**1. 快速判断算法时间复杂度**

**简单情况**

1. 确定问题规模n

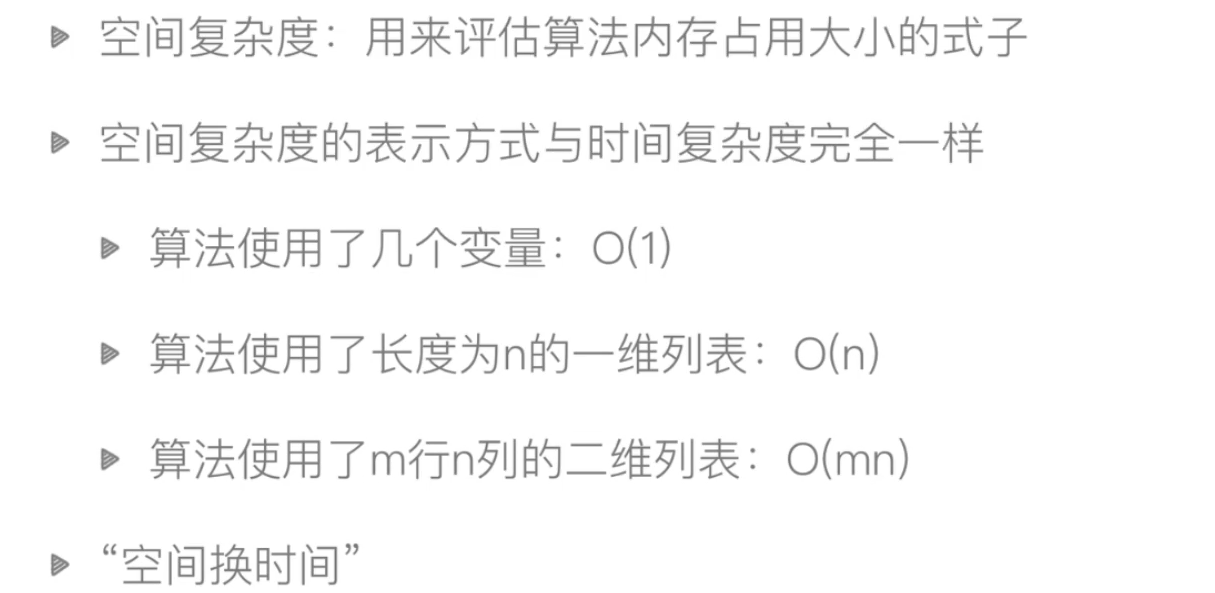
2. 循环减半过程——>logn

3. k曾关于n的循环——>n的k次方

**复杂情况：**

根据算法执行过程

**2. 空间复杂度**

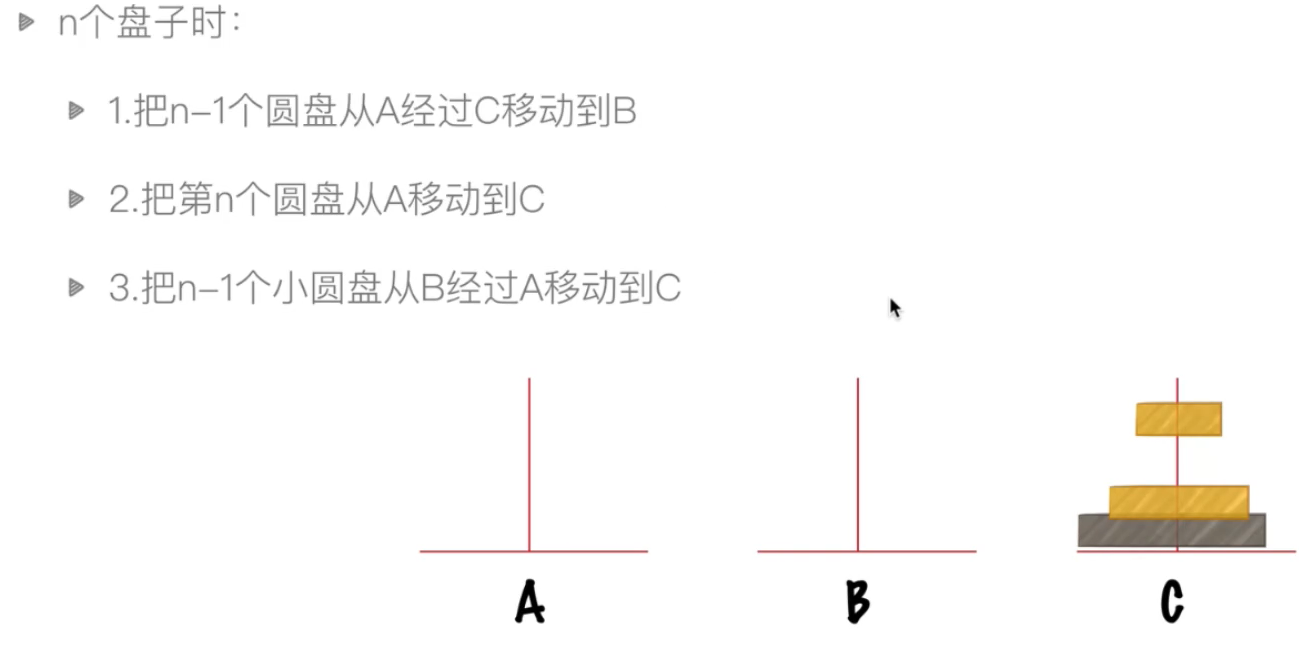


**3. 递归**

1. 循环自身

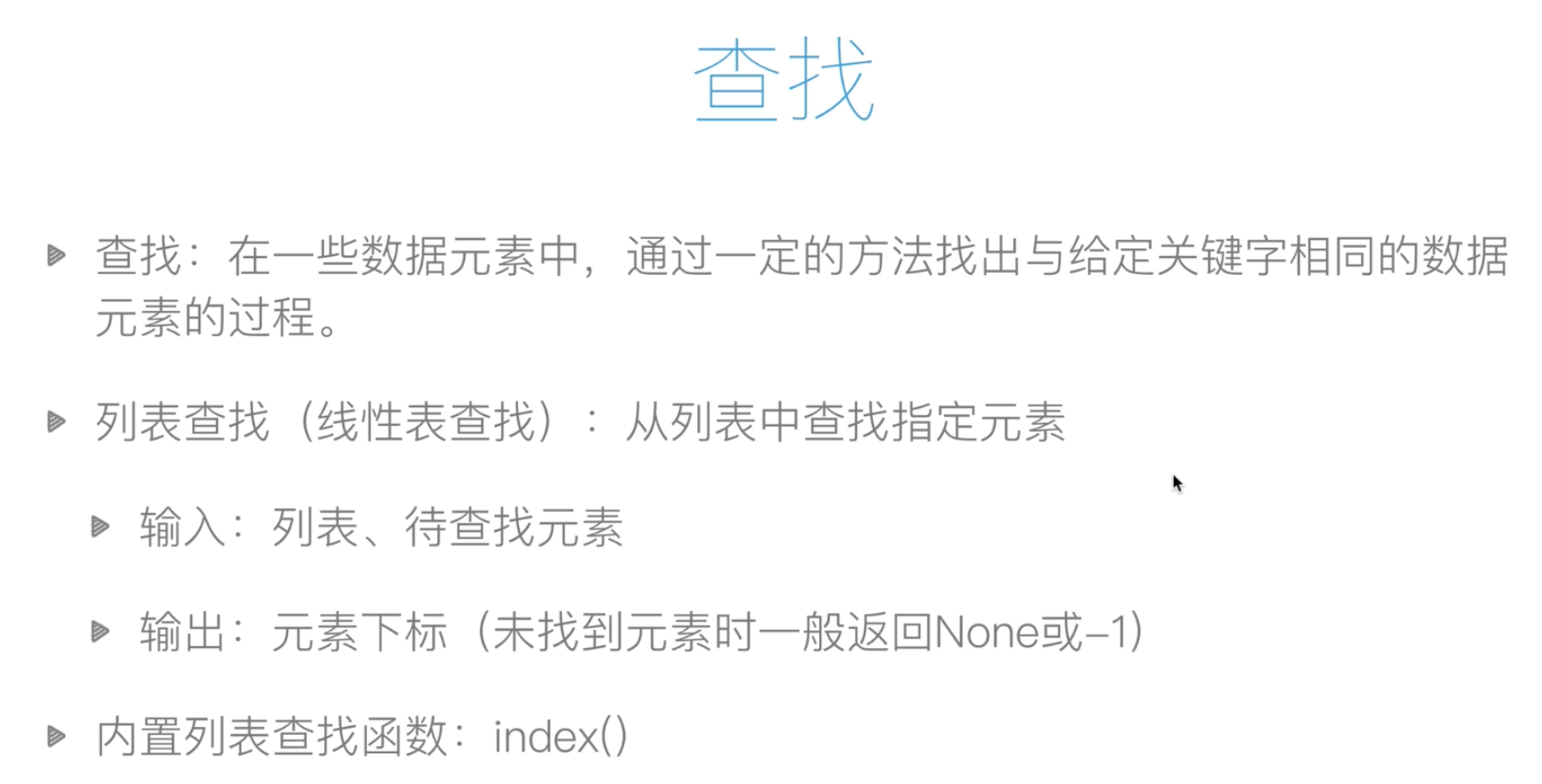
2. 结束条件

汉诺塔问题：



**4. 查找**

**1.顺序查找**



def search\_val(lis, v):

for index, val in enumerate(lis):

if val == v:

return index

else:

return None

a = [1, 2, 3, 4, 5]

b = search\_val(a, 5)

print(b)

# 列表顺序查找enumerate(lis， i) lis为可迭代对象，i表示下标开始位置默认为0

# dic = {

# "k1":"v1",

# "k2":"v2",

# "k3":"v3",

# }

# for i, key in enumerate(dic, 1):

# print(i,"\t",key)

**1.二分查找**

列表内置函数index()是顺序查找，因为二分查找需要有序

def binary\_search(lis, v):

left = 0

right = len(a)-1

while left <= right:

mid = (left+right)//2

# 表示

if v > lis[mid]:

left = mid + 1

if v < lis[mid]:

right = right - 1

if v == lis[mid]:

return v

else:

return None

c =binary\_search(a, 1)

print(c)

**5. 排序**

**1.冒泡排序**

# 冒泡排序O(n\*n)

def bubble\_sort(lis):

for i in range(len(lis)-1):

is\_sort = False # 优化算法，可能不用循环n-1次就已有序

for j in range(len(lis)-i-1):

if lis[j] > lis[j+1]:

is\_sort = True

lis[j], lis[j+1] = lis[j+1], lis[j]

if not is\_sort:

return

print(i, lis)

a = [1, 4, 5, 3, 7, 9, 6, 2, 8]

print('..........', a)

bubble\_sort(a)

**2.选择排序**



# 选择排序 O(n\*n)

def select\_sort(lis):

for i in range(len(lis)-1):

# 假如lis\_min为list无序区最小数的下标，默认为第一个数

lis\_min = i

# i之前为有序区，i之后为无序区，通过循环找到无序列表最小值然后获取下标

for j in range(i, len(lis)):

if lis[j] < lis[lis\_min]:

lis\_min = j

# 找到比lis\_min还小的值，并交换这个值

if lis\_min != i:

lis[i], lis[lis\_min] = lis[lis\_min], lis[i]

print(a)

a = [1, 4, 5, 3, 7, 9, 6, 2, 8]

print('..........', a)

select\_sort(a)

**3. 插入排序**



def insert\_sort(li):

for i in range(1, len(li)):

tmp = li[i] # i表示摸到的牌的下标

print(tmp)

j = i - 1 # 手里牌的下标

while j >= 0 and li[j] > tmp:

li[j+1] = li[j]

j -= 1

li[j+1] = tmp

print(li)

li = [5,7,4,6,3,1,2,9,8]

# print(li)

insert\_sort(li)

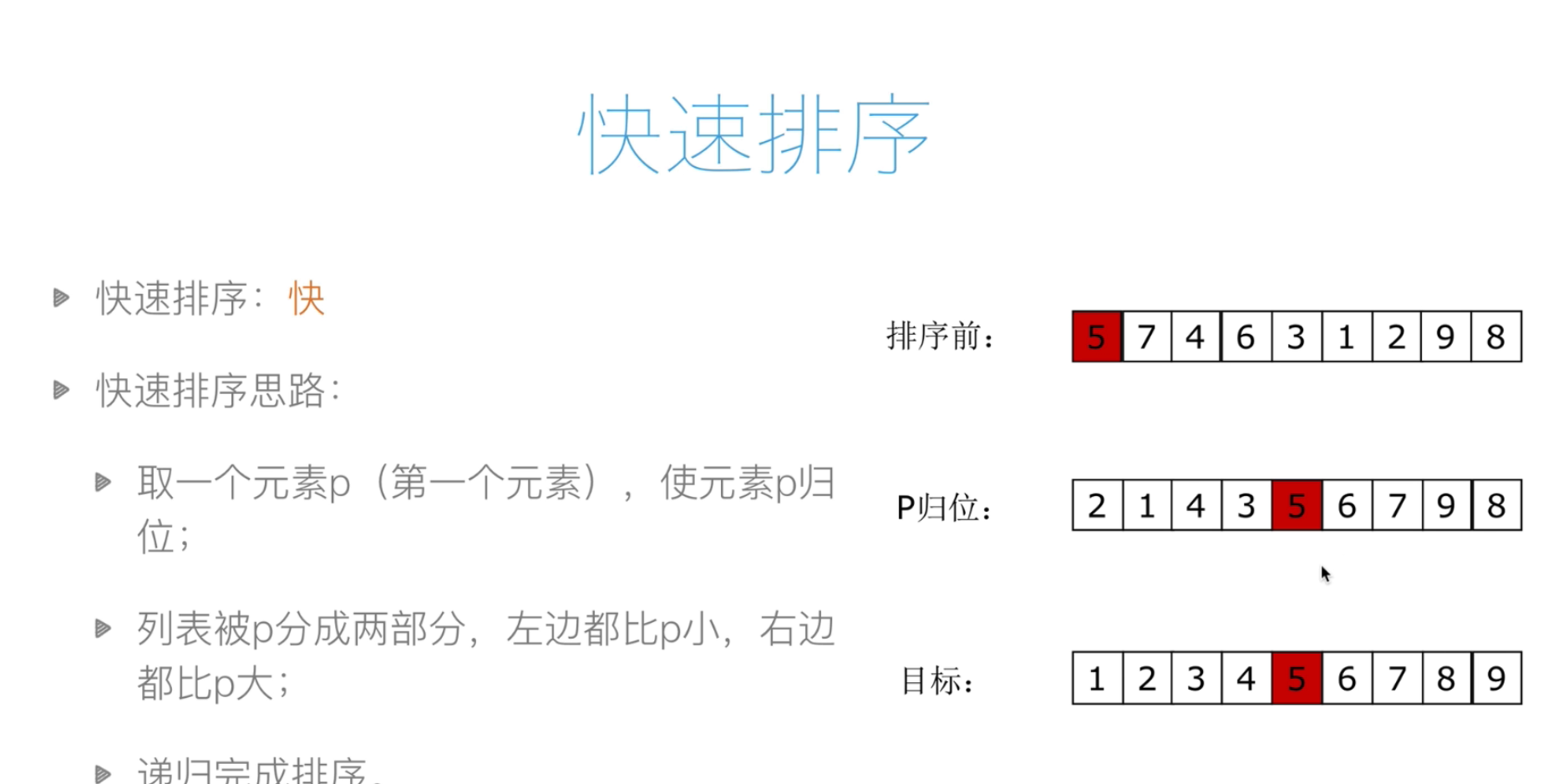
简单理解：

有列表li = [5,7,4,6,3,1,2,9,8]

1. 有序区为5，手里的牌为5，准备摸无序区(5之后所有元素)第一个元素7，然后判断，因为5<7 则不换位置，往下走

2. 此时有序区为5，7。手里的牌为7，准备摸无序区(7之后所有元素)第一个元素4，然后判断，因为7>4。交换位置，此时手里牌位置下标减一，比较5和4，5>4,交换位置，此时此时手里牌位置下标减一已经小于0,则退出循环，找下一元素

**4.快速排序**



# partition归位函数

# 目的是把小于tmp的值放在左边,大于tmp的值放在右边

# 思想是第一位元素值保存到tmp,然后变成一个空位,然后通过右往左找值或者是左往右找值填空位，最后把tmp放在left==right的位置

import random

import sys

import time

sys.setrecursionlimit(100000) # 设置递归最大深度

**def partition(li, left, right):**

# 把第一位元素值保存到tmp,然后变成一个空位

**tmp = li[left]**

**while left < right:**

# 从右面找比tmp小的数，li[right] >= tmp则right下标往前移,left==right时应该终止循环

**while left < right and li[right] >= tmp:**

**right -= 1**

# 循环结束后找到比tmp小的数(或者是因为右边数比tmp都大,此时下标left和right重合),即自己给自己赋值

**li[left] = li[right]**

# 从左面找比tmp大的数，li[left] <= tmp则left下标往后移，,left==right时应该终止循环

**while left < right and li[left] <= tmp:**

**left += 1**

# 循环结束后找到比tmp大的数(或者是因为左边数比tmp都小,此时下标left和right重合),即自己给自己赋值

**li[right] = li[left]**

# 最后空位指向left==right,要把值填回来

**li[left] = tmp**

# 返回此时mid下标

**return left**

**def quick\_sort(li, left, right):**

**if left < right**:# 至少两个元素

**mid = partition(li, left, right)**

**quick\_sort(li, left, mid-1)**

**quick\_sort(li, mid+1, right)**

li = list(range(10000))

random.shuffle(li)

a = time.time()

quick\_sort(li, 0, len(li)-1)

print(li)

print(time.time()-a)

**4. 希尔排序：是一组分组插入排序算法。**

先分组，然后组内进行插入排序

