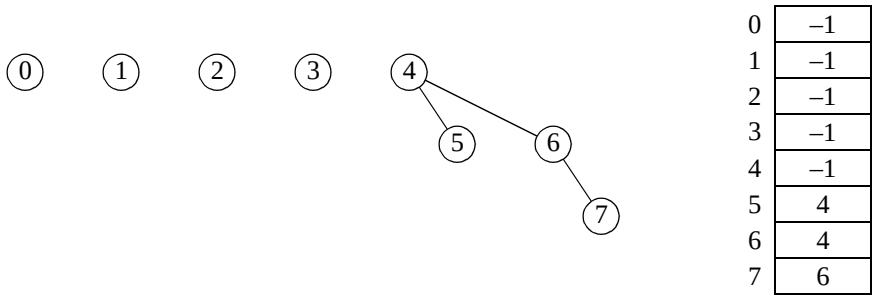

Figure 23.10 The forest after the `union` of trees with roots 4 and 6

23.4.1 Smart Union Algorithms

The previous `union`s were performed rather arbitrarily by making the second tree a subtree of the first. A simple improvement is always to make the smaller tree a subtree of the larger, breaking ties by any method; this approach is called *union-by-size*. The three `union`s in the preceding section were all ties, so we can consider that they were performed by size. If the next operation is `union`(3, 4), then the forest in Figure 23.11 will form. Had the size heuristic not been used, a deeper forest would have been formed (three nodes rather than one would have been one level deeper).

Union-by-size guarantees logarithmic finds.

We can prove that if the `union` operation is done by size, the depth of any node is never more than $\log N$. To see this, note that a node is initially at depth 0. When its depth increases as a result of a `union`, it is placed in a tree that is at least twice as large as before. Thus its depth can be increased at most $\log N$ times. (We used this argument in the quick-find algorithm in Section 23.3.) This implies that the running time for a `find` operation is $O(\log N)$ and a sequence of M operations takes at most $O(M \log N)$. The tree in Figure 23.12 shows the worst tree possible after 15 `union`s and is obtained if all `union`s are between equal-sized trees. (The worst-case tree is called a *binomial tree*. Binomial trees have other applications in advanced data structures.)

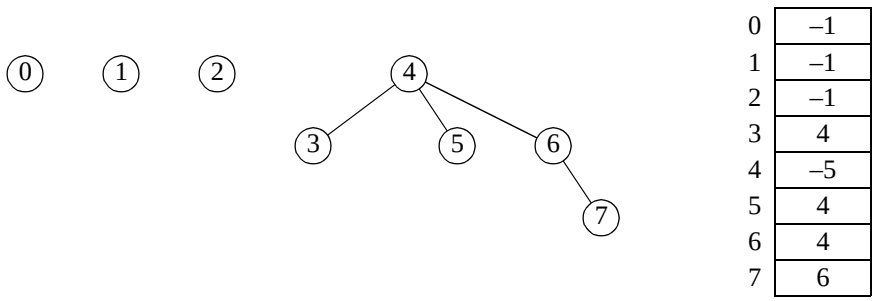

Figure 23.11 The forest formed by union-by-size, with the sizes encoded as negative numbers