# 数据结构

## 二叉树

### 构造二叉树

#### 构造平衡二叉树

def build(self, ans，l, r): # 利用二分法建立平衡二叉树，其中ans是升序的  
 if l == r:  
 return TreeNode(self.ans[l]) # 建立叶子结点  
 if l > r:  
 return  
 mid = (l + r) // 2  
 node = TreeNode(self.ans[mid]) # 建立根节点和非叶子节点  
 node.left = self.build(l, mid - 1)  
 node.right = self.build(mid + 1, r)  
 return node # 返回一根子树

#### 构造普通二叉树

def buildTree(self, start, A): # 从start处开始构造普通二叉树，一般从0开始，A无序  
 if start >= len(A):  
 return  
 node = TreeNode(A[start])  
 node.left = self.buildTree(2 \* start + 1, A) # 下一个节点为2 \* start + 1  
 node.right = self.buildTree(2 \* start + 2, A)  
 return node

### 2.前、中、后遍历递归与非递归实现：

#### 前序递归遍历：

def pre\_order1(self, root, result):  
 if root == None:  
 return  
 result.append(root.val)  
 self.pre\_order1(root.left, result)  
 self.pre\_order1(root.right, result)  
 return result

#### 前序遍历非递归实现：

def pre\_order2(self, root):  
 result = []  
 stack = []  
 pos = root  
 while pos is not None or stack:  
 if pos is not None:  
 result.append(pos.val)  
 stack.append(pos)  
 pos = pos.left  
 else:  
 pos = stack.pop()  
 pos = pos.right  
 return result

#### 中序遍历递归实现：

def mid\_order1(self, root, result):  
 if root == None:  
 return  
 self.mid\_order1(root.left, result)  
 result.append(root.val)  
 self.mid\_order1(root.right, result)

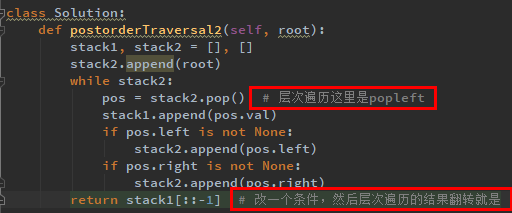
#### 中序遍历非递归实现：

def mid\_order2(self, root):  
 result = []  
 stack = []  
 pos = root  
 while pos is not None or stack:  
 if pos is not None:  
 stack.append(pos)  
 pos = pos.left  
 else:  
 pos = stack.pop()  
 result.append(pos.val)  
 pos = pos.right  
 return result

#### 后序遍历递归实现：

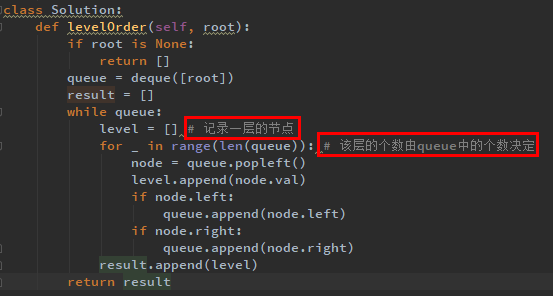
def post\_order1(self, root, result):  
 if root == None:  
 return  
 self.post\_order1(root.left, result)  
 self.post\_order1(root.right, result)  
 result.append(root.val)

#### 后序遍历非递归实现：



### 3.BFS遍历实现

def BFS(self, root):  
 result = []  
 queue = collections.deque()  
 queue.append(root)  
 while queue:  
 root = queue.popleft()  
 result.append(root.val)  
 if root.left is not None:  
 queue.append(root.left)  
 if root.right is not None:  
 queue.append(root.right)  
 return result



### 4.求最大深度

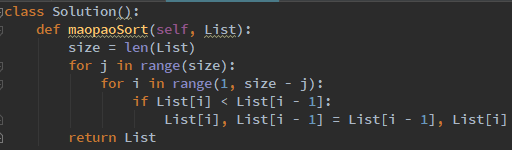
def max\_depth(self, root):  
 if root == None:  
 return 0  
 ldepth = self.max\_depth(root.left)  
 rdepth = self.max\_depth(root.right)  
  
 return max(ldepth, rdepth) + 1

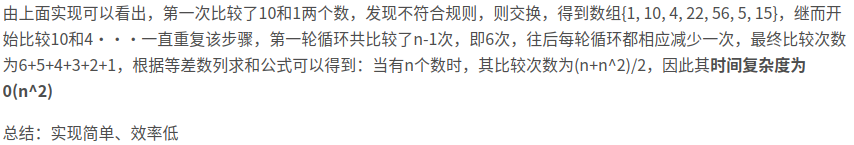
### 5.求所有节点数

def nums(self, root):  
 if root == None:  
 return 0  
  
 lnums = self.nums(root.left)  
 rnums = self.nums(root.right)  
  
 return lnums + rnums + 1

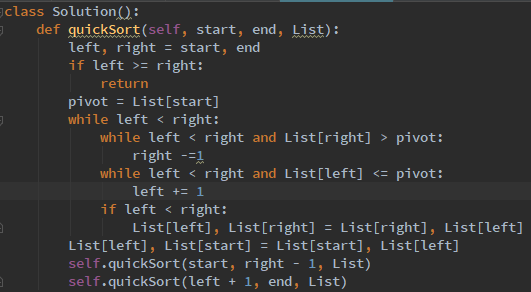
## 2.排序

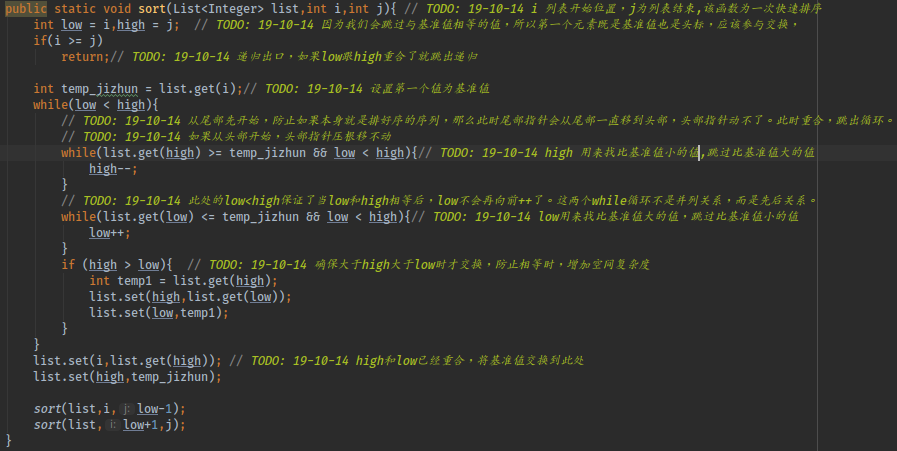
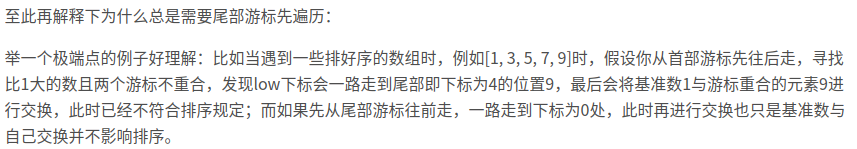
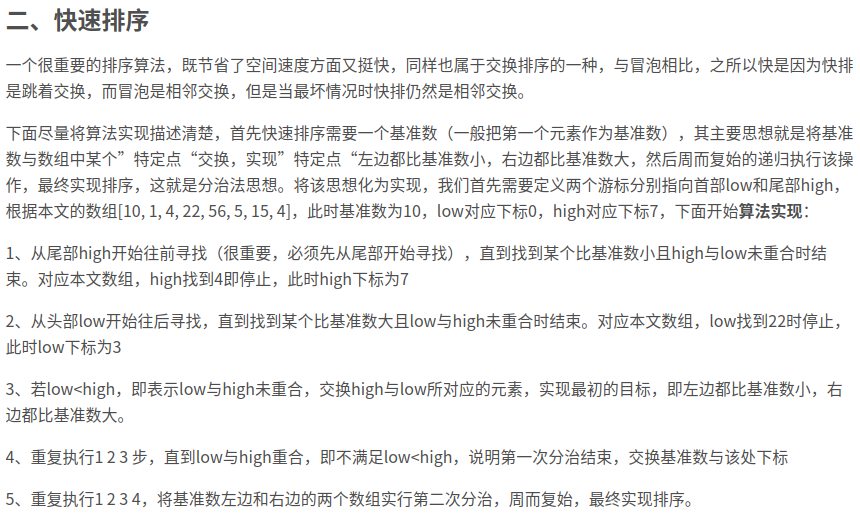
### 1.冒泡排序



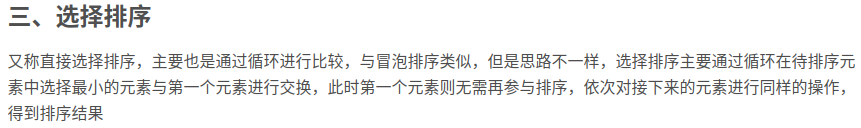


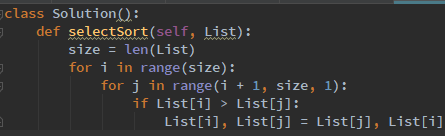
### 快速排序



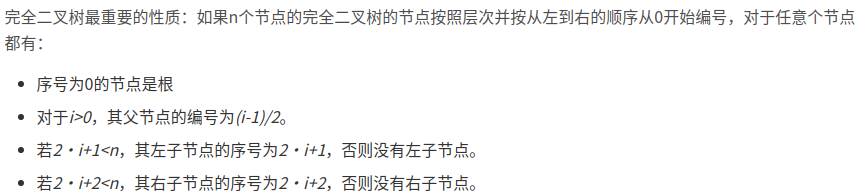
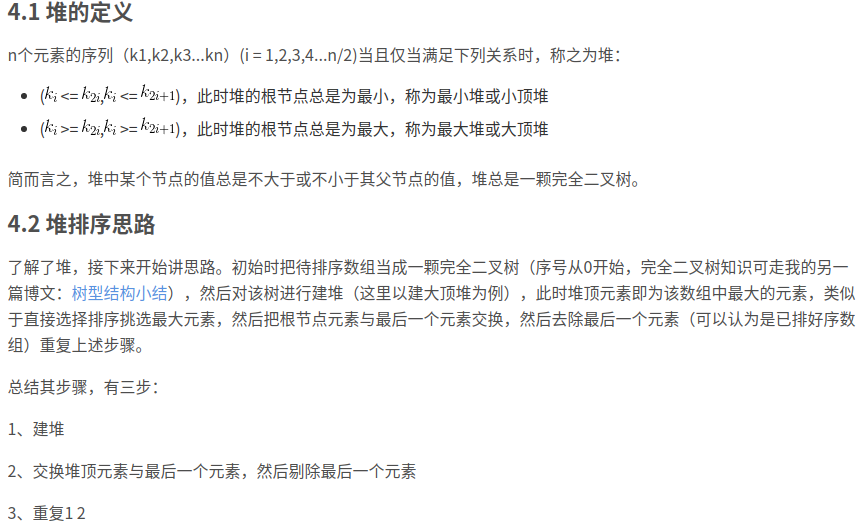
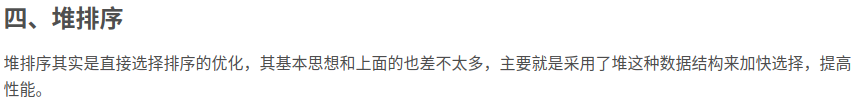


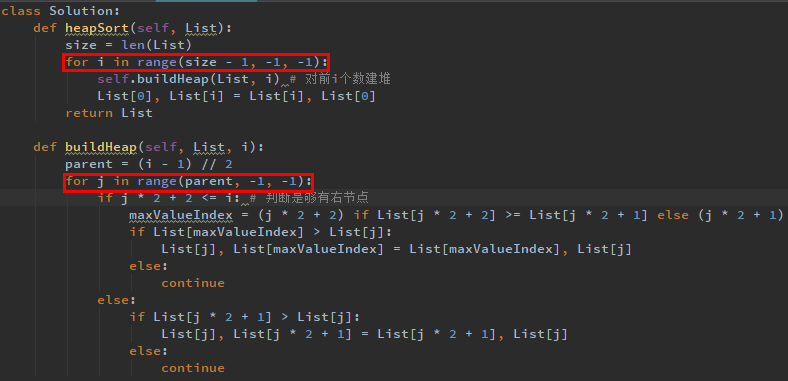
### 3.选择排序

2019-10-14 15-13-16 的屏幕截图

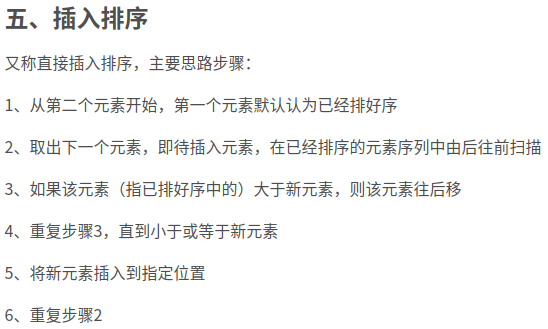


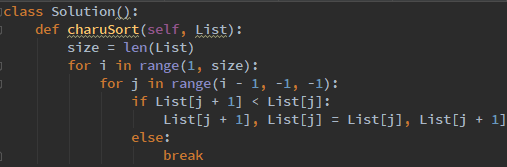
### 4.堆排序（选择排序的高级实现）

2019-10-14 17-17-08 的屏幕截图

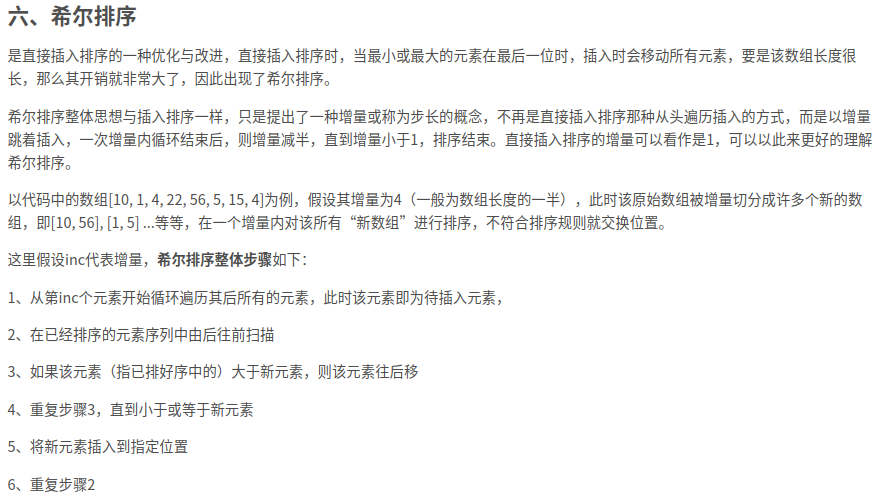


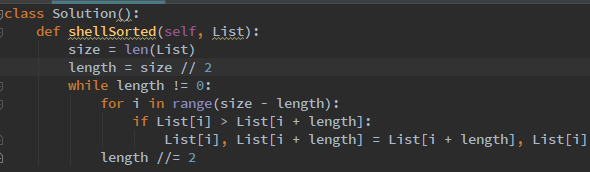
### 5.插入排序

2019-10-14 19-29-54 的屏幕截图

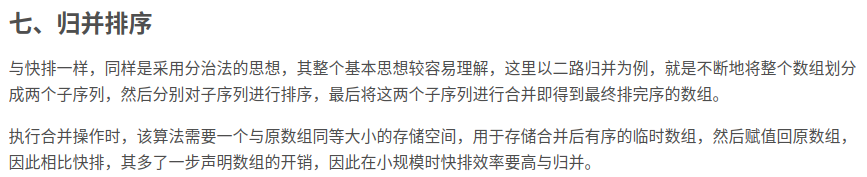


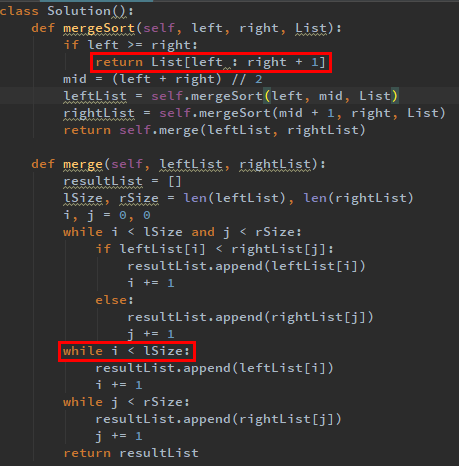
### 6.希尔排序（插入排序的高级实现）



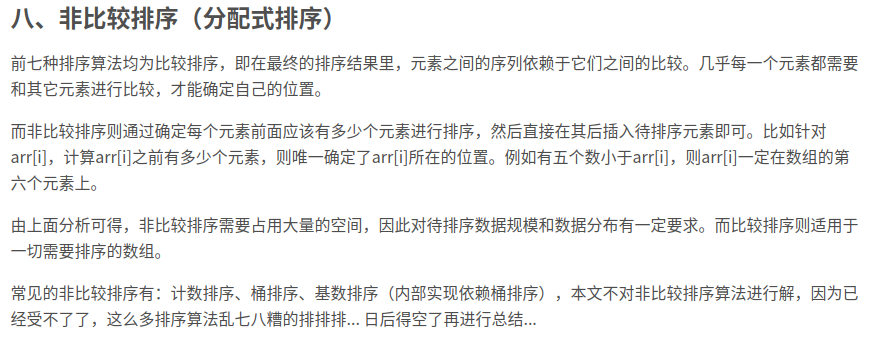


### 7.归并排序





### 8.非比较排序



#### 8.1、计数排序

<https://www.jianshu.com/p/86c2375246d7>

### 9.各种排序算法的时间复杂度

