线性时间排序

基于比较的排序方法都可以归结为决策树，其最优的时间复杂度为$O(nlgn)$，而不基于比较的排序方法可以有更低的时间复杂度，线性时间排序的时间复杂度为$O(n)$

决策树模型

决策树是一种完全二叉树。对于给定规模的输入数组，决策树可以表示比较大小的过程；

决策树的每个叶子节点表示原数组一种可能的排列，因此对于输入为n的数组，决策树应有$n!$
个叶子节点，每个由根节点通向某个叶子节点的道路都表示了一个排序过程，因此最长路径（即决策树的高度）表示了算法的时间复杂度；

由于决策树的叶子节点数量为$n!$，故有斯特林公式可以算出，树高为$O(nlgn)$,从而可知基于比较的排序算法时间复杂度不会优于$O(nlgn)$。

计数排序

计数排序要求输入数组的元素取值范围在[0,k]且均为整数，当$k = O(n)$时，计数排序时间复杂度为$O(n)$。

思路

基本思路：
在递增排序的情形下，对于数组中的某个元素x，只许确定小于x的元素的个数，就可以直到x在结果数组中的索引，从而完成对元素x的排序。
计数排序不是原址排序，假设输入数组为A，A.lenth=n，还需要一个数组B[1..n]来存储结果，以及一个数组C[1..k]来存储中间变量。
具体实现：
首先需要确定k的大小，即找出输入数组的最大数，这是一个$O(n)$的操作

```
FIND-K(A):
    k=A[1]
    for i from 2 to A.lenth:
        if A[i]>big:
            k=A[i]
    return k
```

然后我们要将A中每个元素的个数记录到C中，C的索引对应着A中元素的值

```
COUNTING-SORT(A,B,k):
    let C[0..k] be a new array    //申请内存,指定C数组的索引从0开始
    for i from 1 to k:
        C[i]=0                    //初始化数组C
    for j from 1 to A.lenth:
        C[A[j]] += 1              //用j遍历数组A，查看A[j]的值，该值即为C对应索引，同一个数查到一次
```

接着对C做一些变换，让C[i]记录小于等于C[i]的元素的个数

```
for i from 2 to k:
    C[i]=C[i-1]+C[i]
```

然后只需要按照C的记录把结果填入B即可

```
p=1
for i from 0 to k:
    while C[i] > 0:
        B[p]=i
        p += 1
        C[i] -= 1
```

或者

```
for j from A.lenth down to 1:
    B[C[A[j]]]=A[j]     //C[A[j]]记录了所有A[j]中最后一个在B中的索引
    C[A[j]] -= 1
```

伪代码

完整伪代码如下：

```
FIND-K(A):
    k=A[1]
    for i from 2 to A.lenth:
        if A[i]>big:
            k=A[i]
    return k

COUNTING-SORT(A,B,k):
    let C[0..k] be a new array    //申请内存,指定C数组的索引从0开始
    for i from 1 to k:
        C[i]=0                    //初始化数组C
    for j from 1 to A.lenth:
        C[A[j]] += 1              //用j遍历数组A，查看A[j]的值，该值即为C对应索引，同一个数查到一次
    for i from 2 to k:
        C[i]=C[i-1]+C[i]
    for j from A.lenth down to 1:
        B[C[A[j]]]=A[j]           //C[A[j]]记录了所有A[j]中最后一个在B中的索引
        C[A[j]] -= 1
```

时间复杂度分析

总体时间复杂度为$O(n)$。