北京大学
PEKING UNIVERSITY

计算概论B

上机课（6）

2021-2022年秋

5:二分法求函数近似零点

总时间限制: 2000ms　　内存限制: 65535kB

描述

二分法——循环缩小需查找的区间

有函数形如

$$f(x) = a_1*x^5 + a_2*x^4 + a_3*x^3 + a_4*x^2 + a_5*x + b.$$

现给定上述的六个参数并给定一个区间，保证在该区间上该函数单调并有且只有一个零点。请使用二分法求该零点的近似值，精确到小数点后六位。

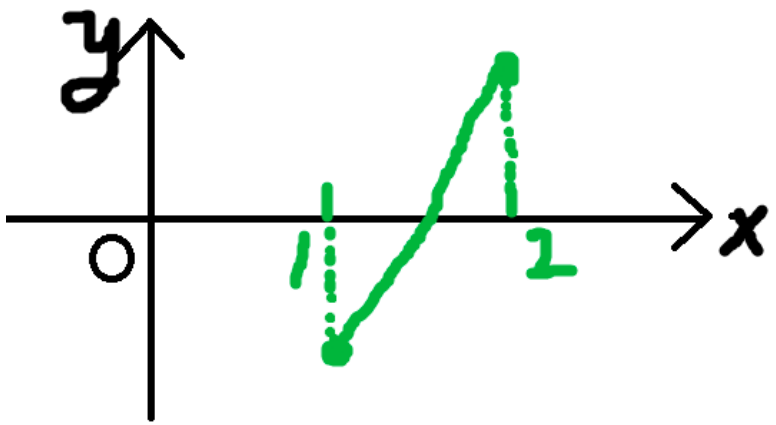
二分法

上界　　下界

```
while inter[1] - inter[0] > 0.0000001:
    mid = (inter[1] + inter[0]) / 2
    num = 0
    for i in range(6):
        num += params[i] * math.pow(mid, 5 - i)
    # print(mid, num)
    if num * delta > 0:
        inter[1] = mid
    elif num * delta < 0:
        inter[0] = mid
    else:
        break
```

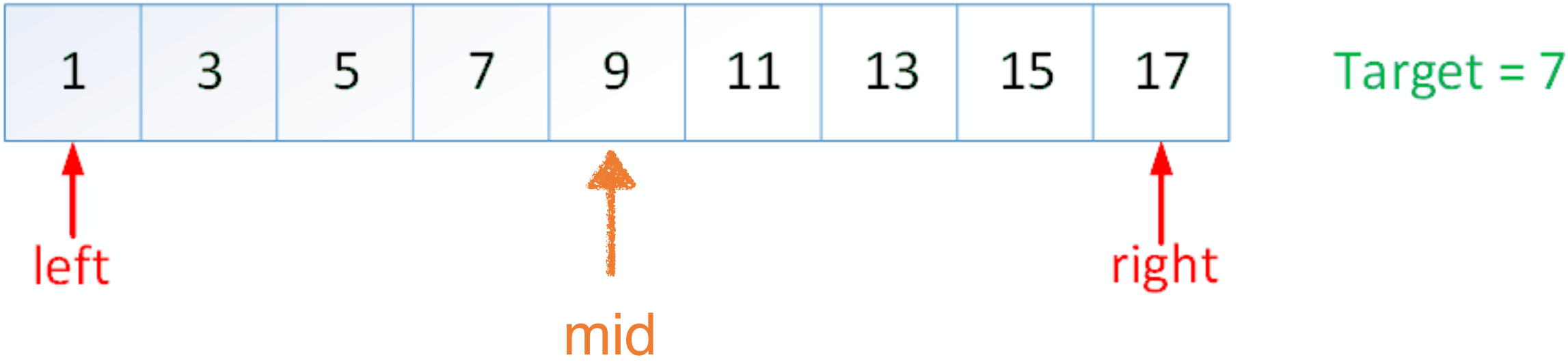
循环条件

函数单调上升或下降，如果判断根在左边，则上界向二分点mid移动缩小范围，否则下界向mid移动缩小范围，（delta代表上升或者下降，上升为正数，下降为负数）



二分查找

在一个单调上升的有序数组里找到是否存在Target的值



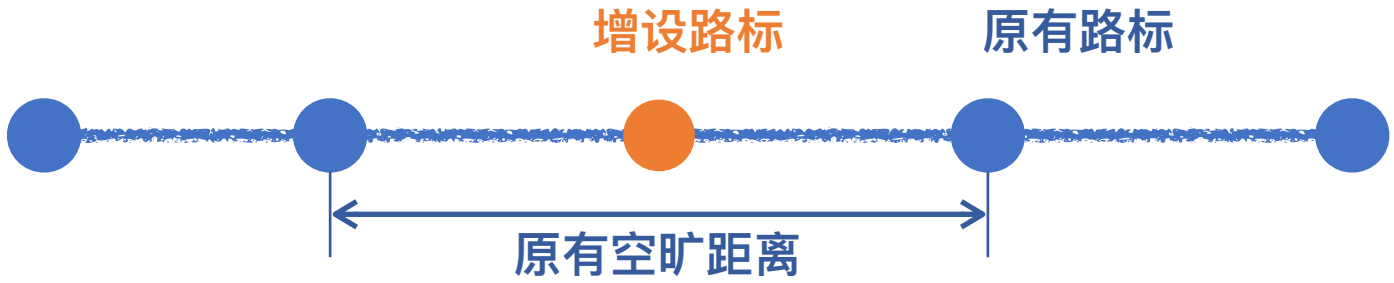
二分查找

```
def BinarySearch(nums,target):
    left,right = 0,len(nums)-1
    while left <= right:
        mid = left+(right-left)//2 #计算数组中间位置
        if nums[mid] == target:
            return mid #找到并返回
        # 中间元素大于目标值，搜索数组左半部分
        elif nums[mid] > target:
            right = mid - 1
        # 中间元素小于目标值，搜索数组右半部分
        elif nums[mid] < target:
            left = mid + 1
    return -1 #没找到目标值
```

Tips：二分查找有很多种写法，由于整数整除的特性，尤其是最后边界的情况需要考虑，建议模拟推演几组数据

二分答案

- 二分答案，指一种利用二分搜索对一类答案单调求最值的问题的算法。例如：求使得最大值最小的解法



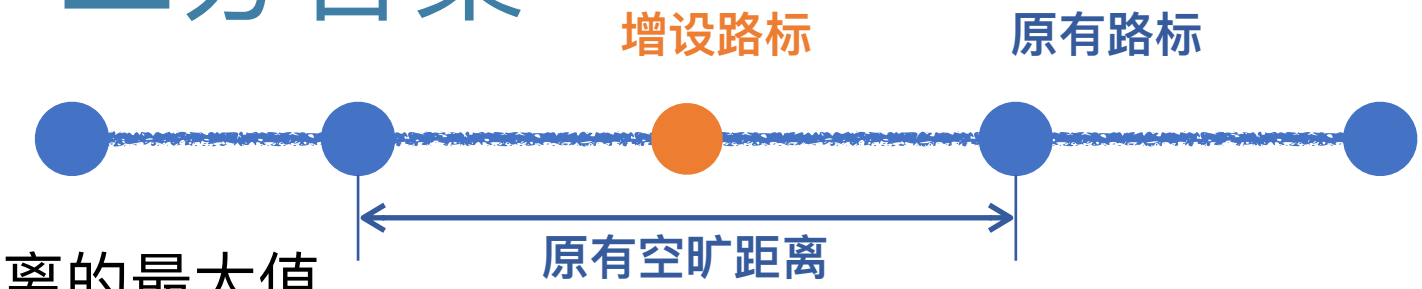
题目背景

B 市和 T 市之间有一条长长的高速公路，这条公路的某些地方设有路标，但是大家都感觉路标设得太少了，相邻两个路标之间往往隔着相当长的一段距离。为了便于研究这个问题，我们把公路上相邻路标的最大距离定义为该公路「空旷指数」。

题目描述

现在政府决定在公路上增设一些路标，使得公路的「空旷指数」最小。他们请求你设计一个程序计算能达到的最小值是多少。

二分答案



分析：

空旷距离定义为求相邻路标距离的最大值。

因为**最多可以增设k个路标**，可以知道“空旷距离”越大，所需要放的路标数越少；“空旷距离”越小，所需要放的路标数越多（增设路标不会使得空旷距离变大）。

因为我们直接去求解空旷距离的最小值是难以设计算法的，所以可以利用“空旷距离”答案的单调性，尝试用二分答案解决，即，给定一个空旷距离的要求，如果设置不超过k个路标可以满足该要求，则“空旷距离”还可以更小；否则如果无法找到，则“空旷距离”还需要变大。

二分答案法经常用于：求最大值的最小解or最小值的最大解。

枚举

- **水仙花数**（Narcissistic number）也被称为超完全数字不变数（pluperfect digital invariant, PPDI）
- 水仙花数是指一个 3 位数，它的每个位上的数字的 3次幂之和等于它本身。例如：1^3 + 5^3+ 3^3 = 153。枚举代码示例：

```
for i in range(100,1000):
    a = i//100
    b = (i-a*100)//10
    c = (i-a*100-b*10)

    if i == pow(a,3)+pow(b,3)+pow(c,3):
        print(i)
```

1:输出所有素因子

总时间限制: 1000ms　内存限制: 65536kB

描述

请定义一个函数 prime_factors(n)，它设法找出正整数参数 n 的所有素因子，并调用 print 逐个增序输出，重复的因子重复输出。

样例输入

```
12103
```

样例输出

```
7
7
13
19
```

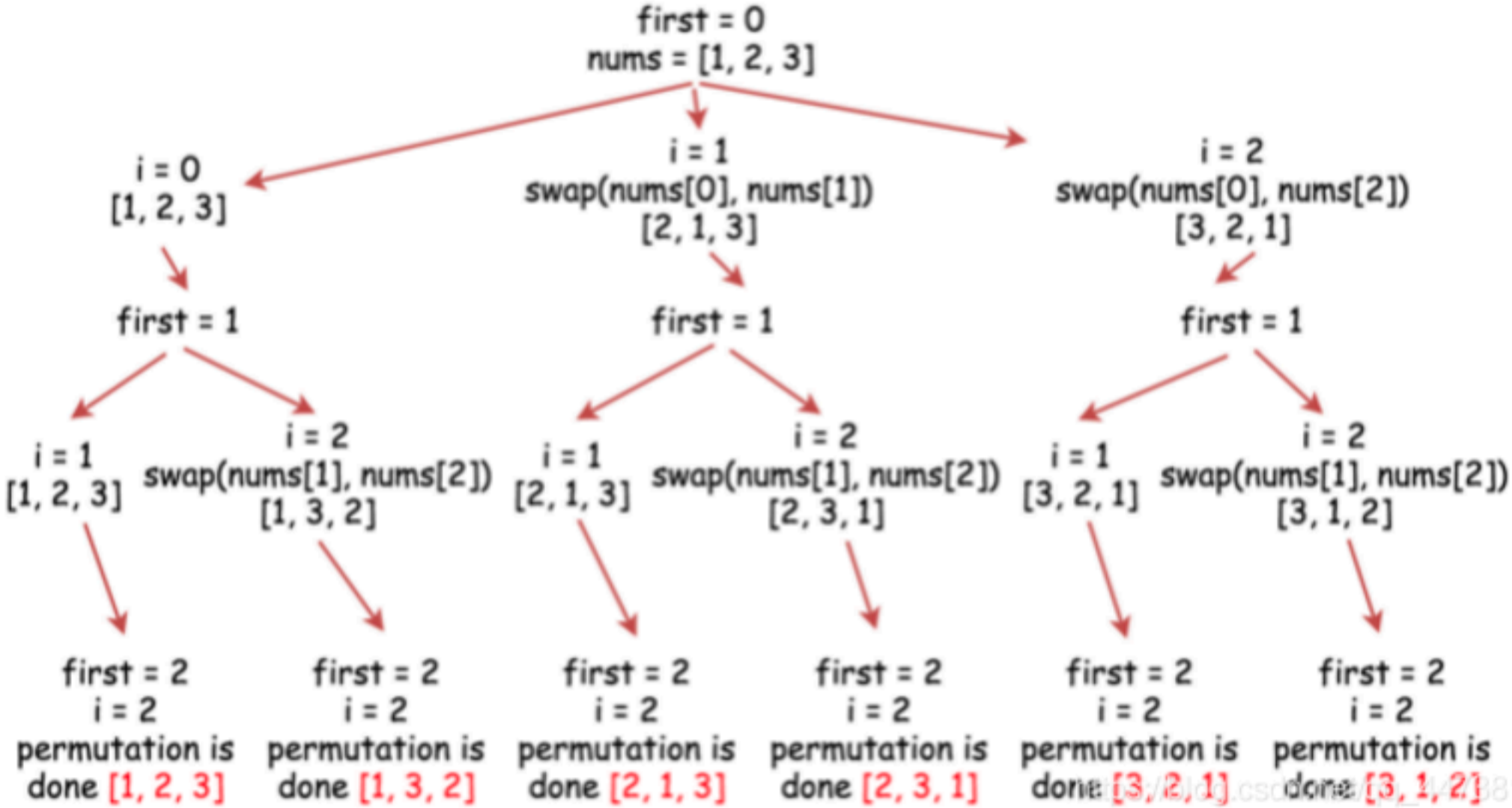
```
def prime_factors(num):
    fac = []
    for i in range(2, int(math.sqrt(num) + 1)):
        if num % i == 0:
            fac.append(i)
            fac.extend(prime_factors(num / i))
            break
    else:
        fac.append(int(num))
    return fac
```

全排列的python简便方法

```
import itertools

array = [1, 2, 3, 4]
p = list(itertools.permutations(array))
for item in p:
    for y in item:
        print(y, end=' ') # 将一个排列里的每个元素打印出来，用空格进行分割
    print() # 输出一个回车符
```

以全排列为例

回溯：模拟全排列的交换，每次递归返回后要把数组恢复为原来的样子



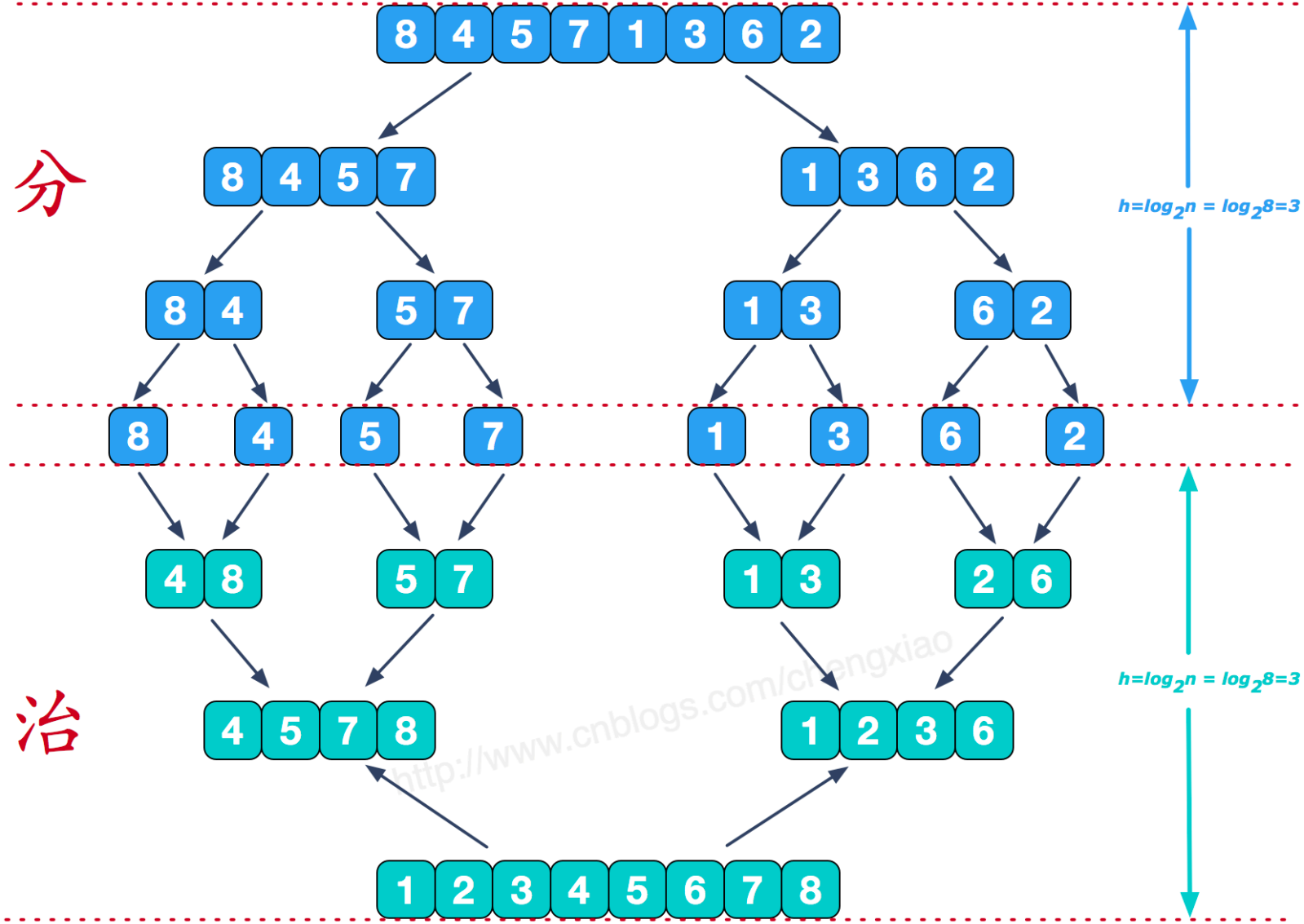
递归+枚举，以全排列为例

```
visit = [True, True, True, True]
permutation_arr = ["" for x in range(0, 4)]
arr = [1, 2, 3, 4]

def permutations(position, end):
    if position == end:
        print(permutation_arr)
        return
    for index in range(0, end):
        if visit[index] == True:
            permutation_arr[position] = arr[index]
            visit[index] = False
            permutations(position+1, end)
            visit[index] = True

permutations(0, len(arr))
```

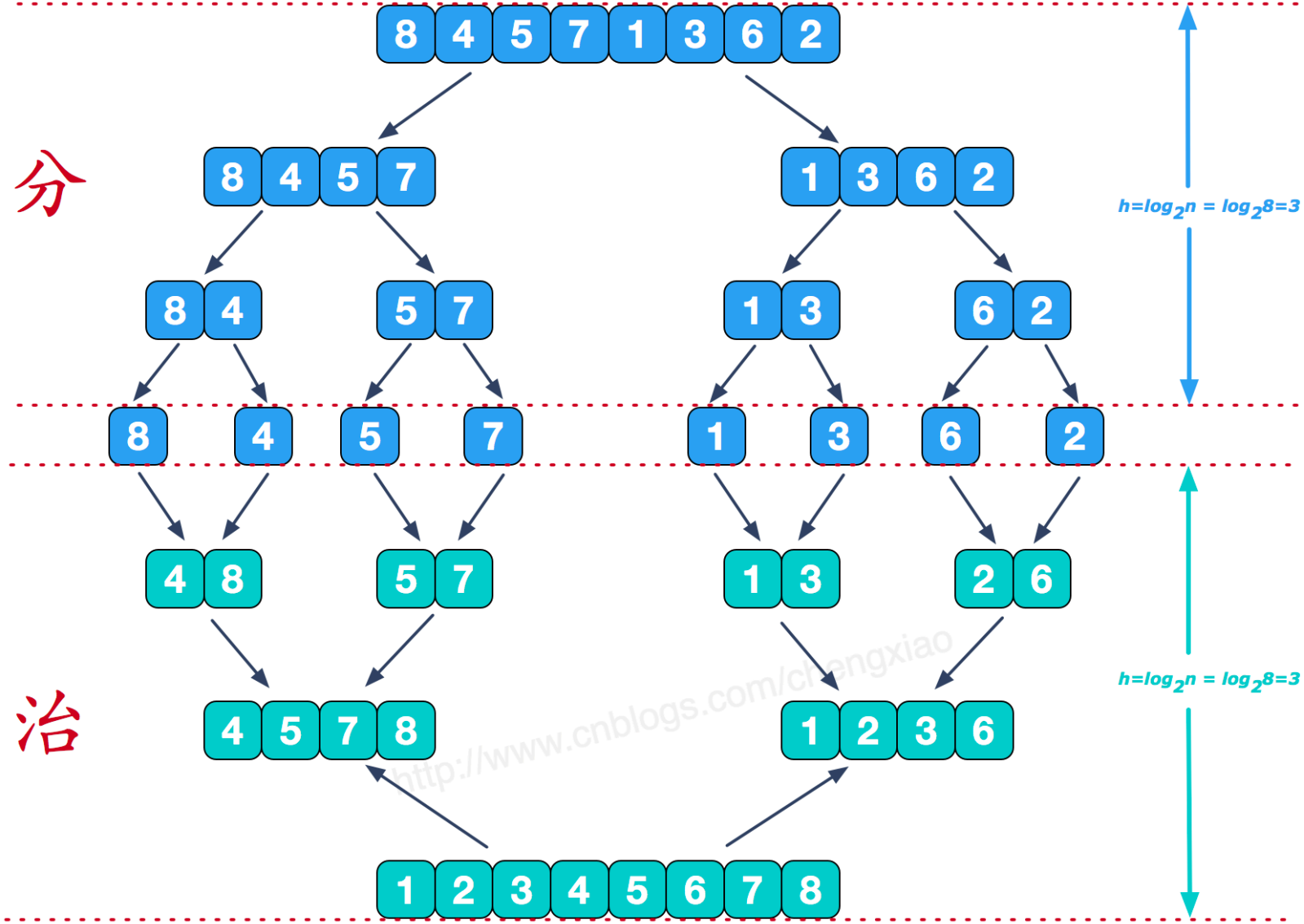
递归+回溯，以全排列为例

```python
def permutations(arr, position, end):
    if position == end:
        print(arr)
    else:
        for index in range(position, end):
            arr[index], arr[position] = arr[position], arr[index]
            permutations(arr, position + 1, end)
            arr[index], arr[position] = arr[position], arr[index] # 还原到交换前的状态，为了进行下一次交换

arr = [1, 2, 3, 4]
permutations(arr, 0, len(arr))
```

递归+二分——归并排序


分
治
8 4 5 7 1 3 6 2
8 4 5 7
1 3 6 2
8 4
5 7
1 3
6 2
8
4
5
7
1
3
6
2
4 8
5 7
1 3
2 6
4 5 7 8
1 2 3 6
1 2 3 4 5 6 7 8
h=log2n = log28=3
h=log2n = log28=3

归并排序求逆序对


分
治
8 4 5 7 1 3 6 2
8 4 5 7
1 3 6 2
8 4
5 7
1 3
6 2
8
4
5
7
1
3
6
2
4 8
5 7
1 3
2 6
4 5 7 8
1 2 3 6
1 2 3 4 5 6 7 8
h=log2n = log28=3
h=log2n = log28=3

北京大学
PEKING UNIVERSITY

谢谢大家！