最常用的 8 个排序算法：从原理到改进，再到代码兑现透彻解析

(https://gitbook.cn/gitchat/author/59e86f8281f8273047f174b2)

zhen.guo (https://gitbo…

向作者提问 (https://gitbook.cn/m/mazi/author/59e86f8281f8273047f174b2/question)

工作4年，某知名互联网
公司算法工程师。乐于分享，公众号《Python与机器学习算法频道》已推200篇原创文章。

查看本场Chat
(https://gitbook.cn/gitchat/activity/59faa033edf8562cf5d29c1e)

ed)
        - 2-5-4 归并排序评价 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)

- 3. JAVA中Sort()为什么没选择如下算法？ (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 3-1 希尔排序 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-1-1 希尔排序思想 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-1-2 希尔排序举例 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-1-3 希尔排序评价 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 3-2 冒泡排序 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-2-1 冒泡排序思想 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-2-2 冒泡排序举例 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-2-3 冒泡排序评价 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 3-3 选择排序 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-3-1 选择排序思想 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-3-2 选择排序举例 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-3-3 选择排序评价 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 3-4 堆排序 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-4-1 堆排序思想 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-4-2 堆排序举例 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
        - 3-4-3 堆排序评价 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 3-5 基数排序 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
- 4. 算法兑现 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 4-1 冒泡排序源码 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)
    - 4-2 快速排序源码 (https://gitbook.cn/books/5a099f419bfa9d42d08c81ea/index.html#undefined)

1. 关于排序

很高兴与大家一起探讨计算机科学中的基础算法之排序算法。排序算法是非常基础同时又应用非常广泛的算法，无论在工作还是在生活中，比如：

- 数据库脚本，如MSSql, MySql, NoSql 中按多个关键词的升序或降序排序，例如，学生按照考试分数排名次，分数相等的再按照学号排序。
- 前端界面和后端写服务时经常要调用排序接口。
- 计算机科学很多算法都是基于排序算法的，例如二分查找算法实现逻辑中一般都会先对原序列应用排序操作。
- 还有很多其他应用.....

2. JAVA中Sort()实现解析

在JAVA中，Sort这个排序函数是如何实现的呢？或许我们在工作学习中，对于如此底层的API，很少去想它是怎么实现的，只是拿来使用它。但是，你可能会有兴趣知道它的神秘面纱。现在让我们认识下它，这个Sort函数结合了我们接下来要讨论的几种排序算法。从这个入口了解常用的排序算法，或许是不错的方法，让我们开启研究排序算法的旅程吧！

2-1 实现思路

Java的Sort函数，按照对象是基本类型和引用类型，将排序算法分为两类实现。对于基本类型，排序算法使用了插入排序和快速排序两种算法的结合；而对于对象是引用类型的，主要使用了归并排序算法。

2-2 为什么结合了三种排序算法？

排序算法的选择主要考虑哪些因素？如果想清楚了这个问题，JAVA中按照如上的实现思路也就清楚了。主要考虑如下因素：

- 算法的执行效率
- 排序的稳定性
- 排序元素的个数
- 排序关键码的类型
- 递归调用的开销

首先根据关键码的类型选择排序算法，当为基本类型时，排序实现逻辑如下：

- 待排序的数组中的元素个数小于 7 时，采用插入排序 （不用快排是因为递归开销代价更大）；

- 当待排序的数组中的元素个数大于 或等于7 时，采用快速排序，选择合适的划分元是极为重要的：
    - 当数组大小 `7<size<=40` 时，取首、中、末 三个元素中间大小的元素作为划分元；
    - 当数组大小 `size>40` 时，从待排数组中较均匀的选择9个元素，选出一个伪中数做为划分元。

当为引用类型时，排序实现逻辑如下：

- 采用的是归并排序，为什么，因为对于引用类型排序的稳定性非常重要，而快速排序是无法保证排序的稳定性的，但是归并排序是稳定的排序算法，并且时间复杂也为nlog(n) 。

接下来先介绍JAVA的排序算法中用到的这三个排序：插入排序，快速排序，归并排序。

2-3 插入排序

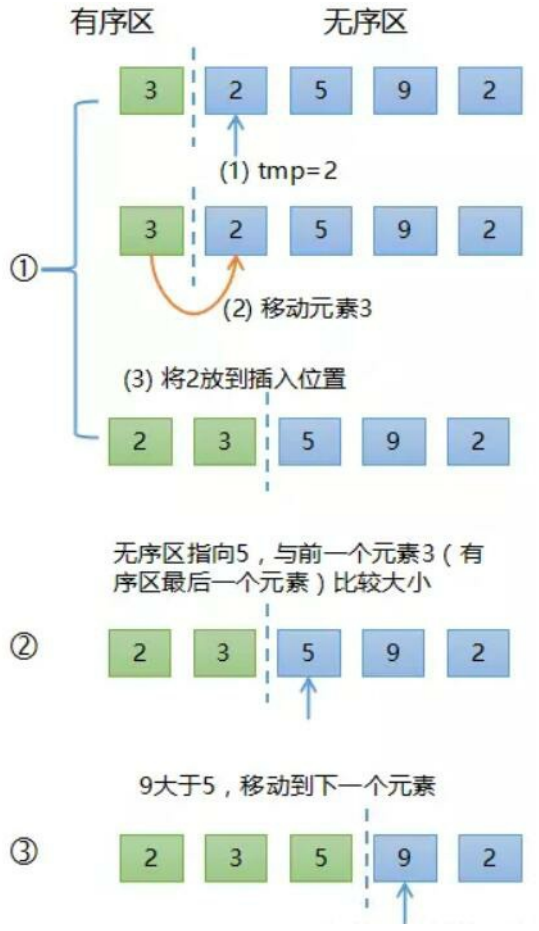
直接插入排序，英文名称 straight insertion sort，它是一种依次将无序区的元素在有序区内找到合适位置依次插入的算法。

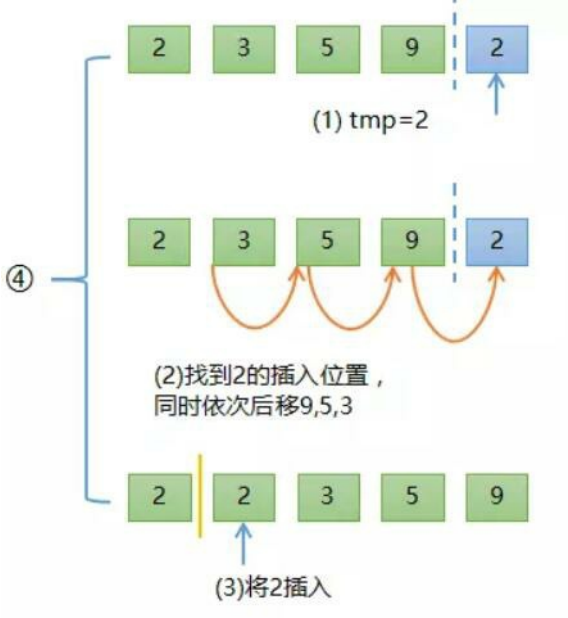
2-3-1 基本思想

每次从无序表中取出第一个元素，把它插入到有序表的合适位置，使有序表仍然有序，直到无序表内所有元素插入为止。首先在当前有序区R[0..i-1]中查找R[i]的正确插入位置 k(0≤k≤i-1)；然后将R[k．．i-1]中的记录均后移一个位置，腾出 k 位置上的空间插入R[i]。

2-3-2 插入排序举例

用待排序列 **3 2 5 9 2** ，演示直接插入排序的过程，至此结束插入排序的过程，可以看到直接插入排序共经过4轮的操作。





2-3-3 插入排序评价

插入排序的最坏时间复杂度为 O(n^2)，属于稳定排序算法，对于处理小批量数据时高效，因此JAVA在排序元素个数小于7时，选择了这种算法。

2-4 快速排序

快速排序（Quicksort）是对冒泡排序的一种改进。

2-4-1 基本思想

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其中一部分数据进行快速排序，这是递归调用。再按此方法对另一部分数据进行快速排序，这也是递归调用。

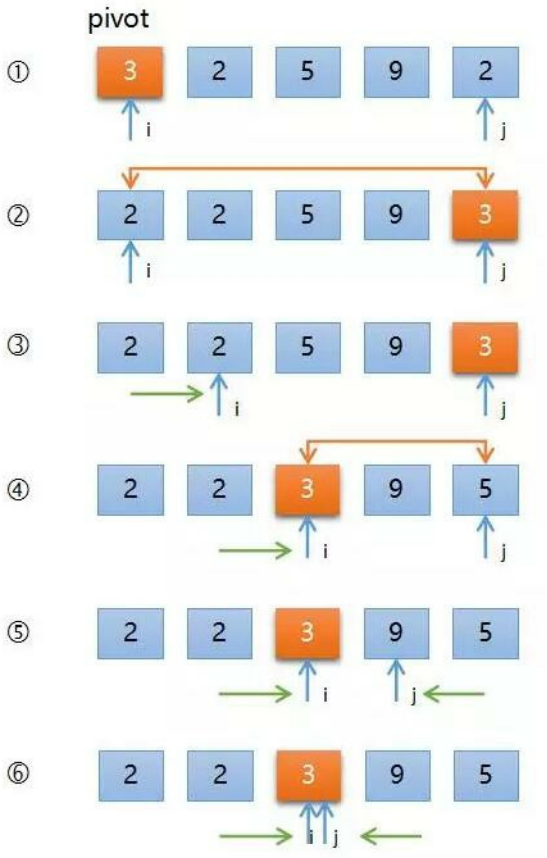
2-4-2 算法介绍

设要排序的数组是A[0]……A[n-1]，首先任意选取一个数据作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。一趟快速排序算法所完成的工作：

1. 设置两个变量 i、j，排序开始的时候：i=0，j=n-1；
2. 以第一个数组元素作为关键数据，赋值给 pivot，即 pivot =A[0]；
3. 从 j 开始向前搜索，即由后开始向前搜索(j--)，找到第一个小于 pivot 的值A[j]，将 A[j] 和 A[i] 互换（互换保证了 A[j] < A[i]，也就是保证了要趋向于前方的关键码都小于 pivot)；
4. 从 i 开始向后搜索，即由前开始向后搜索(i++)，找到第一个大于pivot 的A[i]，将 A[i] 和 A[j] 互换（互换保证了 A[j] < A[i]，也就是保证了后方的关键码都大于 pivot ） 重复第3、4步，直到 i=j 。
5. 完成本轮比较

2-4-3 快速排序例子

假设待排序的序列仍为：**3 2 5 9 2** 。第一轮比较，选取第一个关键码 3 为pivot，初始值 i =0, j =4，整个的比较过程如下图所示：



2-4-4 快速排序评价

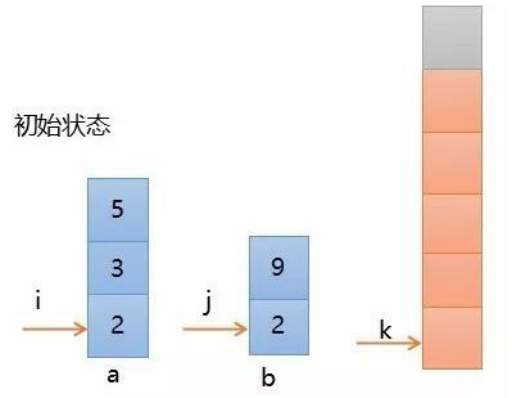
快速排序的最坏时间复杂度为 O(n^2)，这是一种退化的情况，在大多数情况下只要选取了合适的划分元后，时间复杂度为 nlog(n)，快速排序通常比其他 O (n log n) 算法更快，属于非稳定排序算法。

2-5 归并排序

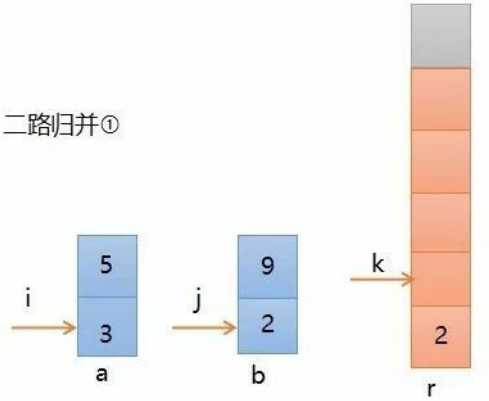
归并排序，英文名称是MERGE-SORT。它是建立在归并操作上的一种有效的排序算法，该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。

2-5-1 二路归并

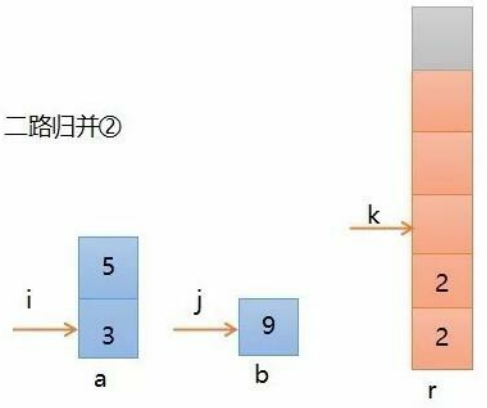
比较 a[i] 和 b[j] 的大小，若 `a[i]≤b[j]`，则将第一个有序表中的元素 a[i] 复制到 r[k] 中，并令 i 和 k 分别加上1；否则将第二个有序表中的元素b[j]复制到 r[k] 中，并令 j 和 k 分别加上1；如此循环下去，直到其中一个有序表取完；然后再将另一个有序表中剩余的元素复制到 r 中从下标 k 到下标 t 的单元。 这个过程，请见下面的例子演示。



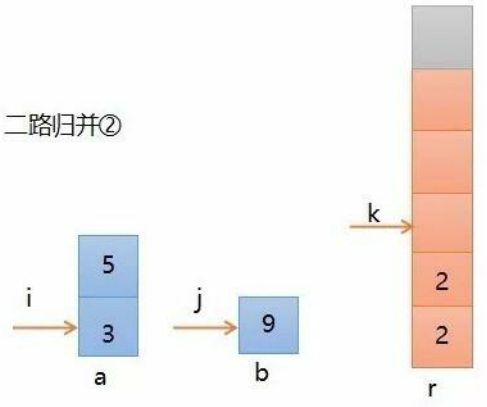
二路归并①



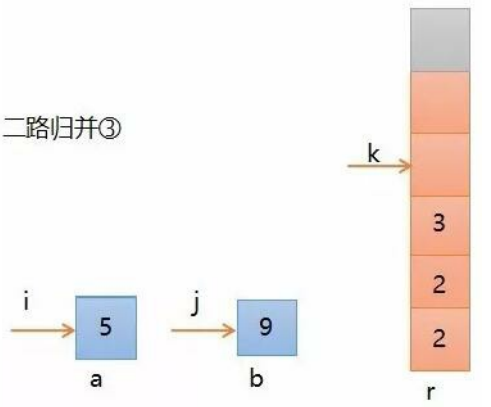
二路归并②



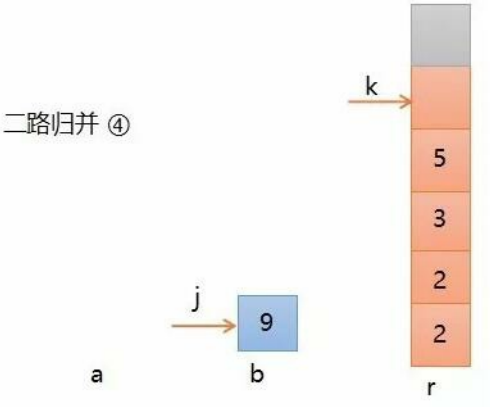
二路归并②

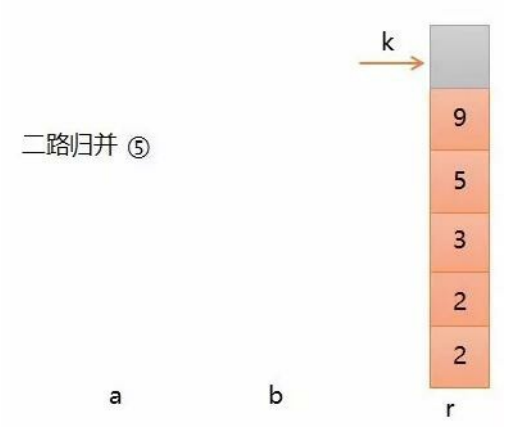


二路归并③



二路归并 ④





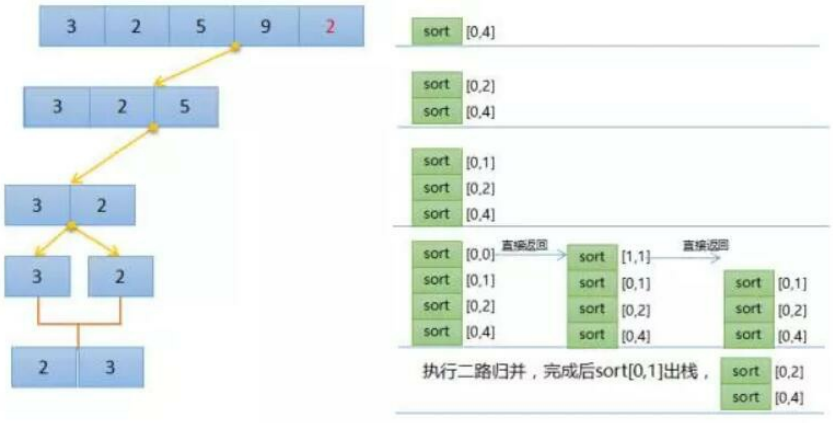
2-5-2 归并算法

归并排序的算法我们通常用递归实现。先把待排序区间 [s,t] 以中点二分；接着把左边子区间排序；再把右边子区间排序；最后把左区间和右区间用一次归并操作合并成有序的区间 [s,t] 。

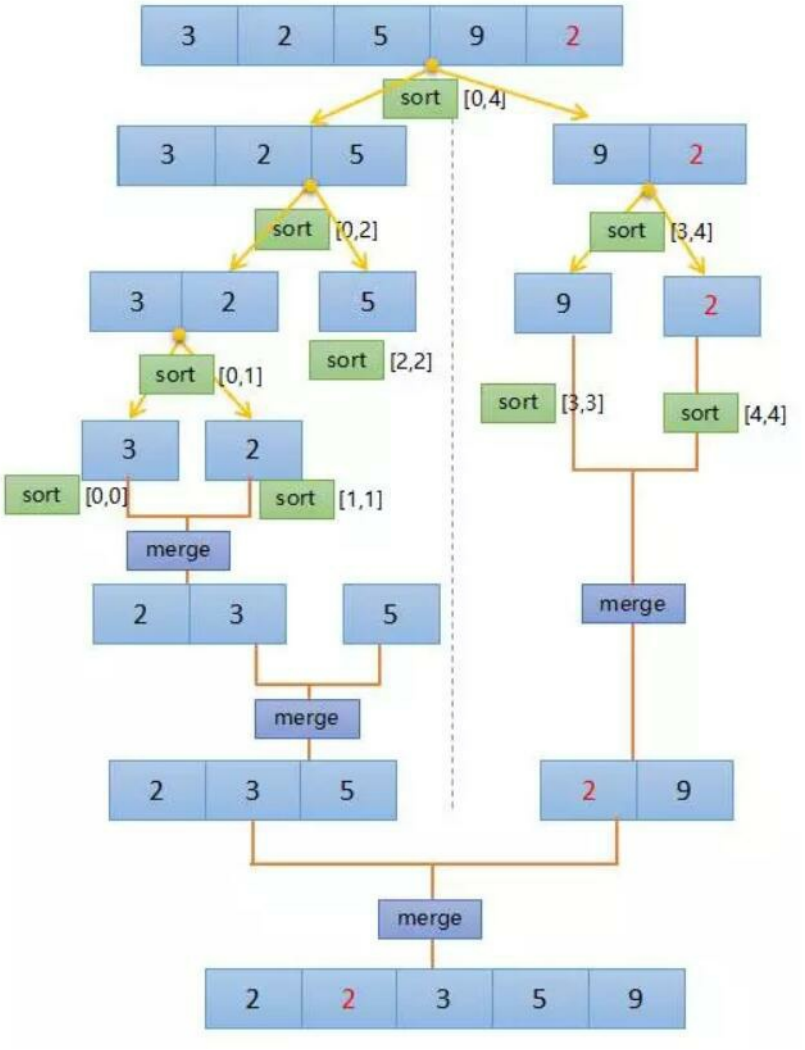
2-5-3 递归过程

待排序列 **3 2 5 9 2** 。下图演示的是归并排序递归版，第一次执行二路归并时的示意图，注意观察右图的栈的入栈顺序。

可以看到 sort 的入栈顺序，当执行一次 merge 时，一定是有2个sort返回并有序了，如下图，sort[0,0]和sort[1,1]（递归返回的条件是 `start<end` ）都返回了，然后执行到merge，执行完 merge 后，sort[0,1]出栈，此时的栈顶为 sort[0,2] 函数，可以看出它的前半部分已经计算完，只需要计算后半部分，即第二个 sort，然后再次merge，再 sort[0,2] 出栈。。。



如下为上个例子的归并排序的完整示例，sort 和 merge 的示意图，可以看到最后一次merge，正是上面说到的二路 [2,3,5] 和 [2,9] 的归并排序，如果不熟悉的，可以回过头再看看。



2-5-4 归并排序评价

归并排序的最坏时间复杂度为O(nlogn)，是一种稳定排序算法。

3. JAVA中Sort()为什么没选择如下算法？

以上我们介绍了JAVA中Sort()的主要实现逻辑，那么为什么没有引用其他常见的排序算法呢？像希尔排序算法，冒泡排序，选择排序和堆排序呢？下面我们试着找找原因。

3-1 希尔排序

缩小增量排序，是以上介绍的插入排序算法的一种更高效的改进版本，可以看做是分组版插入排序算法。

3-1-1 希尔排序思想

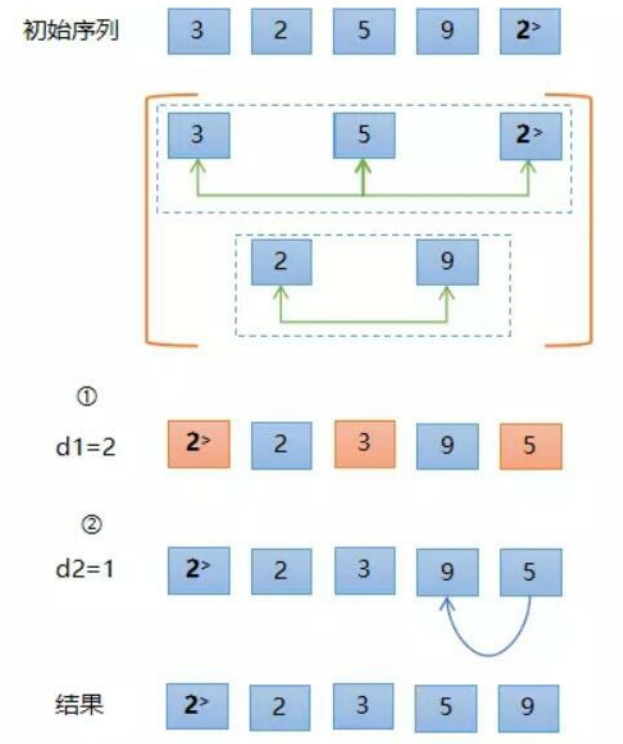
先取一个正整数 `d1<n`，把所有序号相隔 d1 的数组元素放一组，组内进行直接插入排序，然后取 `d2< d1`，重复上述分组和直接插入排序操作；直至 di = 1，即所有记录放进一个组中排序为止。

3-1-2 希尔排序举例

仍然用待排序列 **3 2 5 9 2** 。在第一轮中，选取增量为2，即分为两组，第一组为 [3 5 2]，另一组为 [2 9]，分别对这两组做直接插入排序，第一组插入排序的结果为[2 3 5]，第二组不动，这样导致的直接结果是原来位于最后的2经过第一轮插入排序后，跑到最头里了，这样两个2的相对位置发生改变了，所以希尔排序不是稳定的排序算法。

再经过第二轮排序，此时的增量为1，所以一共只有一组了，相当于直接插入排序，9后移1步，5插入到原9的位置。

这样整个的希尔排序结束，得到如下图所示的非降序序列。



3-1-3 希尔排序评价

希尔排序的最坏时间复杂度为 `O(n^2)`， 对中等大小规模表现良好，但对规模非常大的数据排序不是最优选择，并且如上所述希尔排序不是稳定的排序算法，所以JAVA弃用它也是再所难免的。

3-2 冒泡排序

JAVA中使用的快排是在冒泡排序的基础上的改进，而冒泡排序一般都是我们最先接触的排序算法，英文名称是 bubble sort 。

3-2-1 冒泡排序思想

已知一组无序数据a[0]、a[1]、……a[n-1]，需将其用冒泡排序按升序排列。

首先比较a[0]与a[1]的值，若a[0]大于a[1]则交换两者的值，否则不变，以此类推，最后比较a[n-2]与a[n-1]的值。这样处理一轮后，a[n-1]的值一定是这组数据中最大的。

再对a[0]~a[n-2]以相同方法处理一轮，则a[n-2]的值一定是a[0]~a[n-2]中最大的。

以此类推，这样共处理 n-1 轮后a[0]、a[1]、……a[n-1]就以升序排列了。

3-2-2 冒泡排序举例

待排序列 **3 2 5 9 2**

- 第一轮
  - 第1次比较 2 3 5 9 2
  - 第2次比较 2 3 5 9 2
  - 第3次比较 2 3 5 9 2
  - 第4次比较 2 3 5 2 | 9
- 第二轮
  - 第5次比较 2 3 5 2 9
  - 第6次比较 2 3 5 2 9
- 第7次比较 2 3 2 | 5 9
- 第三轮

- 第8次比较 2 3 2 5 9
- 第9次比较 2 2 | 3 5 9

- 第四轮
  - 第10次比较 2 | 2 3 5 9

3-2-3 冒泡排序评价

算法的时间复杂度为 O(n^2)，对于大批量数据处理效率低，所以JAVA弃用也是再所难免，是稳定的排序算法。

3-3 选择排序

直接选择排序，英文名称：Straight Select Sorting，是一个直接从未排序序列选择最值到已排序序列的过程。

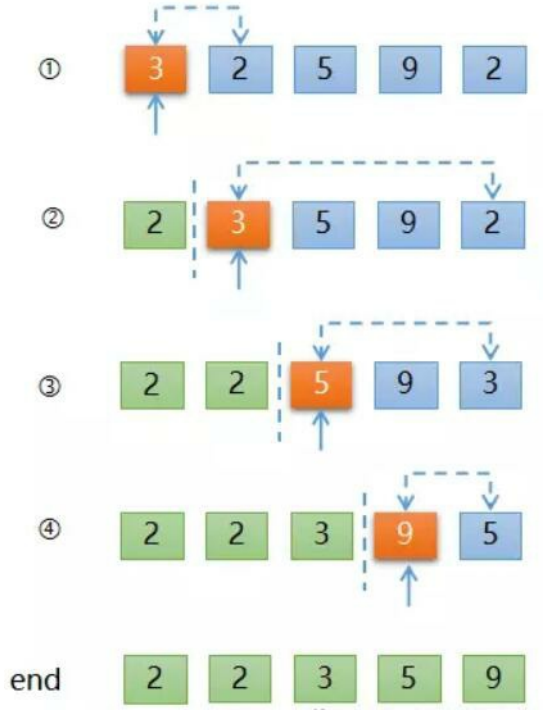
3-3-1 选择排序思想

- 第一次从R[0]~R[n-1]中选取最小值，与R[0]交换；
- 第二次从R[1]~R[n-1]中选取最小值，与R[1]交换，....，
- 第 i 次从R[i-1]~R[n-1]中选取最小值，与R[i-1]交换，.....，
- 总共通过n-1次，得到一个按关键码从小到大排列的有序序列。

3-3-2 选择排序举例

待排序列 3 2 5 9 2。演示如何用直接选择排序得到升序序列。

- 第一轮，从所有关键码中选择最小值与 R[0]交换，3与2交换，如下图所示，
- 第二轮，从 R[1]~R[n-1]中选择最小值与R[1]交换，3与2交换；
- 第三轮，从 R[2]~R[n-1]中选择最小值与R[2]交换，5与3交换；
- 第四轮，从 R[3]~R[n-1]中选择最小值与R[3]交换，9与5交换；
- 终止。



3-3-3 选择排序评价

直接选择排序的最坏时间复杂度为 O(n^2)，效率低也是JAVA不使用它的原因，与处理小批量数据时，直接选择排序所需要的比较次数也比直接插入排序多。它是稳定排序算法。

3-4 堆排序

堆排序，英文名称 Heapsort，利用二叉树（堆）这种数据结构所设计的一种排序算法，是一种对直接选择排序的一种改建算法。在逻辑结构上是按照二叉树存储结构，正是这种结构优化了选择排序的性能，在物理存储上是连续的数组存储，它利用了数组的特点快速定位指定索引的元素。

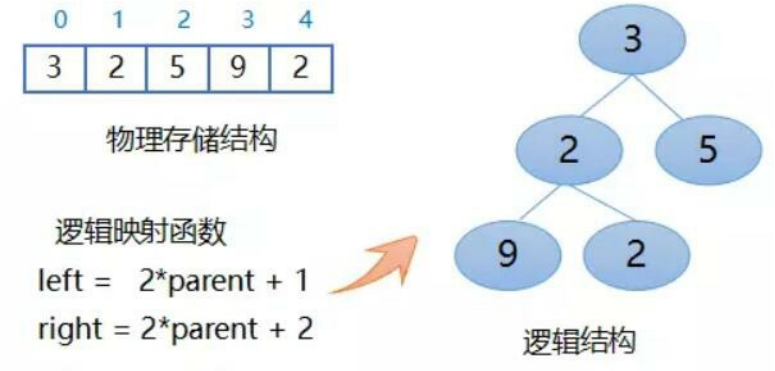
3-4-1 堆排序思想

以大根堆排序为例，即要得到非降序序列：

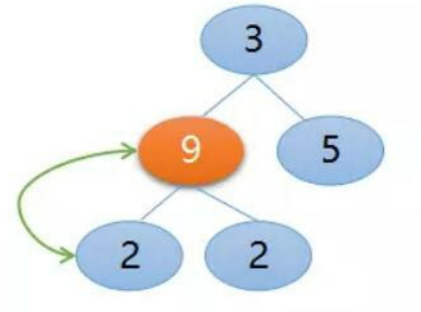
- 先将初始文件R[0..n-1]建成一个大根堆，此堆为初始的无序区。
- 再将关键字最大的记录R[0]（即堆顶）和无序区的最后一个记录R[n-1]交换，由此得到新的无序区 R[0..n-2] 和有序区 R[n-1]，且满足 R[0..n-2] ≤ R[n-1] 。由于交换后新的根R[0]可能违反堆性质，故应将当前无序区R[0..n-2]调整为堆。
- 然后再次将R[0..n-2]中关键字最大的记录R[0]和该区间的最后一个记录R[n-2]交换，由此得到新的无序区R[0..n-3] 和 有序区R[n-2..n-1]，且仍满足关系R[0..n-3] ≤ R[n-2..n-1]。
- 重复步骤2和步骤3，直到无序区只有一个元素为止。

3-4-2 堆排序举例

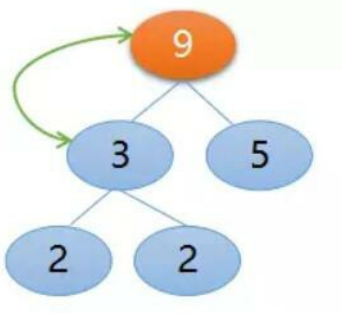
待排序列 **3 2 5 9 2** 。第一步，首先以上待排序列的物理存储结构和逻辑存储结构的示意图如下所示：



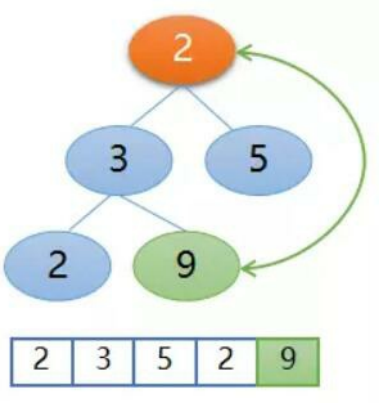
构建初始堆是从length/2 - 1，即从索引1处关键码等于2开始构建，2的左右孩子等于9, 2，它们三个比较后，父节点2与左孩子9交换，如下图所示：



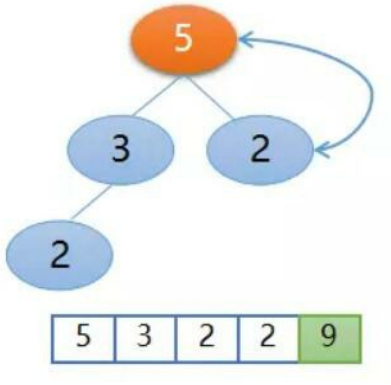
接下来从索引1减1等于0处，即元素3开始与其左右孩子比较，比较后父节点3与左孩子节点9交换，如下所示：



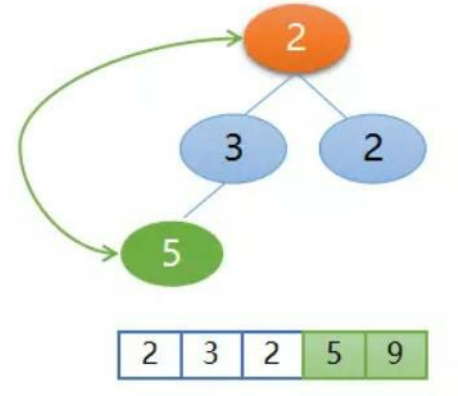
因为索引等于 0 了，所以构建堆结束，得到大根堆，第一步工作结束，下面开始第二步调整堆，也就是不断地交换堆顶节点和未排序区的最后一个元素，然后再构建大根堆，下面开始这步操作，交换栈顶元素9（如上图所示）和未排序区的最后一个元素2，如下图所示，现在排序区9成为了第一个归位的。



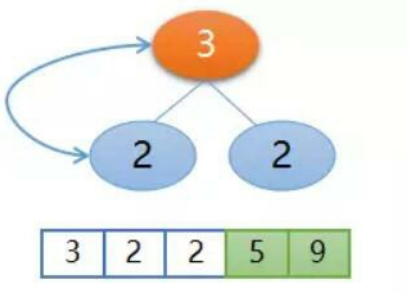
接下来拿掉元素9，未排序区变成了2,3,5,2，然后从堆顶2开始进行堆的再构建，比较父节点2与左右子节点3和5，父节点2和右孩子5交换位置，如下图所示，这样就再次得到了大根堆。



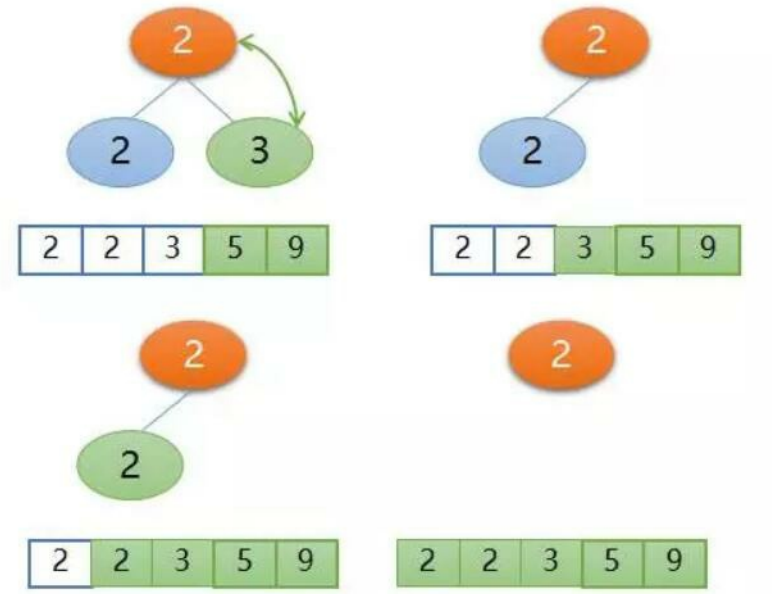
再交换堆顶5和未排序区的最后一个元素2，这样5又就位了，这样未排序区变为了2,3,2，已排序区为 5,9，交换后的位置又破坏了大根堆，已经不再是大根堆了，如下图所示：



所以需要再次调整，然后堆顶2和左孩子3交换，交换后的位置如下图所示，这样二叉树又重新变为了大根堆，再把堆顶3和此时最后一个元素也就是右孩子2交换。



接下来再构建堆，不再赘述，见下图：



3-4-3 堆排序评价

堆排序的时间，主要由建立初始堆和反复重建堆这两部分的时间开销构成，堆排序的最坏时间复杂度是O(nlogn)，堆排序是不稳定的排序方法。由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的排序序列。

为了防止有人使用精心构造的数据来攻击排序算法，有些框架的排序算法会采取先快速排序，如果发现明显退化迹象，则回退到堆排序这样的时间复杂度稳定的排序上，所以堆排序对基本数据类型排序也是很有用的。

3-5 基数排序

在此由于篇幅问题，不再详细介绍，请参考我在微信公众号中的介绍：基数排序 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483799&idx=1&sn=46ace682e1c9ed14ae9caee120c6d368&chksm=eb7c2c5cdc0ba54a7b0a84485f5d006a6cb084a60905e85704239cf858c80e51058e88f8b1a4&scene=21#wechat_redirect)

4. 算法兑现

当我们详细研究了这些常用排序算法的基本实现原理之后，是时候写出这些排序算法的源代码了，也许这些代码在网上有更高效的实现，不过下面写的这些都是和之前说的算法原理和图都解密切相关，一一对应的，主要是方便大家的理解。

几个算法中使用的一个交换函数，源码如下，

```
//swap element at i to at j
private static void swap(int[] array, int i,int j){
   int tmp = array[i];
   array[i] = array[j];
   array[j] = tmp;
}
```

以下排序算法都实现了序列的非降序排列，函数参数代表的含义一般统一定义为：

- array： 待排序的数组，类型为一维整形数组
- n：元素个数
- i：一般为外层循环索引，或表示排序区或未排序的开始或结束索引
- j：一般为内层循环索引，或表示未排序区或排序的结束或开始索引
- lo：数组计算区间的开始索引
- hi：数组计算区间的结束索引
- d：分组长度
- k：分组索引

4-1 冒泡排序源码

冒泡排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

4-2 快速排序源码

快速排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

4-3 插入排序源码

插入排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

4-4 希尔排序源码

希尔排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

4-5 选择排序源码

选择排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

4-6 堆排序源码

堆排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

下面列出堆排序的实现代码。注意大根堆顶与未排序区的最后一个元素不断交换，直至未排序区的个数为0，整个序列完成排序。

堆排序算法比较容易出错的点：

构建堆函数，左右子节点可能都有，也可能只含有左节点；

堆排序函数，while遍历时，buildHeap参数中元素个数每次减1，始终从位置0（堆顶）开始调整。

```
    //heap sort
    public static void heapSort(int[] array, int n){
        for (int i = n / 2 - 1; i >= 0; i--)
            buildHeap(array, n, i);
        int len = n - 1;
        while (len > 0) {
            swap(array, 0, len);
            buildHeap(array, len, 0);
            len--;
        }
    }
    private static void buildHeap(int[] array, int n, int i){
        for (; left(i) < n; i = left(i)) {
            int bigger =
                right(i) < n ? max(array, left(i), right(i)) : left(i);
            if (array[bigger] > array[i]) //swap
                swap(array, bigger, i);
            else break;
        }
    }

private static int left(int i){
    return 2*i+1;
}
private static int right(int i){
    return 2*i+2;
}
private static int max(int[] array, int i, int j){
    return array[i]>array[j]? i:j;
}
```

4-7 归并排序源码

归并排序源码 (https://mp.weixin.qq.com/s?__biz=Mzl3NTkyMjA4NA==&mid=2247483808&idx=1&sn=9d90856f16a20b4a1846eb1680334a4e&chksm=eb7c2c6bdc0ba57dfeb93f78b65952019f12e6371b763bb73f343485c1549e45c8173778ee5a&scene=21#wechat_redirect)

5. 外部排序

以上阐述的这些都是内部排序, 指的是待排序记录存放在计算机存储器中进行的排序过程。但是，另一类是外部排序, 指的是待排序记录的数量很大，以至于内存一次不能容纳全部记录，在排序过程中尚需对外存进行访问的排序过程。

5-1 多路归并排序

外部排序最常用的算法是多路归并排序，即将原文件分解成多个能够一次性装入内存的部分分别把每一部分调入内存完成排序。然后，对已经排序的子文件进行归并排序。

由于受话题篇幅长度的限制，我们在此不再展开对外部排序的讨论，以后有时间，我们再一起交流。

6. 致谢

以上是我个人对内部排序算法的一些理解，如有分析不准确之处，还请大家多包涵，谢谢大家的参与和讨论！

写评论

向作者提问 (https://gitbook.cn/m/mazi/author/59e86f8281f8273047f17

佳殷

哈哈哈 厉害了 继续努力啊～～

2017年11月2日 1 1

zhen.guo: 谢谢鼓励！我会继续努力。。。

↑□ 小霞

不错 不错！

2017年11月2日 1 2

↑□ 小霞: 期待分享哈！👏👏👏

zhen.guo: 谢谢

TTTTTUSKIIIIIII

看算法第四版辅以本文效果会更好

2017年12月2日 1 1

zhen.guo: 嗯，谢谢，欢迎留言交流。

。。

选择排序是不稳定的算法。。。

3月16日 0 1

zhen.guo: 选择排序属于稳定排序吧，你是怎么理解呢？ 欢迎交流

晴天

直接插入排序，第二个while条件是有问题的。

7月20日 0 0