27|递归树：如何借助树来求解递归算法的时间复杂度?

今天，我们来讲树这种数据结构的一种特殊应用，递归树。

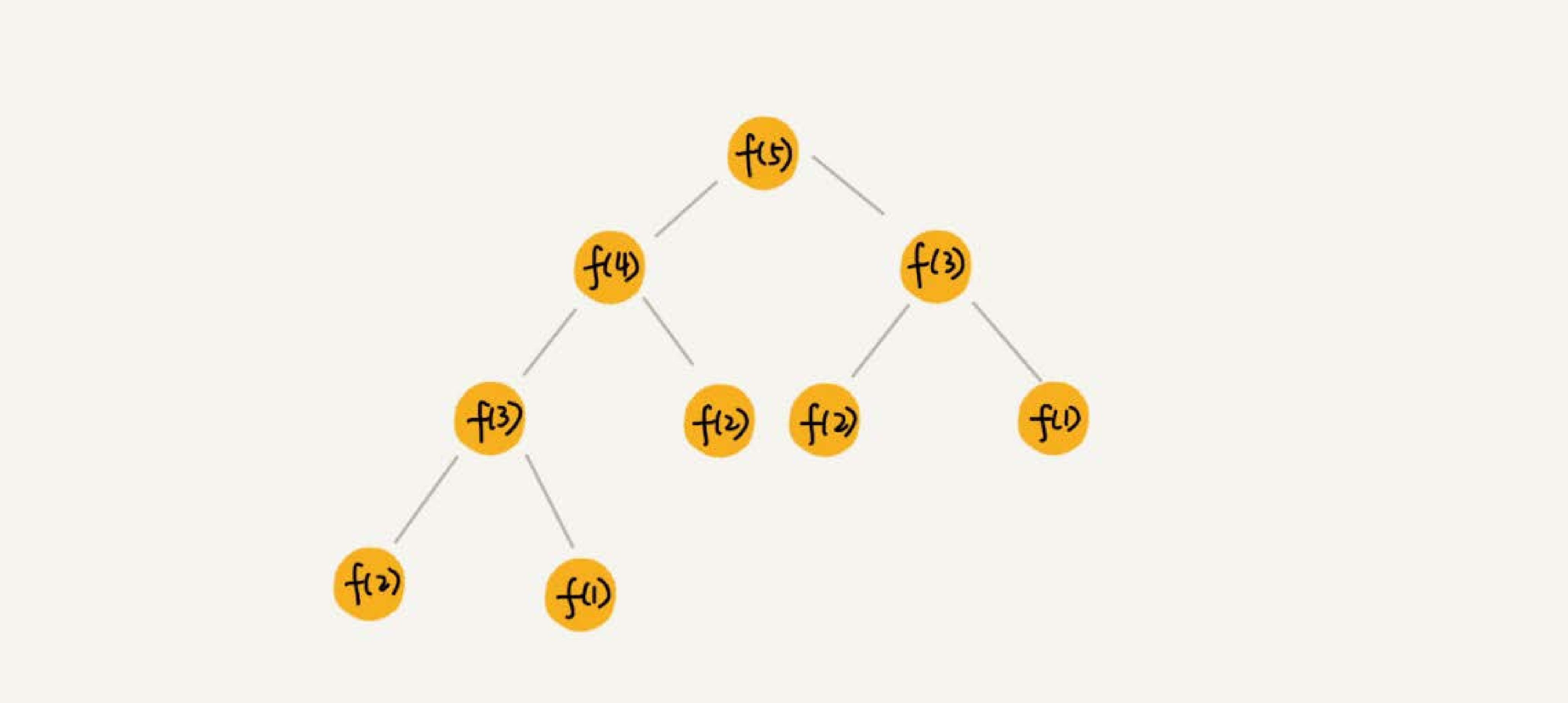
我们都知道，递归代码的时间复杂度分析起来很麻烦。我们在第12节《排序（下）》那里讲过，如何利用递推公式，求解归并排序、快速排序的时间复杂度，但是，有些情况，比如快排的平均时间复杂度的分析，用递推公式的话，会涉及非常复杂的数学推导。

除了用递推公式这种比较复杂的分析方法，有没有更简单的方法呢？今天，我们就来学习另外一种方法，借助递归树来分析递归算法的时间复杂度。

递归树与时间复杂度分析

我们前面讲过，递归的思想就是，将大问题分解为小问题来求解，然后再将小问题分解为小小问题。这样一层一层地分解，直到问题的数据规模被分解得足够小，不用继续递归分解为止。

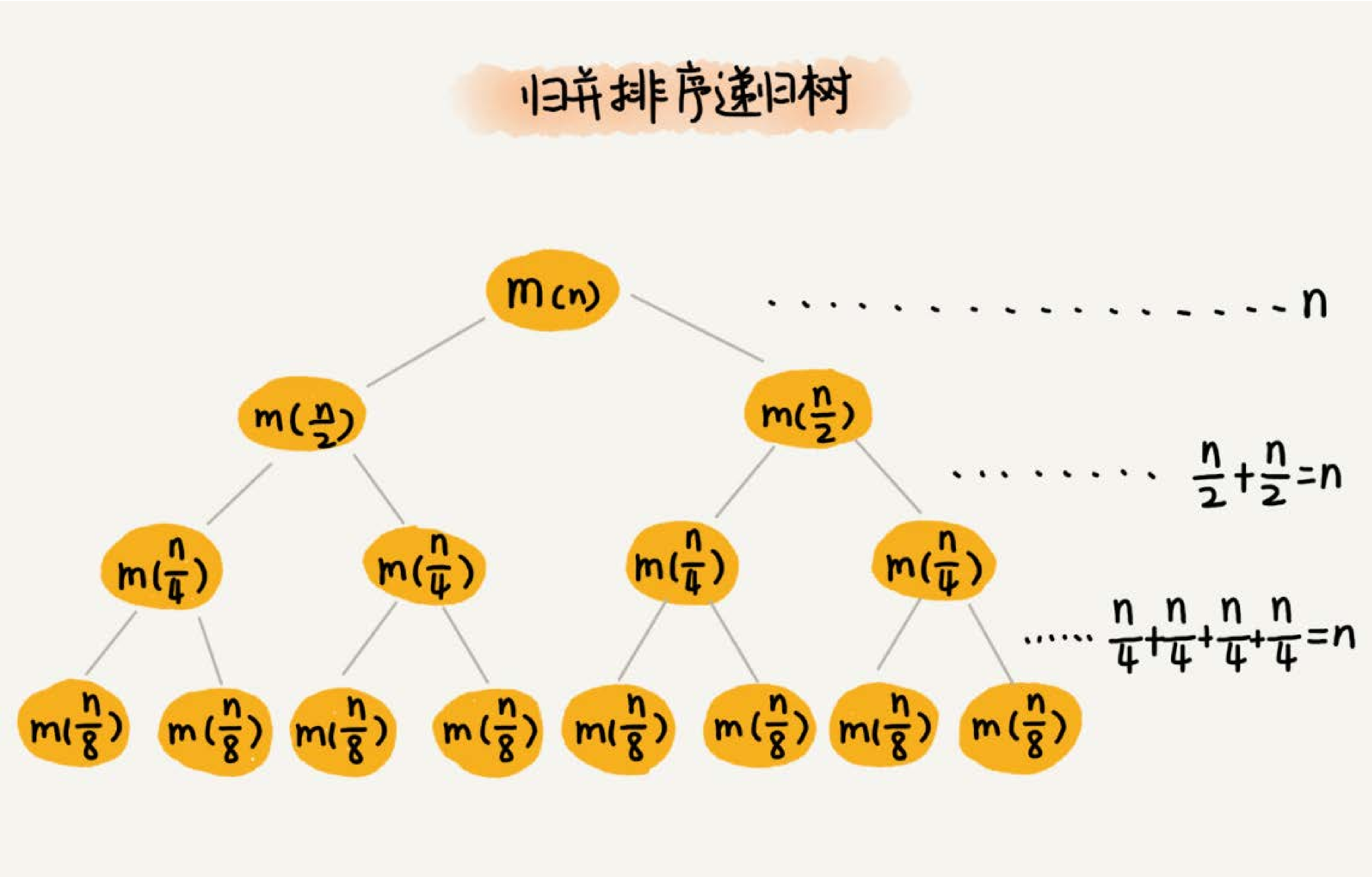
如果我们把这个一层一层的分解过程画成图，它其实就是一棵树。我们给这棵树起一个名字，叫作递归树。我这里画了一棵斐波那契数列的递归树，你可以看看。节点里的数字表示数据的规模，一个节点的求解可以分解为左右子节点两个问题的求解。



通过这个例子，你对递归树的样子应该有个感性的认识了，看起来并不复杂。现在，我们就来看，如何用递归树来求解时间复杂度。

归并排序算法你还记得吧？它的递归实现代码非常简洁。现在我们就借助归并排序来看看，如何用递归树，来分析递归代码的时间复杂度。

归并排序的原理我就不详细介绍了，如果你忘记了，可以回看一下第12节的内容。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$\log_{2}n$，所以，归并排序递归实现的时间复杂度就是$O(n\log n)$。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。

利用递归树的时间复杂度分析方法并不难理解，关键还是在实战，所以，接下来我会通过三个实际的递归算法，带你实战一下递归的复杂度分析。学完这节课之后，你应该能真正掌握递归代码的复杂度分析。

实战一：分析快速排序的时间复杂度

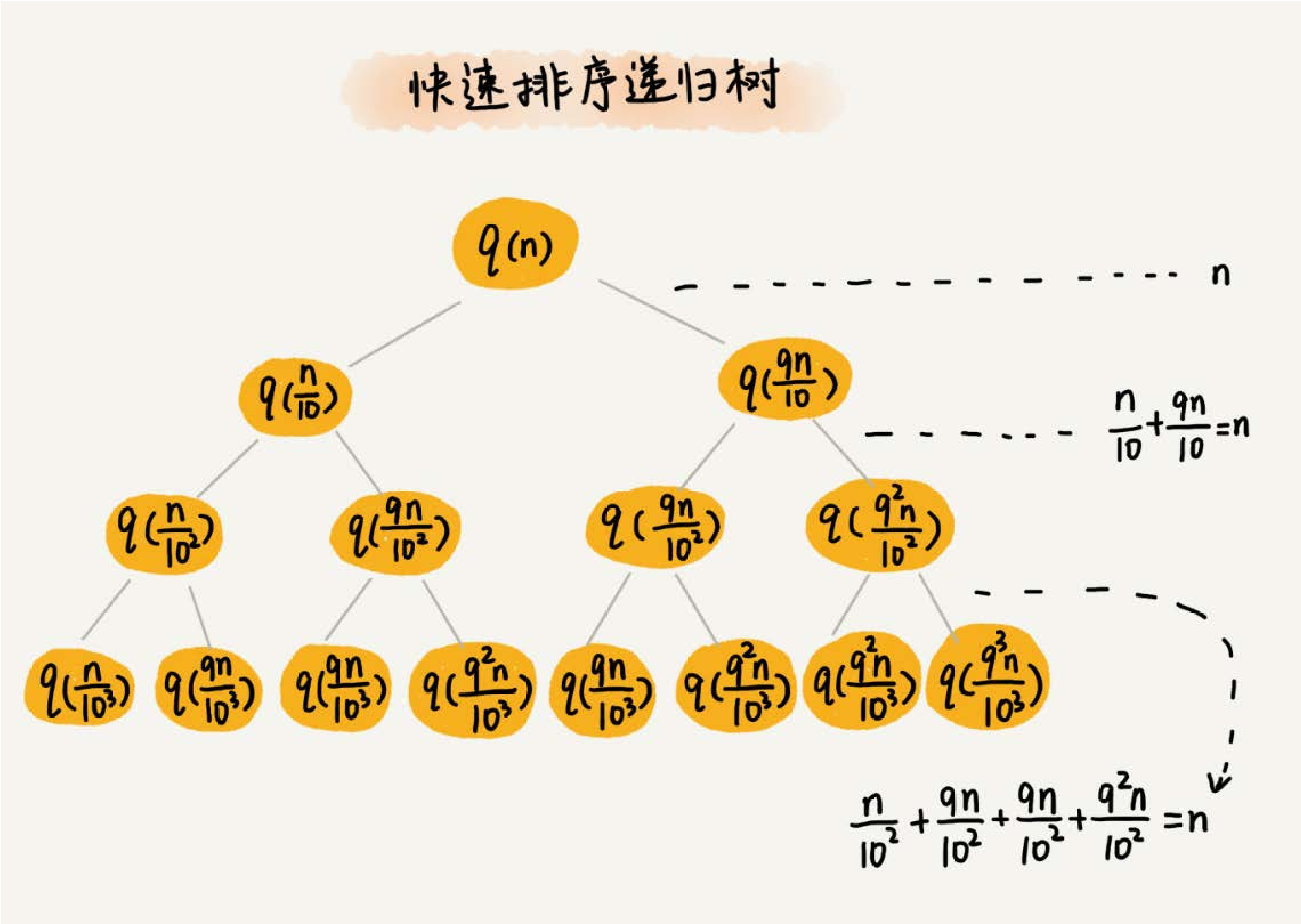
在用递归树推导之前，我们先来回忆一下用递推公式的分析方法。你可以回想一下，当时，我们为什么说用递推公式来求解平均时间复杂度非常复杂？

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$T(n)=2T(\frac{n}{2})+n$，很容易就能推导出时间复杂度是$O(n\log n)$。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$T(n)=T(\frac{n}{10})+T(\frac{9n}{10})+n$。

这个公式可以推导出时间复杂度，但是推导过程非常复杂。那我们来看看，用递归树来分析快速排序的平均情况时间复杂度，是不是比较简单呢？

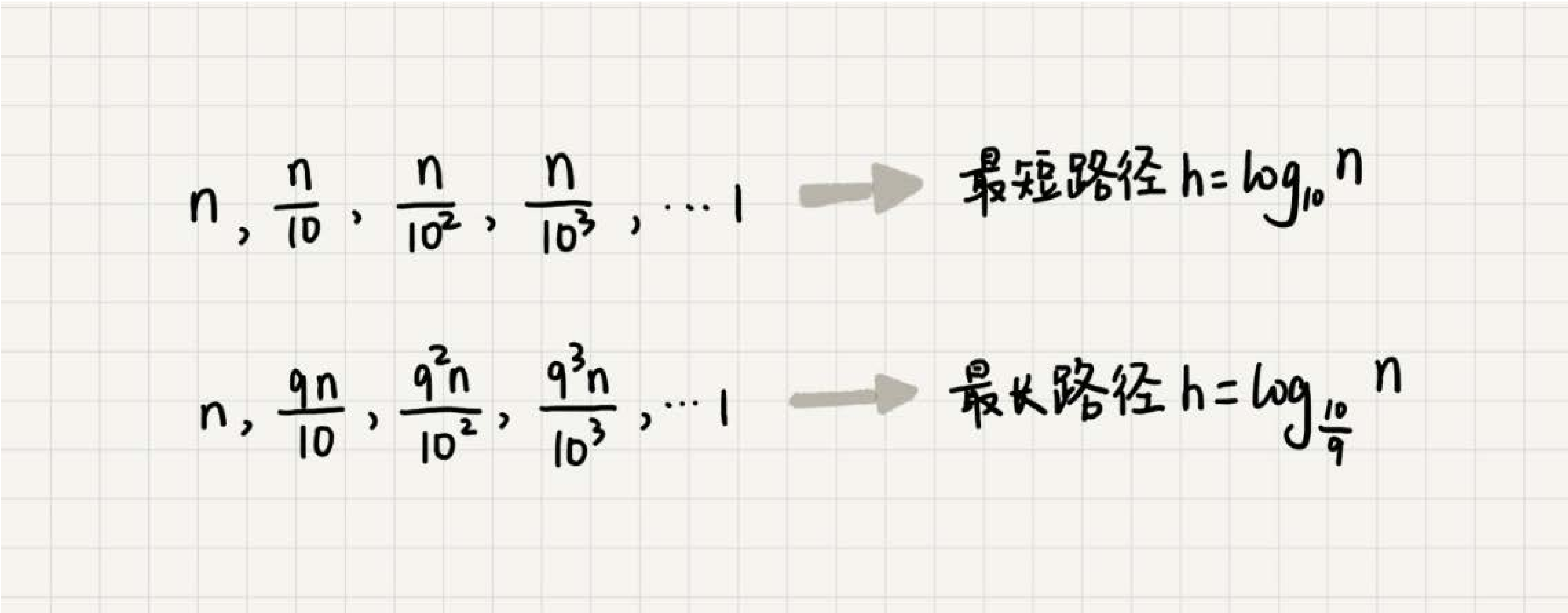
我们还是取k等于9，也就是说，每次分区都很不平均，一个分区是另一个分区的9倍。如果我们把递归分解的过程画成递归树，就是下面这个样子：


快速排序递归树
q(n)
n
$q(\frac{n}{10})$
$q(\frac{9n}{10})$
$\frac{n}{10}+\frac{9n}{10}=n$
$q(\frac{n}{10^2})$
$q(\frac{9n}{10^2})$
$q(\frac{9n}{10^2})$
$q(\frac{9^2n}{10^2})$
$q(\frac{n}{10^3})$
$q(\frac{9n}{10^3})$
$q(\frac{9n}{10^3})$
$q(\frac{9^2n}{10^3})$
$q(\frac{9n}{10^3})$
$q(\frac{9^2n}{10^3})$
$q(\frac{9^2n}{10^3})$
$q(\frac{9^3n}{10^3})$
$\frac{n}{10^2}+\frac{9n}{10^2}+\frac{9n}{10^2}+\frac{9^2n}{10^2}=n$

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。这样一个递归树的高度是多少呢？

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$\frac{1}{10}$，最长的一个路径每次都乘以$\frac{9}{10}$。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$\log_{10}n$，最长的路径是$\log_{\frac{10}{9}}n$。



所以，遍历数据的个数总和就介于$n\log_{10}n$和$n\log_{\frac{10}{9}}n$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$\log n$，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$O(n\log n)$。

刚刚我们假设$k=9$，那如果$k=99$，也就是说，每次分区极其不平均，两个区间大小是$1:99$，这个时候的时间复杂度是多少呢？

我们可以类比上面$k=9$的分析过程。当$k=99$的时候，树的最短路径就是$\log_{100}n$，最长路径是$\log_{\frac{100}{99}}n$，所以总遍历数据个数介于$n\log_{100}n$和$n\log_{\frac{100}{99}}n$之间。尽管底数变了，但是时间复杂度也仍然是$O(n\log n)$。

也就是说，对于k等于9，99，甚至是999，9999……，只要k的值不随n变化，是一个事先确定的常量，那快排的时间复杂度就是$O(n\log n)$。所以，从概率论的角度来说，快排的平均时间复杂度就是$O(n\log n)$。

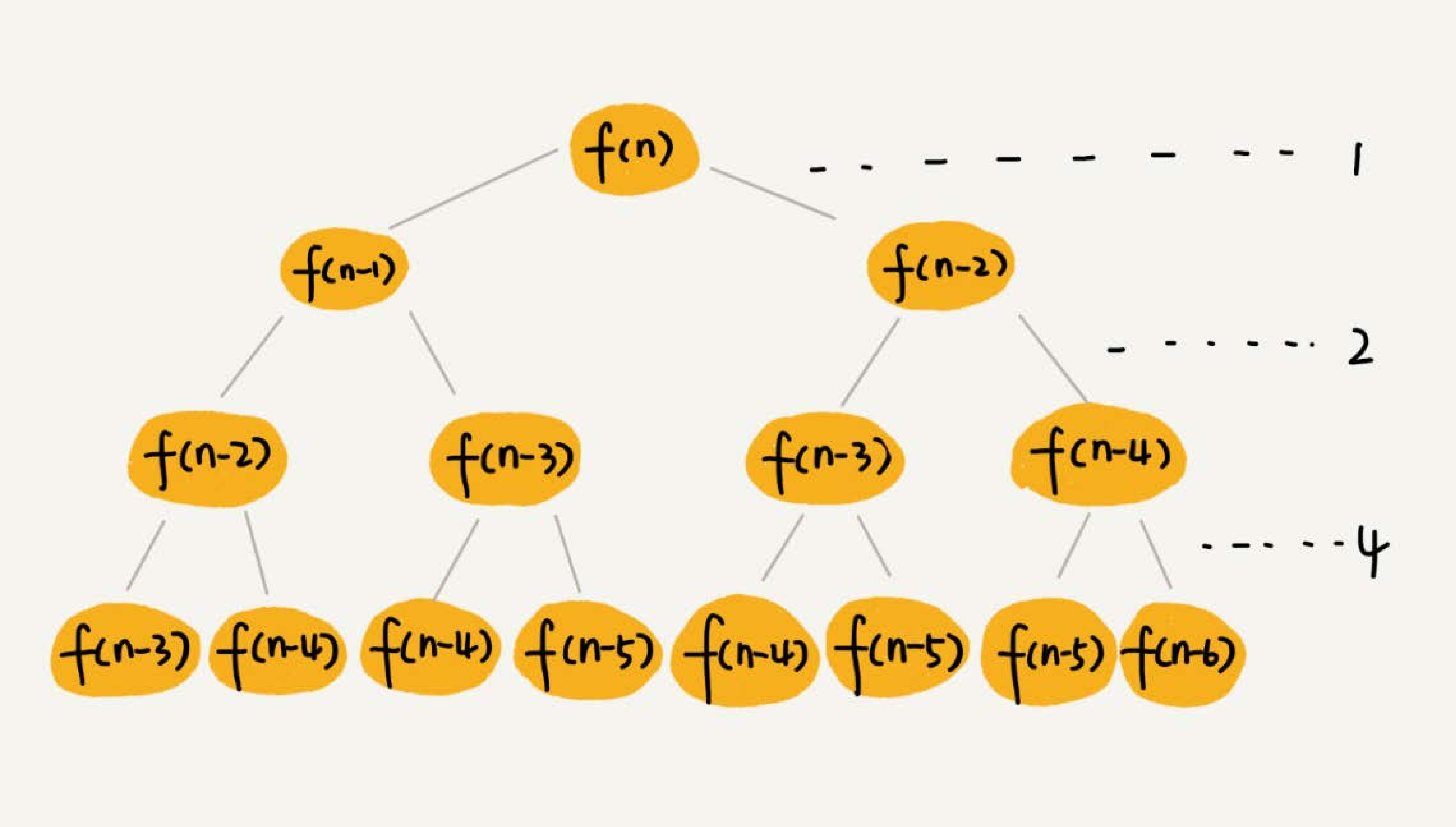
实战二：分析斐波那契数列的时间复杂度

在递归那一节中，我们举了一个跨台阶的例子，你还记得吗？那个例子实际上就是一个斐波那契数列。为了方便你回忆，我把它的代码实现贴在这里。

```
int f(int n) {
  if (n == 1) return 1;
  if (n == 2) return 2;
  return f(n-1) + f(n-2);
}
```

这样一段代码的时间复杂度是多少呢？你可以先试着分析一下，然后再来看，我是怎么利用递归树来分析的。

我们先把上面的递归代码画成递归树，就是下面这个样子：



这棵递归树的高度是多少呢?

$f(n)$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$\frac{n}{2}$。

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$2^{n}-1$。

$$1+2+\cdots+2^{n-1}=2^{n}-1$$

如果路径长度都是$\frac{n}{2}$ ，那整个算法的总的时间消耗就是$2^{\frac{n}{2}}-1$。

$$1+2+\cdots+2^{\frac{n}{2}-1}=2^{\frac{n}{2}}-1$$

所以，这个算法的时间复杂度就介于$O(2^{n})$和$O(2^{\frac{n}{2}})$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。

实战三：分析全排列的时间复杂度

前面两个复杂度分析都比较简单，我们再来看个稍微复杂的。

我们在高中的时候都学过排列组合。“如何把n个数据的所有排列都找出来”，这就是全排列的问题。

我来举个例子。比如，$1， 2，3$这样3个数据，有下面这几种不同的排列：

```
1, 2, 3
1, 3, 2
2, 1, 3
2, 3, 1
3, 1, 2
3, 2, 1
```

如何编程打印一组数据的所有排列呢？这里就可以用递归来实现。

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。

如果我们把它写成递推公式，就是下面这个样子：

假设数组中存储的是1，2， 3...n。

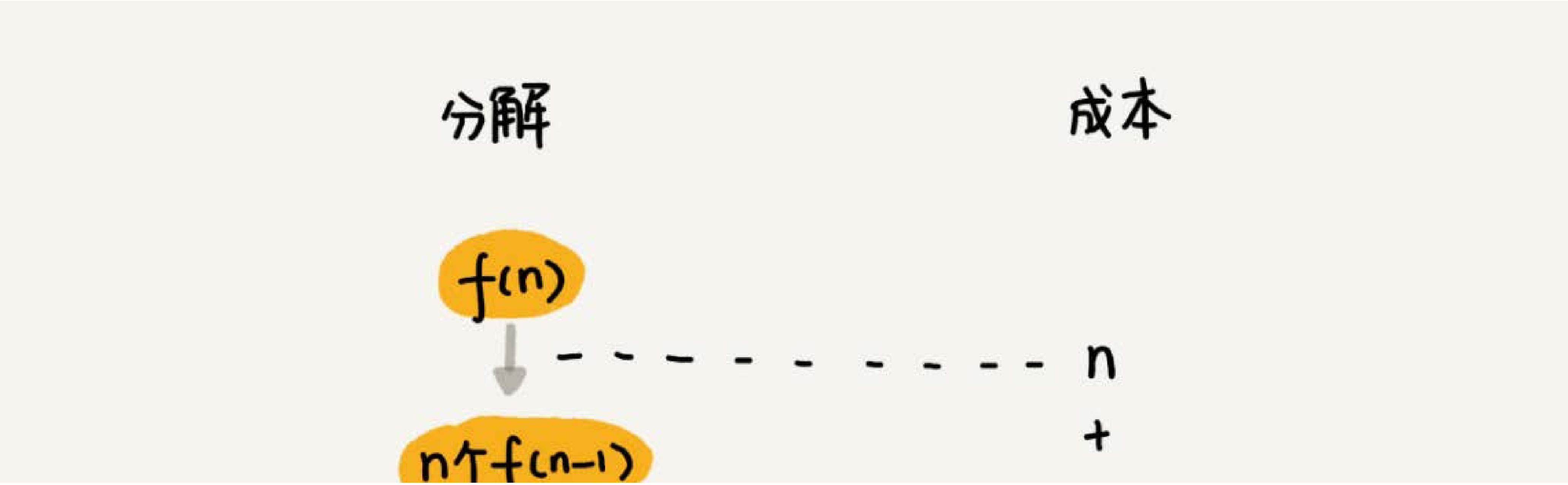
f(1,2,...n) = {最后一位是1, f(n-1)} + {最后一位是2, f(n-1)} +...+{最后一位是n, f(n-1)}。

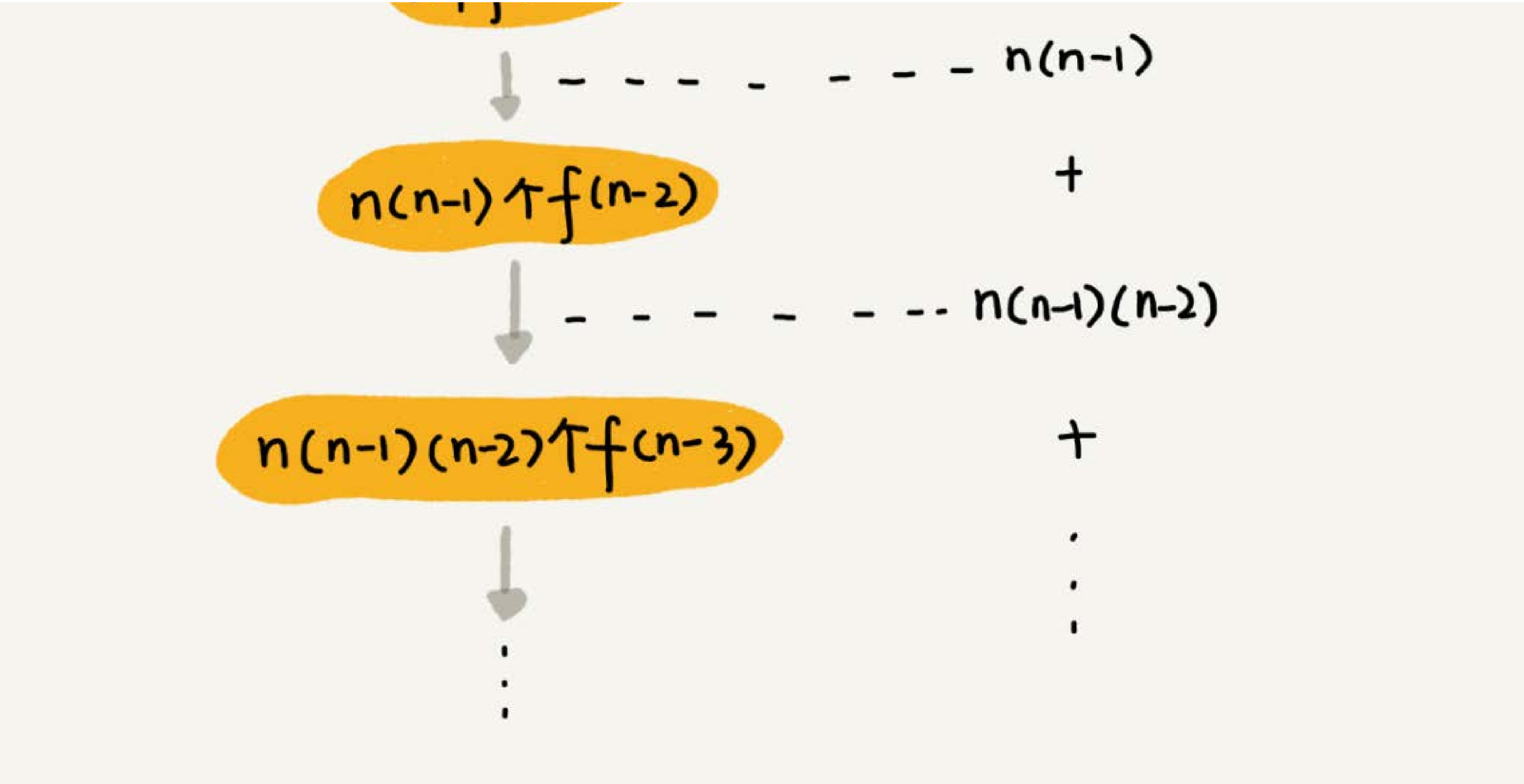
如果我们把递推公式改写成代码，就是下面这个样子：

```
// 调用方式：
// int[]a = a={1, 2, 3, 4}; printPermutations(a, 4, 4);
// k表示要处理的子数组的数据个数
public void printPermutations(int[] data, int n, int k) {
  if (k == 1) {
    for (int i = 0; i < n; ++i) {
      System.out.print(data[i] + " ");
    }
    System.out.println();
  }

  for (int i = 0; i < k; ++i) {
    int tmp = data[i];
    data[i] = data[k-1];
    data[k-1] = tmp;

    printPermutations(data, n, k - 1);

    tmp = data[i];
    data[i] = data[k-1];
    data[k-1] = tmp;
  }
}
```

如果不用我前面讲的递归树分析方法，这个递归代码的时间复杂度会比较难分析。现在，我们来看下，如何借助递归树，轻松分析出这个代码的时间复杂度。

首先，我们还是画出递归树。不过，现在的递归树已经不是标准的二叉树了。





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。

n + n*(n-1) + n*(n-1)*(n-2) +... + n*(n-1)*(n-2)*...*2*1

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。

这里我稍微说下，掌握分析的方法很重要，思路是重点，不要纠结于精确的时间复杂度到底是多少。

内容小结

今天，我们用递归树分析了递归代码的时间复杂度。加上我们在排序那一节讲到的递推公式的时间复杂度分析方法，我们现在已经学习了两种递归代码的时间复杂度分析方法了。

有些代码比较适合用递推公式来分析，比如归并排序的时间复杂度、快速排序的最好情况时间复杂度；有些比较适合采用递归树来分析，比如快速排序的平均时间复杂度。而有些可能两个都不怎么适合使用，比如二叉树的递归前中后序遍历。

时间复杂度分析的理论知识并不多，也不复杂，掌握起来也不难，但是，在我们平时的工作、学习中，面对的代码千差万别，能够灵活应用学到的复杂度分析方法，来分析现有的代码，并不是件简单的事情，所以，你平时要多实战、多分析，只有这样，面对任何代码的时间复杂度分析，你才能做到游刃有余、毫不畏惧。

课后思考

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？请你用已经学过的递归时间复杂度的分析方法，分析一下这个递归问题的时间复杂度。

欢迎留言和我分享，我会第一时间给你反馈。

精选留言:

- farFlight 2018-11-20 23:25:45
  假设细胞到了第三个小时是先分裂完再死亡，那么递推公式就应该是:
  f(n) = f(n-1)*2 - f(n-3)
  一次乘法和一次减法一起看作一次基本操作消耗，那么情况和斐波那契数列很像。

最高的树应该有n层， 最短的是n/3层，每层操作数都是指数增长。
那么时间复杂度应该是在O(2^n)量级的。 [72赞]

- Jerry银银 2018-11-27 14:51:58
  说个有意思的现象，我平时除了看专栏本身的内容，我也会看留言。我发现从专栏开始时，精品留言点赞数达到500多，随着专栏的前行，点赞的人越来越少了

  从中，也能发现端倪。

  这挺有意思的 [23赞]

- qinggeouye 2018-12-19 16:30:44
  有些同学不明白点赞第一的意思，在此试着解释一下。

  假设细胞先分裂再死亡，即，每个细胞分裂三次后死亡（存活三个小时）。

  n 从第 0 个小时开始，

  n = 0， f(0) = 1

  n = 1， f(1) = 2*f(1)

  n = 2， f(2) = 2*f(1)

  n = 3，f(3) = 2*f(2) - f(0) ，减去存活了三个小时的细胞个数。

  n = 4，f(4) = 2*f(3) - f(1)，减去存活了三个小时的细胞个数。

  以此类推：

  f(n) = 2*f(n-1) - f(n-3)，减去存活了三个小时的细胞个数。
  [9赞]

- 纯洁的憎恶 2018-11-21 00:15:14

思考题：

f0=1

f1=1+1=2

f2=1+1+2=4

f3=1+1+2+3-1=6 = f1 + f2

f4=1+1+2+3-1+5-1=10 = f2+f3

f5=1+1+2+4-1+5-1+8-2=16 = f3+f4

f（n） = f（n-1） + f（n-2）

与斐波那契数列的递归复杂度相同。 [8赞]

- 于欣磊 2018-11-23 08:15:47

  打卡，立flag的同学少了一个数量级都不止啊 [7赞]

- komo0104 2018-11-21 17:49:19

  如果到了第三小时先分裂再死亡应该是f(n) = 2*f(n-1) - f(n-4)

  ———————————-

  public static int func(int hour){

  if(hour == 0) return 1;

  if(hour == 1) return 2;

  if(hour == 2) return 4;

  if(hour == 3) return 7;

  return 2*func(hour -1) - func(hour - 4);

  }

  —————-

  带入hour=4

  结果： 2 * func(3)-func(0)= 13 [4赞]

- ppingfann 2018-11-21 02:32:57

  老师，有几个问题不明白：

  1. 求归并排序的时间复杂度中

满二叉树的高度计算公式中的n指的是树中的节点的总个数，而归并排序中的n指的却是叶子节点的个数。所以归并排序中树的高度，我计算出来的是h=log2^2n-1。
2. 实战二中
“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。“
叶子节点为1或者2都不能再往下分叉了，所以，我计算出来的最长路径是n-2。举个具体的例子：n=5时，最长路径为3。
我计算出来的最短路径依据n的不同还会不同，
具体的例子：n=5时，最短路径为2，n=6时，最短路径依然为2。

是我理解的有偏差吗？请老师指点。 [3赞]

- Laughing_Lz 2018-12-05 16:10:10
  假设细胞到了第三个小时是先分裂完再死亡，递推公式为f(n) = 2f(n-1)-f(n-3)
  假设细胞到了第三个小时是先死亡再其余的分裂，递推公式为f(n) = [f(n-1)-f(n-3)]*2 [2赞]

- 沉睡的木木夕 2018-11-21 02:16:09
  有几点问题不懂
  1.实战二：分析斐波那契数列的时间复杂度 一节提到
  “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。”数据规模都是-1，-2 这怎么理解？每次都是-1，最长路径大约就是n 这又是怎么理解的？
  2.实战3中提到
  “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)。”
  交换操作的次数是怎么的出来的？
  这对于我来讲就好比，数学老师讲了一堆看似简单的东西（有同学基础不好），最后老师最后落笔：所以1+1=2，但我还是一脸懵逼 [2赞]

- 小林子 2018-11-21 02:08:47
  f(0) = 1
  f(1) = 2
  f(2) = 4
  f(3) = 8
  f(n) = f(n-1) + f(n-2) + f(n-3) + f(n-4) [2赞]