# 排序算法与分治初步 3

## 概述

计算复杂性理论里有一个定理说，利用比较和交换的排序的时间复杂度的下限不会低于$O(n \log n)$，而快速排序和归并排序是比较和交换的排序算法，从而它们的时间复杂度不能低于这个下限。但从另一方面讲，如果某个排序算法并非利用比较和交换来做，那么是有可能达到低于$O(n \log n)$时间复杂度的。本文会介绍其中一种算法，叫“基数排序”，它的复杂度可以达到$O(nM)$，其中$M$是排序数的最大位数，当$M$比较小的时候可以近似为线性。

为了介绍基数排序，我们先介绍计数排序。

## 计数排序

当数字范围非常小的时候，计数排序是一种很高效的排序方式，并且还是稳定的。我们直接看例子：

假设我们有四种水果，包含名称和个数：`{Apple,3},{Banana,2},{Grape,2},{Watermelon,3}`，我们想对他们按个数排序，并且要求是稳定排序。

1. 先开一个足够长的数组$c$，然后从左到右遍历所有水果，直接数一下个数为$k$的水果种类有多少个，存入$c$中。从而$c[i]$表示个数为$i$的水果种类有多少个。在这个例子中，$c = [0, 0, 2, 2]$，即$c[2] = c[3] = 2$。接着对$c$求前缀和，$s = [0, 0, 2, 4]$，那么$s[i]$表示个数小于等于$i$的水果总共多少种
2. 从后向前遍历水果，第一个遍历到了`{Watermelon,3}`，因为$s[3] = 4$，所以我们知道了个数小于等于$3$的水果一共有$4$种，从而`Watermelon`在最终答案里应该第$4$；接着让$s[3]$减$1$变成$3$；
3. 接着遍历到了`{Grape,2}`，由于$s[2] = 2$，所以我们知道了`Grape`应该排第$2$，接着让$s[2]$减$1$变成$1$；
4. 接着遍历到了`{Banana,2}`，由于$s[2] = 1$，所以我们知道了`Banana`应该排第$1$，接着让$s[2]$减$1$变成$0$；
5. 接着遍历到了`{Apple,3}`，由于$s[3] = 3$，所以我们知道了`Apple`应该排第$3$，接着让$s[3]$减$1$变成$2$；

到这里，所有水果的排位都得到了。并且排序是稳定的，即个数相同的水果的相对位置没有变。

从上面的算法可以看出，计数排序“人如其名”，先记个数，然后再排序。由于计数、求前缀和、遍历数组三个操作都是$O(n)$的，所以总体时间复杂度也是$O(n)$。但是，如果数据的数字范围特别的大，空间要求就非常高了。所以计数排序只适合数字范围比较小的时候使用。

## 基数排序

思想是这样的。给定一个长$n$数组$a$，假设$a$里只有正整数。我们先将所有数按照其个位数做稳定排序，做稳定排序的方法如下：

1. 正序遍历$a$，先统计一下个位数为$k$的数字各有多少个，其中$k=0\sim 9$。记答案为$c$数组
2. 对$c$求前缀和
3. 逆序遍历$a$，如果$a[i]$的个位数等于$r$，那么我们就知道，如果只按个位数排序的话，这个$a[i]$的排名一定是$c[a[i]\%10]$，从而可以开个临时数组，将$a[i]$填到它的排名上。同时让$c[a[i]\%10]$减去$1$。这里的理由也非常简单，比如我们先遍历了$a[n]$，如果它的个位数是$1$，由于个位数是$1$的数总共有$c[1]$个，那$a[n]$只按个位数排的话，它的排名显然就是$c[1]$，接着让$c[1]$减$1$；遍历完$a[n]$之后，假设我们又发现$a[n-1]$个位数也是$1$，那它的排名显然就在$a[n]$的排名的前一个。以此类推。而且由于是逆序遍历$a$，这个排序是稳定的（也就是说，十位数、百位数等等的相对位置不变）
4. 按个位数做稳定排序完毕之后，类似办法，再按十位数、百位数等等继续稳定排序。容易明白，第一趟个位数已经排好序，第二趟是对十位数排好序，排序的同时，十位数相同的数字之间，个位数也已经有序（因为对十位数的排序是个稳定排序），这样以此类推，直到最高位排好序为止。

例题：https://blog.csdn.net/qq_46105170/article/details/113790305。代码如下：

```cpp
#include <algorithm>
#include <cstring>
#include <iostream>
using namespace std;

const int N = 1e5 + 10;
int n, a[N];
int tmp[N], cnt[15];

void radix_sort(int M) {
  for (int rad = 1; rad <= M; rad *= 10) {
    memset(cnt, 0, sizeof cnt);
    for (int i = 1; i <= n; i++) cnt[a[i] / rad % 10]++;
    for (int i = 1; i < 10; i++) cnt[i] += cnt[i - 1];
    for (int i = n; i; i--) tmp[cnt[a[i] / rad % 10]--] = a[i];
    for (int i = 1; i <= n; i++) a[i] = tmp[i];
  }
}

int main() {
  scanf("%d", &n);
  for (int i = 1; i <= n; i++) scanf("%d", &a[i]);

  radix_sort(*max_element(a + 1, a + 1 + n));
  for (int i = 1; i <= n; i++) printf("%d ", a[i]);
}
```

设数据里数字最大位数为$M$，那么总体时间复杂度就是$O(Mn)$。我们看到基数排序比计数排序来讲稍微慢一些，但是空间的要求也要小的多。但基数排序只适合整数的排序（严格意义上来讲，基数排序可以对任何能保序序列化为整数的对象来排序，比如对于只含英文字母的字符串，也可以用基数排序来做。未来讲到后缀数组的时候还要重新提起），我们这里不深入讨论。

## 作业

上面例题。