1. 插入排序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

java 好记忆 复杂算法用java

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **void** insert(**int**[] list){

**int** insertNote;

**for** (**int** i = 1; i < list.length; i++) {

insertNote = list[i];

**int** j = i - 1;

**while** (j >= 0 && insertNote < list[j]) {

list[j + 1] = list[j];

j--;

}

list[j + 1] = insertNote;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Python比较难记忆 不建议

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def insert(data):

i = 1

while i < len(data):

for j in range(0,i):

if data[j]<data[i]:

data.insert(j,data.pop(i))

i+=1

return data

1. 选择排序（第i个与后面每个比较，如果小于他交换，每次找到最大一个） ， 优化后的冒泡， 对比次数不变 ，交换次数减少

def select(data):

for i in range(0,len(data)-1):

for j in range(i+1,len(data)):

if data[i]<data[j]:

data[i],data[j]=data[j],data[i]

return data

1. 快速排序（左右分别递归，每次把第一个数作为基数，插入在列表中间，使左边都比它小，右边都比较大）

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

java 好记忆 复杂算法用java

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**public** **void** quicksort(**int**[] list, **int** L, **int** R){

**int** i, j, key;

**if** (L > R) {

**return**;

}

i = L;

j = R;

key = list[i]; // 用子表的第一个记录做基准

**while** (i < j) { // 从表的两端交替向中间扫描

//外层循环，每一次i和j的值就会往中间汇拢一段

//知道最后 i = j，的时候收敛退出

//一共有5个i<j , 2,4加key判断

**while** (i < j && list[j] >= key)

j--;

**if** (i < j) { // 用比基准小的记录替换低位记录

list[i] = list[j];

i++; //替换后i往后移一位 ，每次变动都是这个坑被填过以后

}

**while** (i < j && list[i] < key)

i++;

**if** (i < j){ // 用比基准大的记录替换高位记录

list[j] = list[i];

j--; //替换后j往前移一位

}

}

list[i] = key;// 此时i=j 将基准数值替换回 a[i]

quicksort(list, L, i - 1); // 对低子表进行递归排序

quicksort(list, i + 1, R); // 对高子表进行递归排序

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Python比较难记忆 不建议

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def quickSort(L, low, high):

i = low

j = high

if i >= j:

return L

key = L[i] #设置第一个为基数

L[i]=None #挖坑帮助理解

while i < j:

while i < j and L[j] >= key: #J从右到左扫描，发现小于基数的放入前面的坑，此坐标为新坑

j = j-1

L[i] = L[j]

L[j] = None

while i < j and L[i] <= key:#i从左到右扫描，发现小于基数的放入前面的坑，此坐标为新坑

i = i+1

L[j] = L[i]

L[i]=None

L[i] = key #填回坑

#此时i=j，左右分别递归

quickSort(L, low, i-1)

quickSort(L, i+1, high)

return L

quickSort(l,0,len(l)-1)

1. 创建链表：

public Node head;

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

}

}

//添加数据

public void add(int data) {

Node current = null;

if (head == null) {

head = new Node(data);

current = head;

} else {

current.next = new Node(data);

current = current.next;

}

}

//遍历链表

public void print(Node node) {

if (node == null) {

return;

}

current = node;

while (current != null) {

System.out.println(current.data);

current = current.next;

}

}

LinkList list = new LinkList();

for (int i = 0; i < 10; i++) {

list.add(i);

}

倒数第k个结点

得到链表的长度之后，直接输出第(size-k)个节点

单链表中的中间结点(不允许你算出链表的长度)

置两个指针first和second，只不过这里是，两个指针同时向前走，second指针每次走两步，first指针每次走一步，直到second指针走到最后一个结点时，

此时first指针所指的结点就是中间结点

while (second != null && second.next != null) {

first = first.next;

second = second.next.next;

}

单链表的反转

public Node reverse(Node node) {

Node pre = null;

Node now = node;

Node next = null;

while (now != null) {

next = now.next;

now.next = pre;

pre = now;

now = next;

}

return pre;

}

单链表反转打印

public void reversePrint(Node head) {

if (head == null)

return;

reversePrint(head.next);

System.out.println(head.data);

}

判断单链表是否有环：

first指针每次走一步，second指针每次走两步，如果first指针和second指针相遇，

while (second != null) {

first = first.next; //first指针走一步

second = second.next.next; second指针走两步

if (first == second) { //一旦两个指针相遇，说明链表是有环的

return true;

}