更快的排序

When sorting needs to be faster

引言

- 在前一章当中，我们已经学习了三种排序方法的工作原理
  - 插入排序
  - 选择排序
  - 冒泡排序
- 尽管它们在不同的条件下会有不同的表现，但是总的来说，三种算法需要的时间相差无几

概要

- 这一章我们将介绍三种更快的排序方法
  - 快速排序
  - 归并排序
  - 排序网络
- 前两种方法拥有更高的执行效率，在运行速度上优势惊人
- 最后一种方法涉及到计算机中很重要的一个概念——并行化

快速排序

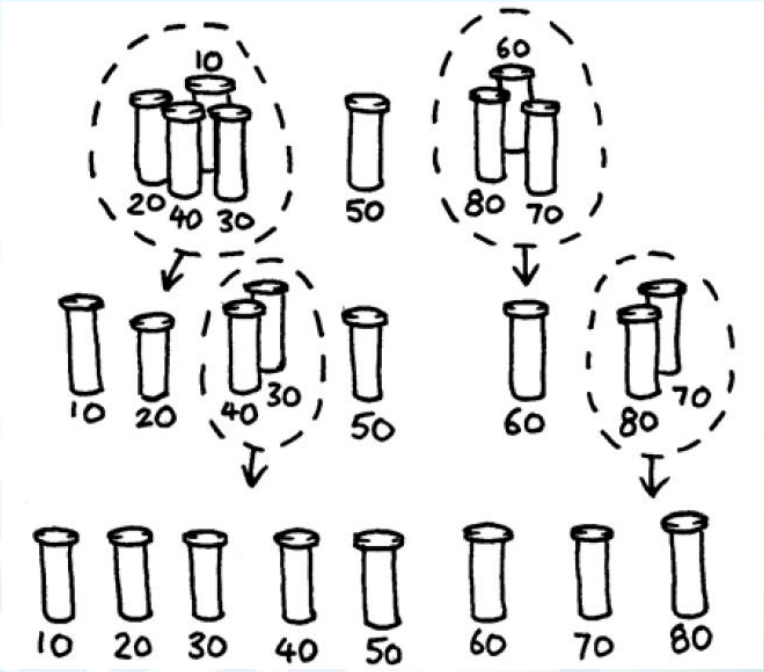
- 对于处理大量待排序对象的工作来说，快速排序（Quicksort）无疑是最佳选择
- 快速排序算法是由Tony Hoare设计的
- 对一百万个对象进行排序，使用快速排序法比冒泡排序法要快将近20000倍
- 我们将继续上一章中用到的重物和天平的例子来描述其工作原理

快速排序

- 第一步，任选一个重物，将其放置在天平的一端
- 第二步，将剩下的所有重物依次和这个重物进行对比，将较轻的放在你的左边，较重的放在右边
- 在所有重物分成两组后，将之前选取的重物放在两组之间

快速排序

- 第一阶段的一种可能的比较结果如下图第一行所示，两组对象中分别包括重量小于50的一组和重量大于50的一组



快速排序

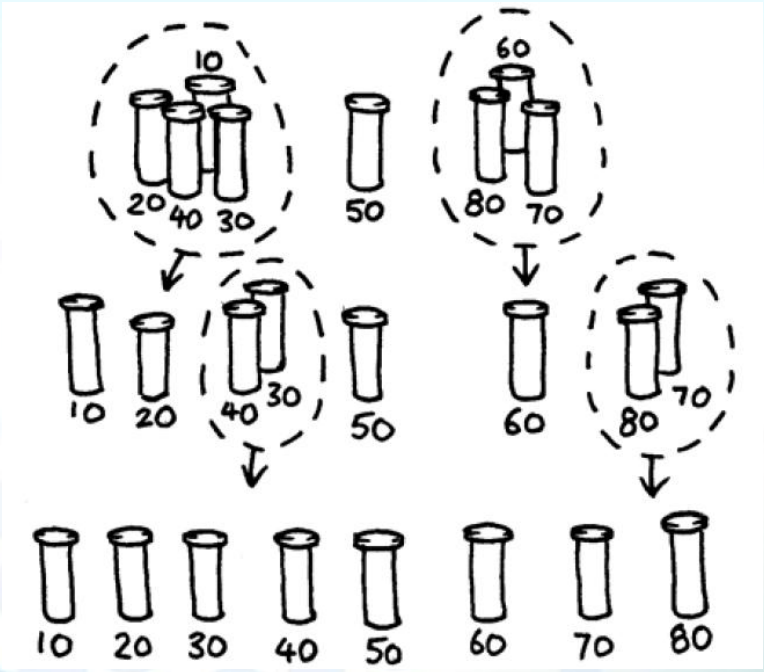
- 一开始被选出的重物我们称它为“基准”（pivot）
- 基准（上图第一行中的50）在最后排好序的序列中已经处于了正确的位置——它的左侧有4个比它小的重物，右侧有3个比它大的重物
- 我们不需要再对基准做任何操作，只要把基准两侧的对象再进行排序即可

快速排序

- 在对基准两侧的对象再进行排序时，我们采用的仍然是之前的方式
- 选择其中的一组，然后重复上述“分裂”的过程，即在该组中随机选出基准对象再将组中剩余的对象“分裂”成两组，比基准对象轻的放左边，比基准对象重的放右边，基准对象放中间
- 对另一组也进行同样的处理

快速排序

- 如下图所示的第二行，即是第二轮排序后的情况，其中左边一组的基准为20，右边一组的基准为60



快速排序

- 接下来只剩下两组只含两个对象的序列需要再次排序了，而其他的对象则已经处于排好序的状态
- 对剩下的各组进行同样的操作，直到每组中只有一个对象
- 当所有组都拆分成单项时，重物序列也完成了从最轻到最重的排序

实践与思考

- 假设有8个重物，使用快速排序法对其进行排序，记录一共比较了多少次
- 你觉得最多需要多少次比较？
- 最多的比较次数发生在怎样的情形下？
- 你觉得最少需要多少次比较？
- 最少的比较次数发生在怎样的情形下？
- 针对这样的情形，你觉得快速排序性能的关键在于什么？

快速排序

- 在快速排序中，我们用到了递归（recursion）的原理，即不断用相同的原则将序列划分成越来越小的部分，并采用相同的步骤对各部分排序
- 实际上，这种特殊的递归法被称为分治（divide and conquer），序列被不断分割知道它足够小到能够直接导出结果
- 未来的章节我们会进一步学习分治的思想

快速排序的伪代码

procedure Quicksort(List)
if (List少于一个元素) **then**
　　(退出并报告排序完毕)
选择基准pivot
分割 (Split) List 使得
　　List[first]…List[splitPoint-1] <= pivot
　　List[splitPoint] = pivot
　　List[splitPoint+1]…List[last] >= pivot
递归调用 Quicksort (List[first]…List[splitPoint-1])
递归调用 Quicksort (List[splitPoint+1]…List[last])

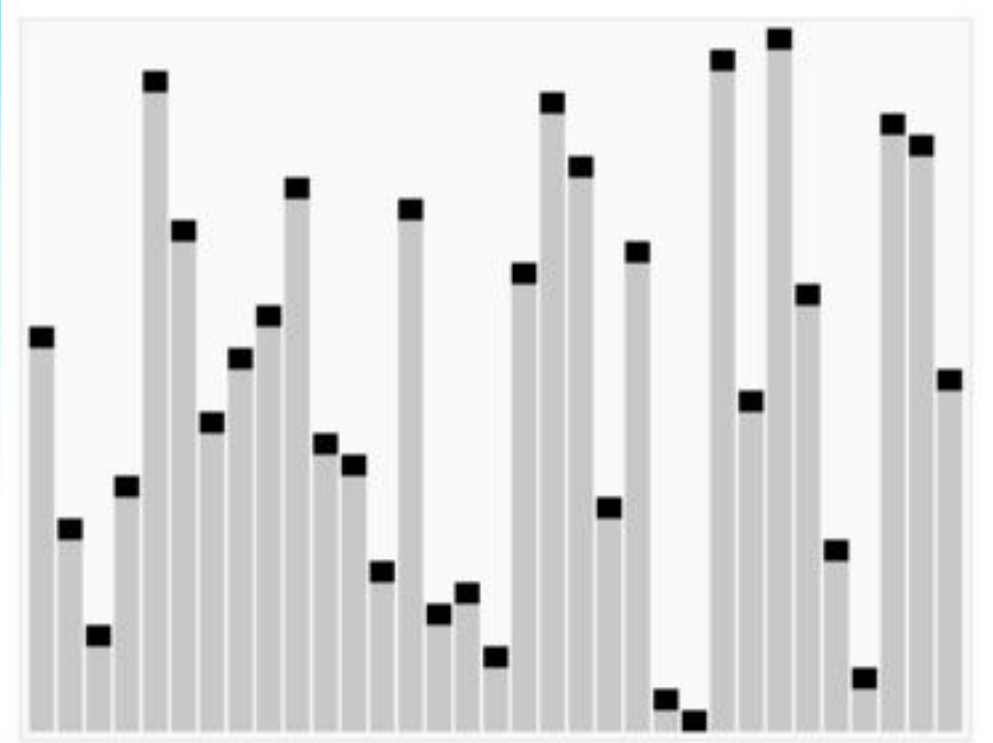
快速排序的伪代码

```
procedure Split(List)
pivot ← List[first]
left ← first + 1
right ← last
while (left <= right) do
        while (List[left]<pivot且left <= right) do  (右移left)
        while (List[right]>pivot且left <= right) do (左移right)
        if (left<right) then
                (交换List[left]与List[right])
splitPoint ← right
交换List[first]与List[right]
```

快速排序的例子

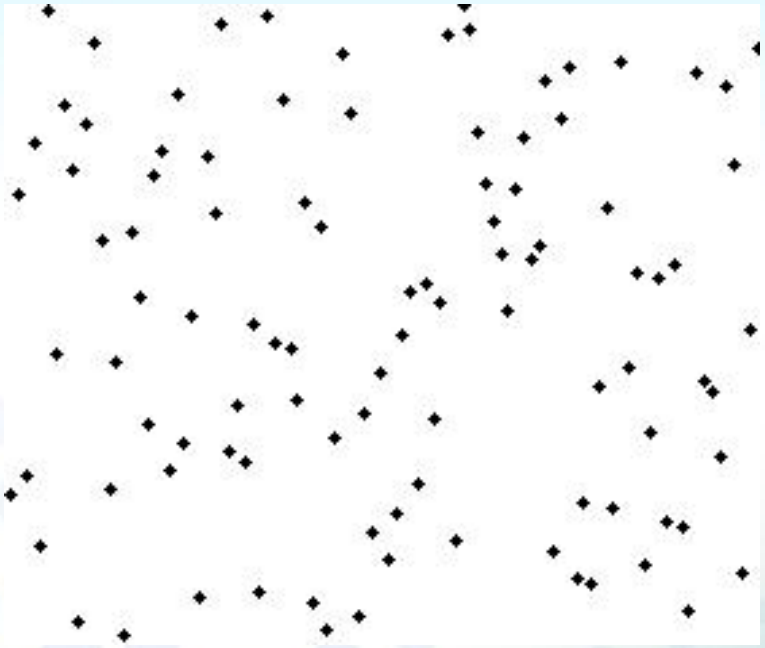

初始化
9 20 6 10 14 8 60 11
增大left 减小right
9 20 6 10 14 8 60 11
交换并往中间移动
9 8 6 10 14 20 60 11
继续向中间移动直到终止条件满足
9 8 6 10 14 20 60 11
交换List[first]与List[right]
6 8 9 10 14 20 60 11

快速排序的例子



归并排序

- 归并排序（Merge sort）是建立在归并操作上的排序算法，它同样体现了“分治”的思想

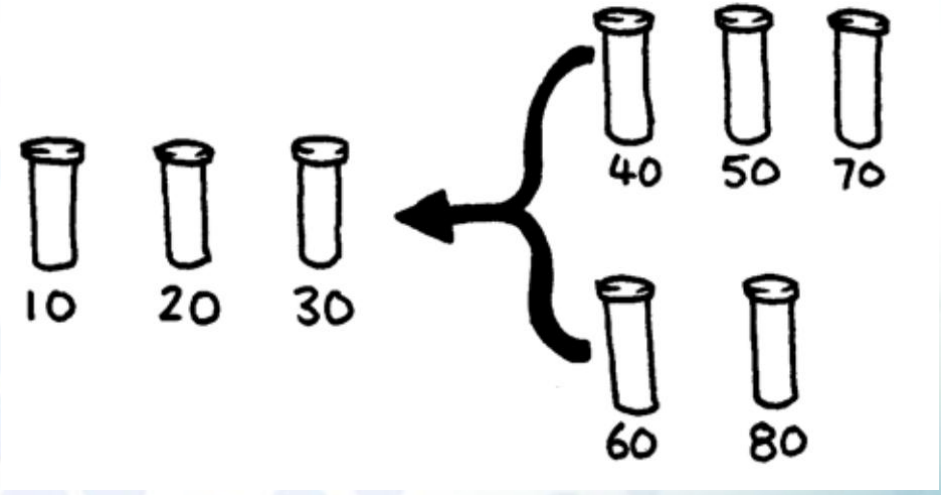


归并排序

- 首先，将待排序序列随机分成两组且对象数量相同（如果总数为奇数，则应让两组数量尽量接近）
- 然后，分别对两组对象进行排序，再将两组对象归并起来
- 归并两个已排好序的序列很容易，你只要不断地移出两组对象最前端较小的那个即可

归并排序

- 在下图中，40克和60克重物位于两个子序列的前端，通过比较，选出较轻的40克重物即可。之后，再比较50克和60克的重物，把50克的重物移出放入排好序的序列中



归并排序

- 思考：应当怎样对刚才的子序列进行排序呢？
- 继续使用归并排序法！
- 即将子序列再分割成两半，对它们进行排序，再归并起来
- 最终，所有的子序列都变成单独的对象，这样便说明此时每个子序列中的对象都已经排好序了

归并排序

- 让我们用另一种方式来理解归并排序法
- 开始，每个重物归并到只有1个对象的小组
- 然后，选2个重物将它们“归并”成拥有2个对象的小组
- 下一步，再选2个排好序的小组（每个小组中均含有2个对象）将它们归并成含有4个对象的小组
- 依次类推，从而最终完成整个排序过程

归并排序的伪代码

```
procedure Mergesort(List)
middle ← List的正中 (N/2)
firstList ← Mergesort(List[1]…List[middle])
lastList ← Mergesort(List[middle+1]…List[N])
i ← 1
while (i <= N) do
        if (firstList中元素较大) then
                (List[i]为firstList中元素并使firstList右移)
        else
                (List[i]为lastList中元素并使lastList右移)
        i ← i + 1
```

选择排序
插入排序
冒泡排序
快速排序
归并排序

思考：我们现在讲过的五种排序方法各有什么特点？

进阶讨论

快速排序 vs 归并排序

- 快速排序是先将序列处理成两个有大小关系的子序列，再对这两个子序列进行排序
- 归并排序是先获得两个已经有序的子序列，再对这两个子序列进行合并
- 从“分治”的角度来说，前者是先“治”再“分”，后者是先“分”再“治”

快速排序 vs 归并排序

- 快速排序对n个对象进行排序平均需要比较$1.39n\log_2 n$次，这一结果要明显优于需要耗费n^2次比较的诸如插入排序这样的算法，因为在n很大的时候，其对数值也很小
- 归并排序法平均只需要$n\log_2 n$次比较，也就是比快速排序法平均少39%，然而在实际应用中差别并不明显，因为归并排序在合并过程中需要临时存储空间，整体性能反而不如快速排序

快速排序 vs 归并排序

- 归并排序一般用于合并大的数据序列，因此它比较适合处理存储在硬盘上的数据，比如用于外部排序的多路归并排序算法
- 快速排序法总是要将数据移动到不同的位置，因此它更适合处理存储在计算机内存中的数据

快速排序 vs 归并排序

- 另外快速排序是不稳定的，而归并排序是稳定的
- 排序中的“稳定”是指对于相同的关键字，在最终的排序序列中保持原来的位置关系
- 冒泡排序和插入排序是稳定的
- 选择排序是不稳定的
- C和C++中标准库使用的是快速排序，而JAVA中使用的是归并排序

快速排序

- 前面已经提到，在快速排序中选择“基准”是非常重要的一个步骤，否则快速排序在极端情况下会退化为n^2的算法
- 一般选择随机化的方法
- 还有一种可行的方法是在第一个元素，最后一个元素和中间元素三个元素当中选取中位数，作为最终的基准

快速排序

- 然而对于真实世界的应用来说，可能会有人特别构造一个序列使你的排序算法退化，从而实现对你提供的服务进行攻击
- 为了避免退化，在SGI标准库中使用了叫做Introsort的方法，在发现可能退化的情况下会使用堆排序（另一种$n\log_2 n$的排序方法）
- 更多关于快速排序的分析可以参考《算法导论》第7章

趣闻

- 在2008年美国大选前，Google采访了总统候选人贝拉克·奥巴马。当奥巴马被问到“对一百万个32位整数进行排序的最佳方法”时，他回答道：“冒泡排序法绝对不是正确答案”。

排序网络

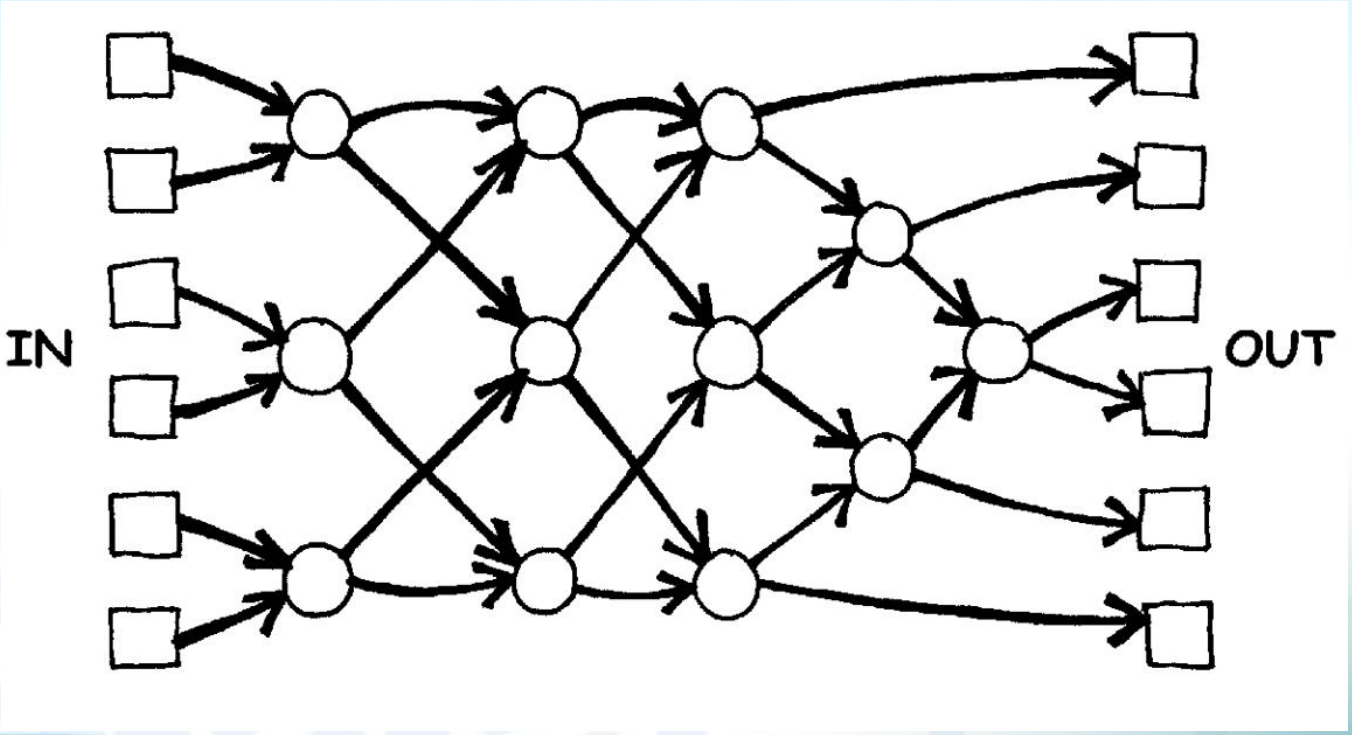
- 人们一直渴望让计算机处理信息的速度变得更快，但现在计算机硬件的发展已经接近了一个极限，“摩尔定律”可能也不再适用
- 尽管如此，计算机的制造成本已经相当廉价。因此人们开始使用多个处理器，让每个处理器同时处理问题的不同部分。这一方法被称为并行计算（parallel computation）

排序网络

- 对于非并行的基于比较的排序算法，归并排序和快速排序已经是最快的了，这可以从理论上进行证明
- 如果想要进一步加快运算速度，并行计算是其中的一种方案——事实上在大多数情况下，它往往也是唯一的选择
- 排序网络就是在这样的背景下诞生的，我们将来看看如何利用排序网络来快速排序

排序网络

- 下图所示的排序网络可以一次对6个数字进行排序

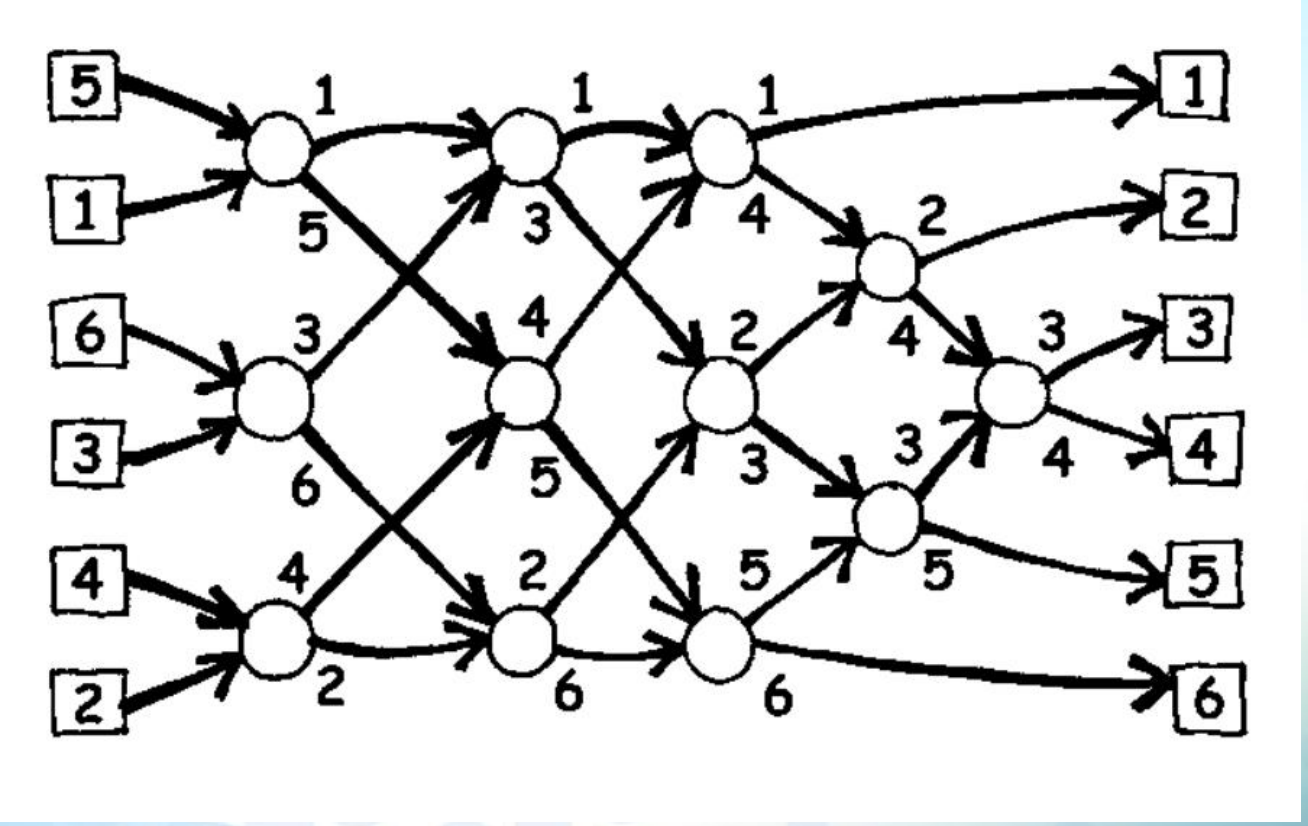


排序网络

- 待排序的6个数字开始被放在左侧的6个框中。每一步，将它们沿着箭头移动到圆圈中（称为一个节点）
- 在节点处比较此处的两个数字，较小的数字沿着向上的箭头移出节点，较大的数字沿着向下的箭头移动

排序网络

- 下图中显示了所有数字在前面所述的排序网络中移动的方式



是否正确？

实践与思考

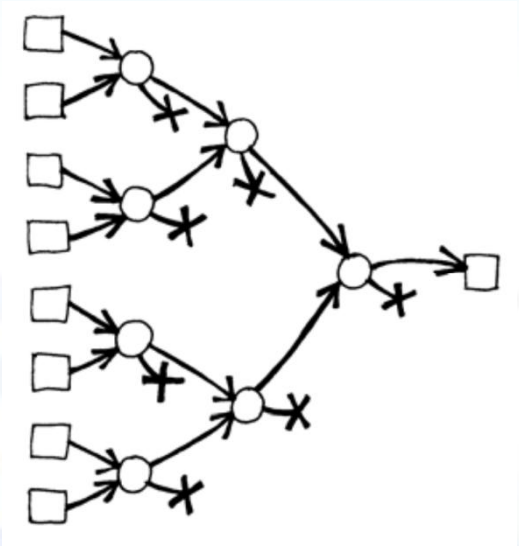
- 上例中，移动到最右侧的数字已经被排好顺序了。试试将数字的顺序重新打乱后再用这个排序网络来排序，是否总能输出排好序的结果呢？
- 数字从左至右穿过网络一共需要多少步？注意一些比较是同时发生的，而另一些比较是需要等前一步比较完成后才能开始

排序网络

- 借助排序网络要比我们前面借助天平排序快得多，因为排序网络能够同时进行多组比较，在这个例子中是3组比较
- 在这个例子中，并行排序网络要比只能执行单次比较的系统快上2倍，但排序网络则需要使用到3倍数量的设备资源

排序网络

- 网络结构也能够被用于处理许多其它大规模输入的问题
- 你知道下面所示网络在做什么运算么？它与生活中的哪个过程很相似？



排序网络

- 如果你想来测试一下你构造的排序网络是否正确，假设待排序的对象有n个，那么意味着你需要测试n!种排列情况
- 幸运的是，可以通过“01原理”来简化我们的问题，并且能够证明其正确性
- 关于排序网络的更多内容，可以参考《算法导论》第27章，上面介绍了如何构造一般性的排序网络以及其所需的比较次数

并行计算

- 并不是所有的问题都能进行并行处理的
- 如果一个人需要用9天时间挖一个9米长的水渠，那么9个人同时挖一天就能完成
- 但是如果是挖9米深的土洞，事情就没有这么简单了
- 一个女人可以九月怀胎产下一个小孩，但并不意味着9个女人一起努力，就能让小孩在一个月之内生出来……

总结

- 递归
- 分治
- 并行
- 这些都是计算机中非常重要的思想，在计算机最重要的问题之一——排序上，我们已经看到了它们的应用
- 在未来还会分别对它们进行具体的介绍

Thank you

Q&A