冒泡排序

两两比较大小，位置互换

void bubble\_sort(int a[], int n)

{

int i, j, temp;

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

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

{

if(a[i] > a[i + 1])

{

temp = a[i];

a[i] = a[i + 1];

a[i + 1] = temp;

}

}

}

int main()

{

int number[SIZE] = {95, 45, 15, 78, 84, 51, 24, 12};

int i;

bubble\_sort(number, SIZE);

for (i = 0; i < SIZE; i++)

{

printf("%d", number[i]);

}

printf("\n");

}

java

**public** **static** void bubbleSort(int[] arr) {

int i, j, temp, len = arr.length;

**for** (i = 0; i < len - 1; i++) {

**for** (j = 0; j < len - 1 - i; j++) {

**if** (arr[j] > arr[j + 1]) {

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

python

def bubble(bubbleList):

listLength = len(bubbleList)

while listLength > 0:

for i in range(listLength - 1):

if bubbleList[i] > bubbleList[i+1]:

bubbleList[i] = bubbleList[i] + bubbleList[i+1]

bubbleList[i+1] = bubbleList[i] - bubbleList[i+1]

bubbleList[i] = bubbleList[i] - bubbleList[i+1]

listLength -= 1

print bubbleList

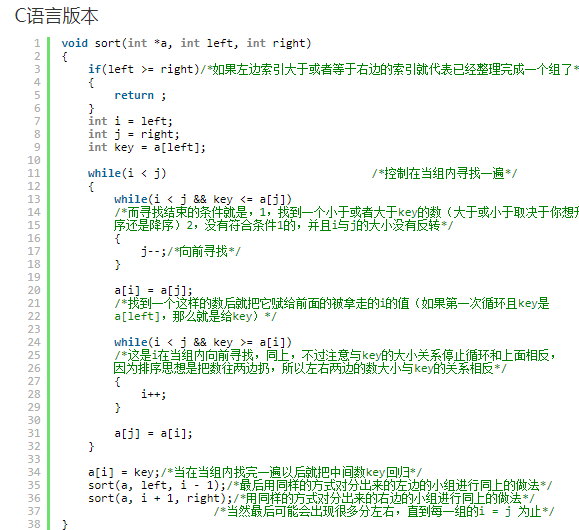
if \_\_name\_\_ == '\_\_main\_\_':

bubbleList = [3, 4, 1, 2, 5, 8, 0]

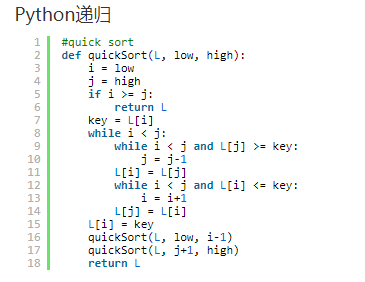
bubble(bubbleList)

快速排序

通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以[递归](https://baike.baidu.com/item/%E9%80%92%E5%BD%92)进行，以此达到整个数据变成有序[序列](https://baike.baidu.com/item/%E5%BA%8F%E5%88%97/1302588)。







**一、插入排序**

**1）算法简介**

        插入排序（Insertion Sort）的算法描述是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间

1、从第一个元素开始，该元素可以认为已经被排序

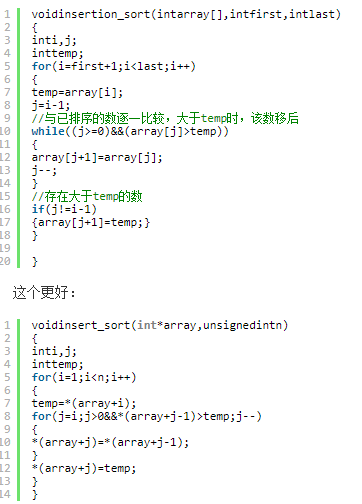
    2、取出下一个元素，在已经排序的元素序列中从后向前扫描

    3、如果该元素（已排序）大于新元素，将该元素移到下一位置

    4、重复步骤3，直到找到已排序的元素小于或者等于新元素的位置

    5、将新元素插入到该位置后

6、重复步骤2~5



二分排序

1）算法简介

       二分（折半）插入（Binary insert sort)排序是一种在直接插入排序算法上进行小改动的排序算法。其与直接排序算法最大的区别在于查找插入位置时使用的是二分查找的方式，在速度上有一定提升。

2）算法描述和分析

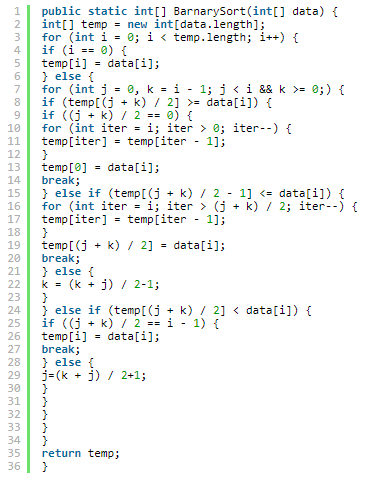
    一般来说，插入排序都采用in-place在数组上实现。具体算法描述如下：

    1、从第一个元素开始，该元素可以认为已经被排序

    2、取出下一个元素，在已经排序的元素序列中二分查找到第一个比它大的数的位置

    3、将新元素插入到该位置后

    4、重复上述两步



# 三、希尔排序

## 1）算法简介

希尔排序，也称递减增量排序算法，因DL．Shell于1959年提出而得名，是插入排序的一种高速而稳定的改进版本。

## 2）算法描述

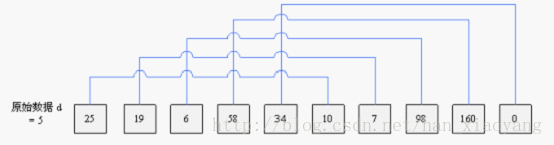
    1、先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。

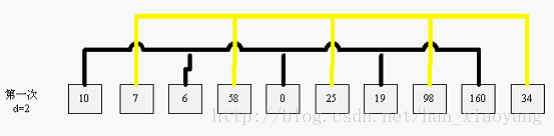
    2、所有距离为d1的倍数的记录放在同一个组中，在各组内进行直接插入排序。

    3、取第二个增量d2<d1重复上述的分组和排序，

    4、直至所取的增量dt=1(dt<dt-l<…<d2<d1)，即所有记录放在同一组中进行直接插入排序为止。

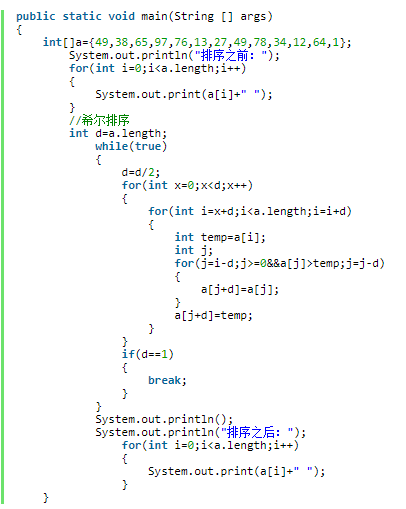
          希尔排序的时间复杂度与增量序列的选取有关，例如希尔增量时间复杂度为O(n^2)，而Hibbard增量的希尔排序的时间复杂度为O(N^(5/4))，但是现今仍然没有人能找出希尔排序的精确下界。







http://img.blog.csdn.net/20130929145839109?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvaGFuX3hpYW95YW5n/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/SouthEast





二分查找

*// while循环*

int binary\_search(**const** int arr[], int start, int end, int khey) {

int mid;

**while** (start <= end) {

mid = start + (end - start) / 2; *//直接平均可能會溢位，所以用此算法*

**if** (arr[mid] < khey)

start = mid + 1;

**else** **if** (arr[mid] > khey)

end = mid - 1;

**else**

**return** mid; *//最後檢測相等是因為多數搜尋狀況不是大於要不就小於*

}

**return** -1;

}