另一方面，用堆实现的优先队列在现代应用程序中越来越重要，因为它能在插入操作和删除最大元素操作混合的动态场景中保证对数级别的运行时间。我们会在本书后续章节见到更多的例子。 326 ~ 327

答疑

问 我还是不明白优先队列是做什么用的。为什么我们不直接把元素排序然后再一个个地引用有序数组中的元素？

答 在某些数据处理的例子里，比如 TopM 和 Multiway，总数据量太大，无法排序（甚至无法全部装进内存）。如果你需要从 10 亿个元素中选出最大的十个，你真的想把一个 10 亿规模的数组排序吗？但有了优先队列，你就只用一个能存储十个元素的队列即可。在其他的例子中，我们甚至无法同时获取所有的数据，因此只能先从优先队列中取出并处理一部分，然后再根据结果决定是否向优先队列中添加更多的数据。

问 为什么不像我们在其他排序算法中那样使用 Comparable 接口，而在 MaxPQ 中使用泛型的 Item 呢？

答 这么做的话 delMax() 的用例就需要将返回值转换为某种具体的类型，比如 String。一般来说，应该尽量避免在用例中进行类型转换。

问 为什么在堆的表示中不使用 a[0] ？

答 这么做可以稍稍简化计算。实现从 0 开始的堆并不困难，a[0] 的子结点是 a[1] 和 a[2]，a[1] 的子结点是 a[3] 和 a[4]，a[2] 的子结点是 a[5] 和 a[6]，以此类推。但大多数程序员更喜欢我们的简单方法。另外，将 a[0] 的值用作哨兵（作为 a[1] 的父结点）在某些堆的应用中很有用。

问 在我看来，在堆排序中构造堆时，逐个向堆中添加元素比 2.4.5.1 节中描述的由底向上的复杂方法更简单。为什么要这么做？

答 对于一个排序算法来说，这么做能够快上 20%，而且所需的代码更少（不会用到 swim() 函数）。理解算法的难度并不一定与它的简洁性或者效率相关。

问 如果我去掉 MaxPQ 的实现中的 extends Comparable<Key> 这句话会怎样？

答 和以前一样，回答这类问题的最简单的办法就是你自己直接试试。如果这么做 MaxPQ 会报出一个编译错误：

```
MaxPQ.java:21: cannot find symbol
symbol  : method compareTo(Item)
```

Java 这样告诉你它不知道 Item 对象的 compareTo() 方法，因为你没有声明 Item extends Comparable<Item>。 328

练习

2.4.1 用序列 P R I O * R * * I * T * Y * * * Q U E * * * U * E （字母表示插入元素，星号表示删除最大元素）操作一个初始为空的优先队列。给出每次删除最大元素返回的字符。

2.4.2 分析以下说法：要实现在常数时间找到最大元素，为何不用一个栈或队列，然后记录已插入的最大元素并在找出最大元素时返回它的值？

2.4.3 用以下数据结构实现优先队列，支持插入元素和删除最大元素的操作：无序数组、有序数组、无序链表和链表。将你的 4 种实现中每种操作在最坏情况下的运行时间上下限制成一张表格。

2.4.4 一个按降序排列的数组也是一个面向最大元素的堆吗？

2.4.5 将EASYQUESTION顺序插入一个面向最大元素的堆中，给出结果。

2.4.6 按照练习2.4.1的规则，用序列P R I O * R * * I * T * Y * * * Q U E * * * U * E操作一个初始为空的面向最大元素的堆，给出每次操作后堆的内容。

2.4.7 在堆中，最大的元素一定在位置1上，第二大的元素一定在位置2或者3上。对于一个大小为31的堆，给出第k大的元素可能出现的位置和不可能出现的位置，其中k=2、3、4（设元素值不重复）。

2.4.8 回答上一道练习中第k小元素的可能和不可能的位置。

2.4.9 给出A B C D E五个元素可能构造出来的所有堆，然后给出A A A B B这五个元素可能构造出来的所有堆。

2.4.10 假设我们不想浪费堆有序的数组pq[]中的那个位置，将最大的元素放在pq[0]，它的子结点放在pq[1]和pq[2]，以此类推。pq[k]的父结点和子结点在哪里？

329 2.4.11 如果你的应用中有大量的插入元素的操作，但只有若干删除最大元素操作，哪种优先队列的实现方法更有效：堆、无序数组、有序数组？

2.4.12 如果你的应用场景中大量的找出最大元素的操作，但插入元素和删除最大元素操作相对较少，哪种优先队列的实现方法更有效：堆、无序数组、有序数组？

2.4.13 想办法在sink()中避免检查j < N。

2.4.14 对于没有重复元素的大小为N的堆，一次删除最大元素的操作中最少要交换几个元素？构造一个能够达到这个交换次数的大小为15的堆。连续两次删除最大元素呢？三次呢？

2.4.15 设计一个程序，在线性时间内检测数组pq[]是否是一个面向最小元素的堆。

2.4.16 对于N=32，构造数组使得堆排序使用的比较次数最多以及最少。

2.4.17 证明：构造大小为k的面向最小元素的优先队列，然后进行N–k次替换最小元素操作（删除最小元素后再插入元素）后，N个元素中的前k大元素均会留在优先队列中。

2.4.18 在MaxPQ中，如果一个用例使用insert()插入了一个比队列中的所有元素都大的新元素，随后立即调用delMax()。假设没有重复元素，此时的堆和进行这些操作之前的堆完全相同吗？进行两次insert()（第一次插入一个比队列所有元素都大的元素，第二次插入一个更大的元素）操作接两次delMax()操作呢？

2.4.19 实现MaxPQ的一个构造函数，接受一个数组作为参数。使用正文2.4.5.1节中所述的自底向上的方法构造堆。

330 2.4.20 证明：基于下沉的堆构造方法使用的比较次数小于$2N$，交换次数小于N。

提高题

2.4.21 基础数据结构。说明如何使用优先队列实现第1章中的栈、队列和随机队列这几种数据结构。

2.4.22 调整数组大小。在MaxPQ中加入调整数组大小的代码，并和命题Q一样证明对于一般性长度为N的队列其数组访问的上限。

2.4.23 Multiway的堆。只考虑比较的成本且假设找到t个元素中的最大者需要t次比较，在堆排序中使用t向堆的情况下找出使比较次数$N\lg N$的系数最小的t值。首先，假设使用的是一个简单通用的sink()方法；其次，假设Floyd方法在内循环中每轮可以节省一次比较。

2.4.24 使用链接的优先队列。用堆有序的二叉树实现一个优先队列，但使用链表结构代替数组。每个结点都需要三个链接：两个向下，一个向上。你的实现即使在无法预知队列大小的情况下也能保证优先队列的基本操作所需的时间为对数级别。

2.4.25 计算数论。编写程序 CubeSum.java，在不使用额外空间的条件下，按大小顺序打印所有 a^3+b^3 的结果，其中 a 和 b 为 0 至 N 之间的整数。也就是说，不要全部计算 N^2 个和然后排序，而是创建一个最小优先队列，初始状态为 $(0^3, 0, 0),(1^3, 1, 0),(2^3, 2, 0),\cdots,(N^3, N, 0)$。这样只要优先队列非空，删除并打印最小的元素 (i^3+j^3, i, j)。然后如果 $j<N$，插入元素 $(i^3+(j+1)^3, i, j+1)$。用这段程序找出 0 到 10^6 之间所有满足 $a^3+b^3=c^3+d^3$ 的不同整数 a,b,c,d。

2.4.26 无需交换的堆。因为 sink() 和 swim() 中都用到了初级函数 exch()，所以所有元素都被多加载并存储了一次。回避这种低效方式，用插入排序给出新的实现（请见练习 2.1.25）。

2.4.27 找出最小元素。在 MaxPQ 中加入一个 min() 方法。你的实现所需的时间和空间都应该是常数。

2.4.28 选择过滤。编写一个 TopM 的用例，从标准输入读入坐标 (x, y, z)，从命令行得到值 M，然后打印出距离原点的欧几里德距离最小的 M 个点。在 $N=10^8$ 且 $M=10^4$ 时，预计程序的运行时间。 331

2.4.29 同时面向最大和最小元素的优先队列。设计一个数据类型，支持如下操作：插入元素、删除最大元素、删除最小元素（所需时间均为对数级别），以及找到最大元素、找到最小元素（所需时间均为常数级别）。提示：用两个堆。

2.4.30 动态中位数查找。设计一个数据类型，支持在对数时间内插入元素，常数时间内找到中位数并在对数时间内删除中位数。提示：用一个面向最大元素的堆再用一个面向最小元素的堆。

2.4.31 快速插入。用基于比较的方式实现 MinPQ 的 API，使得插入元素需要 ~ loglog$N$ 次比较，删除最小元素需要 ~2log$N$ 次比较。提示：在 swim() 方法中用二分查找来寻找祖先结点。

2.4.32 下界。请证明，不存在一个基于比较的对 MinPQ 的 API 的实现能够使得插入元素和删除最小元素的操作都保证只使用 ~Nloglog$N$ 次比较。

2.4.33 索引优先队列的实现。按照 2.4.4.6 节的描述修改算法 2.6 来实现索引优先队列 API 中的基本操作：使用 pq[] 保存索引，添加一个数组 keys[] 来保存元素，再添加一个数组 qp[] 来保存 pq[] 的逆序——qp[i] 的值是 i 在 pq[] 中的位置（即索引 j，pq[j]=i）。修改算法 2.6 的代码来维护这些数据结构。若 i 不在队列之中，则总是令 qp[i] = -1 并添加一个方法 contains() 来检测这种情况。你需要修改辅助函数 exch() 和 less()，但不需要修改 sink() 和 swim()。 332

部分答案：

```
public class IndexMinPQ<Key extends Comparable<Key>>
{
   private int N;           // PQ中的元素数量
   private int[] pq;        // 索引二叉堆，由1开始
   private int[] qp;        // 逆序：qp[pq[i]] = pq[qp[i]] = i
   private Key[] keys;      // 有优先级之分的元素
   public IndexMinPQ(int maxN)
   {
      keys = (Key[]) new Comparable[maxN + 1];
      pq   = new int[maxN + 1];
      qp   = new int[maxN + 1];
      for (int i = 0; i <= maxN; i++) qp[i] = -1;
   }

   public boolean isEmpty()
   {  return N == 0;  }

   public boolean contains(int k)
   {  return qp[k] != -1;  }
```

```
   public void insert(int k, Key key)
   {
      N++;
      qp[k] = N;
      pq[N] = k;
      keys[k] = key;
      swim(N);
   }
   public Key min()
   {  return keys[pq[1]];  }
   public int delMin()
   {
      int indexOfMin = pq[1];
      exch(1, N--);
      sink(1);
      keys[pq[N+1]] = null;
      qp[pq[N+1]] = -1;
      return indexOfMin;
   }
}
```

333

2.4.34 索引优先队列的实现（附加操作）。向练习 2.4.33 的实现中添加 minIndex()、change() 和 delete() 方法。

解答：

```
public int minIndex()
{  return pq[1];  }
public void change(int k, Key Key)
{
   keys[k] = key;
   swim(qp[k]);
   sink(qp[k]);
}
public void delete(int k)
{
   exch(k, N--);
   swim(qp[k]);
   sink(qp[k]);
   keys[pq[N+1]] = null;
   qp[pq[N+1]] = -1;
}
```

2.4.35 离散概率分布的取样。编写一个 Sample 类，其构造函数接受一个 double 类型的数组 p[] 作为参数并支持以下操作：random()——返回任意索引 i 及其概率 p[i]/T（T 是 p[] 中所有元素之和）；change(i, v)——将 p[i] 的值修改为 v。提示：使用完全二叉树，每个结点对应一个权重 p[i]。在每个结点记录其下子树的权重之和。为了产生一个随机的索引，取 0 到 T 之间的一个随机数并根据各个结点的权重之和来判断沿着哪条子树搜索下去。在更新 p[i] 时，同时更新从根结点到 i 的路径上的所有结点。不要像堆的实现那样显式使用指针。

334

实验题

2.4.36 性能测试 I。编写一个性能测试用例，用插入元素操作填满一个优先队列，然后用删除最大元素操作删去一半元素，再用插入元素操作填满优先队列，再用删除最大元素操作删去所有元素。用一列随机的长短不同的元素多次重复以上过程，测量每次运行的用时，打印平均用时或是将其绘制成图表。

2.4.37 性能测试 II。编写一个性能测试用例，用插入元素操作填满一个优先队列，然后在一秒钟之内尽可能多地连续反复调用删除最大元素和插入元素的操作。用一列随机的长短不同的元素多次重复以上过程，将程序能够完成的删除最大元素操作的平均次数打印出来或是绘成图表。

2.4.38 练习测试。编写一个练习用例，用算法 2.6 中实现的优先队列的接口方法处理实际应用中可能出现的高难度或是极端情况。例如，元素已经有序、元素全部逆序、元素全部相同或是所有元素只有两个值。

2.4.39 构造函数的代价。对于 $N=10^3$、10^6 和 10^9，根据经验判断堆排序时构造堆占总耗时的比例。

2.4.40 Floyd 方法。根据正文中 Floyd 的先沉后浮思想实现堆排序。对于 $N=10^3$、10^6 和 10^9 大小的随机不重复数组，记录你的程序所使用的比较次数和标准实现所使用的比较次数。

2.4.41 Multiway 堆。根据正文中的描述实现基于完全堆有序的三叉树和四叉树的堆排序。对于 $N=10^3$、10^6 和 10^9 大小的随机不重复数组，记录你的程序所使用的比较次数和标准实现所使用的比较次数。

2.4.42 堆的前序表示。用前序法而非级别表示一棵堆有序的树，并基于此实现堆排序。对于 $N=10^3$、10^6 和 10^9 大小的随机不重复数组，记录你的程序所使用的比较次数和标准实现所使用的比较次数。

335