亲子算法课（8）
——快速排序算法

作者：叶蒙蒙

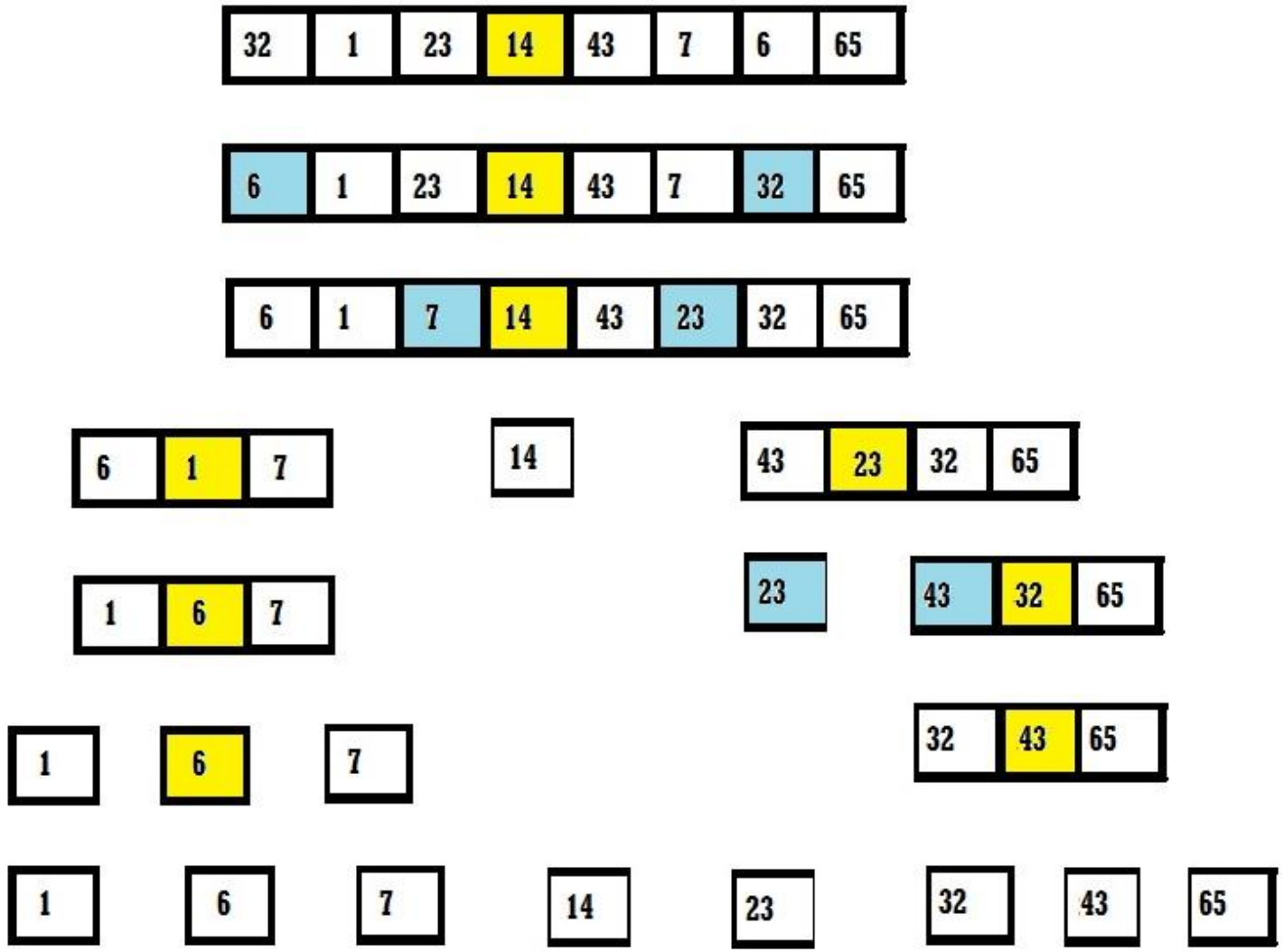
快速排序

快速排序

- 非常常用！！！——程序员面试经常考
- 时间复杂度
  - 最坏时间复杂度（倒序）　　O(n^2)
  - 最优时间复杂度（正序）　　O(n log(n))
  - 平均时间复杂度　　O(n log(n))
- 快速
  - 最坏情况很少见
  - 因为它的内部循环可以在大部分的架构上很有效率地优化，所以通常明显比其他算法更快
- 空间复杂度——根据实现的方式不同而不同

快速排序

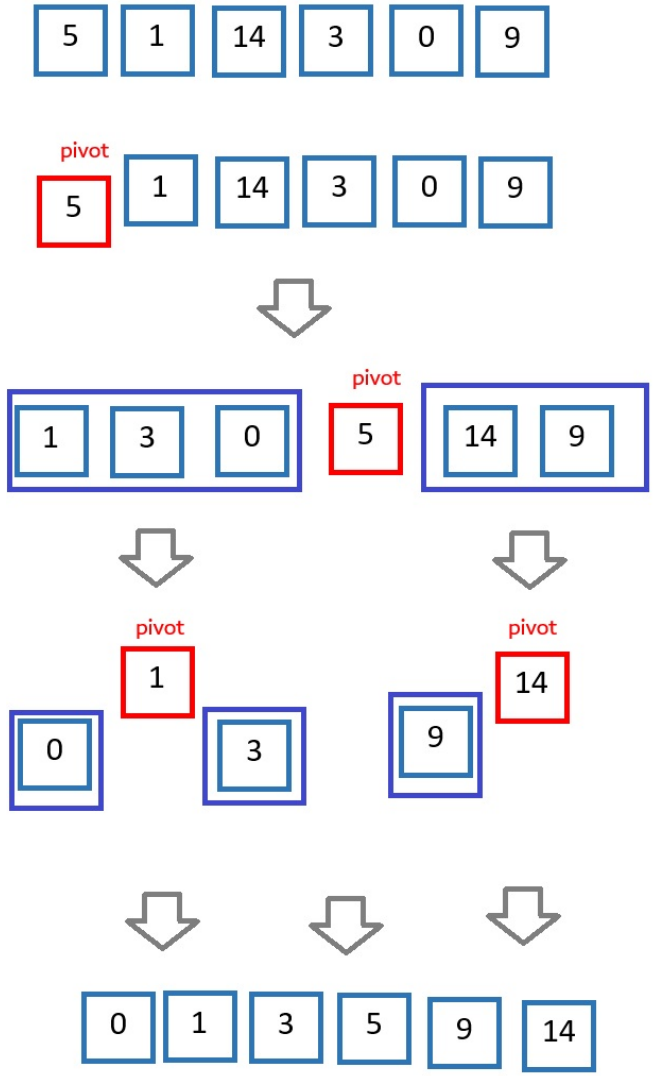
- 采用了分治策略 （Divide and Conquer）
- 分治：把一个大问题，分成几个小问题
- 快排：把一个大序列，分成两个小序列



快速排序原理

- 【第一步】一分为二
  - 在一个待排序序列中选中一个元素作为轴（Pivot）
  - 把比轴元素小的元素都放到轴元素前面
  - 把比轴元素大的元素都放到轴元素后面

  经过这一步后，一个大序列被分为了前后两个子序列，前子序列中的元素都比轴元素小；后子序列中的元素都比轴元素大
- 【第二步】分而治之：对前后子序列分别再次一分为二
- 不断重复上面两步，直到所有分裂出来的子序列都只包含一个元素为止



快速排序的非递归实现

快速排序的“分”和“治”

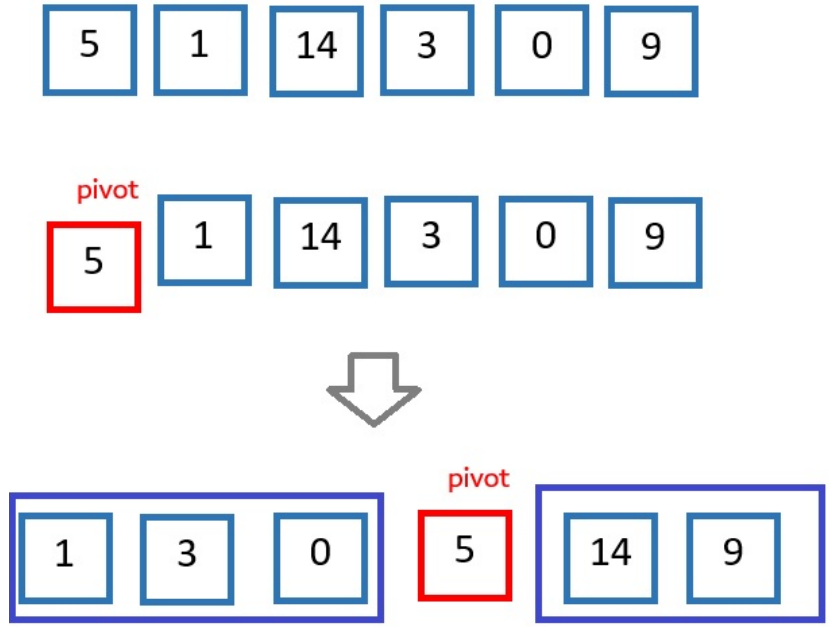
- 分——分区：在当前序列中选择一个元素作为轴，再将所有元素按照与轴的大小比较，分成两个子序列（区）

- 治——处理分出来的子序列（区）

分区函数

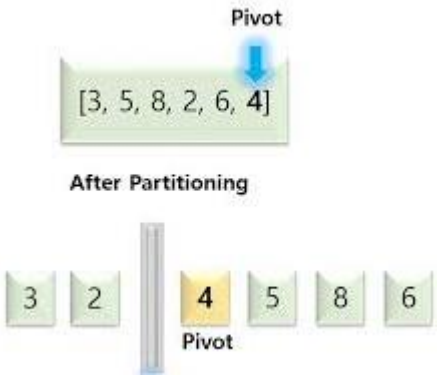
- 在待排序列(arr)中选中一个元素作为轴(Pivot)
  一般习惯性地选第一个元素：arr[0]作为轴
  pivot = 0

- 把比轴元素小的元素都放到轴元素前面
- 把比轴元素大的元素都放到轴元素后面



最简单的分区函数

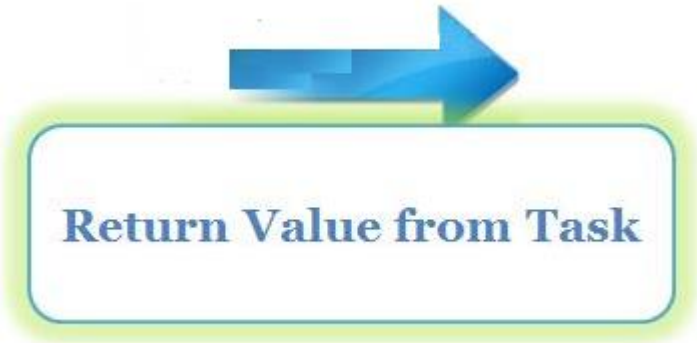
- 直观简单
- 不计存储空间的得失




```
def partition(arr):
    pivot = 0

    left = []
    right = []

    for i in range(1, len(arr)):
        if arr[i] <= arr[pivot]:
            left.append(arr[i])
        else:
            right.append(arr[i])

    left.append(arr[pivot])
    left.extend(right)

    for i in range(0, len(left)):
        arr[i] = left[i]

    return
```

编程Tips

- partition函数，如果变成右侧形式
- 在调用时有何不同？




```
def partition(arr):
    pivot = 0

    left = []
    right = []

    for i in range(1, len(arr)):
        if arr[i] <= arr[pivot]:
            left.append(arr[i])
        else:
            right.append(arr[i])

    left.append(arr[pivot])
    left.extend(right)

    return left
```

编程Tips

- partition函数，如果变成这样，无论最后一行选用（1）还是（2），都会出错。
- 错误是什么？
- 这一现象说明什么？

```
def partition(arr):
    pivot = 0

    left = []
    right = []

    for i in range(1, len(arr)):
        if arr[i] <= arr[pivot]:
            left.append(arr[i])
        else:
            right.append(arr[i])

    left.append(arr[pivot])
    left.extend(right)

    arr = left

(1) return

(2) return arr
```

第一个完整的快速排序程序

为了反复调用，修改分区函数

为了在第一次分区后继续使用partition函数，我们先修改一下partition函数：

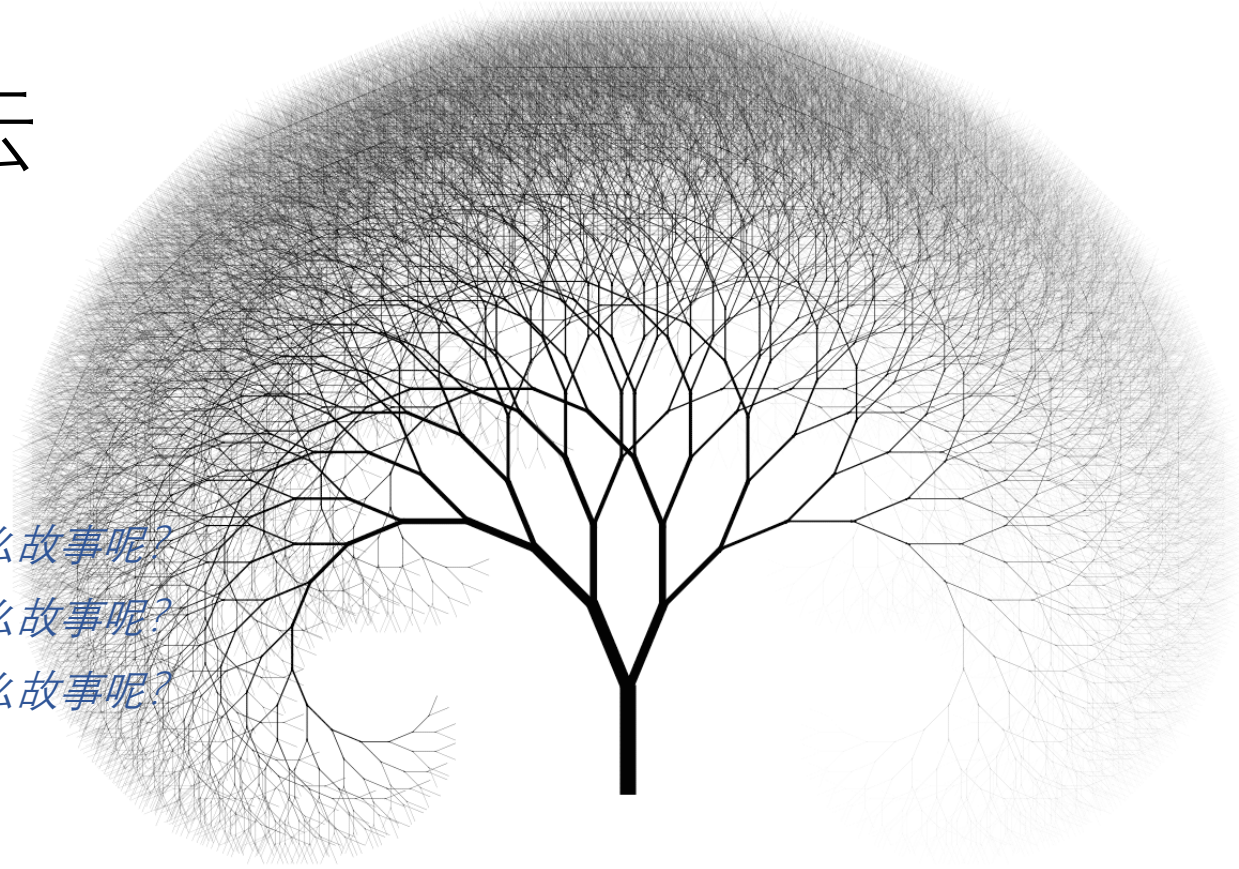
```
def partition(arr, low, high):
    pivot = low

    left = []
    right = []

    leftSize = 0

    for i in range(low + 1, high + 1):
        if arr[i] <= arr[pivot]:
            left.append(arr[i])
            leftSize = leftSize + 1
        else:
            right.append(arr[i])

    left.append(arr[pivot])
    left.extend(right)

    pivot = pivot + leftSize

    for i in range(low, high + 1):
        arr[i] = left[i-low]

    return pivot
```

对分区后的序列继续分区，直到长度为1

```
def quicksort(arr):

    todoList = []
    todoList.append([0, len(arr) - 1])

    index = 0

    while(index < len(todoList)):
        low = todoList[index][0]
        high = todoList[index][1]

        pivot = partition(arr, low, high)

        if(low <= pivot - 1):
            todoList.append([low, pivot -1])

        if(pivot + 1 <= high):
            todoList.append([pivot + 1, high])

        index = index + 1

    return
```

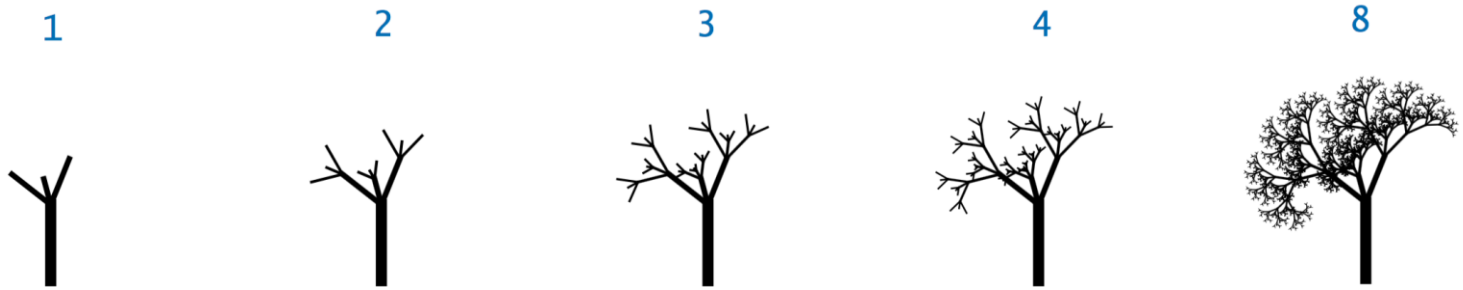
快速排序的递归实现

编程小主题：递归算法



- 自己调用自己的算法
- 递归函数：自己调用自己的函数

从前有座山，山里有座庙，庙里有个老和尚讲故事。讲得什么故事呢？
从前有座山，山里有座庙，庙里有个老和尚讲故事。讲得什么故事呢？
从前有座山，山里有座庙，庙里有个老和尚讲故事。讲得什么故事呢？
……

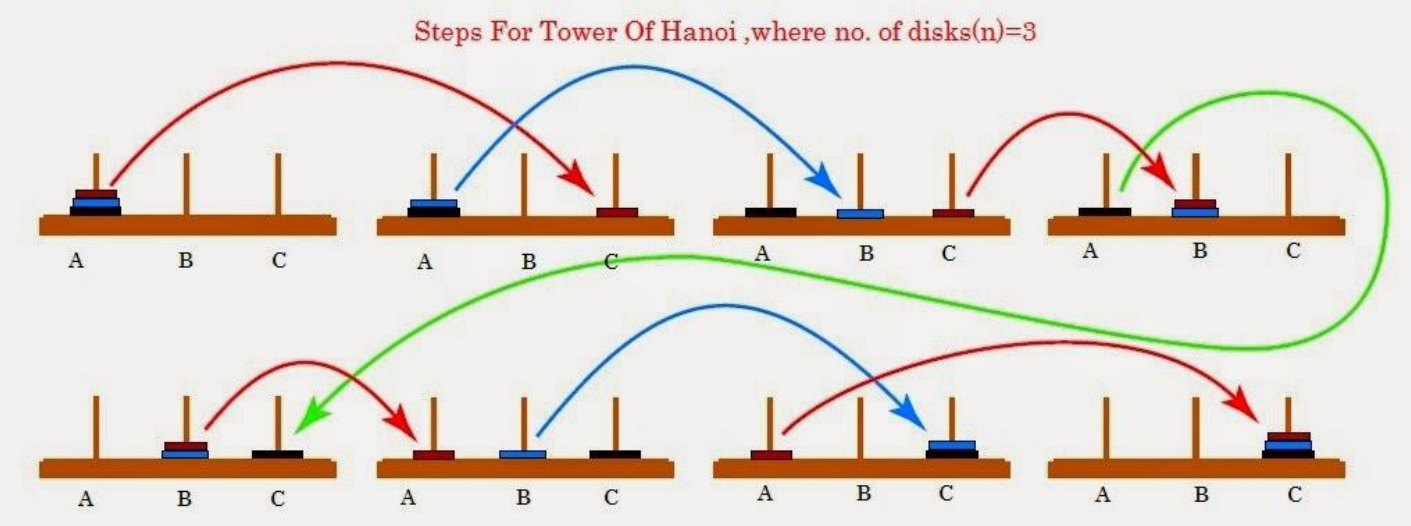


数学归纳法

- 证明当n等于任意一个自然数时某命题成立
- 证明分下面两步:

    1. 证明当n = 1时命题成立;
    2. 证明如果在n = k 时命题成立，那么可以推导出在n = k+1时命题也成立。（k代表任意自然数）

- 多米诺骨牌
    - 第一块会倒
    - 第n 块倒后第 n+1 块一定也会倒 (n>=1)




```
def TowerOfHanoi(n , from_rod, to_rod, aux_rod):
    if n == 1:
        print "Move disk 1 from rod",from_rod,"to rod",to_rod
        return
    TowerOfHanoi(n-1, from_rod, aux_rod, to_rod)
    print "Move disk",n,"from rod",from_rod,"to rod",to_rod
    TowerOfHanoi(n-1, aux_rod, to_rod, from_rod)
```

递归经典问题：汉诺塔问题

神奇的斐波那契额数列

FIBONACCI SEQUENCE

A series of numbers, starting from 0 where every number is the sum of the two numbers preceding it.

0,1,1,2,3,5,8,13,21,34,55.... and so on

Named after

FIBONACCI

An Italian mathematician

Year 1202

The year it was first introduced to the western world in the book "Liber Abaci"

$$x_n = x_{n-1} + x_{n-2}$$

Mathematical formula

1.618

"Phi" or the "Golden Ratio" The ratio of any two consequent numbers of the sequence

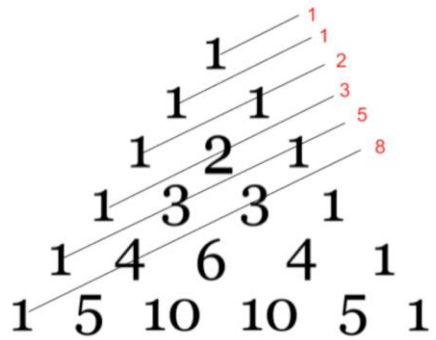
Nature's code

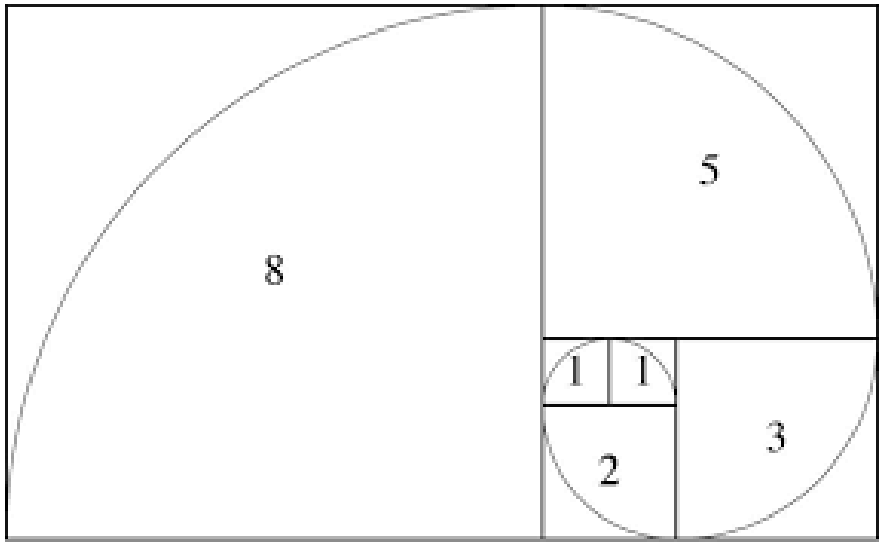
Because it is observed in several natural phenomena

递归算法实现斐波那契数列

- 斐波那契（Fibonacci）数列。在数学上，它是以递归的方法来定义的：
  - F0 = 0
  - F1 = 1
  - Fn = Fn-1 + Fn-2 (n >= 2)

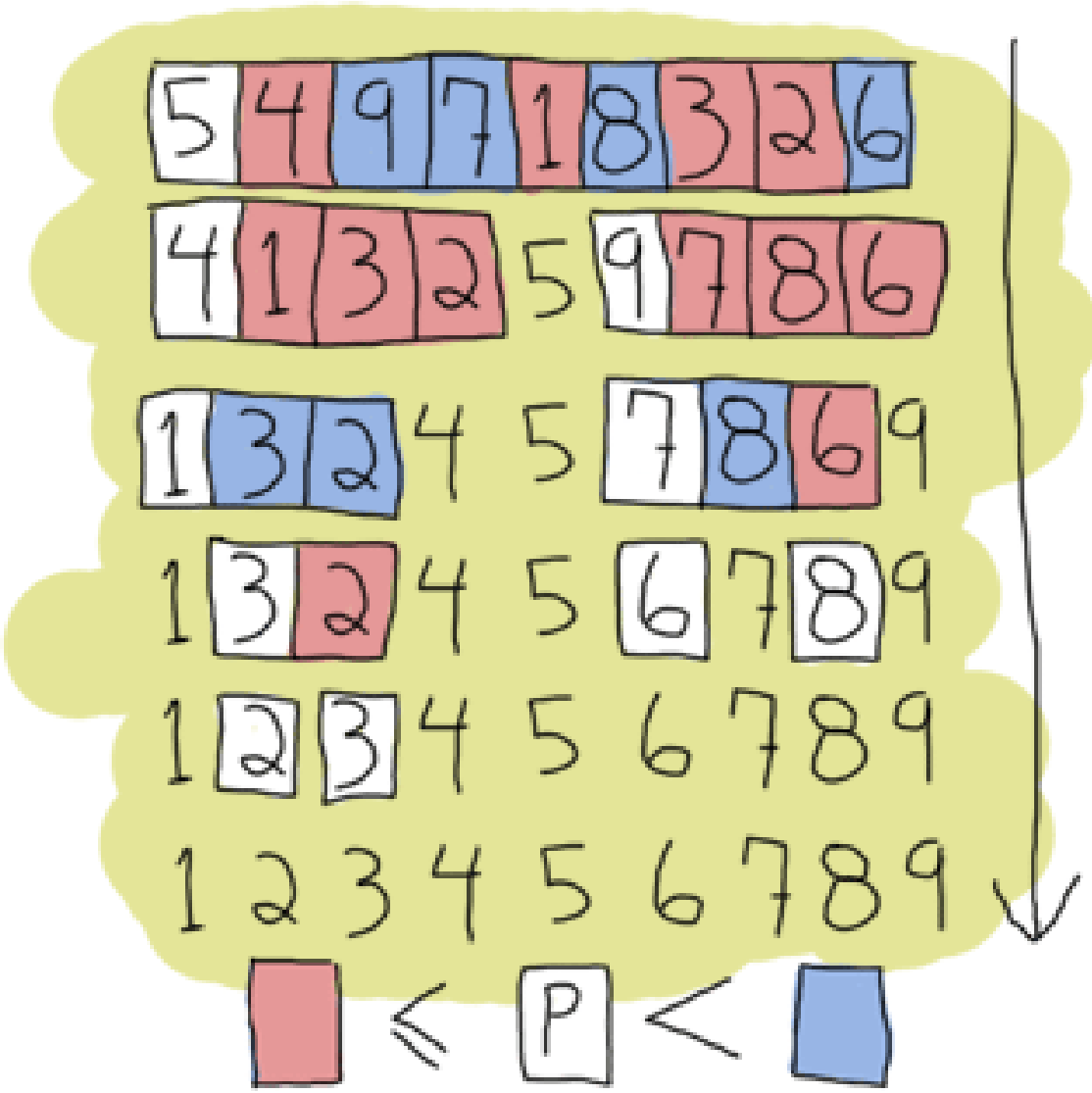
```
def F(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return F(n-1)+F(n-2)
```





再开快速排序的“分”和“治”

- 分——分区：以轴元素为界，将所有元素分为前、后两个子序列
- 治——对子序列再进行快速排序



递归实现的快速排序

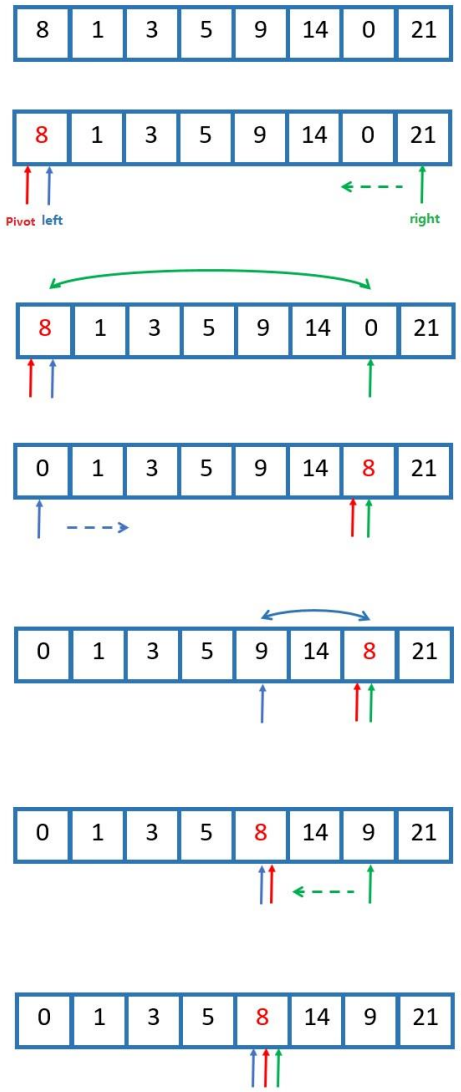
```
def quicksort_recursive(arr, low, high):

    if (low > high):
        return

    pivot = partition(arr, low, high)

    quicksort_recursive(arr, low, pivot - 1)
    quicksort_recursive(arr, pivot + 1, high)

    return
```

优化分区函数

只用一个元素的缓存空间的分区函数

- 选中第一个作为pivot
- left, right分别从两个方向向中间走
- 先从右往左（因为pivot在头部）
- 如果 arr[right] < arr[pivot]，则交换两个元素，之后反向走
- 如果 arr[left] > arr[pivot]，则交换两元素，然后反向走
- 当left和right相遇时为止

NOTE：每次轴元素和其他元素交换，pivot自己也会变



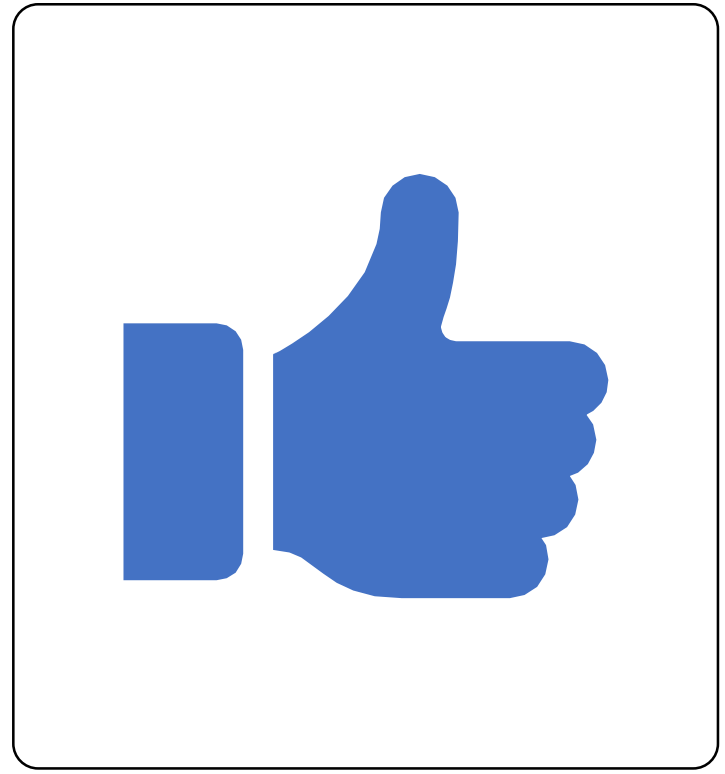
亦步亦趋实现分区函数

```
def partition(arr, low, high):
    pivot = low
    left = low
    right = high

    while(left < right):

        while (arr[right] >= arr[pivot] and left < right ):
            right = right - 1

        if(arr[right] < arr[pivot] ):
            switch(arr, pivot, right)
            pivot = right

        while (arr[left] <= arr[pivot] and left <right):
            left = left + 1

        if (arr[left] > arr[pivot] ):
            switch(arr, left, pivot)
            pivot = left

    return pivot
```

分区函数的进一步优化

```
def partition(arr, low, high):
    pivot = low
    fewer_range = low
    for i in range(low+1, high+1):
        if arr[i] <= arr[low]:
            fewer_range = fewer_range + 1
            switch(arr, i, fewer_range)

    switch(arr, pivot, fewer_range)
    pivot = fewer_range
    return pivot
```

- 你能读懂这段代码?
- 它是怎么运行的?
- 你能用我们的例子模拟一下这段代码的运行吗?

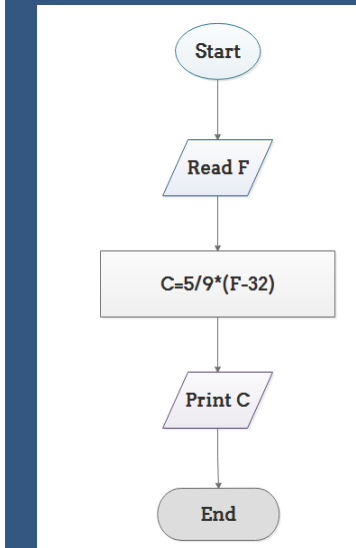


如何阅读一个算法（代码）？

- 直接读读不懂，该怎么办？

1. 阅读接口（输入/输出——参数、返回值）
2. 选取一份很简单的数据作为参数输入到函数中（函数调用）
3. 一步步推演函数体内部每一个变量的变化，直到得到返回值

- 辅助方法：画算法流程图



谢谢