# 算法基础

## 1.算法概念

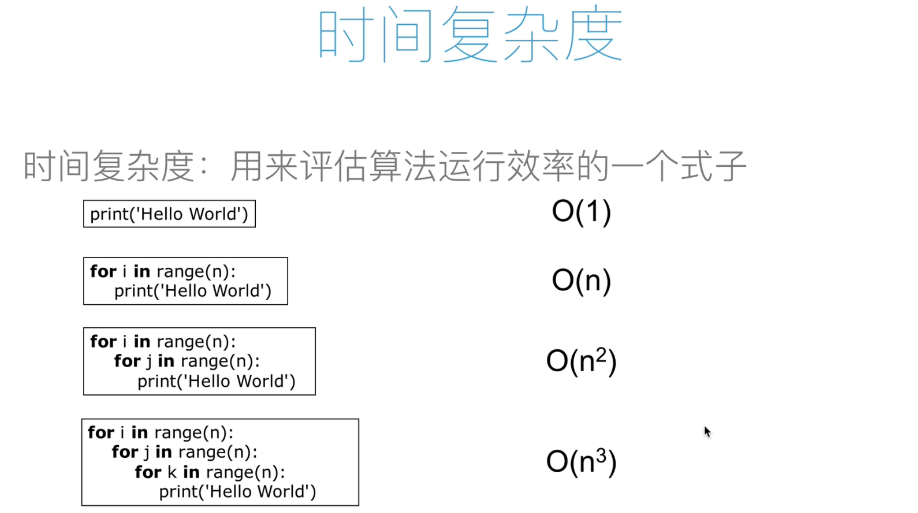
算法(Algorithm)：一个计算过程，解决问题的方法

Niklaus Wirth：“程序=数据结构+算法”

## 2.时间复杂度

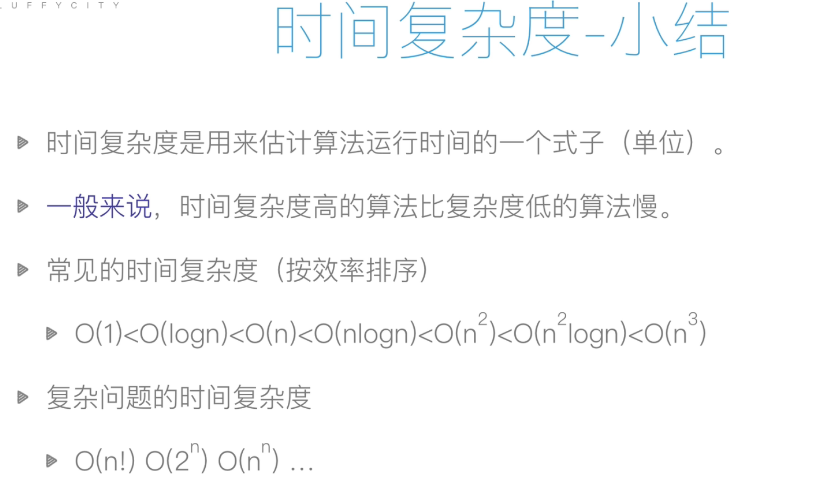
*for* i *in* range(n):  
 print("Hello World")

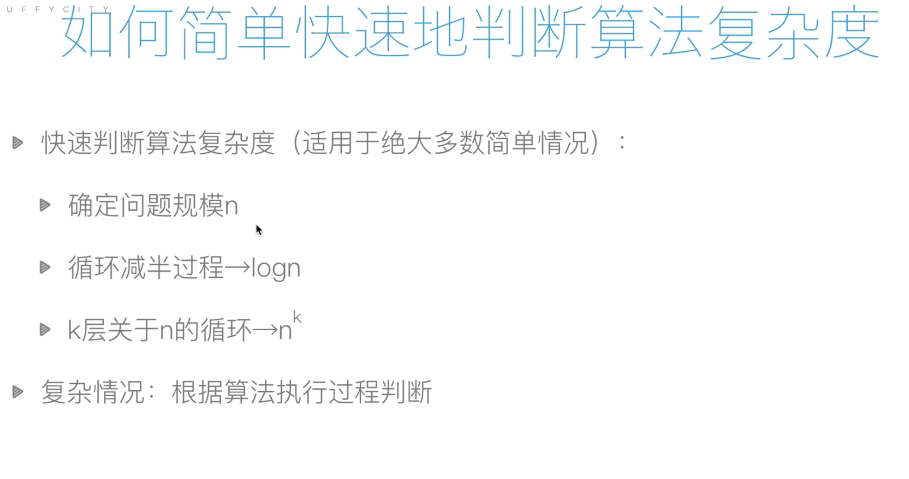
range 中的n ，表示一个问题规模，问题规模的大小影响问题的运行时间。



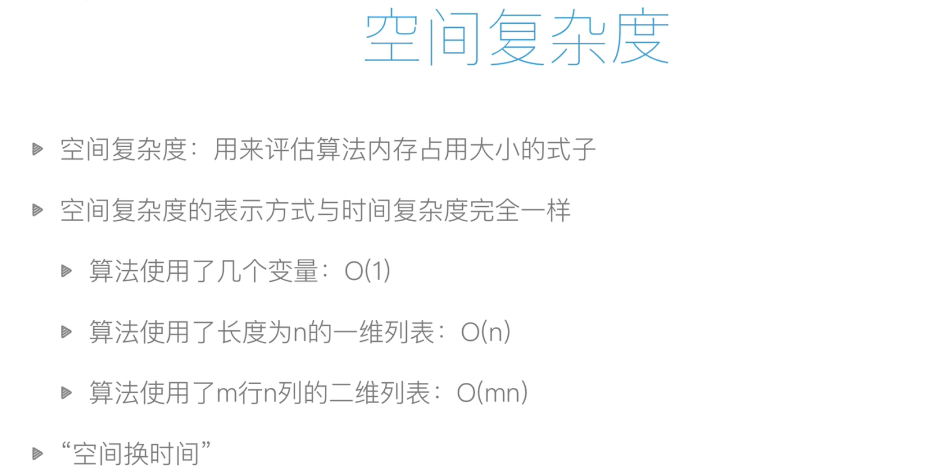
其中的1 n n方 n立方 只是表示一个单位，一个规模的单位







## 3.空间复杂度



## 4.递归

递归的两个特点：

1.调用自身

2.结束条件

汉诺塔：n表示盘子，a，b，c表示柱子,顺序意义是，盘子从 a 经过 b 到c  
*def* hanoi(*n*,*a*,*b*,*c*):  
 *if n* > 0:  
 hanoi(*n*-1,*a*,*c*,*b*)  
 print("moving from %s to %s"%(*a*,*c*))  
 hanoi(*n*-1,*b*,*a*,*c*)  
  
  
hanoi(100,"A","B","C")

## 5.查找

查找：在一些数据元素中，通过一定的方法找出与给定关键字相同对的数据元素的过程

列表查找（线性表查找）：从列表中查找指定元素

输入：列表、待查找元素

输出：元素小标（未找到元素时一般返回None或-1）

内置列表查找函数：index()

### 1.顺序查找（Linear Search）

顺序查找：也叫线性查找，从列表第一个元素开始，顺序进行搜索，知道找到元素或搜索到列表最后一个元素为止。

# li 表示列表；val表示待查找的元素  
# 此时的时间复杂度为：O(n)  
*def* linear\_search(*li*,*val*):  
 *for* ind,v *in* enumerate(*li*):  
 *if* v == *val*:  
 *return* ind  
 *else*:  
 *return None*

### 2.二分查找介绍

二分查找：又叫折半查找，从有序列表的初始候选区li[0:n]开始，通过对待查找的值与候选区中间值的比较，可以使候选区减少一半。

### 3.二分查找代码实现

*def* binary\_search(*li*,*val*):  
 left = 0  
 right = len(*li*) -1  
 *while* left <= right: #候选区还有值  
 mid = (left + right) //2 #mid left right 都表示列表的小标索引  
 *if li*[mid] == *val*:  
 *return* mid  
 *elif li*[mid] > *val*: # 待查找的值在mid左侧  
 right = mid-1  
 *else*: # li[mid] <val 待查找的值在mid右侧  
 left = mid + 1  
  
 *else*: #没有找到，就是不符合 left 不小于等于 right  
 *return None*

二分查找时间复杂度：O(logn)

### 4.二分查找和线性查找比较

*import* time  
  
*def* cal\_time(*func*):  
 *def* wrapper(*\**args,*\*\**kwargs):  
 t1 = time.time()  
 result = func(\**args*,\*\**kwargs*)  
 t2 = time.time()  
 print("%s running time: %s secs."%(func.\_\_name\_\_,t2-t1))  
 *return* result  
 *return* wrapper  
  
@cal\_time  
*def* linear\_search(*li*,*val*):  
 *for* ind,v *in* enumerate(*li*):  
 *if* v == *val*:  
 *return* ind  
 *else*:  
 *return None*@cal\_time  
*def* binary\_search(*li*,*val*):  
 left = 0  
 right = len(*li*) -1  
 *while* left <= right: #候选区还有值  
 mid = (left + right) //2 #mid left right 都表示列表的小标索引  
 *if li*[mid] == *val*:  
 *return* mid  
 *elif li*[mid] > *val*: # 待查找的值在mid左侧  
 right = mid-1  
 *else*: # li[mid] <val 待查找的值在mid右侧  
 left = mid + 1  
  
 *else*: #没有找到，就是不符合 left 不小于等于 right  
 *return None*li = list(range(100000000))  
linear\_search(li,3338900)  
binary\_search(li,3338900)

result：

linear\_search running time: 0.3142251968383789 secs.  
binary\_search running time: 0.0 secs.

结果可以得到：线性查找比二分查找耗时长得多。但是，二分查找弊端，必须是有序列表；

例如：232 如果是按照线性查找某个数字，那可能要查找40多亿次，但是2分查找的话就最多需要32次就可以找到。

如果是一个无序列表，那如果要进行二分查找，那必须先排序，但是排序的时间复杂度就要大于O(n),如果需要一直查找，就可以先排序。

## 6.排序

### 1.列表排序

排序：将一组“无序”的记录序列调整为“有序”的记录序列

列表排序：将无序列表变为有序列表

输入：列表

输出：有序列表

升序与降序

内置排序函数：sort()

### 2.常见排序算法



### 3.冒泡排序（Bubble Sort）

列表每两个相邻的输，如果前面比后面打，则交换这个数。

一趟排序完成后，则无序区减少一个数，有序区增加一个数。

代码关键点：趟、无序区范围

*def* bubble\_sort(*li*):  
 *for* i *in* range(len(*li*)-1): # 第i趟，从0开始  
 *for* j *in* range(len(*li*)-i-1): #列表内自循环，相当于列表里面的元素进行比较时的次数  
 *if li*[j] > *li*[j+1]: # 升序 > ,降序 <  
 *li*[j],*li*[j+1] = *li*[j+1],*li*[j] # 两个值进行交换  
 print(*li*)  
  
  
li = [9,8,7,1,2,3,4,5,6]  
bubble\_sort(li)  
运行结果：  
 [8, 7, 1, 2, 3, 4, 5, 6, 9]  
 [7, 1, 2, 3, 4, 5, 6, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

时间复杂度：O(n2) 。可以发现，在第四趟有，无序区其实已经排好了序，因此，后面的循环是没有必要的。下面进行改进。

**冒泡改进：**

*def* bubble\_sort(*li*):  
 *for* i *in* range(len(*li*)-1): # 第i趟，从0开始  
 exchange = *False  
 for* j *in* range(len(*li*)-i-1): #列表内自循环，相当于列表里面的元素进行比较时的次数  
 *if li*[j] > *li*[j+1]: # 升序 > ,降序 <  
 *li*[j],*li*[j+1] = *li*[j+1],*li*[j] # 两个值进行交换  
 exchange = *True* print(*li*)  
 *if not* exchange:  
 *return*li = [9,8,7,1,2,3,4,5,6]  
bubble\_sort(li)  
运行结果：  
 [8, 7, 1, 2, 3, 4, 5, 6, 9]  
 [7, 1, 2, 3, 4, 5, 6, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]  
 [1, 2, 3, 4, 5, 6, 7, 8, 9]

加一个exchange标志位的理解：意思就是说，在进行排序的过程中前后有数字的位置进行交换，那么，就把exchange=true , 后面的判断语句就不执行。当9,8,7,6都排序好了以后，就会发现，后面的循环冒泡排序其实是没有进行数字的变动的，那此过程中 exchange 就是false，那么，判断语句就会被执行。

提示：每次循环都是遍历列表，对每一个数字进行前后比较。