合并操作基本上是通过将两个队列加到一起来完成的。令 H_3 是新的二项队列。由于 H_1 没有高度为 0 的二项树而 H_2 有，因此我们就用 H_2 中高度为 0 的二项树作为 H_3 的一部分。然后，将两个高度为 1 的二项树相加。由于 H_1 和 H_2 都有高度为 1 的二项树，因此可以将它们合并，让大的根成为小的根的子树，从而建立高度为 2 的二项树，见图 6-37。这样，H_3 将没有高度为 1 的二项树。现在存在 3 棵高度为 2 的二项树，即 H_1 和 H_2 原有的两棵二项树以及由上一步形成的一棵二项树。我们将一棵高度为 2 的二项树放到 H_3 中并合并其他两个二项树，得到一棵高度为 3 的二项树。由于 H_1 和 H_2 都没有高度为 3 的二项树，因此该二项树就成为 H_3 的一部分，合并结束。最后得到的二项队列如图 6-38 所示。

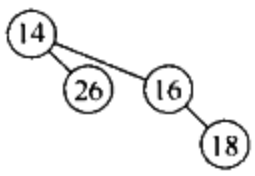

图 6-37 H_1 和 H_2 中两棵 B_1 树合并

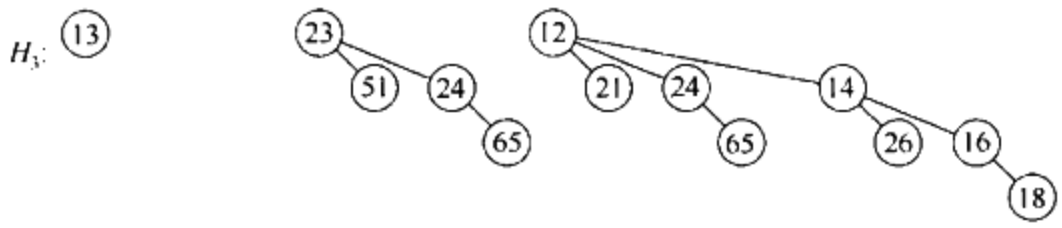

图 6-38 二项队列 H_3：合并 H_1 和 H_2 的结果

由于几乎使用任意合理的实现方法合并两棵二项树均花费常数时间，而总共存在 $O(\log N)$ 棵二项树，因此合并操作在最坏情形下花费时间 $O(\log N)$。为使该操作更有效，我们需要将这些树放到按照高度排序的二项队列中，当然这是一项简单的操作。

插入实际上就是特殊情形的合并，因为我们只要创建一棵单节点树并执行一次合并即可。这种操作的最坏情形运行时间也是 $O(\log N)$。更准确地说，如果元素将要插入的那个优先队列中不存在的最小的二项树是 B_i，那么运行时间与 $i+1$ 成正比。例如，H_3(见图 6-38)缺少高度为 1 的二项树，因此插入将进行两步终止。由于二项队列中的每棵树均以概率 1/2 出现，于是我们预计插入在两步后终止，因此，平均时间是常数。不仅如此，分析将指出，对一个初始为空的二项队列进行 N 次 insert 将花费 $O(N)$最坏情形时间。事实上，只用 $N-1$ 次比较就有可能进行该操作；我们把它留作练习。

作为一个例子，我们用图 6-39 到图 6-45 演示通过依序插入 1 到 7 来构成一个二项队列。4 的插入展现一种坏的情形。我们把 4 与 B_0 合并，得到一棵新的高度为 1 的树。然后将该树与 B_1 合并，得到一棵高度为 2 的树，它是新的优先队列。我们把这些算作 3 步(两棵树合并加上终止情形)。在插入 7 以后的下一次插入又是一个坏情形，需要 3 次树的合并操作。

图 6-39 在 1 插入之后

图 6-40 在 2 插入之后

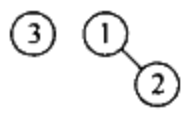

图 6-41 在 3 插入之后

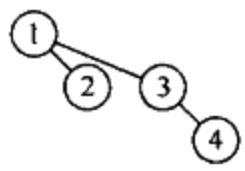

图 6-42 在 4 插入之后

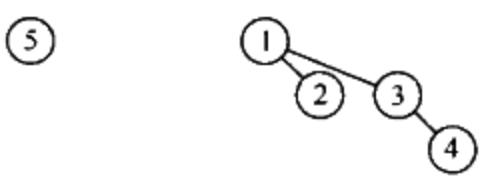

图 6-43 在 5 插入之后

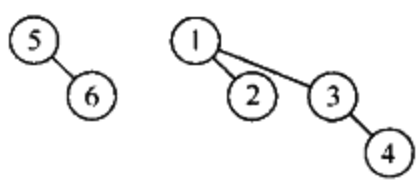

图 6-44 在 6 插入之后

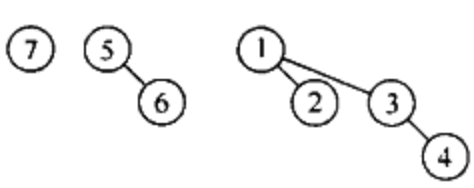

图 6-45 在 7 插入之后

deleteMin 可以通过首先找出一棵具有最小根的二项树来完成。令该树为 B_k，并令原始的优先队列为 H。我们从 H 的树的森林中除去二项树 B_k，形成新的二项树队列 H'。再除去 B_k 的根，得到一些二项树 B_0，B_1，...，B_{k-1}，它们共同形成优先队列 H''。合并 H' 和 H''，操作结束。