Algorithm I (Treap Insertion). Given a set of nodes which form a treap \( T \), and a key to insert \( K \), this algorithm will insert the node into the treap while maintaining it’s heap properties. Each node is assumed to contain \( \text{KEY} \), \( \text{PRIO} \), \( \text{LLINK} \), \( \text{RLINK} \), and \( \text{PARENT} \) fields. For any given node \( N \), \( \text{KEY}(N) \) gives the key field of \( N \), \( \text{PRIO}(N) \) gives the priority field of \( N \), \( \text{LLINK}(N) \) and \( \text{RLINK}(N) \) are pointers to \( N \)’s left and right subtrees, respectively, and \( \text{PARENT}(N) \) is a pointer to the node of which \( N \) is a subtree. Any or all of these three link fields may be \( \Lambda \), which for \( \text{LLINK}(N) \) respectively, and \( \text{PARENT}(N) \) is a pointer to the node of which \( N \) is a subtree. You can find an implementation of this algorithm, as well as many others, in libdict, which is available on the web at http://www.crazycoder.org/libdict.html.

1. [Initialize.] Set \( N \leftarrow \text{ROOT}(T) \), \( P \leftarrow \Lambda \).
2. [Find insertion point.] If \( N = \Lambda \), go to step I3. If \( K = \text{KEY}(N) \), the key is already in the treap and the algorithm terminates with an error. Set \( P \leftarrow N \); if \( K < \text{KEY}(N) \), then set \( N \leftarrow \text{LLINK}(N) \), otherwise set \( N \leftarrow \text{RLINK}(N) \). Repeat this step.
3. [Insert.] Set \( N \leftarrow \text{AVAIL} \). If \( N = \Lambda \), the algorithm terminates with an out of memory error. Set \( \text{KEY}(N) \leftarrow K \), \( \text{LLINK}(N) \leftarrow \text{RLINK}(N) \leftarrow \Lambda \), and \( \text{PARENT}(N) \leftarrow P \). Set \( \text{PRIO}(N) \) equal to a random integer. If \( P = \Lambda \), set \( \text{ROOT}(T) \leftarrow N \), and go to step I5. If \( K < \text{KEY}(P) \), set \( \text{LLINK}(P) \leftarrow N \); otherwise, set \( \text{RLINK}(P) \leftarrow N \).
4. [Sift up.] If \( P = \Lambda \) or \( \text{PRIO}(P) < \text{PRIO}(N) \), go to step I5. If \( \text{LLINK}(P) = N \), rotate \( P \) right; otherwise, rotate \( P \) left. Then set \( P \leftarrow \text{PARENT}(N) \), and repeat this step.
5. [All done.] The algorithm terminates successfully.

Rotations

Algorithm R (Right Rotation). Given a treap \( T \) and a node in the treap \( N \), this routine will rotate \( N \) right.

1. [Do the rotation.] Set \( L \leftarrow \text{LLINK}(N) \) and \( \text{LLINK}(N) \leftarrow \text{RLINK}(L) \). If \( \text{RLINK}(L) \neq \Lambda \), then set \( \text{PARENT}(	ext{RLINK}(L)) \leftarrow N \). Set \( P \leftarrow \text{PARENT}(N) \), \( \text{PARENT}(L) \leftarrow P \). If \( P = \Lambda \), then set \( \text{ROOT}(T) \leftarrow L \); if \( P \neq \Lambda \) and \( \text{LLINK}(P) = N \), set \( \text{LLINK}(P) \leftarrow L \), otherwise set \( \text{RLINK}(P) \leftarrow L \). Finally, set \( \text{RLINK}(L) \leftarrow N \), and \( \text{PARENT}(N) \leftarrow L \).

The code for a left rotation is symmetric. At the risk of being repetitive, it appears below.

Algorithm L (Left Rotation). Given a treap \( T \) and a node in the treap \( N \), this routine will rotate \( N \) left.

1. [Do the rotation.] Set \( R \leftarrow \text{RLINK}(N) \) and \( \text{RLINK}(N) \leftarrow \text{LLINK}(R) \). If \( \text{LLINK}(R) \neq \Lambda \), then set \( \text{PARENT}(\text{LLINK}(R)) \leftarrow N \). Set \( P \leftarrow \text{PARENT}(N) \), \( \text{PARENT}(R) \leftarrow P \). If \( P = \Lambda \), then set \( \text{ROOT}(T) \leftarrow R \); if \( P \neq \Lambda \) and \( \text{LLINK}(P) = N \), set \( \text{LLINK}(P) \leftarrow R \), otherwise set \( \text{RLINK}(P) \leftarrow R \). Finally, set \( \text{LLINK}(R) \leftarrow N \), and \( \text{PARENT}(N) \leftarrow R \).