13 | 线性排序：如何根据年龄给100万用户数据排序?

2018-10-19 王争

13 | 线性排序：如何根据年龄给100万用户数据排序?
朗读人：修阳　16′39″ | 7.63M

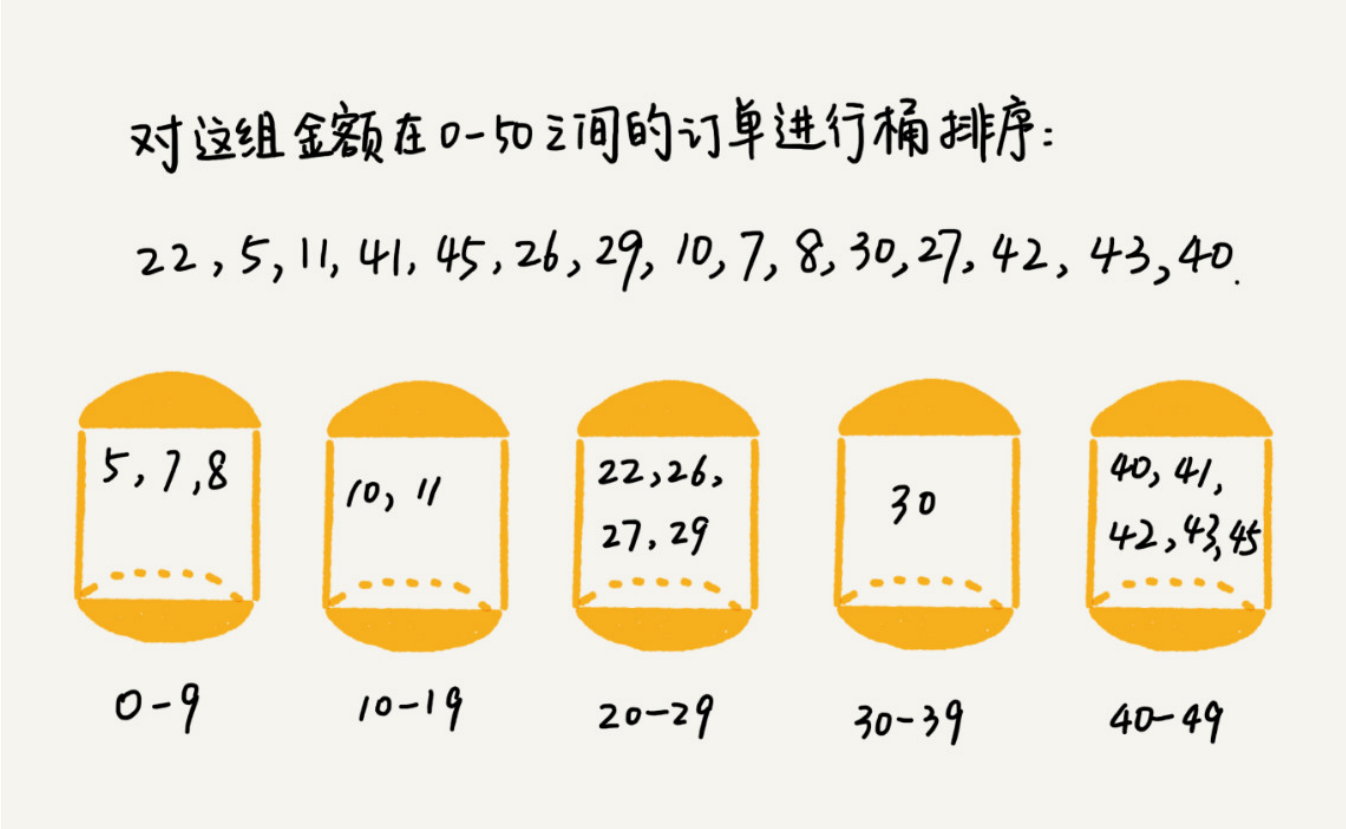
上两节中，我带你着重分析了几种常用排序算法的原理、时间复杂度、空间复杂度、稳定性等。今天，我会讲三种时间复杂度是 O(n) 的排序算法：桶排序、计数排序、基数排序。因为这些排序算法的时间复杂度是线性的，所以我们把这类排序算法叫作**线性排序**（Linear sort）。之所以能做到线性的时间复杂度，主要原因是，这三个算法是非基于比较的排序算法，都不涉及元素之间的比较操作。

这几种排序算法理解起来都不难，时间、空间复杂度分析起来也很简单，但是对要排序的数据要求很苛刻，所以我们**今天学习重点的是掌握这些排序算法的适用场景。**

按照惯例，我先给你出一道思考题：如何根据年龄给 100 万用户排序？ 你可能会说，我用上一节课讲的归并、快排就可以搞定啊！是的，它们也可以完成功能，但是时间复杂度最低也是 O(nlogn)。有没有更快的排序方法呢？让我们一起进入今天的内容！

桶排序（Bucket sort）

首先，我们来看桶排序。桶排序，顾名思义，会用到“桶”，核心思想是将要排序的数据分到几个有序的桶里，每个桶里的数据再单独进行排序。桶内排完序之后，再把每个桶里的数据按照顺序依次取出，组成的序列就是有序的了。



桶排序的时间复杂度为什么是 O(n) 呢？我们一块儿来分析一下。

如果要排序的数据有 n 个，我们把它们均匀地划分到 m 个桶内，每个桶里就有 k=n/m 个元素。每个桶内部使用快速排序，时间复杂度为 O(k * logk)。m 个桶排序的时间复杂度就是 O(m * k * logk)，因为 k=n/m，所以整个桶排序的时间复杂度就是 O(n*log(n/m))。当桶的个数 m 接近数据个数 n 时，log(n/m) 就是一个非常小的常量，这个时候桶排序的时间复杂度接近 O(n)。

桶排序看起来很优秀，那它是不是可以替代我们之前讲的排序算法呢？

答案当然是否定的。为了让你轻松理解桶排序的核心思想，我刚才做了很多假设。实际上，桶排序对要排序数据的要求是非常苛刻的。

首先，要排序的数据需要很容易就能划分成 m 个桶，并且，桶与桶之间有着天然的大小顺序。这样每个桶内的数据都排序完之后，桶与桶之间的数据不需要再进行排序。

其次，数据在各个桶之间的分布是比较均匀的。如果数据经过桶的划分之后，有些桶里的数据非常多，有些非常少，很不平均，那桶内数据排序的时间复杂度就不是常量级了。在极端情况下，如果数据都被划分到一个桶里，那就退化为 O(nlogn) 的排序算法了。

桶排序比较适合用在外部排序中。所谓的外部排序就是数据存储在外部磁盘中，数据量比较大，内存有限，无法将数据全部加载到内存中。

比如说我们有 10GB 的订单数据，我们希望按订单金额（假设金额都是正整数）进行排序，但是我们的内存有限，只有几百 MB，没办法一次性把 10GB 的数据都加载到内存中。这个时候该怎么办呢？

现在我来讲一下，如何借助桶排序的处理思想来解决这个问题。

我们可以先扫描一遍文件，看订单金额所处的数据范围。假设经过扫描之后我们得到，订单金额最小是 1 元，最大是 10 万元。我们将所有订单根据金额划分到 100 个桶里，第一个桶我们存储金额在 1 元到 1000 元之内的订单，第二桶存储金额在 1001 元到 2000 元之内的订单，以此类推。每一个桶对应一个文件，并且按照金额范围的大小顺序编号命名（00，01，02...99）。

理想的情况下，如果订单金额在 1 到 10 万之间均匀分布，那订单会被均匀划分到 100 个文件中，每个小文件中存储大约 100MB 的订单数据，我们就可以将这 100 个小文件依次放到内存中，用快排来排序。等所有文件都排好序之后，我们只需要按照文件编号，从小到大依次读取每个小文件中的订单数据，并将其写入到一个文件中，那这个文件中存储的就是按照金额从小到大排序的订单数据了。

不过，你可能也发现了，订单按照金额在 1 元到 10 万元之间并不一定是均匀分布的 ，所以 10GB 订单数据是无法均匀地被划分到 100 个文件中的。有可能某个金额区间的数据特别多，划分之后对应的文件就会很大，没法一次性读入内存。这又该怎么办呢？

针对这些划分之后还是比较大的文件，我们可以继续划分，比如，订单金额在 1 元到 1000 元之间的比较多，我们就将这个区间继续划分为 10 个小区间，1 元到 100 元，101 元到 200 元，201 元到 300 元...901 元到 1000 元。如果划分之后，101 元到 200 元之间的订单还是太多，无法一次性读入内存，那就继续再划分，直到所有的文件都能读入内存为止。

计数排序（Counting sort）

我个人觉得，计数排序其实是桶排序的一种特殊情况。当要排序的 n 个数据，所处的范围并不大的时候，比如最大值是 k，我们就可以把数据划分成 k 个桶。每个桶内的数据值都是相同的，省掉了桶内排序的时间。

我们都经历过高考，高考查分数系统你还记得吗？我们查分数的时候，系统会显示我们的成绩以及所在省的排名。如果你所在的省有 50 万考生，如何通过成绩快速排序得出名次呢？

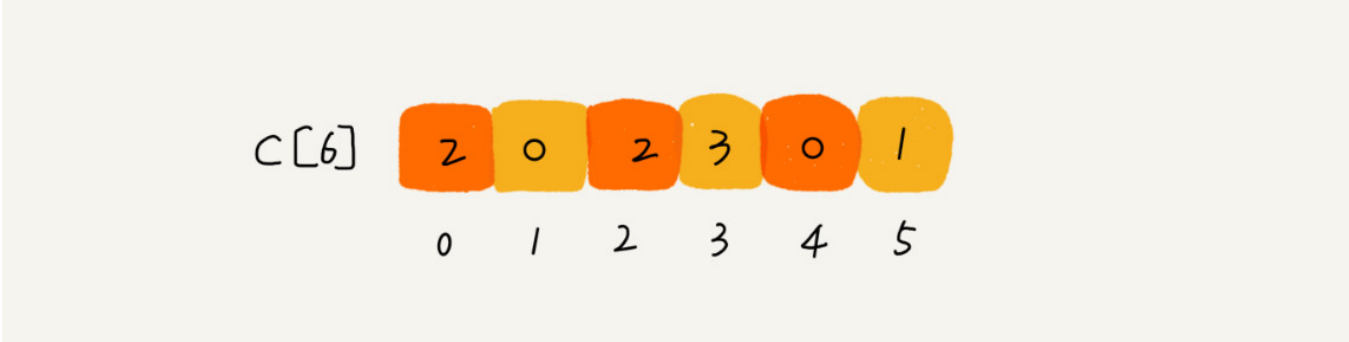
考生的满分是 900 分，最小是 0 分，这个数据的范围很小，所以我们可以分成 901 个桶，对应分数从 0 分到 900 分。根据考生的成绩，我们将这 50 万考生划分到这 901 个桶里。桶内的数据都是分数相同的考生，所以并不需要再进行排序。我们只需要依次扫描每个桶，将桶内的考生

依次输出到一个数组中，就实现了 50 万考生的排序。因为只涉及扫描遍历操作，所以时间复杂度是 O(n)。

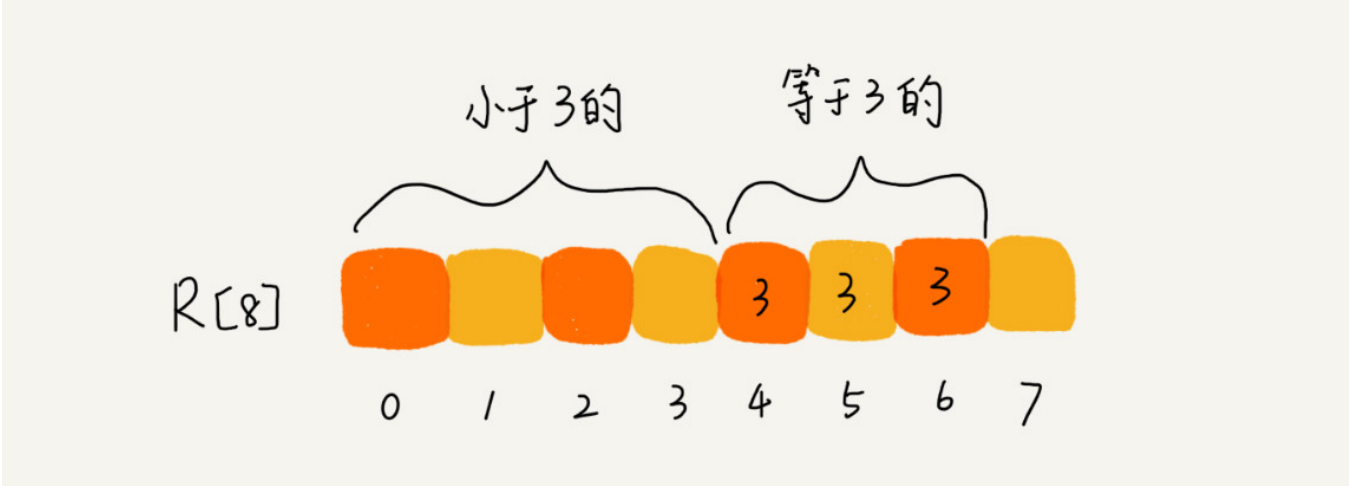
计数排序的算法思想就是这么简单，跟桶排序非常类似，只是桶的大小粒度不一样。**不过，为什么这个排序算法叫“计数”排序呢？“计数”的含义来自哪里呢？**

想弄明白这个问题，我们就要来看计数排序算法的实现方法。我还拿考生那个例子来解释。为了方便说明，我对数据规模做了简化。假设只有 8 个考生，分数在 0 到 5 分之间。这 8 个考生的成绩我们放在一个数组 A[8] 中，它们分别是：2，5，3，0，2，3，0，3。

考生的成绩从 0 到 5 分，我们使用大小为 6 的数组 C[6] 表示桶，其中下标对应分数。不过，C[6] 内存储的并不是考生，而是对应的考生个数。像我刚刚举的那个例子，我们只需要遍历一遍考生分数，就可以得到 C[6] 的值。

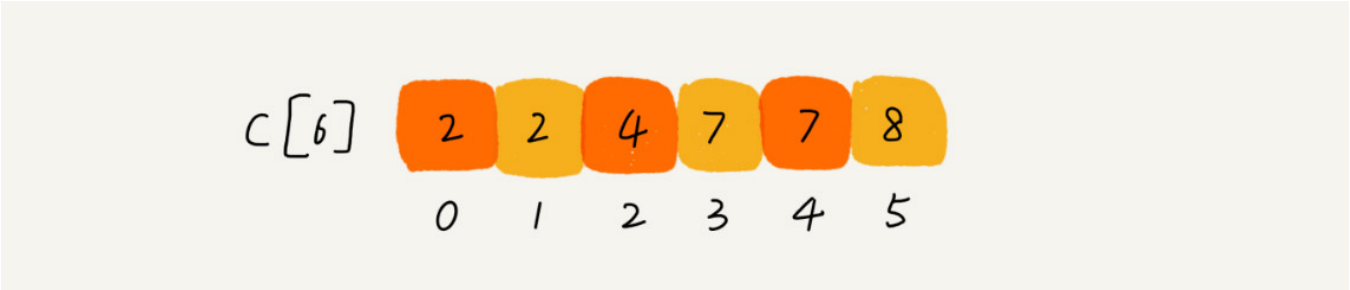


从图中可以看出，分数为 3 分的考生有 3 个，小于 3 分的考生有 4 个，所以，成绩为 3 分的考生在排序之后的有序数组 R[8] 中，会保存下标 4，5，6 的位置。



那我们如何快速计算出，每个分数的考生在有序数组中对应的存储位置呢？这个处理方法非常巧妙，很不容易想到。

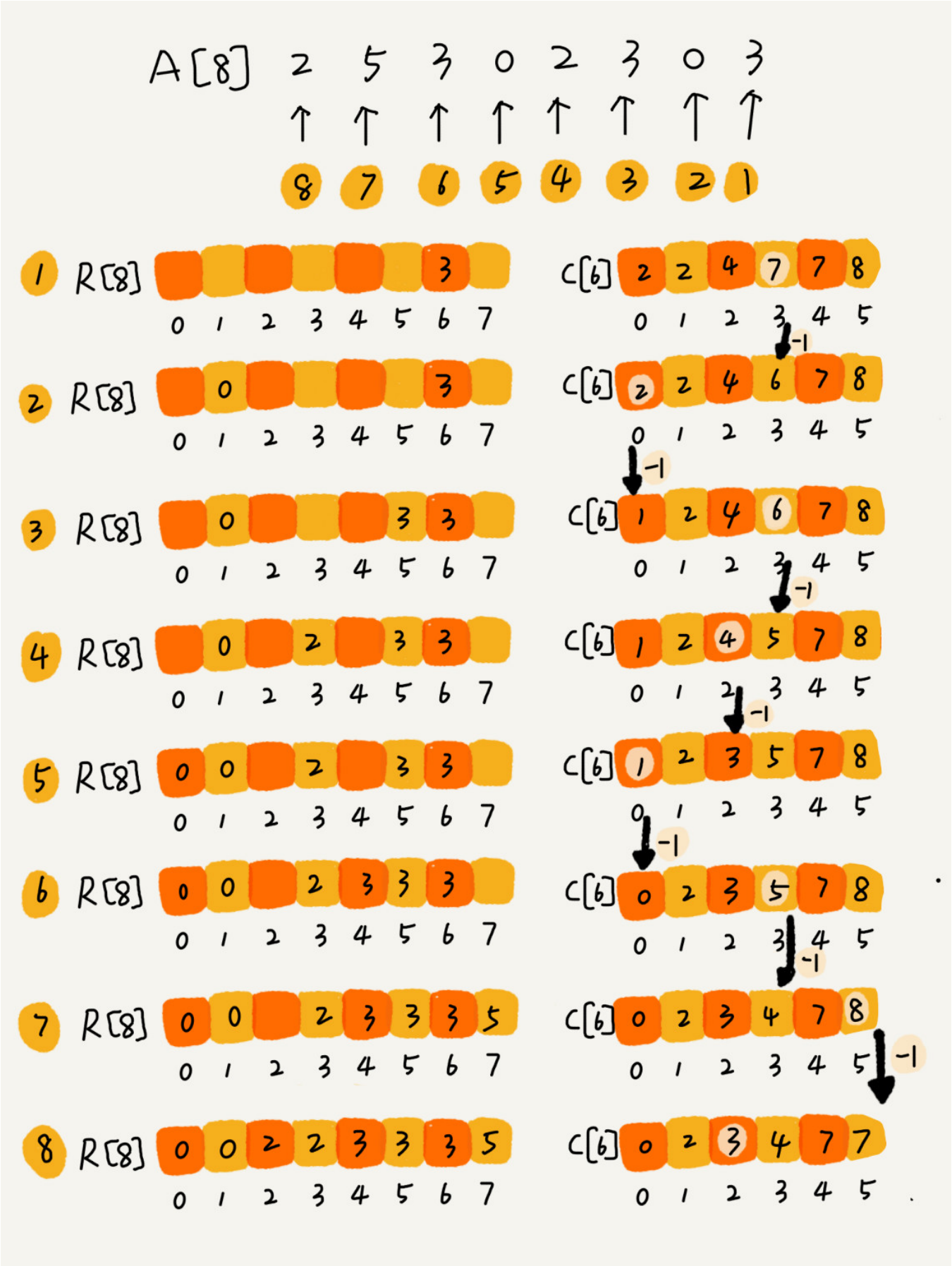
思路是这样的：我们对 C[6] 数组顺序求和，C[6] 存储的数据就变成了下面这样子。C[k] 里存储小于等于分数 k 的考生个数。



有了前面的数据准备之后，现在我就要讲计数排序中最复杂、最难理解的一部分了，请集中精力跟着我的思路！

我们从后到前依次扫描数组 A。比如，当扫描到 3 时，我们可以从数组 C 中取出下标为 3 的值 7，也就是说，到目前为止，包括自己在内，分数小于等于 3 的考生有 7 个，也就是说 3 是数组 R 中的第 7 个元素（也就是数组 R 中下标为 6 的位置）。当 3 放入到数组 R 中后，小于等于 3 的元素就只剩下了 6 个了，所以相应的 C[3] 要减 1，变成 6。

以此类推，当我们扫描到第 2 个分数为 3 的考生的时候，就会把它放入数组 R 中的第 6 个元素的位置（也就是下标为 5 的位置）。当我们扫描完整个数组 A 后，数组 R 内的数据就是按照分数从小到大有序排列的了。



上面的过程有点复杂，我写成了代码，你可以对照着看下。

复制代码

```
// 计数排序，a 是数组，n 是数组大小。假设数组中存储的都是非负整数。
public void countingSort(int[] a, int n) {

  if (n <= 1) return;
```

```
4
5   // 查找数组中数据的范围
6   int max = a[0];
7   for (int i = 1; i < n; ++i) {
8     if (max < a[i]) {
9       max = a[i];
10    }
11  }
12
13  int[] c = new int[max + 1]; // 申请一个计数数组 c，下标大小 [0,max]
14  for (int i = 0; i <= max; ++i) {
15    c[i] = 0;
16  }
17
18  // 计算每个元素的个数，放入 c 中
19  for (int i = 0; i < n; ++i) {
20    c[a[i]]++;
21  }
22
23  // 依次累加
24  for (int i = 1; i <= max; ++i) {
25    c[i] = c[i-1] + c[i];
26  }
27
28  // 临时数组 r，存储排序之后的结果
29  int[] r = new int[n];
30  // 计算排序的关键步骤，有点难理解
31  for (int i = n - 1; i >= 0; --i) {
32    int index = c[a[i]]-1;
33    r[index] = a[i];
34    c[a[i]]--;
35  }
36
37  // 将结果拷贝给 a 数组
38  for (int i = 0; i < n; ++i) {
39    a[i] = r[i];
40  }
41 }
```

这种利用另外一个数组来计数的实现方式是不是很巧妙呢？这也是为什么这种排序算法叫计数排序的原因。不过，你千万不要死记硬背上面的排序过程，重要的是理解和会用。

我总结一下，计数排序只能用在数据范围不大的场景中，如果数据范围 k 比要排序的数据 n 大很多，就不适合用计数排序了。而且，计数排序只能给非负整数排序，如果要排序的数据是其他类型的，要将其在不改变相对大小的情况下，转化为非负整数。

比如，还是拿考生这个例子。如果考生成绩精确到小数后一位，我们就需要将所有的分数都先乘以 10，转化成整数，然后再放到 9010 个桶内。再比如，如果要排序的数据中有负数，数据的

范围是 [-1000, 1000]，那我们就需要先对每个数据都加 1000，转化成非负整数。

基数排序（Radix sort）

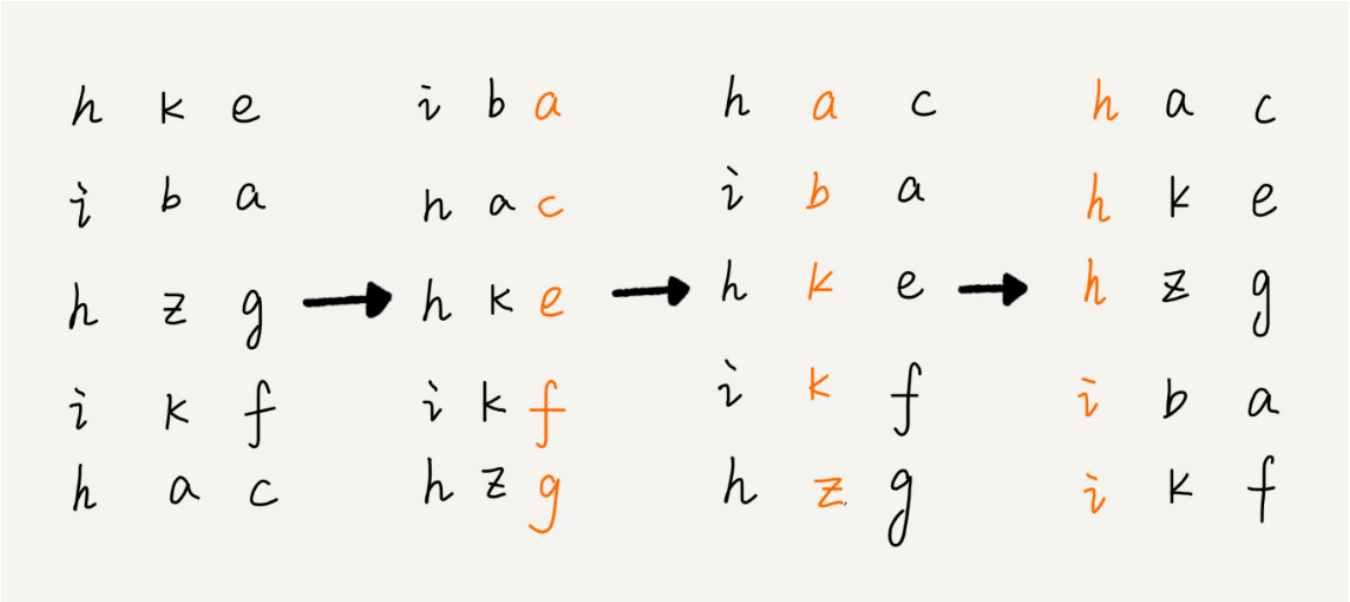
我们再来看这样一个排序问题。假设我们有 10 万个手机号码，希望将这 10 万个手机号码从小到大排序，你有什么比较快速的排序方法呢？

我们之前讲的快排，时间复杂度可以做到 O(nlogn)，还有更高效的排序算法吗？桶排序、计数排序能派上用场吗？手机号码有 11 位，范围太大，显然不适合用这两种排序算法。针对这个排序问题，有没有时间复杂度是 O(n) 的算法呢？现在我就来介绍一种新的排序算法，基数排序。

刚刚这个问题里有这样的规律：假设要比较两个手机号码 a，b 的大小，如果在前面几位中，a 手机号码已经比 b 手机号码大了，那后面的几位就不用看了。

借助稳定排序算法，这里有一个巧妙的实现思路。还记得我们第 11 节中，在阐述排序算法的稳定性的时候举的订单的例子吗？我们这里也可以借助相同的处理思路，先按照最后一位来排序手机号码，然后，再按照倒数第二位重新排序，以此类推，最后按照第一位重新排序。经过 11 次排序之后，手机号码就都有序了。

手机号码稍微有点长，画图比较不容易看清楚，我用字符串排序的例子，画了一张基数排序的过程分解图，你可以看下。



注意，这里按照每位来排序的排序算法要是稳定的，否则这个实现思路就是不正确的。因为如果是非稳定排序算法，那最后一次排序只会考虑最高位的大小顺序，完全不管其他位的大小关系，那么低位的排序就完全没有意义了。

根据每一位来排序，我们可以用刚讲过的桶排序或者计数排序，它们的时间复杂度可以做到 O(n)。如果要排序的数据有 k 位，那我们就需要 k 次桶排序或者计数排序，总的时间复杂度是

O(k*n)。当 k 不大的时候，比如手机号码排序的例子，k 最大就是 11，所以基数排序的时间复杂度就近似于 O(n)。

实际上，有时候要排序的数据并不都是等长的，比如我们排序牛津字典中的 20 万个英文单词，最短的只有 1 个字母，最长的我特意去查了下，有 45 个字母，中文翻译是尘肺病。对于这种不等长的数据，基数排序还适用吗？

实际上，**我们可以把所有的单词补齐到相同长度，位数不够的可以在后面补“0”**，因为根据 ASCII 值，所有字母都大于“0”，所以补“0”不会影响到原有的大小顺序。这样就可以继续用基数排序了。

我来总结一下，**基数排序对要排序的数据是有要求的，需要可以分割出独立的“位”来比较，而且位之间有递进的关系，如果 a 数据的高位比 b 数据大，那剩下的低位就不用比较了。除此之外，每一位的数据范围不能太大，要可以用线性排序算法来排序，否则，基数排序的时间复杂度就无法做到 O(n) 了。**

解答开篇

今天的内容学完了。我们再回过头来看看开篇的思考题：如何根据年龄给 100 万用户排序？现在思考题是不是变得非常简单了呢？我来说一下我的解决思路。

实际上，根据年龄给 100 万用户排序，就类似按照成绩给 50 万考生排序。我们假设年龄的范围最小 1 岁，最大不超过 120 岁。我们可以遍历这 100 万用户，根据年龄将其划分到这 120 个桶里，然后依次顺序遍历这 120 个桶中的元素。这样就得到了按照年龄排序的 100 万用户数据。

内容小结

今天，我们学习了 3 种线性时间复杂度的排序算法，有桶排序、计数排序、基数排序。它们对要排序的数据都有比较苛刻的要求，应用不是非常广泛。但是如果数据特征比较符合这些排序算法的要求，应用这些算法，会非常高效，线性时间复杂度可以达到 O(n)。

桶排序和计数排序的排序思想是非常相似的，都是针对范围不大的数据，将数据划分成不同的桶来实现排序。基数排序要求数据可以划分成高低位，位之间有递进关系。比较两个数，我们只需要比较高位，高位相同的再比较低位。而且每一位的数据范围不能太大，因为基数排序算法需要借助桶排序或者计数排序来完成每一个位的排序工作。

课后思考

我们今天讲的都是针对特殊数据的排序算法。实际上，还有很多看似是排序但又不需要使用排序算法就能处理的排序问题。

假设我们现在需要对 D，a，F，B，c，A，z 这个字符串进行排序，要求将其中所有小写字母都排在大写字母的前面，但小写字母内部和大写字母内部不要求有序。比如经过排序之后为 a，c，z，D，F，B，A，这个如何来实现呢？如果字符串中存储的不仅有大小写字母，还有数字。要将小写字母的放到前面，大写字母放在最后，数字放在中间，不用排序算法，又该怎么解决呢？

欢迎留言和我分享，我会第一时间给你反馈。

我已将本节内容相关的详细代码更新到 Github，戳此即可查看。

上一篇 12 | 排序（下）：如何用快排思想在O(n)内查找第K大元素?

下一篇 14 | 排序优化：如何实现一个通用的、高性能的排序函数?

写留言

精选留言

wucj 51

用两个指针a、b：a指针从头开始往后遍历，遇到大写字母就停下，b从后往前遍历，遇到小写字母就停下，交换a、b指针对应的元素；重复如上过程，直到a、b指针相交。
对于小写字母放前面，数字放中间，大写字母放后面，可以先将数据分为小写字母和非小写字母两大类，进行如上交换后再在非小写字母区间内分为数字和大写字母做同样处理

2018-10-19

伟忠 86

课后思考，利用桶排序思想，弄小写，大写，数字三个桶，遍历一遍，都放进去，然后再从桶中取出来就行了。相当于遍历了两遍，复杂度O(n)

2018-10-19

□ 79

渐渐有些掉队的趋势

2018-10-19

传说中的成大大 33

排序算法基本上算是学完了，昨天我在实践快排的时候 我就发现这样一个问题，虽然理解了原理 但是写起来还不是很顺畅，如果写出来的代码跟老师的一模一样 那就不叫理解了原理 那叫背代码，我昨天也去翻了大话数据结构里面的快排 发现代码又不一样，所以我觉得接下来的时间就应该根据思路多实现一下这些排序代码，不能死记硬背代码，多理解原理

2018-10-19

胡二 13

计数排序中，从数组A中取数，也是可以从头开始取的吧

2018-10-19

作者回复

可以的 只不过就不是稳定排序算法了

2018-10-20

徐凯 8

包含数字的话。其实就是一个荷兰国旗问题 思路与第一题类似 一个指针控制左边界 一个指针控制右边界 左边界控制小写字母 右边界控制大写字母 另外一个指针扫描 遇到小写字母跳过 遇到大写字母则将右边界-1的元素交换过来 此时q指针应保持原位置不动 因为右边还未扫描过 交换过来的元素无法保证就是小写字母

2018-10-19

seniusen 6

计数排序中可以从头向后取数据吗？个人感觉似乎是一样的过程。

2018-10-19

作者回复

可以的 但就不是稳定排序算法了

2018-10-20

伟忠 6

以前了解桶排序，以为计数排序就是桶排序的优化，很简单，没想到里面用"顺序求和"快速得出值对应的原排序对象位置(有多个相同值的时候是这个值在排序后的数组中的最后位置，用一次以后减少1)，这样就可以用对象属性来将对象进行排序了，这波操作666

基数排序用了排序算法的稳定性，排多次

2018-10-19

Monday 5

老师，个人感觉这节没有以往章节的细致了，拿桶排序来举例：
1、自问的三个问题只有了时间复杂度分析，少了是否为稳定排序算法和空间复杂度两个问题。
1.1）稳定性，若单个桶内用归并排序，则可保证桶排序的稳定性；若使用快排则无法保证稳定性。
1.2）空间复杂度，桶都是额外的存储空间，只有确定了单个桶的大小才能确定空间复杂度；n个元素假设分为m个桶，每个桶分配n/m个元素的大小？个人觉得单个桶的大小不好确定，但是范围应该在n/m~n之间
2、没有伪代码实现，自己在代码实现中遇到了一些问题
2.1）桶个数的设置依据什么原则？
2.2）桶大小的设置，让桶的能动扩容？
望回复，谢谢！

2018-10-22

作者回复

1）这一节课的重点是应用场景

2）关于时间 空间 稳定性分析确实没有前面两节详细。不过通过前两节的学习 这三种排序算法的时间 空间 稳定性分析应该简单多了

3）你说的对 要用归并

4）桶的大小设置的原则 权衡空间 时间复杂度 在你能接受的执行时间和内存占用下完成就可以 并没有一个标准答案

5）是的 要么用链表 要么用动态扩容的数组

2018-10-22

Kudo 4

关于思考题：
如果分为三个桶（大写、小写、数字），那么时间复杂度应该不会达到O(n)，因为O(nlog(n/m))中的m只有3，时间复杂度会退化到O(nlogn)。如果要达到O(n)的复杂度，我认为应该使用计数排序，将A-Za-z0-9作为62个桶，这样遍历一次就可以完成排序。（如果上述理解有偏差，请作者务必指出，多谢！）

2018-10-19

峰 4

思考题，快排划分的思想分成两半，分成三份(双轴)，只不过固定选取一个合适的pivot就ok。

2018-10-19

刘強 3

根据acill码的天然顺序，分三个桶就可以把

2018-10-19

GrubbyLu 2

王老师，看完专栏之后，对于桶排序有个疑问，您文中讲到“当桶的个数 m 接近数据个数 n 时，这个时候桶排序的时间复杂度接近 O(n)。” 但是在您下面的举例中，10GB 的订单数据只分100个桶，即便增加到1千或者1万个桶，桶和数据个数之间还是有很大的差距。而且您最后的总结中也说到“桶排序是对针对范围不大的数据”，是不是可以理解为桶应该设置的尽量小，这样在大数量的情况下时间复杂度就很难接近 O(n)了，希望您能予以解答，多谢。

2018-10-19

作者回复

嗯嗯 可能例子举的导致你误解了。我的订单的例子主要是体现可以用在外排序中。当然 如果能分成10万个桶最好不过了。

2018-10-20

Feliscatus 2

基数排序那个图最后一步不是h在i前面吗(从小到大排序)

2018-10-19

kakasi 1

桶排序:

1. 原理: 根据数据范围，分成若干个数据段的桶，通过遍历讲数据放到对应的桶中。每个桶里都进行快排或归并。
2. 时间复杂度: 最好o(n), 最坏o(nlogn), 平均o(n)，一般桶分的越细越多复杂度就会最好。
3. 内存消耗: o(n)
4. 稳定性: 取决于每个桶的排序方式，快排就不稳定，归并就稳定。
5. 适用场景: 数据范围不大的。内存吃紧的，如磁盘的读写可以分成多个小文件并对每个小文件排序，然后直接写到大文件里，这个时候内存消耗不再是o(n)了。

计数排序:

1. 原理: 特殊的桶排序，即每个下标代表一个数据范围，其值就是这个数据的个数。
2. 时间复杂度: 都是o(n)
3. 内存消耗: o(n)
4. 稳定性: 稳定，只要整理最后结果时从后开始遍历即可。
5. 适用场景: 数据范围不大的，如年龄排序。

基数排序:

1. 原理: 对数据的每一位进行桶排序或计数排序，对每位排序后结果就是有序的。
2. 时间复杂度: 最好o(n), 最坏o(nlogn), 平均o(n)
3. 内存消耗: o(n)
4. 稳定性: 稳定。否则就排不成的。

5. 适用场景: 是在桶排序和计数排序基础上进行的，保证每位数据范围不大，并且位数也不是很多。

2018-11-04

Ant 1

看了俩小时

2018-11-02

作者回复

如果之前没基础 想掌握牢固 起码看一个礼拜吧😄

2018-11-02

在路上 1

我觉着可以把大小写字母根据对应的ASCII值转化成数字，然后遍历一遍就可以了吧。

2018-11-02

coulson 1

老师，你讲的一会数组C,一会数组R，一会数组A，已经被绕晕了，咋办？跟不上节奏了

2018-10-30

作者回复

多看几遍 确实不好理解

2018-10-31

李靖峰 1

遍历，遇到小写并且上一个不是小写扔前面，遇到数字继续遍历，遇到大写扔后面。序列最好用链表

2018-10-28

竹林清风 1

非常棒，理解更深刻了，关于课后思考我首先想到的是应用快排的分区思想，然后看了评论，感觉桶排序更好理解！

2018-10-22

作者回复

实际上 左右指针移动交换更巧妙 留言区也有人给出答案

2018-10-22