```
1       /**
2        * Remove the smallest item from the priority queue.
3        * @return the smallest item, or throw UnderflowException if empty.
4        */
5       public AnyType deleteMin( )
6       {
7           if( isEmpty( ) )
8               throw new UnderflowException( );
9
10          AnyType minItem = root.element;
11          root = merge( root.left, root.right );
12
13          return minItem;
14      }
```

图 6-30 左式堆的 deleteMin 例程

## 6.7 斜堆

斜堆(skew heap)是左式堆的自调节形式，实现起来极其简单。斜堆和左式堆间的关系类似于伸展树和 AVL 树间的关系。斜堆是具有堆序的二叉树，但不存在对树的结构限制。不同于左式堆，关于任意节点的零路径长的任何信息都不再保留。斜堆的右路径在任何时刻都可以任意长，因此，所有操作的最坏情形运行时间均为 $O(N)$。然而，正如伸展树一样，可以证明(见第 11 章)对任意 $M$ 次连续操作，总的最坏情形运行时间是 $O(M \log N)$。因此，斜堆每次操作的摊还开销(amortized cost)为 $O(\log N)$。

与左式堆相同，斜堆的基本操作也是合并操作。merge 例程还是递归的，我们执行与以前完全相同的操作，但有一个例外，即：对于左式堆，我们查看是否左儿子和右儿子满足左式堆结构性质，并在不满足该性质时将它们交换。但对于斜堆，交换是无条件的，除那些右路径上所有节点的最大者不交换它的左右儿子的例外外，我们都要进行这种交换。这个例外就是在自然递归实现时所发生的情况，因此它实际上根本不是特殊情形。此外，证明时间界也是不必要的，但是，由于这样的节点肯定没有右儿子，因此执行交换是不明智的(在我们的例子中，该节点没有儿子，因此我们不必为此担心)。另外，仍设我们的输入是与前面相同的两个堆，见图 6-31。

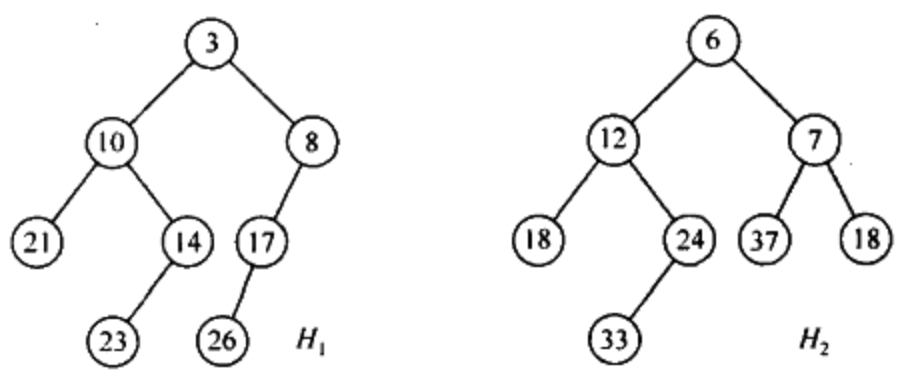


图 6-31 两个斜堆 $H_1$ 和 $H_2$

如果我们递归地将 $H_2$ 与 $H_1$ 的根在 8 处的子堆合并，那么将得到图 6-32 中的堆。

这又是递归完成的，因此，根据递归的第三个法则(1.3 节)我们不必担心它是如何得到的。这个堆碰巧是左式的，不过不能保证情况总是如此。我们使这个堆成为 $H_1$ 的新的左儿子，而 $H_1$ 的老的左儿子变成了新的右儿子(见图 6-33)。