对递归的一个新的理解，应用一个已知情况，经过过程相同的推演不断推断出其他的情况。

示例：

当我们查词典的时候，不理解对该词的意思，于是我们不得不去通过对该词的解释的词进一步进行查询，直到找到一个我们理解的单词

计算机的底层硬件实现影响计算机程序的快慢，例如 一个判断语句总是比一个赋值语句花费的时间更长（判断语句类似于调用一个 判断符号函数，从而会衍生出 函数栈的申请，参数的传递，通常都是值传递，而一次值传递就相当于一次赋值，而通常都是多个值传递，计算结果，最后返回值貌似还要传递参数），

为什么只需要 用1到根号下n的数去检验n就可以验证它是否为质数了呢？

1. 假设有一个大于根号下n的数可以整除n，那么除完的数m一定小于根号下n，从而得出不需要去验证大于根号下n的数
2. 为什么选 根号下n这个数，第一个已经证明了，根号下n之后的数不需要验证了，那有比根号下n更小的数来缩小检查范围么。

答案是没有。傻啊，如果这个数是一个数的平方，例如36，49，那么根号下n不就是最小的可判断解么。

欧几里德算法：已知两个正整数是某个数的整数倍，找出这个数，并使这个数尽量的大。

为什么不断的取余数可以的到最终答案：我们可以假设，还没有得到最终答案，这两个整数必然还是这个未知数的整数倍，并且有大有小，那么我们总可以继续利用取余数使这个整数倍减小，而无穷论貌似 指出整数不断的减小最终会有一个先变成零。

如果其中一个还不能整除另一个数，那么只有一种情况，那就是这个较大的数是这个较小的数的一个某些倍再加上

哦！天哪！数学的语言是如此的巧妙，假设a=m1c1,b=m2c1,而a%b的余数c一定是比b小的，继续递推，b%c的余数d一定是比c小的，从而得出每一次的余数都比前一次小，而我们的算法只需要当余数为零即两个数可以整除使即可，而即使每次都比前一次小一个数，c也只和0差了c次运算，从而证明了只要不断的取余数就可以得到最终答案

对一个问题进行处理，应该对这个问题有着完全，清晰，准确的理解

考虑这样一个问题，求一个无序数组中大小最接近的两个元素的差

1. 自然思维，既然是求大小最接近，那么我们需要比较所有的成对的元素之差
2. 先排序，因为大小最接近的两个元素必然相邻（利用排序我们给这个无序的元素数组创造了一种性质，猜想：给元素即要处理的数据组织一定的形式，可以简化问题）

先排序后比较元素真的对第一个方法进行优化了么，如果优化了，第二种方法对那些步骤进行了简化呢

分析：貌似利用比较，第一二种方法我们都使用比较（排序基数排序这种不需要比较的算法），貌似我们要知道最接近的两个元素，一定要知道任意一对的元素之差，而第二种方法则把效率变成（注意我们还不知道是快还是慢）nlog2n+n，而第一种算法则是n的平法阶的，貌似在处理大量随机的数组中，后一种方法的增长速度明显的比第一种慢，到底慢在哪里呢

慢在这里：很有意思，我们拿快速排序来举例子，在第一种方法中，我们总是在问这样的问题，你这一对的差是多少，记录下来，然后和其他成对的元素进行比较，而在第二种方法中，我们之所以只需要加一个n，即只需要比较相邻的两个元素的差的值，是因为我们在排序时总是问这样的问题，（拿快速排序举例），在一趟和中值的比较中，我们问，你是否和中值后面的元素的差总是过大？从而把整个数组分成两个阵营，他们分别只能同和自己阵营里的元素才有可能产生最接近的元素之差，而排除了和另一个阵营产生最接近的元素之差的可能，从而不断的减少需要比较的次数，最终只需要比较n次即可了。

准确的数学推导是应该进行的（而不仅仅是看到他们所被教课书定义的数量阶就迷惑了，即使它是近似准确的），以及对两个方法之中计算机所进行操作的快慢也是需要比较的（第一种方法我们需要的计算机操作只有作差，比较，赋值，并且他们的分布比较均匀，而第二种方法看起来貌似数量阶小得多，但它的作差，比较，赋值不是那么均匀），我们应该对它们所做的操作进行细分（有可能第一种的作差较多，而第二种的比较较多，这两种运算的速度对于计算机当然是不同的，我们需要仔细考虑），才能真正看出两种算法的优劣情况。

怎么样才能想到这种优化呢：it's difficult but meaningful，还有前面的农夫和狼，白菜以及4个人过桥的问题

二分思想：一个问题看待角度的不同，一个本来用循环遍历的方式解决的问题怎么样用其他方式看待

示例：求一个数的幂，一个数的幂可以是多个数不断的相乘，也可以是两个数相乘，而这两个数又可以写成四个数相乘。

1. 欧几里德算法，怎么求最大公约数，
2. 以及二分查找，问：你是不是这个数和问你比这个数小，还是比这个数大所产生的效果。这个问题很有意思，感觉和上边的问题性质不同，它本质的区别是这样问：你是这个数，还是你后边的数中有这个数，另一个问法：你是这个数，还是你前边有这个数，还是你后边有这个数，多了一问（可能多了两个操作，因为前边的问法是潜在的问你后边的数），但却排除了一半的可能性。

一个复杂问题的分析，往往甚至可以说绝对不是一步到位的，我们学习的基础算法往往在复杂问题进行分解之后就可以应用的上，而这个分解通常是十分困难的

情况策略

好的方法为什么难想？因为它常常不是直接向目标方向直接进行，而是对局部做出调整后在去趋向目标方向。

解决问题的方法有时会在大脑形成模糊的形象，为了去具象化这个方法，应该反复思考深化印象