Exercise 4.1  Advanced Search Trees.

In addition to the value of the key, we store at each node $v$ of the tree the number $g_v$ of even elements that are in the left subtree of $v$.

When asked for the number of elements that are even and smaller than $k$, we look for the element $k$ in the tree. Every time we go to the right subtree of a node $v$ (in the case where $k$ is greater than the key in $v$), we increment a counter by $g_v$ (initially the counter is zero), or by $g_v + 1$ if $v$ is even and smaller than $k$. If we go to the left subtree, we keep the counter unchanged. If we find $k$ in a node $v$, then we add one last time $g_v$ to the counter. Otherwise, we end up in a leaf $v$ of the tree. Then, if the key of $v$ is greater or equal than $k$, the counter is already set to the correct number. Otherwise, we increase the counter by one if the key in $v$ is even.

To answer how many even numbers are between $k_1$ and $k_2$ in the tree, we need to determine only the values $L_1, L_2$ of even elements that are smaller then $k_1$ and $k_2$ respectively. We have just seen how to find these values. Subtracting $L_1$ from $L_2$, we obtain how many even numbers between $k_1 - 1$ and $k_2$ are in the tree. If $k_1$ exists in the tree and is even, we have to remove one from this number. We can easily check for this case when computing $L_1$.

The insertion of an odd element $i$ works as usual, and we set $g_i ← 0$. If $i$ is even, additionally, whenever we meet a node $v$ containing a greater key, we have to increase $g_v$ by one (since $i$ is inserted in its left subtree).

Also the removal of an element $i$ works as usual if $i$ is not present in the tree. Otherwise, if $i$ is even, during the traversal of the tree we decrease the value $g_v$ for every node $v$ at which we go in the left subtree. If we find $i$, then we look for its symmetrical successor $n$, that will take the position of $i$. If $n$ is even, we decrease the counters on the path from $i$ to $n$ at every ,,left turn“. Due to the properties of the symmetric successor, it holds that $g_n = 0$. We remove $i$, and we set $n$ in the old position of $i$. We set $g_n ← g_i$, since $n$ was originally in the right subtree of $i$.

All new tree operations require an additional effort that is constant, therefore there is no impact on the asymptotic running time of Insert, Search and Remove.

Exercise 4.2  Traversal Rules for Trees.

a) The sequence generated when the tree is visited using the pre-order traversal is: 9, 5, 4, 2, 1, 3, 7, 6, 8, 20, 13, 24.

b) The binary search tree with the given post-order traversal is:

```
       5
      / 
     3   8
    /   / 
   2  4 6  9
  /   / 
 7  10
```

*Hint:* A simple way to construct this tree is to insert all the keys in the sequence into an initially empty binary search tree in reverse order, i.e. 5, 8, 9, . . .
Exercise 4.3  AVL-Trees (Part of an exam in August 2010).

After the insertion of 6:

After the removal of 23 — using the symmetric predecessor:

After the removal of 23 — using the symmetric successor:

Exercise 4.4  Blum Median.

a) We divide the first sequence in \(\lceil N/5 \rceil\) groups of exactly 5 elements, and one group with 2 elements:

\[7, 12, 17, 3, 10\quad 1, 6, 2, 4, 8\quad 11, 9, 9, 6, 5\quad 14, 20, 13, 1, 7\quad 19, 8.\]

The first recursive call is on the medians of these groups (for 19, 8 the median is, by definition of the algorithm, 19):

\[10, 4, 9, 13, 19\]
The result of this call is the median-of-medians 10. We use 10 as Pivot, and we perform a pivoting step similarly to the one in Quicksort:

\[ 7, 8, 7, 3, 1, 6, 2, 4, 8, 1, 9, 9, 6, 5 \quad 10 \quad 14, 20, 13, 11, 17, 19, 12. \]

Since the first sequence has more elements than the second one (and we look for the element in position \( i = \lceil N/2 \rceil \)), the second recursive call of the procedure is made on sequence:

\[
\begin{array}{c}
7, 8, 7, 3, 1, 6, 2, 4, 8, 1, 9, 9, 6, 5
\end{array}
\]

Note: Depending on the pivoting step, the elements of the sequence could be in a different order.

b) The first recursive call of the procedure Selection is always invoked on \( \lceil N/5 \rceil \) elements.

In the best case, the median-of-medians is exactly the median that we are looking for, then the second call is not needed at all. If the median-of-medians is exactly „next“ to the pivot, the second recursive call is on \( \lceil N/2 \rceil \) elements.

In the worst case, we have at least \( 3 \left( \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil - 2 \right) + 2 + 1 \) elements smaller (or greater) than the median-of-medians \( 3 \left( \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil - 2 \right) \) groups of 5 contribute 3 each, the group of the pivot contributes 2, the group with fewer than 5 elements might only contribute 1). The number of elements in the second recursive call in this case is then:

\[
N - 3 \left( \frac{1}{2} \left\lceil \frac{N}{5} \right\rceil - 1 \right) \approx \frac{7}{10} N + 3.
\]