么说都是非常必要的。而且，在适用的情况下，关键在于计算复杂度会影响优秀软件的开发。首先，准确的上界为软件工程师保证性能提供了空间。很多例子表明，平方级别排序的性能低于线性排序。其次，准确的下界可以为我们节省很多时间，避免因不可能的性能改进而投入资源。

但归并排序的最优性并不是结束，也不代表在实际应用中我们不会考虑其他的方法了，因为本节中的理论还是有许多局限性的，例如：

- 归并排序的空间复杂度不是最优的；
- 在实践中不一定会遇到最坏情况；
- 除了比较，算法的其他操作（例如访问数组）也可能很重要；
- 不进行比较也能将某些数据排序。

因此在本书中我们还将继续学习其他一些排序算法。 282

答疑

问 归并排序比希尔排序快吗？

答 在实际应用中，它们的运行时间之间的差距在常数级别之内（希尔排序使用的是像算法 2.3 中那样的经过验证的递增序列），因此相对性能取决于具体的实现。

```
% java SortCompare Merge Shell 100000
For 100000 random Double values
    Merge is 1.2 times faster than Shell
```

理论上来说，还没有人能够证明希尔排序对于随机数据的运行时间是线性对数级别的，因此存在平均情况下希尔排序的性能的渐进增长率[①]更高的可能性。在最坏情况下，这种差距的存在已经被证实了，但这对实际应用没有影响。

问 为什么不把数组 `aux[]` 声明为 `merge()` 方法的局部变量？

答 这是为了避免每次归并时，即使是归并很小的数组，都创建一个新的数组。如果这么做，那么创建新数组将成为归并排序运行时间的主要部分（请见练习 2.2.26）。更好的解决方案是将 `aux[]` 变为 `sort()` 方法的局部变量，并将它作为参数传递给 `merge()` 方法（为了简化代码我们没有在例子中这么做，请见练习 2.2.9）。

问 当数组中存在重复的元素时归并排序的表现如何？

答 如果所有的元素都相同，那么归并排序的运行时间将是线性的（需要一个额外的测试来避免归并已经有序的数组）。但如果有多个不同的重复值，这样做的性能收益就不是很明显了。例如，假设输入数组的 N 个奇数位上的元素都是同一个值，另外 N 个偶数位上的元素都是另一个值，此时算法的运行时间是线性对数的（这样的数组和所有元素都不重复的数组满足了相同的循环条件），而非线性的。 283

练习

2.2.1 按照本节开头所示轨迹的格式给出原地归并的抽象 `merge()` 方法是如何将数组 `A E Q S U Y E I N O S T` 排序的。

① 即运行时间的近似函数。——译者注

2.2.2 按照算法 2.4 所示轨迹的格式给出自顶向下的归并排序是如何将数组 E A S Y Q U E S T I O N 排序的。

2.2.3 用自底向上的归并排序解答练习 2.2.2。

2.2.4 是否当且仅当两个输入的子数组都有序时原地归并的抽象方法才能得到正确的结果？证明你的结论，或者给出一个反例。

2.2.5 当输入数组的大小 N=39 时，给出自顶向下和自底向上的归并排序中各次归并子数组的大小及顺序。

2.2.6 编写一个程序来计算自顶向下和自底向上的归并排序访问数组的准确次数。使用这个程序将 N=1 至 512 的结果绘成曲线图，并将其和上限 $6N\lg N$ 比较。

2.2.7 证明归并排序的比较次数是单调递增的（即对于 $N>0$，$C(N+1)>C(N)$）。

2.2.8 假设将算法 2.4 修改为：只要 `a[mid] <= a[mid+1]` 就不调用 `merge()` 方法，请证明用归并排序处理一个已经有序的数组所需的比较次数是线性级别的。

2.2.9 在库函数中使用 `aux[]` 这样的静态数组是不妥当的，因为可能会有多个程序同时使用这个类。实现一个不用静态数组的 `Merge` 类，但也不要将 `aux[]` 变为 `merge()` 的局部变量（请见本节的答
284 疑部分）。提示：可以将辅助数组作为参数传递给递归的 `sort()` 方法。

提高题

2.2.10 快速归并。实现一个 `merge()` 方法，按降序将 `a[]` 的后半部分复制到 `aux[]`，然后将其归并回 `a[]` 中。这样就可以去掉内循环中检测某半边是否用尽的代码。注意：这样的排序产生的结果是不稳定的（请见 2.5.1.8 节）。

2.2.11 改进。实现 2.2.2 节所述的对归并排序的三项改进：加快小数组的排序速度，检测数组是否已经有序以及通过在递归中交换参数来避免数组复制。

2.2.12 次线性的额外空间。用大小 M 将数组分为 N/M 块（简单起见，设 M 是 N 的约数）。实现一个归并方法，使之所需的额外空间减少到 $\max(M, N/M)$：(i) 可以先将一个块看做一个元素，将块的第一个元素作为块的主键，用选择排序将块排序；(ii) 遍历数组，将第一块和第二块归并，完成后将第二块和第三块归并，等等。

2.2.13 平均情况的下限。请证明任意基于比较的排序算法的预期比较次数至少为 $\sim N\lg N$（假设输入元素的所有排列的出现概率是均等的）。提示：比较次数至少是比较树的外部路径的长度（根结点到所有叶子结点的路径长度之和），当树平衡时该值最小。

2.2.14 归并有序的队列。编写一个静态方法，将两个有序的队列作为参数，返回一个归并后的有序队列。

2.2.15 自底向上的有序队列归并排序。用下面的方法编写一个自底向上的归并排序：给定 N 个元素，创建 N 个队列，每个队列包含其中一个元素。创建一个由这 N 个队列组成的队列，然后不断用练习 2.2.14 中的方法将队列的头两个元素归并，并将结果重新加入到队列结尾，直到队列的队列只剩下一个元素为止。

2.2.16 自然的归并排序。编写一个自底向上的归并排序，当需要将两个子数组排序时能够利用数组中已经有序的部分。首先找到一个有序的子数组（移动指针直到当前元素比上一个元素小为止），
285 然后再找出另一个并将它们归并。根据数组大小和数组中递增子数组的最大长度分析算法的运行时间。

2.2.17 链表排序。实现对链表的自然排序（这是将链表排序的最佳方法，因为它不需要额外的空间，且运行时间是线性对数级别的）。

2.2.18 *打乱链表*。实现一个分治算法，使用线性对数级别的时间和对数级别的额外空间随机打乱一条链表。

2.2.19 *倒置*。编写一个线性对数级别的算法统计给定数组中的“倒置”数量（即插入排序所需的交换次数，请见 2.1 节）。这个数量和 *Kendall tau* 距离有关，请见 2.5 节。

2.2.20 *间接排序*。编写一个不改变数组的归并排序，它返回一个 `int[]` 数组 `perm`，其中 `perm[i]` 的值是原数组中第 *i* 小的元素的位置。

2.2.21 *一式三份*。给定三个列表，每个列表中包含 *N* 个名字，编写一个线性对数级别的算法来判定三份列表中是否含有公共的名字，如果有，返回第一个被找到的这种名字。

2.2.22 *三向归并排序*。假设每次我们是把数组分成三个部分而不是两个部分并将它们分别排序，然后进行三向归并。这种算法的运行时间的增长数量级是多少？ 286

实验题

2.2.23 *改进*。用实验评估正文中所提到的归并排序的三项改进（请见练习 2.2.11）的效果，并比较正文中实现的归并和练习 2.2.10 所实现的归并之间的性能。根据经验给出应该在何时为子数组切换到插入排序。

2.2.24 *改进的有序测试*。在实验中用大型随机数组评估练习 2.2.8 所做的修改的效果。根据经验用 *N*（被排序的原始数组的大小）的函数描述条件语句（`a[mid] <= a[mid+1]`）成立（无论数组是否有序）的平均次数。

2.2.25 *多向归并排序*。实现一个 *k* 向（相对双向而言）归并排序程序。分析你的算法，估计最佳的 *k* 值并通过实验验证猜想。

2.2.26 *创建数组*。使用 `SortCompare` 粗略比较在你的计算机上在 `merge()` 中和在 `sort()` 中创建 `aux[]` 的性能差异。

2.2.27 *子数组长度*。用归并将大型随机数组排序，根据经验用 *N*（某次归并时两个子数组的长度之和）的函数估计当一个子数组用尽时另一个子数组的平均长度。

2.2.28 *自顶向下与自底向上*。对于 $N=10^3$、10^4、10^5 和 10^6，使用 `SortCompare` 比较自顶向下和自底向上的归并排序的性能。

2.2.29 *自然的归并排序*。对于 $N=10^3$、10^6 和 10^9，类型为 `Long` 的随机主键数组，根据经验给出自然的归并排序（请见练习 2.2.16）所需要的遍数。*提示*：不需要实现这个排序（甚至不需要生成所有完整的 64 位主键）也能完成这道练习。 287