**1 你的优点和缺点:**

**优点：**学习东西比较快，有责任心。

**缺点：**办事情比较急，准确完成任务可能会不稳定。这既说明自己的办事责任心强，也间接引出自己的缺点。

**2 将来想留在哪？**

实事求是的说，想留在那，有发展等。

**3 排序算法：**

<https://blog.csdn.net/liqinzhe11/article/details/78743743>

排序算法稳定性：两个相同的元素排序前后的相对位置关系不会发生改变。



**3.1冒泡排序**

**排序过程**

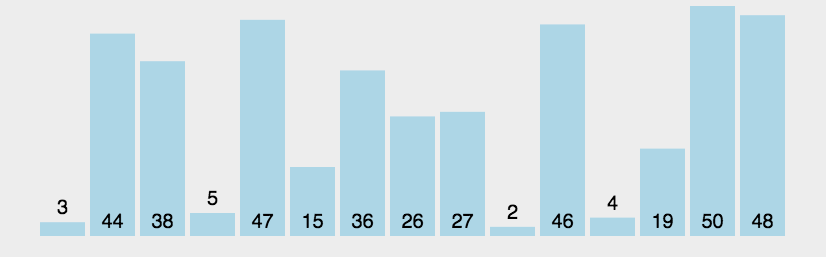
将第一个元素与第二个元素比较大小，如果第一个元素大于第二个元素则调换他们两的位置；

比较第二个元素和第三个元素的大小，如果第二个元素大于第三个元素则调换他们两的位置；

依次类推，进行两两元素的比较和交换，**最终最大的元素排在了最后面**；

重复1到3过程，直到所有元素都排序。

图片演示：



3.1//冒泡排序

//平均时间复杂度:O(N^2)

//最坏情况复杂度:O(N^2)

//空间复杂度:O(1)

//稳定排序

void bubblesort**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** n **-** 1 **-** i**;** j**++)**

**{**

**if** **(**a**[**j**]** **>** a**[**j**+**1**])**

swap**(**a**[**j**],** a**[**j**+**1**]);**

**}**

**}**

**}**

3.2//冒泡排序升级版1

//设置一个标记来标志一趟比较是否发生交换

//如果没有发生交换，则数组已经有序

void bubblesort2**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

bool flag**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

flag **=** **false;**

**for** **(**int j **=** 0**;** j **<** n **-** 1 **-** i**;** j**++)**

**{**

**if** **(**a**[**j**]** **>** a**[**j**+**1**])**

**{**

swap**(**a**[**j**],** a**[**j**+**1**]);**

flag **=** **true;**

**}**

**}**

**if** **(!**flag**)**

**break;**

**}**

**}**

3.3//冒泡排序优化二

//用一个变量记录下最后一个发生交换的位置，后面没有发生交换的已经有序

//所以可以用这个值来作为下一次比较结束的位置

void bubblesort3**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

int flag **=** n**;**

int stop\_pos**;**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)**

**{**

stop\_pos **=** flag **-** 1**;**

flag **=** 0**;**

**for** **(**int j **=** 0**;** j **<** stop\_pos**;** j**++)**

**{**

**if** **(**a**[**j**]** **>** a**[**j**+**1**])**

**{**

swap**(**a**[**j**],** a**[**j**+**1**]);**

flag **=** j **+** 1**;**//随着j++，flag也在增加，最终保证stop\_pos的值在j的最大值处即可

**}**

**}**

**}**

**}**

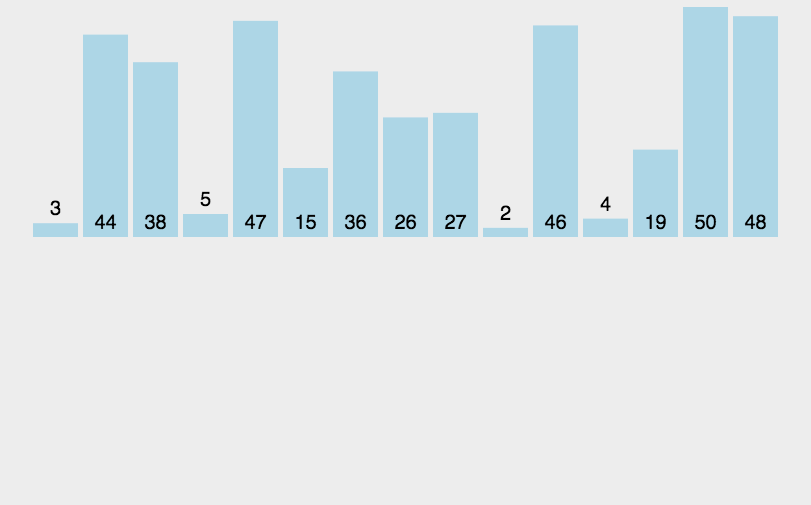
**3.2 插入排序**

从第二个元素开始，与前一个元素比较，前一个大于此元素则将前元素向后移动一位。对于第K个元素，将该元素的值存储在临时变量中，比较第前一个元素与该元素的大小，如果大于该元素就将前一个元素往后移动一步；

比较前面第二个元素与该元素的大小，如果大于该元素就将前第二个元素往后移动一步；

重复上述过程直到找到小于等于原来第K个元素（保存在零时变量中）的位置，并将第K个元素插入到这个元素的后面。或者找不到小于等于第K个元素的位置，就将原来第K个元素插入到数组的首地址。（直到最后一个元素也执行同样的操作为止）

**图片演示**



//插入排序//平均时间复杂度:O(N^2)//最坏情况复杂度:O(N^2)//最好情况复杂度:O(N)

//空间复杂度:O(1)//最多需要n(n−1)/2次比较//最少需要n−1次比较//稳定排序

void insertsort**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

**for** **(**int i **=** 1**;** i **<** n**;** i**++)**

**{**

int insert\_num **=** a**[**i**],** j**;**//当前元素

**for** **(**j **=** i **-** 1**;** j **>=** 0**;** j**--)**

**{**

**if** **(**a**[**j**]** **>** insert\_num**)**

a**[**j **+** 1**]** **=** a**[**j**];**//前一个元素向后移动一位

**else**

**break;**

**}**

a**[**j **+** 1**]** **=** insert\_num**;**//当前元素插入到小于等于当前元素的后边

**}**

**}**

**3.3 选择排序**

1、每一次从后面选择出一个最小的值（swap\_pos），替换到前面来(i)。

//选择排序，每一次从后面选择出一个最小的值（swap\_pos），替换到前面来(i)。

//平均时间复杂度 O(n^2)

//最坏时间复杂度 O(n^2)

//最好时间复杂度 O(n^2)

//空间复杂度 O(1)

//我这个写法 是稳定排序

void select\_sort**(**vector**<**int**>&** vt**)**

**{**

**for** **(**int i **=** 0**;** i **<** vt**.**size**()** **-** 1**;** i **++)**

**{**

int swap\_pos **=** i**;**

**for** **(**int j **=** i **+** 1**;** j **<** vt**.**size**();** j**++)**

**{**

**if** **(**vt**[**swap\_pos**]** **>** vt**[**j**])**

**{**

swap\_pos **=** j**;**

**}**

**}**

**if** **(**swap\_pos **!=** i**)**

**{**

swap**(**vt**[**swap\_pos**],** vt**[**i**]);**

**}**

**}**

**}**

**3.4希尔排序**

**希尔排序**是在插入排序的基础上进行发展的，通过一个希尔增量先排序一定间隔的数据。

**排序过程**

插入排序每次与前面一个比较，然后再往前一个，而希尔排序每次往前K个；

当增量为1的时候，希尔排序与插入排序就完全是一样的过程；

所以代码也很好实现，将插入排序中增1的地方改为增K就行。

**图片演示**



复杂度分析

希尔排序的时间复杂度比较复杂，选用不同的希尔增量也会导致复杂度不同。

//希尔排序

//最坏情况复杂度:O(N^2)

//不稳定排序

void shellsort**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

**for** **(**int increment **=** n **/** 2**;** increment **>** 0**;** increment **/=** 2**)**

**{**

**for** **(**int i **=** increment**;** i **<** n**;** i**++)**

**{**

int insert\_num **=** a**[**i**],** j**;**

**for** **(**j **=** i **-** increment**;** j **>=** 0**;** j **-=** increment**)**

**{**

**if** **(**a**[**j**]** **>** insert\_num**)**

a**[**j **+** increment**]** **=** a**[**j**];**

**else**

**break;**

**}**

a**[**j **+** increment**]** **=** insert\_num**;**

**}**

**}**

**}**

**3.5 堆排序**

**排序过程**

1建立最大堆，建堆的过程是从N/2的位置开始，将父节点与子节点比较，如果子节点大于父节点则交换。为什么是N/2，是因为堆中树叶的个数是N/2。

2从堆中删除堆顶元素，对于最大堆而言，堆顶元素也就是最大元素。每删除一个堆顶元素，就将堆顶元素放在数组的后面，因为每删除一个就出现一个空位，所以数组后面是有地方存放的。

3进行N-1次的删除以后，整个数组就是排序的状态了。

**思路：**把当前最大元素移动到堆顶，递减数组的个数，将当前最大的数移动到后面作为排好序的元素。

//堆排序

//建堆的平均时间是：O(N)

//建堆的最坏情况是：O(NlogN)

//删除元素的时间是：O(logN)

//整个排序平均时间复杂度：O(N+NlogN)=O(NlogN)

//最坏情况复杂度：O(NlogN)

//不稳定排序

//建立一个大顶堆O(n),要求就是 把最大的元素 移动到堆顶 也就是a[0]

void make\_heap**(**vector**<**int**>&** a**,** int size**)** //size的当前堆的大小，也就是数组的前size个数

//（将当前最大移动到后面作为排好序的元素）

**{**

**for** **(**int i **=** size **-** 1**;** i **>** 0**;** i**--)**

**{**

**if** **(**i **%** 2 **&&** a**[**i**]** **>** a**[(**i **-** 1**)** **/** 2**])**//奇数,(i-1)/2为i节点的父节点下标

swap**(**a**[**i**],** a**[(**i **-** 1**)** **/** 2**]);**//交换父节点与子节点

**else** **if** **(**i **%** 2 **==** 0 **&&** a**[**i**]** **>** a**[(**i **-** 2**)** **/** 2**])**//偶数

swap**(**a**[**i**],** a**[(**i **-** 2**)** **/** 2**]);**

**}**

**}**

void heapsort**(**vector**<**int**>&** a**)**

**{**

int n **=** a**.**size**();**

**while** **(**n**)**

**{**

make\_heap**(**a**,** n**);** //每次把新的最大元素移到堆顶，也就是a[0]

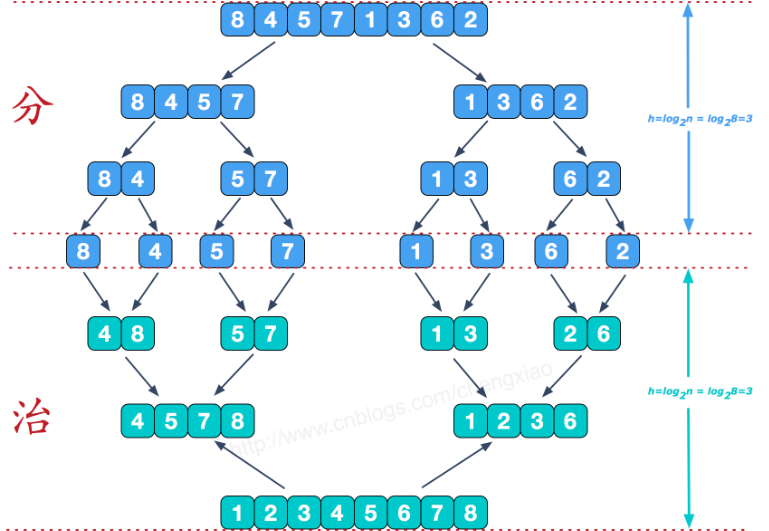
n**--;**

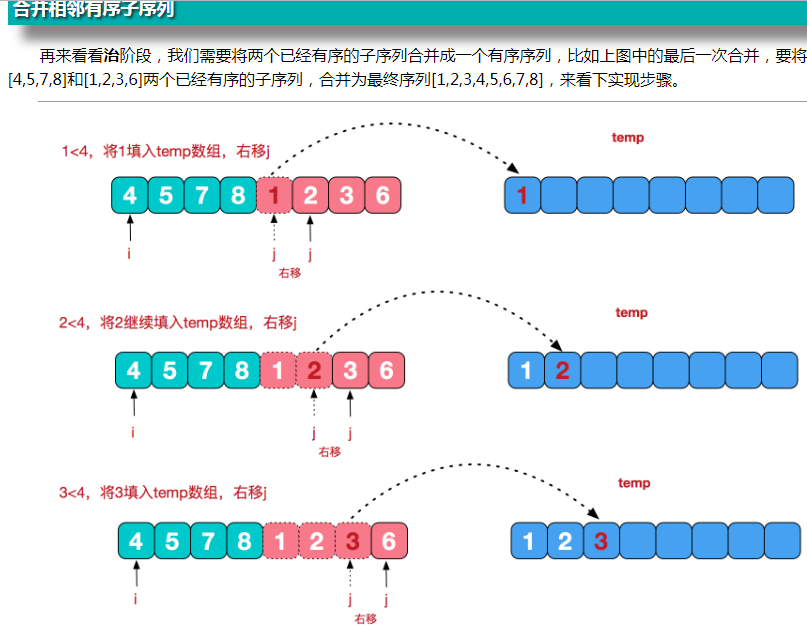
swap**(**a**[**0**],** a**[**n**]);** //然后把当前最大移动到后面来作为排好序的元素

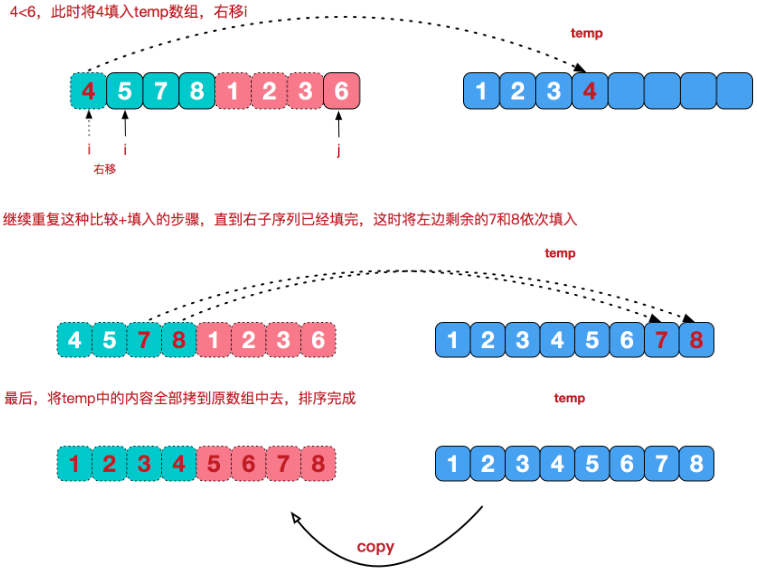
**}**

**}**

**3.6 归并排序**







**排序过程**

1将数组Ｎ从中间分成两个数组N1和N2；

2将N1和N2分别递归用归并排序来排序；

3归并N1与N2。

//归并排序

//平均时间复杂度：O(NlogN)

//稳定排序

vector**<**int**>** mergeHelper**(**vector**<**int**>** **&**a**,** int left**,** int right**)**

**{**

**if** **(**left **==** right**)** **return** vector**<**int**>** **(**1**,** a**[**left**]);**

int mid **=** **(**right **-** left**)** **/** 2 **+** left**;**

vector**<**int**>** l **=** mergeHelper**(**a**,** left**,** mid**);**

vector**<**int**>** r **=** mergeHelper**(**a**,** mid **+** 1**,** right**);**//将数组从中间的分为两个数组left和right

//merge

vector**<**int**>** ret**;**

int ll **=** 0**,** rr **=** 0**;**

**while** **(**ll **<** l**.**size**()** **&&** rr **<** r**.**size**())**//将左半部分或右半部分插入完毕

**{**

**if** **(**l**[**ll**]** **<=** r**[**rr**])** ret**.**push\_back**(**l**[**ll**++]);**

**else** ret**.**push\_back**(**r**[**rr**++]);**

**}**

**while** **(**ll **<** l**.**size**())** ret**.**push\_back**(**l**[**ll**++]);**//将剩余的直接插入进数组

**while** **(**rr **<** r**.**size**())** ret**.**push\_back**(**r**[**rr**++]);**

**return** ret**;**

**}**

void mergesort**(**vector**<**int**>&** a**)**

**{**

a **=** mergeHelper**(**a**,** 0**,** a**.**size**()** **-** 1**);**

**}**

**3.7 快速排序（非常重要，必背（第二种方法吧，第一种不太理解））**

**（1）取中算法**

**排序过程**

1 选择一个枢纽元，可以选择首，尾，中三个数的中位数作为枢纽元；

2 将枢纽元的为止与数组的尾地址进行交换；

3 定义两个指针，P1指向数组首地址，P2指向数组倒数第二个位置，P1所指元素的值与枢纽元比较，如果小于枢纽元则后移一位，如果大于就停下来。P1所指元素的值与枢纽元比较，如果大于枢纽元则前移一位，如果小于就停下来；

4 交换P1和P2所指向的元素；

5 重复３和４直到P1大于P2；

6 对数组的分割过程同样采用递归的方法。

**快排是必须背住的排序！**

1 //快速排序

//平均时间复杂度：O(NlogN)

//最坏情况复杂度：O(N^2)

//不稳定排序

void quicksortHelper**(**vector**<**int**>&** a**,** int start**,** int end**)**

**{**

**if** **(**start **>=** end**)** **return;**

int l **=** start**,** r **=** end**;**

int pivot **=** a**[(**end **-** start**)** **/** 2 **+** start**];**

**while** **(**l **<=** r**)**

**{**

**while** **(**l **<=** r **&&** a**[**r**]** **>** pivot**)** r**--;**

**while** **(**l **<=** r **&&** a**[**l**]** **<** pivot**)** l**++;**

**if** **(**l **<=** r**)** swap**(**a**[**l**++],** a**[**r**--]);**

**}**

quicksortHelper**(**a**,** start**,** r**);**

quicksortHelper**(**a**,** l**,** end**);**

**}**

void quicksort**(**vector**<**int**>&** a**)**

**{**

quicksortHelper**(**a**,** 0**,** a**.**size**()** **-** 1**);**

**}**

2 //快排的最差时间复杂度为O(n²)

//通常出现在选择的轴值(pivot)不能将数组划分为两个长度相等的子数组的时候

//一个较好的办法是“三数取中”，查看当前数组的第一个、中间一个和最后一个位置的数组，取其中位数，以此来降低轴值选择得不好的可能性。

int findmiddle**(**int a**,** int b**,** int c**)**

**{**

**if** **(**a **>=** b **&&** a **<=** c**)**

**return** a**;**

**else** **if** **(**b **>=** a **&&** b **<=** c**)**

**return** b**;**

**else**

**return** c**;**

**}**

void quicksortHelper**(**vector**<**int**>&** a**,** int start**,** int end**)**

**{**

**if** **(**start **>=** end**)** **return;**

int l **=** start**,** r **=** end**;**

int pivot **=** findmiddle**(**a**[**start**],** a**[**end**],** a**[(**end **-** start**)** **/** 2 **+** start**]);**

**while** **(**l **<=** r**)**

**{**

**while** **(**l **<=** r **&&** a**[**r**]** **>** pivot**)** r**--;**

**while** **(**l **<=** r **&&** a**[**l**]** **<** pivot**)** l**++;**

**if** **(**l **<=** r**)** swap**(**a**[**l**++],** a**[**r**--]);**

**}**

quicksortHelper**(**a**,** start**,** r**);**

quicksortHelper**(**a**,** l**,** end**);**

**}**

void quicksort**(**vector**<**int**>&** a**)**

**{**

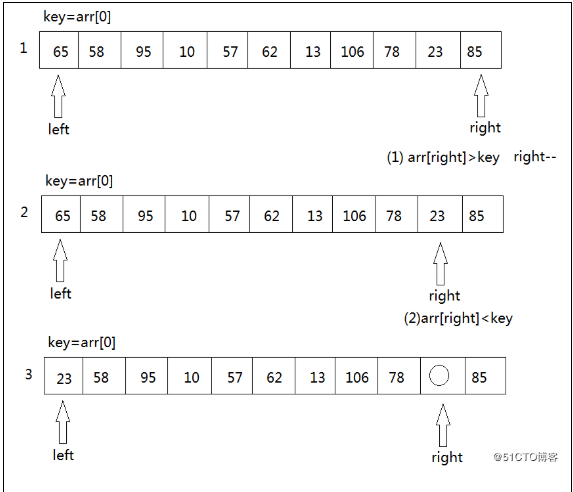
quicksortHelper**(**a**,** 0**,** a**.**size**()** **-** 1**);**

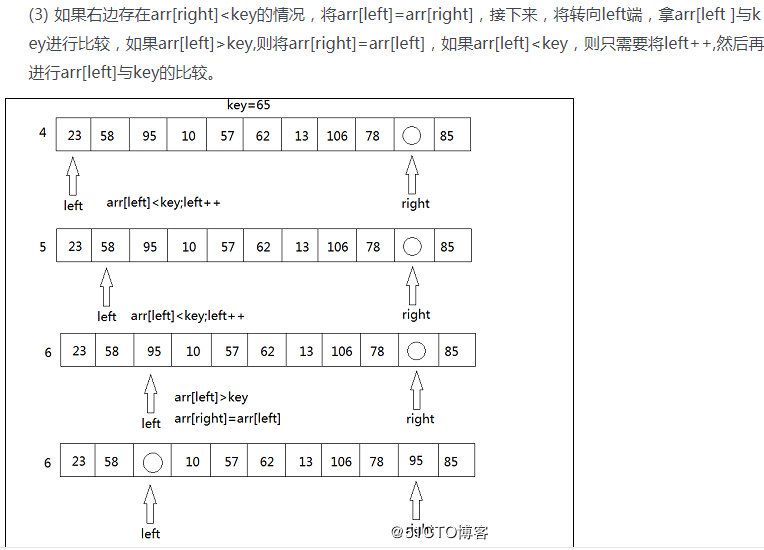
**}**

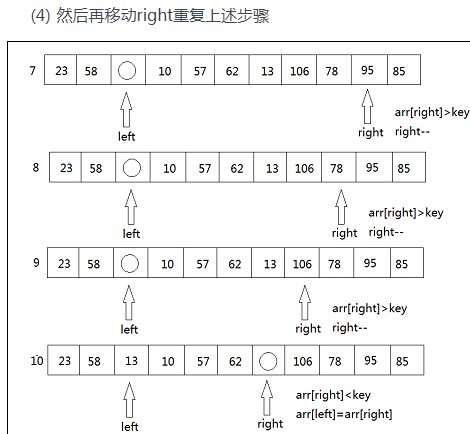
**（2）从头开始的算法**

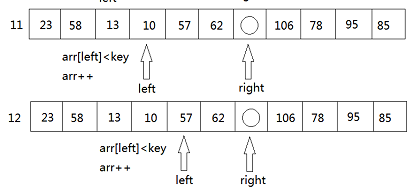
<https://blog.51cto.com/13733462/2113397>

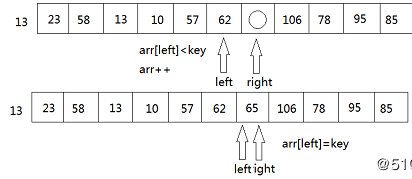














//在快速排序中，切分元素的选取很关键，通常我们可以选取输入数组的第一个元素作为切

//分元素，然后把它交换到数组中的合适位置使得它左边的元素都小于等于它，右边的元素都大于

//等于它，而后对其左右两边的子数组递归执行切分过程，即可完成对整个数组的排序

//切片最后分成了一个元素一个元素的

#include<stdio.h>

int partition**(**int a**[],**int left**,**int right**)**//使得所有a[0..j-1]的元素都小于等于a[j]，

//所有a[j+1..N-1]的元素都大于等于a[j]

**{**

int i**=**left**;**

int j**=**right**;**

int temp**=**a**[**i**];**

**while(**i**<**j**)**

**{**

**while(**i**<**j **&&** a**[**j**]>=**temp**)**

j**--;**

**if(**i**<**j**)**

a**[**i**]=**a**[**j**];**

**while(**i**<**j **&&** a**[**i**]<=**temp**)**

i**++;**

**if(**i**<**j**)**

a**[**j**]=**a**[**i**];**

**}**

a**[**i**]=**temp**;**

**return** i**;**

**}**

void quickSort**(**int a**[],**int left**,**int right**)**

**{**

int dp**;**

**if(**left**<**right**)**

**{**

dp**=**partition**(**a**,**left**,**right**);**//找到那个j值，左边的数小于它，右边的数大于它

quickSort**(**a**,**left**,**dp**-**1**);**

quickSort**(**a**,**dp**+**1**,**right**);**

**}**

**}**

int main**()**

**{**

int a**[**9**]={**5**,**4**,**9**,**1**,**7**,**6**,**2**,**3**,**8**};**

quickSort**(**a**,**0**,**8**);**

**for(**int i**=**0**;**i**<**9**;**i**++)**

**{**

printf**(**"%d "**,**a**[**i**]);**

**}**

**return** 0**;**

**}**

**4 申请和释放内存**

**4.1删除单变量地址空间**

int **\***a **=** **new** int**;**

**delete** a**;** //释放单个int的空间

**4.2.删除数组空间**

int **\***a **=** **new** int**[**5**];**

**delete** **[]**a**;** //释放int数组空间

sizeof 不是函数，然后举出一堆的理由来证明 sizeof 不是函数。在这里，和 sizeof 类似，new 和 delete 也不是函数，它们都是 C++ 定义的关键字，通过特定的语法可以组成表达式。和 sizeof 不同的是，sizeof 在编译时候就可以确定其返回值，new 和 delete 背后的机制则比较复杂。

**4.3 new与delete使用的过程原理**

4.3.1 new与delete配套使用，new int时如果不直接初始化，调用operator new，new string时，由于string具有默认构造函数会调用new直接默认初始化，而不是调用operator new。

4.3.2 new a[]与delete[] a配套使用，new一个数组时会在分配的空间中创建一个空间保存数组的大小，这样删除数组时才知道调用几次析构函数来删除数组元素。

**5 数组和指针的区别**

**5.1 指针与数组的关系：**

当一个指针变量被初始化成数组名时，就说该指针变量指向了数组。

char str[20], \*ptr;

ptr=str;

ptr被置为数组str的第一个元素的地址，因为数组名就是该数组的首地址，也是数组第一个元素的地址。此时可以认为指针ptr就是数组str（反之不成立），这样原来对数组的处理都可以用指针来实现。如对数组元素的访问，既可以用下标变量访问，也可以用指针访问。

**5.2 指向数组元素的指针**

任何由下标完成的操作，都可以用指针来实现

int a**[**10**],** **\***pa**;**

pa**=**a**;**

p**=&**a**[**0**];**

则p=&a[0]是将数组第1个元素的地址赋给了指针变量p。

　　实际上，C语言中数组名就是数组的首地址，所以第一个元素的地址可以用两种方法获得：p=&a[0]或p=a。

这两种方法在形式上相像，其区别在于：pa是指针变量，a是数组名。值得注意的是：pa是一个可以变化的指针变量，而a是一个常数。因为数组一经被说明，数组的地址也就是固定的，因此a是不能变化的，不允许使用a＋＋、＋＋a或语句a＋=10，而pa＋＋、＋＋pa、pa＋=10则是正确的。由此可见，此时指针与数组融为一体。

**5.3** 数组是开辟一块连续的内存空间,数组本身的标示符代表整个数组,可以用sizeof取得真实的大小；指针则是只分配一个指针大小的内存,并可把它的值指向某个有效的内存空间。

**5.4** 把数组作为参数传递的时候

**6 C语言和c++的区别**

**6.1 类：**类对于初学者，它是一个累赘。类的封装使得初学者对程序产生厌倦，感到不适和麻烦。

**6.2 引用：**引用是C++中最好尽量不要用它，除非万不得已。引用对于初学者就更容易产生混淆，不知道哪个是引用，哪个是变量。

**6.3 函数的重载：**初学者学函数的重载好像没什么坏处，但是，这会使初学者潜意识里对C语言的变量类型的重要性产生淡化，要记住C语言是对变量类型最敏感了的，变量的类型在C语言里的重要性是不言而喻的。

**6.4 流操作符：**和上面同样的道理，使得对变量类型的重要性产生淡化，有时会产生使初学者莫名其妙的结果。

**6.5 操作符重载：**典型的高级应用，初学者可能根本用不着，这个东东会让他们觉得C++很难，门槛高，看不懂。

**6.6 继承：**以及虚函数，看起来深奥，实用价值很低。还有些东东我就不发表评论了，如：new,delete操作符等

**6.7 C语言与C++相比的优势：**C是结构性语言，算法结构便于修改，可移植性高，体型小巧，简洁高效并且接近汇编语言，C效率更高，更统一（更符合标准）。代码效率也比较高。

**6.8 C++功能在C的基础上增加面向对象的特点，代码可读性好，运行效率高。**

**7 树莓派与电脑什么通信（SSH连接，无线通信，交叉编译实现的）**

电脑与树莓派在同一个局域网，通过SSH连接，电脑和树莓派就可以通信了。