

讲堂 > 数据结构与算法之美 > 文章详情

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。

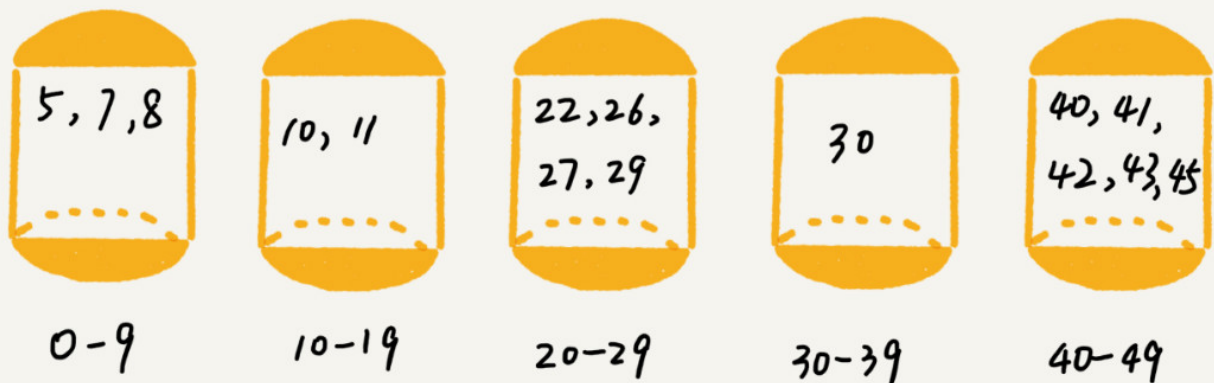
按照惯例，我先给你出一道思考题：**如何根据年龄给 100 万用户排序？**你可能会说，我用上一节课讲的归并、快排就可以搞定啊！是的，它们也可以完成功能，但是时间复杂度最低也是 $O(n \log n)$ 。有没有更快的排序方法呢？让我们一起进入今天的内容！

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。

对这组金额在 0-50 之间的订单进行桶排序：

22, 5, 11, 41, 45, 26, 29, 10, 7, 8, 30, 27, 42, 43, 40.



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 $O(k * \log k)$ 。 m 个桶排序的时间复杂度就是 $O(m * k * \log k)$ ，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 $O(n \log n)$ 的排序算法了。

桶排序比较适合用在外部分序中。所谓的外部分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的排序的订单数据了。

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？

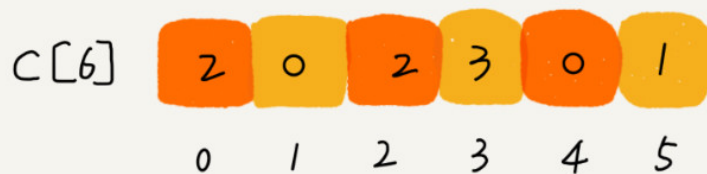
考生的满分是 900 分，最小是 0 分，这个数据的范围很小，所以我们可以分成 901 个桶，对应分数从 0 分到 900 分。根据考生的成绩，我们将这 50 万考生划分到这 901 个桶里。桶内的数据都是分数相同的考生，所以并不需要再进行排序。我们只需要依次扫描每个桶，将桶内的考生

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 $O(n)$ 。

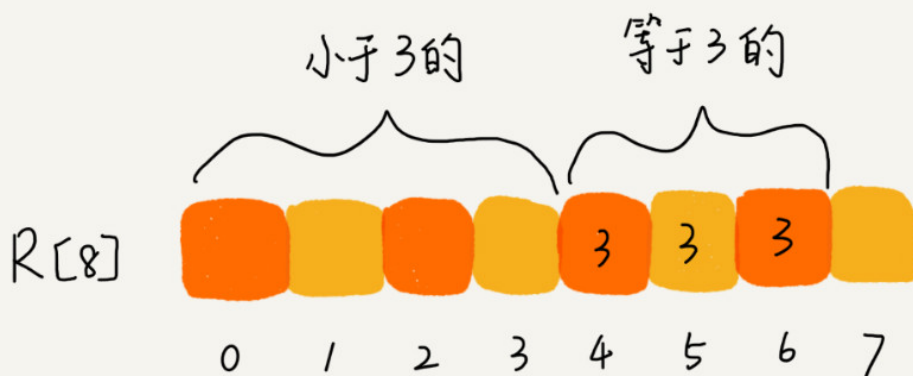
计数排序的算法思想就是这么简单，跟桶排序非常类似，只是桶的大小粒度不一样。不过，为什么这个排序算法叫“计数”排序呢？“计数”的含义来自哪里呢？

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！

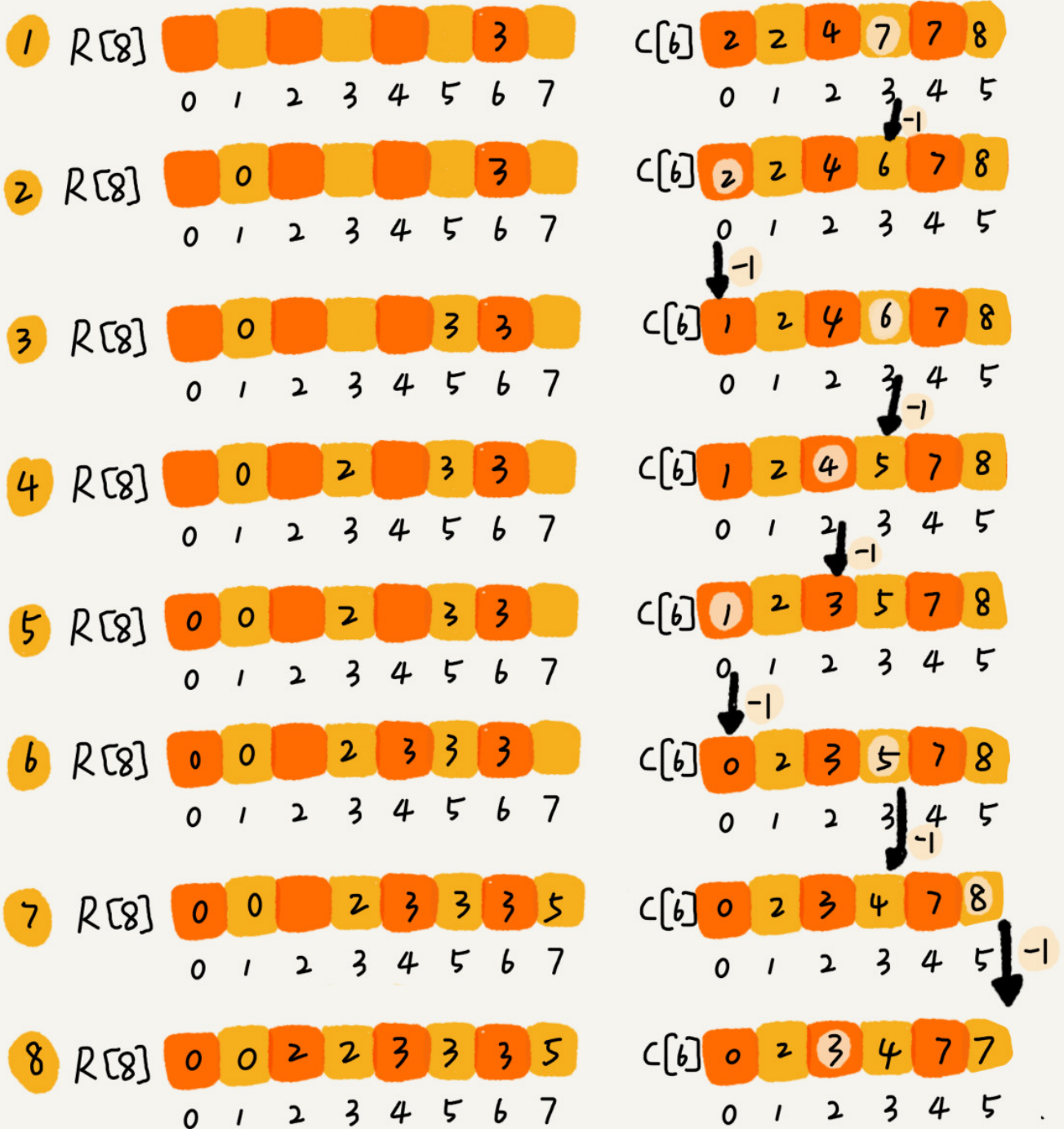
我们从后到前依次扫描数组 A 。比如，当扫描到 3 时，我们可以从数组 C 中取出下标为 3 的值 7，也就是说，到目前为止，包括自己在内，分数小于等于 3 的考生有 7 个，也就是说 3 是数组 R 中的第 7 个元素（也就是数组 R 中下标为 6 的位置）。当 3 放入到数组 R 中后，小于等于 3 的元素就只剩下了 6 个了，所以相应的 $C[3]$ 要减 1，变成 6。

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。

A[8] 2 5 3 0 2 3 0 3

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

8 7 6 5 4 3 2 1



上面的过程有点复杂，我写成了代码，你可以对照着看下。

```
1 // 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。
2 public void countingSort(int[] a, int n) {
3     if (n <= 1) return;
```

复制代码


```
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)

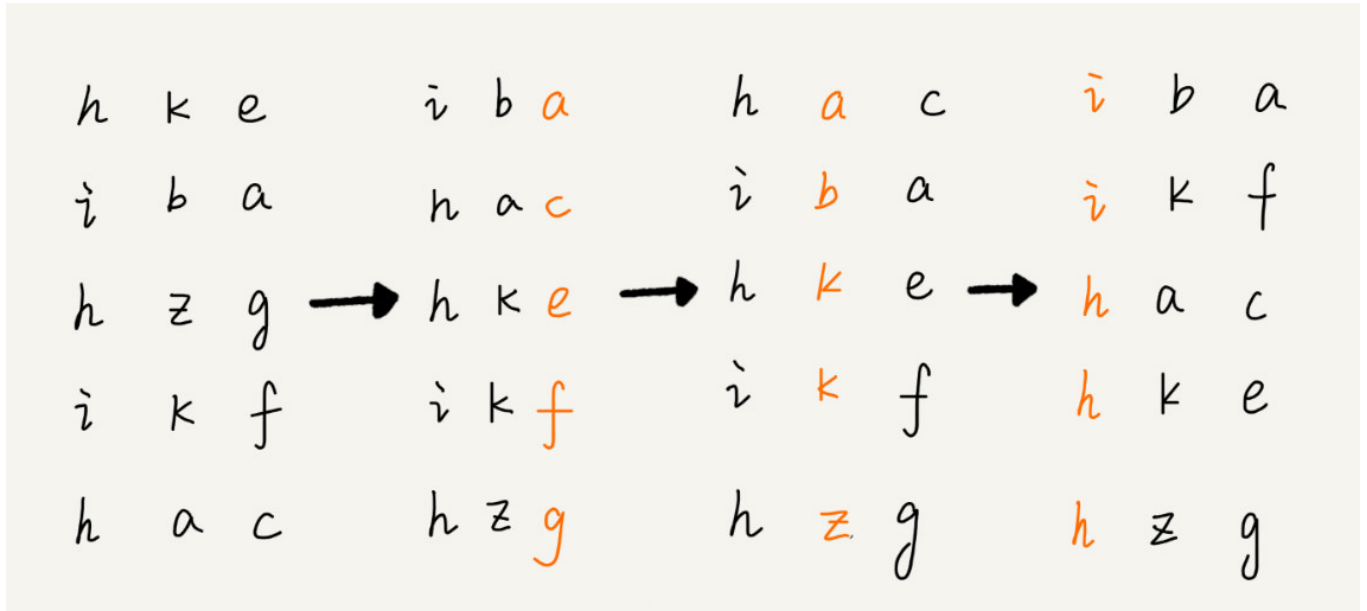
我们再来看这样一个排序问题。假设我们有 10 万个手机号码，希望将这 10 万个手机号码从小到大排序，你有什么比较快速的排序方法呢？

我们之前讲的快排，时间复杂度可以做到 $O(n\log n)$ ，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。



版权归极客邦科技所有，未经许可不得转载

写留言

精选留言



伟忠

👍 18

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



wucj

👍 13

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



传说中的成大大

👍 6

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



徐凯

👍 5

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



□

👍 5

渐渐有些掉队的趋势

2018-10-19



伟忠

👍 3

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



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



峰

👍 2

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



胡二

👍 1

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



Kudo

👍 1

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 $O(n\log(n/m))$ 中的 m 只有3，时间复杂度会退化到 $O(n\log n)$ 。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



kaka

👍 1

感谢 @徐凯，搜索了下荷兰国旗问题，按照该思路写了份 C++ 代码

```
#include <iostream>
#include <algorithm>

using namespace std;

int main(int argc, char **argv) {
    vector<char> test = {'D', 'a', '3', 'F', '1', 'B', 'c', 'A', 'z', '2'};

    int lower_letter = -1, upper_letter = test.size() - 1, current = 0;
    while (test[upper_letter] >= 'A' && test[upper_letter] <= 'Z') upper_letter--;
    while (current <= upper_letter) {
        if (test[current] >= '0' && test[current] <= '9') {
            current++;
        }
        else if (test[current] >= 'a' && test[current] <= 'z') {
            swap(test[current], test[lower_letter + 1]);
            current++;
            lower_letter++;
        } else {
            swap(test[current], test[upper_letter]);
            while (test[upper_letter] >= 'A' && test[upper_letter] <= 'Z') upper_letter--;
        }
    }

    for (auto c : test) {
        cout << c << " ";
    }
    cout << endl;
    return 0;
}
```

2018-10-19



格曼

👍 1

文中提到基数排序可由每一位进行稳定排序算法的排序，比如计数排序，可是文中计数排序算法的实现方式是不稳定的， $c[a[i]]$ 记录的是小于等于 $a[i]$ 的数的个数，每次将 $a[i]$ 放到 $r[c[a[i]]-1]$ 的位置实际上就是将该数放在了最末尾的位置，也就是说如果几个数相等，排序的过程中先出现的数反而排完序之后会到后面去，这样是不稳定的。实现的过程可以改成这样，每次将 $a[i]$ 放到 $r[c[a[i]-1]]$ 处，也就是所有小于 $a[i]$ 的数的个数的后面一位，执行完之后 $c[a[i]-1]++$ ，这样可以保证相同的数是按照原有顺序一个个放置的，从而保证算法的稳定性。

2018-10-19



刘強

👍 1

根据acill码的天然顺序，分三个桶就可以把

2018-10-19



刘強

👍 1

学了算法以后，很容易把简单问题搞复杂😄

2018-10-19



Farflight

👍 1

小写字母26个，大写26个，数字0-9一共10个，那么用计数排序，把它们分入62个桶里就行了

2018-10-19



赵永森

👍 0

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



子非口

👍 0

对于链表的反转问题，看了您的git代码，用的是双重指针，因为对这块不太熟悉，所以没看明白。就从网上找到了其他算法，可能不是最简单的，但是还比较清晰

```
linkList reverse(linkList head){
    linkList p,q,pr;
    p = head->next;//p一直往下走，当指向null时结束
    q = NULL;//q指向反转后第一个结点
    head->next = NULL;
    while(p){
        pr = p->next;//pr暂存p的原始下一个结点
        p->next = q;
        q = p;
        p = pr;
    }
    head->next = q;//头指针指向反转后的第一个结点
    return head;
}
```

参考链接:<https://blog.csdn.net/blloo/article/details/62050967>

2018-10-19



勤劳的小胖子-libo

0

使用三个桶（可以是三个数组或者链表），一个放数字，一个放小写字母，一个放大写字母；每读取一个字符，判断然后决定放入哪一个桶，最后把三个桶的数据拷贝到原数组中。

时间复杂度就是 $O(N)$,空间复杂度也是 $O(N)$ 。

2018-10-19



liangjf

0

今天的思考题终于很快想出来了。借助桶排序思想，分为大，小写，数字三个桶，遍历一遍，放进各自的桶，直到没数据。再从三个桶中取出放进一个数组中。遍历了两次，时间复杂度是 $2n$ ，去掉常系数，时间复杂度是 $O(n)$ 。

如果是要求各自排序，加多一个步骤:各个桶内部排序。

2018-10-19



Tenderness

0

桶排序，数组长度为3，a-z、数字、A-Z。遍历一边，就知道对应的有多少个了，然后利用计数排序知道了位置，然后新建N个长度的数组放入就行了。至于仅有大小写字母，利用这种方式也可以的。不用排序的话，那就新建n长的数组，两个指针i和j，一前一后，对应大小写字母插入。如果中间要插入数字的话，那记录下i j的位置，再遍历一边就可以咯

2018-10-19



唯她命

0

对于学生成绩计数排序，我觉得根本没有必要找到每个分数在数组里的位置，，因为每个元素出现的次数已经知道了，只有for循环每个数字出现的次数，依次填充数组就可以了！，希望老师解答

2018-10-19



小洋洋

0

关于第二个计数排序的栗子，遍历原数组把它们按照正确的顺序位置放入临时数组的时候，为什么要从后向前遍历呢？从前向后遍历也是一样的呀，是因为程序执行时最近刚被使用的将来很可能也会被使用吗，增加CPU缓存命中率？

2018-10-19



有铭

0

很奇怪，为啥我看到别人描述的说的计数排序不是这样的，不是要顺序求和，而是求出C[6]中每个位置分数出现过的次数，然后按照次数，直接从C[0]遍历到C[6]，每个位置的分数出现过几次就遍历几次，就完成排序了

2018-10-19



asdf100

0

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。

这个数组顺序求和是什么意思？ c[0] + c[1] 这类的？

2018-10-19



Smallfly

0

计数排序的空间复杂度应该怎么算呢？似 $O(N)$ 又非 $O(N)$ ，如果数据量比较小可以全部读入内存，那么空间复杂度是 $O(N)$ ，如果数据量非常大，只能分批读取再处理，那临时数组的空间可以复用上一次的，空间复杂度该怎么算呢？

2018-10-19



Rain

0

这个是不是会改变元素本来的排列顺序（不稳定）？

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。

2018-10-19



姚君国cium

0

```
public class Sort { //字数限制，注释和main函数都被我删除了
    public static String digitSort(String str, int n) {
        int digitNums = 0;
        int lowerCaseNums = 0;
        int capitalCaseNums = 0;
        for (int i=0;i<n;i++){
            int temp = str.charAt(i);
            if (temp >= 48 && temp <=57){ digitNums++; }
            else if(temp >= 65 && temp <=90){ capitalCaseNums++; }
            else{ lowerCaseNums++; }
        }
        char [] digit = new char[digitNums];
        char [] lowerCase = new char[lowerCaseNums];
        char [] capitalCase = new char[capitalCaseNums];
        int b = 0;
        int m = 0;
        int p = 0;
        for (int l=0;l<n;l++){
            int temp = str.charAt(l);
            if (temp >= 48 && temp <=57){ digit[b++] = (char)temp; }
            else if(temp >= 65 && temp <=90){ capitalCase[m++] = (char)temp; }
            else{ lowerCase[p++] = (char)temp; }
        }
        String digitSorted = insertionSorts(digit,digit.length);
        String lowerCaseSorted = insertionSorts(lowerCase,lowerCase.length);
        String capitalCaseSorted = insertionSorts(capitalCase,capitalCase.length);
        return lowerCaseSorted + digitSorted + capitalCaseSorted;
    }
    private static String insertionSorts(char [] a,int n){
        if (n <= 1){ return ""; }
        for (int i=1;i<n;i++){
            int j = i-1;
            int value = a[i];
            for(;j >=0 ;j--){
                if (a[j] < value){
                    a[j+1] = a[j];
                }
            }
            a[j+1] = (char)value;
        }
        StringBuffer sb = new StringBuffer();
        for (int i=a.length-1;i>=0;i--){ sb.append((char)a[i]); }
        return sb.toString();
    }
}
```

2018-10-19



Feliscatus

0

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



闲鱼有梦想

0

计数排序逻辑写的真的好复杂。为什么不直接遍历C数组（不累加），从第一个下标开始，如果不为0，在R数组添加C数组的下标，如果为0，则忽略。比如C数组=[2,0,2,3,0,1],从第一个下标0开始，C[0]=2，那么可以在R数组，连续前两位添加0，即：R[0]=0,R[1]=0。到第二个下标1，C[1]=0,则不添加。到第三个下标2，C[2]=2,那么R数组，连续添加两位，即R[2]=2,R[3]=2。依次类推，到结束。逻辑很简单（也就是数C数组中，每个下标对应的值为n,那么在R数组中连续添加n个该下标。

2018-10-19



猫头鹰爱拿铁

0

周三习题应该也可以用桶排序实现，遍历每份大文件时候再将其根据桶的数据范围放到不同桶里面，这个桶排序应该还可以利用文件是有序的再优化，内部排序文件的时候可以二分查询减少数据的比较。将一个范围的数据放到桶里面，每个桶再进行快排。

电话号码排序可以用bitmap去实现。具体是从10000000到99999999划分在这个bitmap里面。

文中对单词排序还可以考虑字典排序，使用trie树建立好所有数据，再遍历。

本文中习题使用桶排序，将字符串转换为ascii码，数字范围为48-57，小写字母为97-122，大写字母为65-90，分为三个桶，将这些数据放到桶里。再按桶的顺序遍历数据。

2018-10-19



Ricky

0

思考题，如果只有大小写字母，则可以采用稳定比较算法进行排序，比较规则是大小写字母比较，小写优于大写或者采用桶排序，大写一个桶，小写一个桶，感觉这里有点类似用哈希表存储；如果还有数字可以用桶排序，分为三个桶，大写字母桶，小写字母桶，数字桶，其中桶的顺序为小写←数字←字母，亦或采用稳定的比较排序算法，比较规则是桶的顺序规则，或者直接哈希表存储即可，个人理解感觉桶就与哈希表有点类似，不同在于数据结构不同罢了

2018-10-19



123456789十百千万

0

计数排序，和数一数每个值有多少个数字，然后从小到大重新构造数组有啥区别？真的是。。折腾

2018-10-19



乘坐Tornado的线程魔法师

0

看到有读者说可以用计数排序 给对象排序，求具体方法

2018-10-19



传说中的成大大

0

思考题中 只有小写字母和大写字母中我们可以取两个数组 然后用ascii码把小写字母放到前面的数组中，大写字母放到后面的数组中 有点类似桶排序，只是数组内部不在需要排序了

如果是小写字母 和数字 加大写字母 我则选择用三个数组(桶)然后合并就行了

2018-10-19



港

0

课后思考：当只有小写字母与大写字母的时候，我们使用a和b分别存储字符串第一个位置指针和最后一个位置指针，a往后遍历，b往前遍历，如果a所指的是小写字母，b所指的是大写字母，那就a往后移一位，b往前移一位，如果a不是小写字母，那就停止。如果b不是也同理。当a,b所指同时不符合，那就将a,b所指地址里的内容交换，然后继续遍历，直到a>=b停止。时间复杂度为O(n)

当有数字时，我们可以先把数字和小写字母归为一类，按照上述方法遍历完成，然后再针对数字和小写字母，再用上述方法将数字和小写字母分开。

2018-10-19



刘远通

0

最后一个基数排序是考虑的字典排序，而不是数字排序。比如20>123?

2018-10-19



侯金彪

0

课后思考：在字符串开始和结束的地方设置两个指针，p1从前往后走，p2从后往前走，如果p1是大写并且p2是小写，则互换位置，否则移动p1或p2。时间复杂度O(n)，空间O(1)。如果有数字，先遍历一次统计大写字母，小写字母和数字的个数，计算新字符串中分别对应的区间范围，然后再分别遍历原字符串的三个区间，把不属于本区间的放到该放的区间，时间O(n)空间O(1)。

2018-10-19



曹源

0

大写一个桶，小写一个桶，加多数字就加多一个桶。

2018-10-19



朱月俊

0

可以利用基数排序的一些步骤，统计小写字母，大写字母，特殊字符的数量，从而确定在最终数组中的位置范围，然后遍历原来数据存放即可。

也可以利用快排思想，先将小写字母和大写字母以及特殊字符区分开，然后再将大写字母和特殊字符区分开来。

2018-10-19



Rain

0

看到要求不用排序算法，我用单链表写了一下：

```
char[] arr = "D a F B c A z"
sortArr(arr);
// -----
// 按照小写字母-数字-大写字母排序
private static char[] sortArr(char[] arr) {
```

```
Node lowerHead = null;
Node lowerTail = null;

Node numberHead = null;
Node numberTail = null;

Node upperHead = null;
Node upperTail = null;

int len = arr.length;

for (int i = 1; i < len; i++) {
    char c = arr[i];
    Node cNode = createNode(c);

    if (isLowerCase(c)) {
        if (lowerHead == null) {
            lowerHead = cNode;
            lowerTail = cNode;
        } else {
            lowerTail.next = cNode;
            lowerTail = cNode;
        }
    } else if (isNumber(c)) {
        if (numberHead == null) {
            numberHead = cNode;
            numberTail = cNode;
        } else {
            numberTail.next = cNode;
            numberTail = cNode;
        }
    } else if (isUpperCase(c)) {
        if (upperHead == null) {
            upperHead = cNode;
            upperTail = cNode;
        } else {
            upperTail.next = cNode;
            upperTail = cNode;
        }
    }
}

// concat numberPart and upperPart
```

```
if (numberHead != null) {
    numberTail.next = upperHead;
} else {
    numberHead = upperHead;
}
if (lowerTail != null) {
    lowerTail.next = numberHead;
}
```

```
Node head = null;
if (lowerHead != null) {
    head = lowerHead;
} else if (numberHead != null) {
    head = numberHead;
}
```

```
SinglyLinkedList sll = new SinglyLinkedList(head);
char[] sortedArr = convertToArray(sll);
return sortedArr;
}
```

```
static boolean isLowerCase(char c) {
    return c >= 'a' && c <= 'z';
}
```

```
static boolean isUpperCase(char c) {
    return c >= 'A' && c <= 'Z';
}
```

```
static boolean isNumber(char c) {
    return c >= 0 && c <= 9;
}
```

```
static char[] convertToArray(SinglyLinkedList sll) {
    char[] arr = new char[sll.length];
    Node cNode = sll.head;
    int i = 0;
    while (cNode != null) {
        arr[i] = cNode.value;
        cNode = cNode.next;
        i++;
    }
}
```

```
return arr;  
}
```

2018-10-19



李相彬

👍 0

对于只有大小写字母的情况，可以利用快排的partition思想，选择'a'对应的ASCII码作为pivot；对于有大小写字母和数字的情况，可以用桶排序，分成三个桶

2018-10-19



徐凯

👍 0

思考题：采用类似快排的思路 p指针控制左边界范围内的小写字母 另外一个q指针不断扫描 扫描到大写字母则跳过 扫描到小写字母 将p边界右边一个元素与当前q所指元素交换 直至q扫描到结尾

2018-10-19



王先统

👍 0

思考题：分桶当然可以，但是对于题中分组数小于等于3的情况目测可以原地完成划分，大于3就需要额外分配空间了，请老师指正一下

2018-10-19



伟忠

👍 0

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。

C[6] 2 2 4 7 7 8 顺序求和后

C[6] 0 1 2 3 4 5 序号

2018-10-19



大可可

👍 0

正则表达式替换为空再连起来就可以吧

2018-10-19