of-three Quicksort.

The basic idea in the construction of such an instance is to place the the largest and the second largest element in each recursive call in such a way that the next recursive call will need to process all the remaining elements. Let \( n \in \mathbb{N} \) be an arbitrary integer. We construct an array \( A^{(n)} \) of length \( 2n + 1 \), on which Median-of-three Quicksort has a quadratic running time. We can construct such an array as follows.

The array \( A^{(n)} \) of length \( 2n + 1 \) contains every of the numbers \( 1, 2, \ldots, 2n + 1 \) exactly once. We want to ensure that the smallest element (i.e., 1) is always located at the beginning, the largest one in the middle, and the second largest one at the end of \( A^{(n)} \). For \( n = 1 \), these conditions are satisfied for the array \( A^{(1)} = [1, 3, 2] \). We now assume that we are already able to construct the array \( A^{(n-1)} \). Assume that it had the following structure:

\[
A^{(n-1)} = 
\begin{bmatrix}
A & b & C \\
1 & \cdots & n & n+1 & n+2 & \cdots & 2n-1
\end{bmatrix}
\]

The sub-array \( A \) consists of the first \( n \) elements, \( b \) is only the \( n + 1 \)-th element, and \( C \) consists of the other \( n - 2 \) elements. In particular, the first element of \( A \) is exactly the 1, and \( b \) is less than \( 2n - 2 \) (because \( b \) was neither the medium nor the last element of \( A^{(n-1)} \)). From this array, we now generate \( A^{(n)} \) with the following structure:

\[
A^{(n)} = 
\begin{bmatrix}
A & 2n + 1 & C & b & 2n \\
1 & \cdots & n & n+1 & n+2 & \cdots & 2n-1 & 2n & 2n+1
\end{bmatrix}
\]

Since \( A \) is at the beginning, the first element of \( A^{(n)} \) is always 1. The middle, i.e., the position \( n + 1 \), contains the largest element, and the end contains the second largest one. When Median-of-three Quicksort is invoked on the array \( A^{(n)} \), then we first compute the median of the elements 1 (position 1), \( 2n + 1 \) (at position \( n + 1 \)) and \( 2n \) (at position \( 2n + 1 \)). This is exactly \( 2n \), so \( 2n \) is used as the pivot element. In the partition step only the elements \( b \) and \( 2n + 1 \) are exchanged. Since the pivot element is the second largest element in \( A^{(n)} \), after the partition step the array has the following structure:

\[
\begin{bmatrix}
A & b & C & 2n & 2n + 1 \\
1 & \cdots & n & n+1 & n+2 & \cdots & 2n-1 & 2n & 2n+1
\end{bmatrix}
\]

In particular, the first \( 2n - 1 \) elements represent exactly \( A^{(n-1)} \). The array \( A^{(n)} \) will be generated after \( A^{(2)} \) from \( A^{(1)} \) is produced, then \( A^{(3)} \) from \( A^{(2)} \), \( A^{(4)} \) of \( A^{(3)} \), etc.

When Median-of-three Quicksort is called on the array \( A^{(n)} \), then there is a recursive call on the array \( A^{(n-1)} \), which in turn starts a recursive call on \( A^{(n-2)} \), etc. We now determine the number of key comparisons \( T(n) \), when Median-of-three Quicksort with the input \( A^{(n)} \) is called. For \( n = 1 \), there are \( T(1) = 3 \) comparisons performed, because \( A^{(1)} \) consists of exactly three elements, and there are three comparisons needed to determine the median. For general \( n \), the first three comparisons on the input \( A^{(n)} \) serve to determine the median, then \( 2n - 1 \) comparisons are still needed to compare the pivot element with the elements at positions 2, ..., \( 2n \) (the first element was already compared to the pivot element when determining the median of the three).

Since the resulting array has only one element on the right of the pivot element, there will be no
further comparisons for that part, and we get 

\[ T(n) = (2n - 1) + 3 + T(n - 1) = 2n + 2 + T(n - 1). \]

Thus, the recursion yields

\[ T(n) = \begin{cases} 
2n + 2 + T(n - 1) & \text{falls } n > 1 \\
3 & \text{falls } n = 1 
\end{cases} \quad (1) \]

By telescoping we obtain the conjecture \( T(n) \geq n^2 \), which we prove by induction on \( n \).

**Base step** \((n = 1)\): For \( n = 1 \) the statement is correct, because \( T(1) = 3 \geq 1^2 \).

**Inductive hypothesis**: Assume that the statement is correct for \( n \), so \( T(n) \geq n^2 \).

**Inductive step** \((n \to n + 1)\): We have

\[ T(n + 1) = 2(n + 1) + 2 + T(n) \geq 2n + 4 + n^2 \geq n^2 + 2n + 1 = (n + 1)^2. \quad (2) \]

Thus, Median-of-three Quicksort compares at least \( n^2 \) many keys when the array \( A^{(n)} \) is used as input.

**Solution 6.2  Number of different Search Trees.**

Let \( K_n = \{1, ..., n\} \) be a set of keys, and let \( T(n) \) be the number of different search trees for the key set \( K_n \). For \( n = 0 \) we have \( K_0 = \emptyset \) and \( T(0) = 1 \), because only the empty tree corresponds to \( K_0 \). For \( n = 1 \) we have \( K_1 = \{1\} \), and the only possible search tree contains just the first key, so \( T(1) = 1 \).

For general \( n \), we proceed as follows. Assume that the key \( k \) is stored at the root of a search tree. Then the left subtree contains the keys \( \{1, ..., k - 1\} \), and analogously the right subtree contains the keys \( \{k + 1, ..., n\} \). There are \( T(k - 1) \) possible subtrees on the left and \( T(n - k) \) possible subtrees on the right. Since every possible subtree on the left can be combined with every possible subtree on the right, the numbers have to be multiplied. Thus, there are \( T(k - 1) \cdot T(n - k) \) possible search trees for key set \( K_n \) with the key \( k \) at the root. We now only need to sum over all the possible keys \( k \) and obtain

\[ T(n) = \sum_{k=1}^{n} T(k - 1)T(n - k). \quad (3) \]

**Note:** These numbers are known under the name Catalan numbers (after the Belgian Mathematician Eugène Charles Catalan). The closed formula for \( T(n) \) (without proof) is

\[ T(n) = \frac{1}{n + 1} \binom{2n}{n}. \quad (4) \]

**Solution 6.3  Traversal of Trees.**

a)  
- Preorder traversal: 7, 5, 3, 10, 8, 9, 11, 15
- Postorder traversal: 3, 5, 9, 8, 15, 11, 10, 7

b) A binary search tree can be reconstructed from its preorder traversal \( k_1, \ldots, k_n \) by inserting the keys \( k_1, \ldots, k_n \) in this order into an initially empty binary search tree. This produces the following search tree:
Solution 6.4  Hash functions.

a)  
- $h(k) = \text{Digit sum of } k$. This function is not suitable for hashing. The value of the hash function must lie between 0 and $p - 1$, but the digit sum can be arbitrarily large.
- $h(k) = k \cdot (1 + p + p^2) \mod p$. Since $p \mod p = 0$, and $p^2 \mod p = 0$, we have $h(k) = k \mod p$. At the lecture it was explained that $h(k) = k \mod p$ is a suitable hashing function.
- $h(k) = \lfloor p(rk - \lfloor rk \rfloor) \rfloor, r \in \mathbb{R}^+ \setminus \mathbb{Q}$. This function is also suitable for hashing, because it corresponds to the multiplicative method presented in the lecture.

b)  
- $h'(k) = [\ln(k + 1)] \mod q$. This function is not suitable as the second hash function, because for the key $k = 0$ we have $h'(0) = [\ln(1)] = 0$.
- $s(j, k) = k^j \mod p$. This function is not suitable as a probing function, because for the keys $k = 0$ and $k = 1$, the function $s(j, k)$ has constant value of 0 and 1.
- $s(j, k) = ((k \cdot j) \mod q) + 1$. This function is also not suitable as a probing function because its value is constant 1 if the key $k$ is a multiple of $q$. Moreover, for all other keys, the image of $s(j, k)$ is $\{1, \ldots, q\}$, i.e., $p - q$ addresses of the hash table can not be reached.