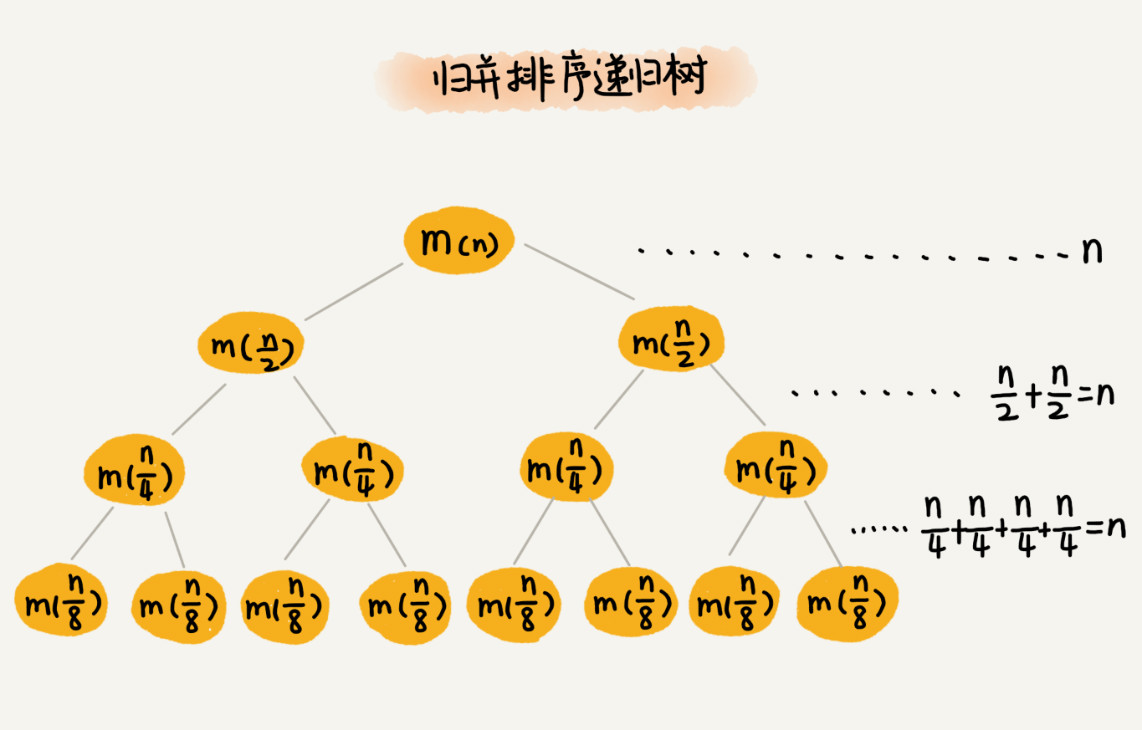
**一. 递归树与时间复杂度分析**

1. 归并排序的时间复杂度

归并排序每次会将数据规模一分为二。我们把归并排序画成递归树，就是下面这个样子：



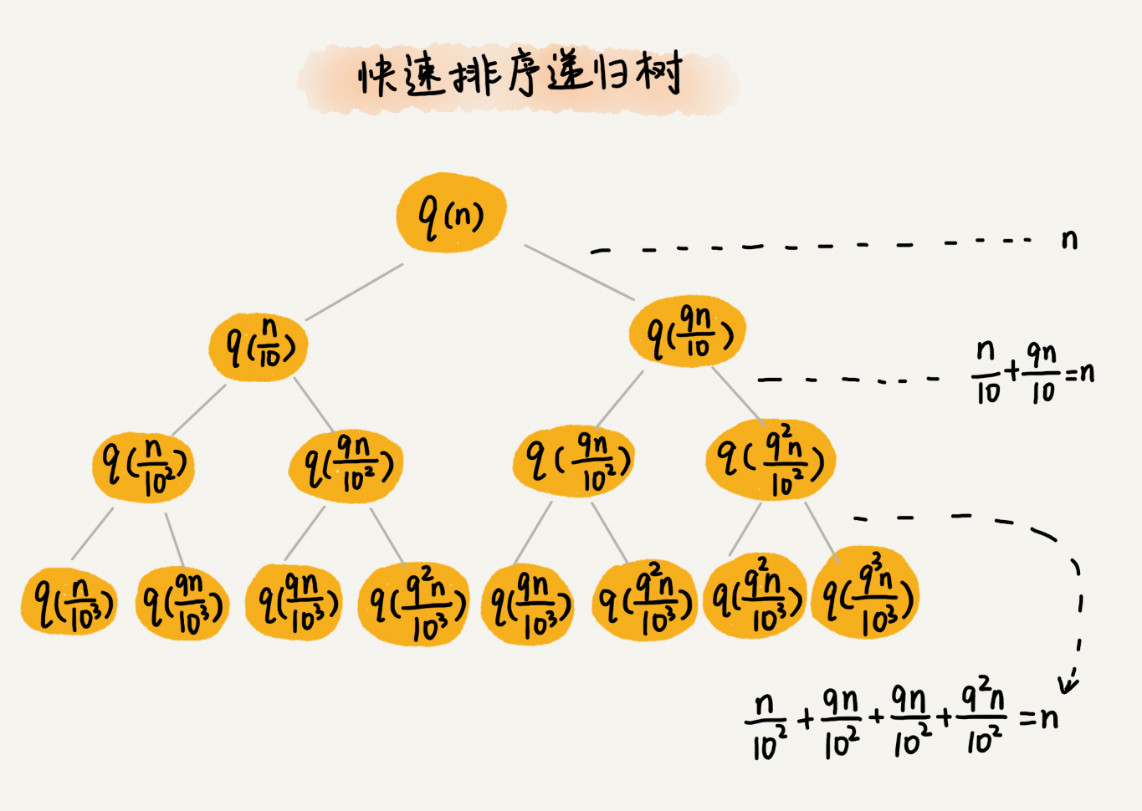
因为每次分解都是一分为二，所以代价很低，我们把时间上的消耗记作常量 1。归并算法中比较耗时的是归并操作，也就是把两个子数组合并为大数组。从图中我们可以看出，每一层归并操作消耗的时间总和是一样的，跟要排序的数据规模有关。

我们把每一层归并操作消耗的时间记作 n。现在，我们只需要知道这棵树的高度 h，用高度 h 乘以每一层的时间消耗 n，就可以得到总的时间复杂度 O(n∗h)。从归并排序的原理和递归树，可以看出来，归并排序递归树是一棵满二叉树。我们前两节中讲到，满二叉树的高度大约是 log2​n，（层数int（log2n）+1，从叶子节点层向上归并算作一次O(n)，所以乘以高度等于层数-1，log2n，叶子节点层高度为0）所以，归并排序递归实现的时间复杂度就是 O(nlogn)。我这里的时间复杂度都是估算的，对树的高度的计算也没有那么精确，但是这并不影响复杂度的计算结果。

2. 快速排序的时间复杂度

快速排序在最好情况下，每次分区都能一分为二，这个时候用递推公式 T(n)=2T(2n​)+n，很容易就能推导出时间复杂度是 O(nlogn)。但是，我们并不可能每次分区都这么幸运，正好一分为二。

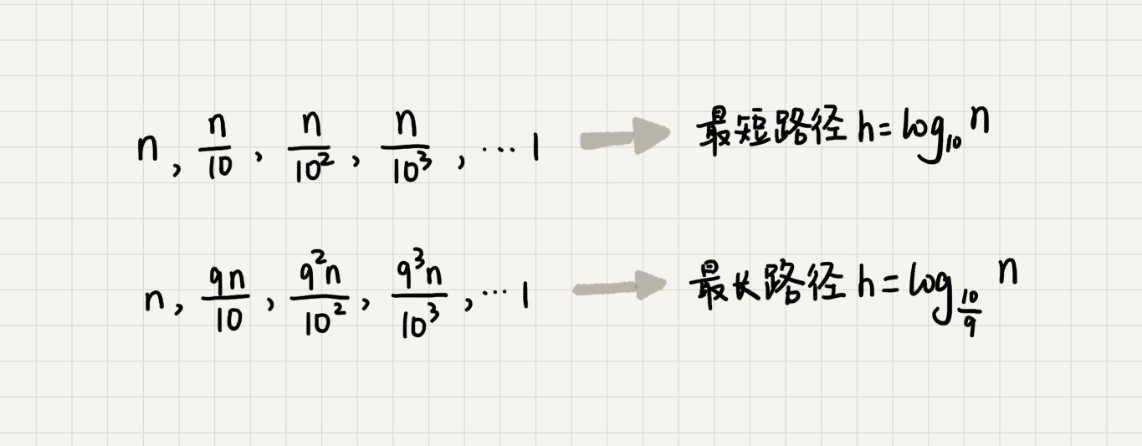
我们假设平均情况下，每次分区之后，两个分区的大小比例为 1:k。当 k=9 时，如果用递推公式的方法来求解时间复杂度的话，递推公式就写成 T(n)=T(​)+T(​)+n。



快速排序的过程中，每次分区都要遍历待分区区间的所有数据，所以，每一层分区操作所遍历的数据的个数之和就是 n。我们现在只要求出递归树的高度 h，这个快排过程遍历的数据个数就是 h∗n ，也就是说，时间复杂度就是 O(h∗n)。

因为每次分区并不是均匀地一分为二，所以递归树并不是满二叉树。

我们知道，快速排序结束的条件就是待排序的小区间，大小为 1，也就是说叶子节点里的数据规模是 1。从根节点 n 到叶子节点 1，递归树中最短的一个路径每次都乘以 ​​，最长的一个路径每次都乘以 ​​。通过计算，我们可以得到，从根节点到叶子节点的最短路径是 ，最长的路径是



所以，遍历数据的个数总和就介于 nlog10​n 和 nlog910​​n 之间。根据复杂度的大 O 表示法，对数复杂度的底数不管是多少，我们统一写成 logn，所以，当分区大小比例是 1:9 时，快速排序的时间复杂度仍然是 O(nlogn)。

3. 斐波那契数列的时间复杂度

int f(int n) {

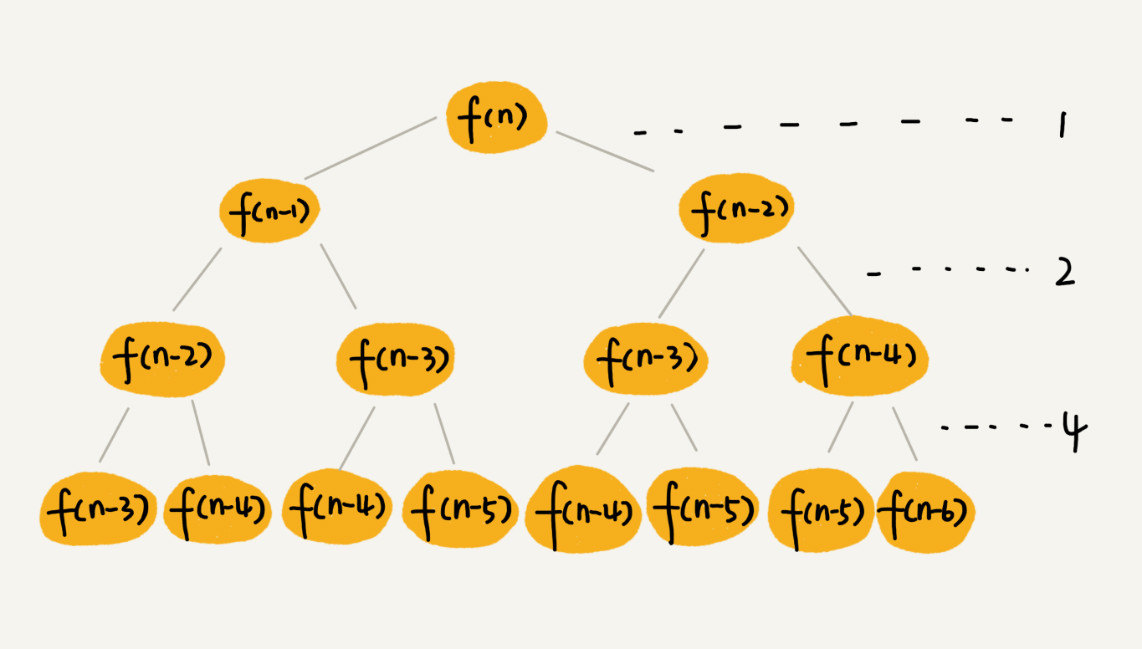
if (n == 1) return 1;

if (n == 2) return 2;

return f(n-1) + f(n-2);

} 斐波那契，跨台阶问题：一共n级台阶，一次可以上1级或2级，总共有几种上台阶的方案。

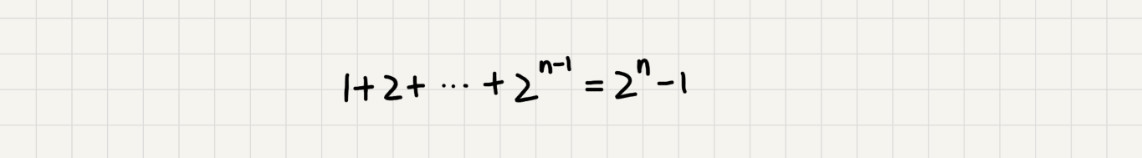
可以分为两种情况，一是第一次上1级台阶，之后n-1级台阶的上发；二是第一次上2级台阶剩下n-2级台阶的上法。所以f（n）=f(n-1)+ f(n-2);



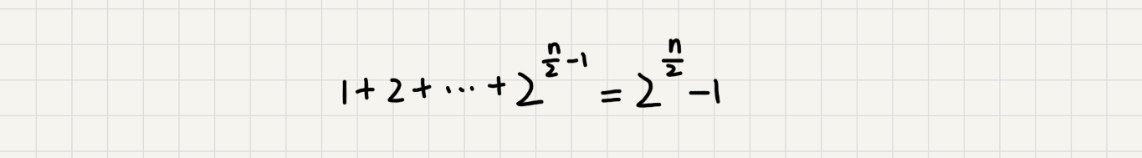
f(n) 分解为 f(n−1) 和 f(n−2)，每次数据规模都是 −1 或者 −2，叶子节点的数据规模是 1 或者 2。所以，从根节点走到叶子节点，每条路径是长短不一的。如果每次都是 −1，那最长路径大约就是 n；如果每次都是 −2，那最短路径大约就是 n/2​。

每次分解之后的合并操作只需要一次加法运算，我们把这次加法运算的时间消耗记作 1。所以，从上往下，第一层的总时间消耗是 1，第二层的总时间消耗是 2，第三层的总时间消耗就是 2^2。依次类推，第 k 层的时间消耗就是 2^(k-1)，那整个算法的总的时间消耗就是每一层时间消耗之和。

如果路径长度都为 n，那这个总和就是 2^n−1。



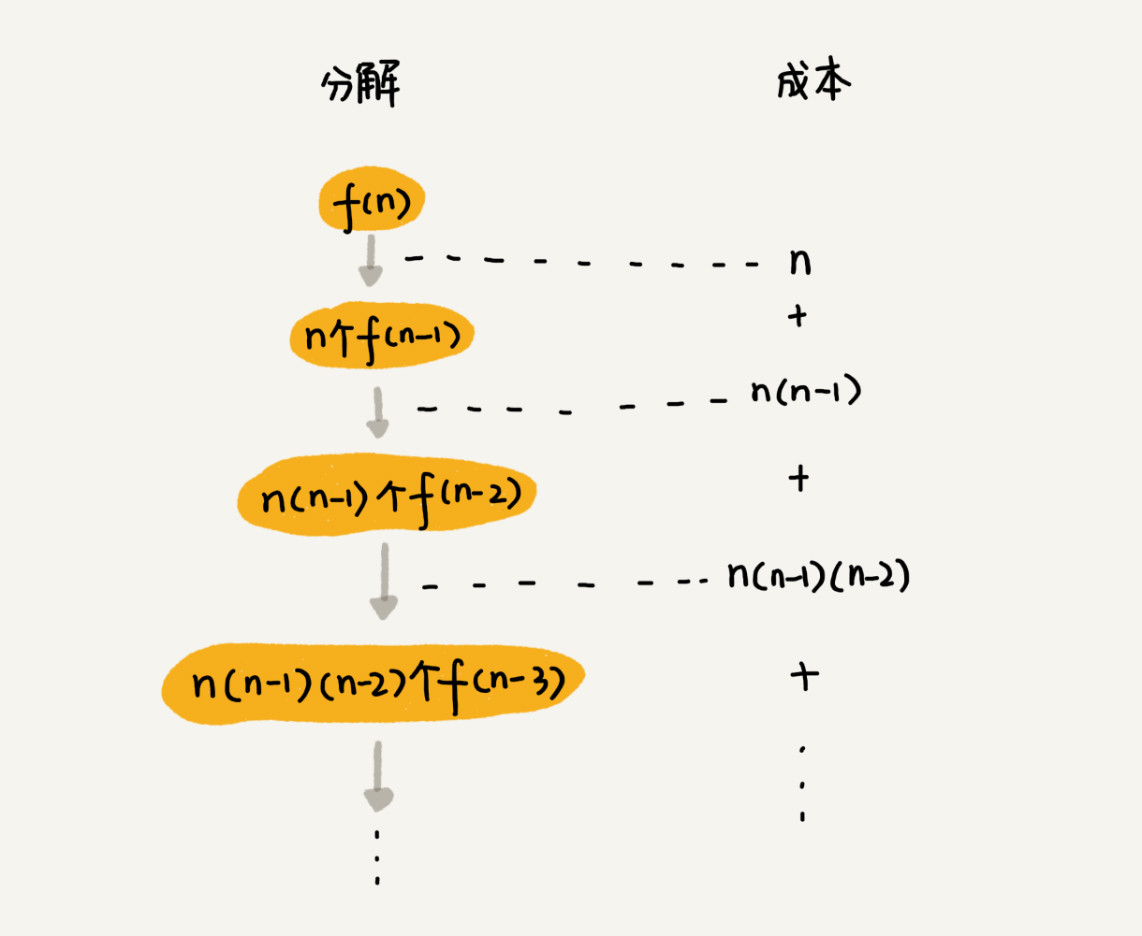
如果路径长度都是n/2​ ，那整个算法的总的时间消耗就是 2^(n/2)​−1。



所以，这个算法的时间复杂度就介于 O(2^n) 和 O(2^(n/2)​) 之间。虽然这样得到的结果还不够精确，只是一个范围，但是我们也基本上知道了上面算法的时间复杂度是指数级的，非常高。

3. 全排列的时间复杂度

如果我们确定了最后一位数据，那就变成了求解剩下 n−1 个数据的排列问题。而最后一位数据可以是 n 个数据中的任意一个，因此它的取值就有 n 种情况。所以，“n 个数据的排列”问题，就可以分解成 n 个“n−1 个数据的排列”的子问题。

第一层分解有 n 次交换操作，第二层有 n 个节点，每个节点分解需要 n−1 次交换，所以第二层总的交换次数是 n∗(n−1)。第三层有 n∗(n−1) 个节点，每个节点分解需要 n−2 次交换，所以第三层总的交换次数是 n∗(n−1)∗(n−2)。以此类推，第 k 层总的交换次数就是 n∗(n−1)∗(n−2)∗…∗(n−k+1)。最后一层的交换次数就是 n∗(n−1)∗(n−2)∗…∗2∗1。每一层的交换次数之和就是总的交换次数。

这个公式的求和比较复杂，我们看最后一个数，n∗(n−1)∗(n−2)∗…∗2∗1 等于 n!，而前面的 n−1 个数都小于最后一个数，所以，总和肯定小于 n∗n!，也就是说，全排列的递归算法的时间复杂度大于 O(n!)，小于 O(n∗n!)，虽然我们没法知道非常精确的时间复杂度，但是这样一个范围已经让我们知道，全排列的时间复杂度是非常高的。

4. 1 个细胞的生命周期是 3 小时，1 小时分裂一次。求 n 小时后，容器内有多少细胞？

If n < =0: return 0

If n == 1: return 1

If n == 2: return 2

If n == 3: return 4

If n == 4: return 7

f(n) = f(n-1)\*2 – f(n-4)