排序

1. 概述
2. 交换排序冒泡排序 Bubble Sort

3. 快速排序

4. 插入排序-直接插入排序

5. 插入排序-希尔排序

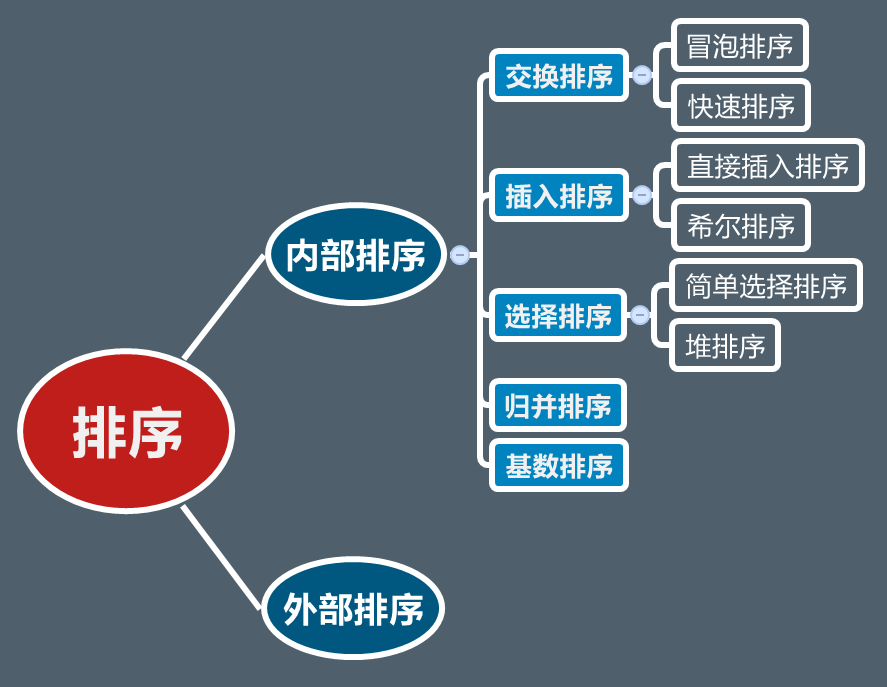
6. 选择排序-简单选择排序

7. 选择排序-堆排序

8. 归并排序

# 概述

排序有内部排序和外部排序，内部排序是数据记录在内存中进行排序，而外部排序是因排序的数据很大，一次不能容纳全部的排序记录，在排序过程中需要访问外存。



# 交换排序-冒泡排序

冒泡排序是一种交换排序。

什么是交换排序呢？

交换排序：两两比较待排序的关键字，并交换不满足次序要求的那对数，直到整个表都满足次序要求为止。

## 基本思想

在要排序的一组数中，对当前还未排好序的范围内的全部数，自上而下对相邻的两个数依次进行比较和调整，让较大的数往下沉，较小的往上冒。即：每当两相邻的数比较后发现它们的排序与排序要求相反时，就将它们互换

## 排序流程

1. 比较相邻的元素，如果前一个比后一个大，就把它们两个调换位置。

2. 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对。这步做完后，最后的元素会是最大的数。

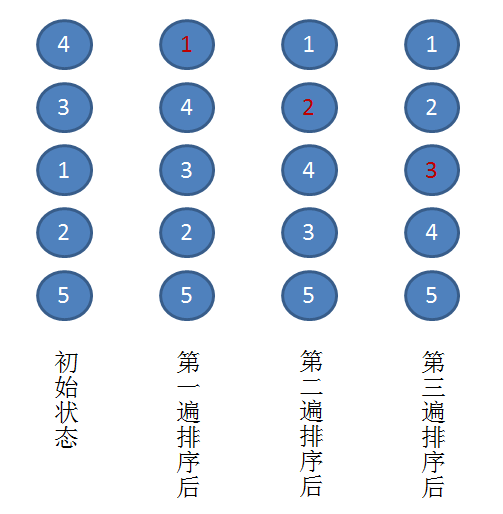
3. 针对所有的元素重复以上的步骤，除了最后一个。

4. 持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较

假设有一个大小为 N 的无序序列。冒泡排序就是要每趟排序过程中通过两两比较，找到第 i 个小（大）的元素，将其往上排。

## 示例图

**图-冒泡排序示例图**



以上图为例，演示一下冒泡排序的实际流程：

假设有一个无序序列 { 4. 3. 1. 2, 5 }

第一趟排序：通过两两比较，找到第一小的数值 1 ，将其放在序列的第一位。

第二趟排序：通过两两比较，找到第二小的数值 2 ，将其放在序列的第二位。

第三趟排序：通过两两比较，找到第三小的数值 3 ，将其放在序列的第三位。

至此，所有元素已经有序，排序结束。

要将以上流程转化为代码，我们需要像机器一样去思考，不然编译器可看不懂。

假设要对一个大小为 N 的无序序列进行升序排序（即从小到大）。

(1) 每趟排序过程中需要通过比较找到第 i 个小的元素。

所以，我们需要一个外部循环，从数组首端(下标 0) 开始，一直扫描到倒数第二个元素（即下标 N - 2) ，剩下最后一个元素，必然为最大。

(2) 假设是第 i 趟排序，可知，前 i-1 个元素已经有序。现在要找第 i 个元素，只需从数组末端开始，扫描到第 i 个元素，将它们两两比较即可。

所以，需要一个内部循环，从数组末端开始（下标 N - 1），扫描到 (下标 i + 1)。

## 稳定性

冒泡排序就是把小的元素往前调或者把大的元素往后调。比较是相邻的两个元素比较，交换也发生在这两个元素之间。

所以相同元素的前后顺序并没有改变，所以冒泡排序是一种稳定排序算法

## 代码实现

# 交换排序-快速排序

快速排序是一种交换排序

## 基本思想

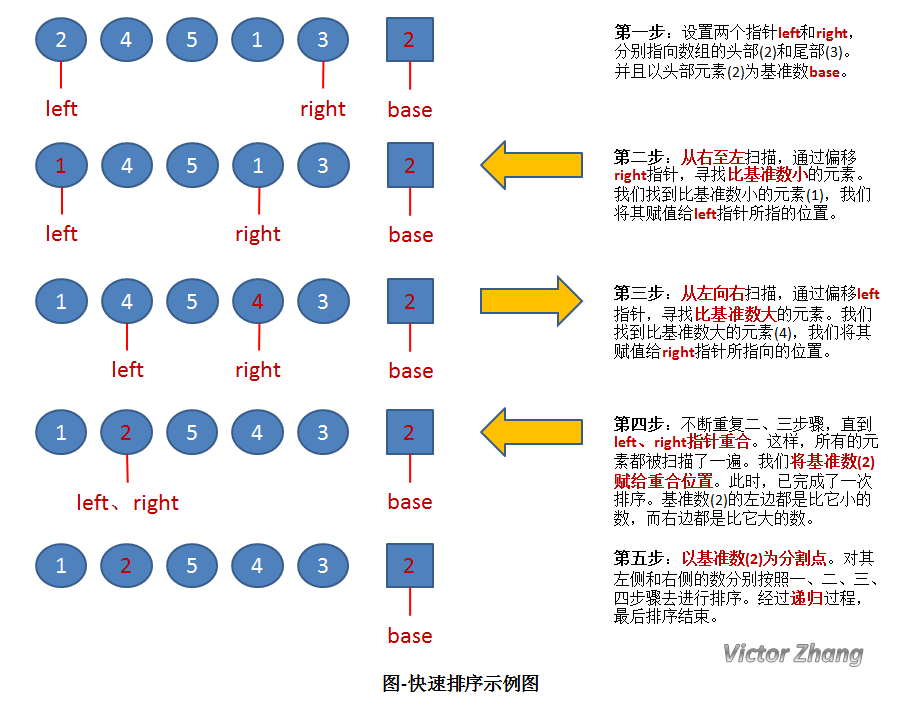
1）选择一个基准元素,通常选择第一个元素或者最后一个元素,

2）通过一趟排序将待排序的记录分割成独立的两部分，其中一部分记录的元素值均比基准元素值小。另一部分记录的 元素值比基准值大。

3）此时基准元素在其排好序后的正确位置

4）然后分别对这两部分记录用同样的方法继续进行排序，直到整个序列有序。

## 示例图



上图中，演示了快速排序的处理过程：

初始状态为一组无序的数组：2、4、5、1、3。

经过以上操作步骤后，完成了第一次的排序，得到新的数组：1、2、5、4、3。

新的数组中，以2为分割点，左边都是比2小的数，右边都是比2大的数。

因为2已经在数组中找到了合适的位置，所以不用再动。

2左边的数组只有一个元素1，所以显然不用再排序，位置也被确定。（注：这种情况时，left指针和right指针显然是重合的。因此在代码中，我们可以通过设置判定条件left必须小于right，如果不满足，则不用排序了）。

而对于2右边的数组5、4、3，设置left指向5，right指向3，开始继续重复图中的一、二、三、四步骤，对新的数组进行排序。

## 稳定性

在快速排序中，相等元素可能会因为分区而交换顺序，所以它是不稳定的算法。

## 代码实现

# 插入排序-直接插入排序

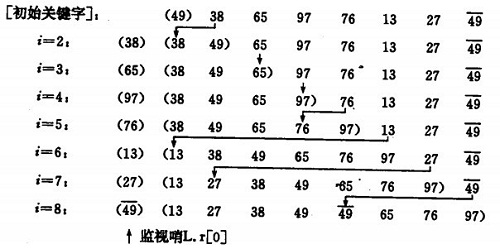
## 基本思想

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。

第一趟比较前两个数，然后把第二个数按大小插入到有序表中； 第二趟把第三个数据与前两个数从后向前扫描，把第三个数按大小插入到有序表中；依次进行下去，进行了(n-1)趟扫描以后就完成了整个排序过程。

直接插入排序是由两层嵌套循环组成的。外层循环标识并决定待比较的数值。内层循环为待比较数值确定其最终位置。直接插入排序是将待比较的数值与它的前一个数值进行比较，所以外层循环是从第二个数值开始的。当前一数值比待比较数值大的情况下继续循环比较，直到找到比待比较数值小的并将待比较数值置入其后一位置，结束该次循环

要点：设立哨兵，作为临时存储和判断数组边界之用。



## 稳定性

直接插入排序的过程中，不需要改变相等数值元素的位置，所以它是稳定的算法。

## 代码实现

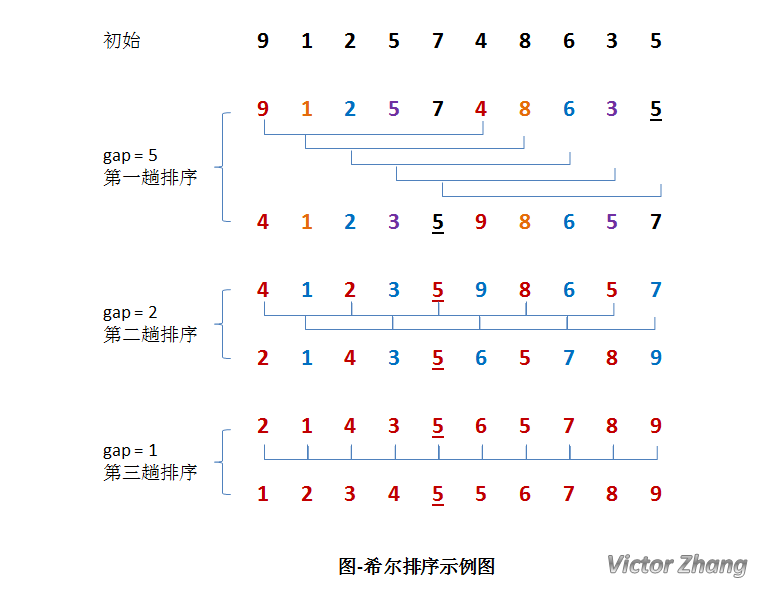
# 插入排序-希尔排序

希尔(Shell)排序又称为缩小增量排序，它是一种插入排序。它是直接插入排序算法的一种加强版

## 基本思想

希尔排序是指记录按下标的一定增量分组，对每一组使用 直接插入排序 ，随着增量逐渐减少，每组包含的关键字越来越多，当增量减少至 1 时，整个序列恰好被分成一组，算法便终止。

## 示例图



在上面这幅图中：

初始时，有一个大小为 n = 10 的无序序列。

在第一趟排序中，我们假设 gap1 = N / 2 = 5，即相隔距离为 5 的元素组成一组，可以分为 5 组。

接下来，按照直接插入排序的方法对每个组进行排序。

在第二趟排序中，我们把上次的 gap 缩小一半，即 gap2 = gap1 / 2 = 2 (取整数)。这样每相隔距离为 2 的元素组成一组，可以分为 2 组。

按照直接插入排序的方法对每个组进行排序。

在第三趟排序中，再次把 gap 缩小一半，即gap3 = gap2 / 2 = 1。 这样相隔距离为 1 的元素组成一组，即只有一组。

按照直接插入排序的方法对每个组进行排序。此时，排序已经结束。

需要注意一下的是，图中有两个相等数值的元素 5 和 5 。我们可以清楚的看到，在排序过程中，两个元素位置交换了。

所以，希尔排序是不稳定的算法。

## 算法稳定性

由上文的希尔排序算法演示图即可知，希尔排序中相等数据可能会交换位置，所以希尔排序是不稳定的算法。

## 直接插入排序和希尔排序的比较

直接插入排序是稳定的；而希尔排序是不稳定的。

直接插入排序更适合于原始记录基本有序的集合。

希尔排序的比较次数和移动次数都要比直接插入排序少，当N越大时，效果越明显。

在希尔排序中，增量序列gap的取法必须满足：最后一个步长必须是 1 。

直接插入排序也适用于链式存储结构；希尔排序不适用于链式结构。

## 代码实现

# 简单选择排序

每趟从待排序的记录中选出关键字最小的记录，顺序放在已排序的记录序列末尾，直到全部排序结束为止

## 排序流程

(1)从待排序序列中，找到关键字最小的元素；

(2)如果最小元素不是待排序序列的第一个元素，将其和第一个元素互换；

(3)从余下的 N - 1 个元素中，找出关键字最小的元素，重复(1)、(2)步，直到排序结束。

## 示意图



如图所示，每趟排序中，将当前第 i 小的元素放在位置 i 上。

## 稳定性

不稳定的算法

## 代码实现

# 堆排序

在介绍堆排序之前，首先需要说明一下，堆是个什么玩意儿。

堆是一棵顺序存储的完全二叉树。

其中每个结点的关键字都不大于其孩子结点的关键字，这样的堆称为小根堆。

其中每个结点的关键字都不小于其孩子结点的关键字，这样的堆称为大根堆。

举例来说，对于n个元素的序列{R0, R1, ... , Rn}当且仅当满足下列关系之一时，称之为堆：

(1) Ri <= R2i+1 且 Ri <= R2i+2 (小根堆)

(2) Ri >= R2i+1 且 Ri >= R2i+2 (大根堆)

其中i=1,2,…,n/2向下取整;

## 基本思想

（1）根据初始数组去构造初始堆（构建一个完全二叉树，保证所有的父结点都比它的孩子结点数值大）。

（2）每次交换第一个和最后一个元素，输出最后一个元素（最大值），然后把剩下元素重新调整为大根堆。

当输出完最后一个元素后，这个数组已经是按照从小到大的顺序排列了。

**堆排序分成三块：**

**第一块,什么是堆,什么是最大堆**

**第二块,怎么将堆调整为最大堆,这部分是重点**

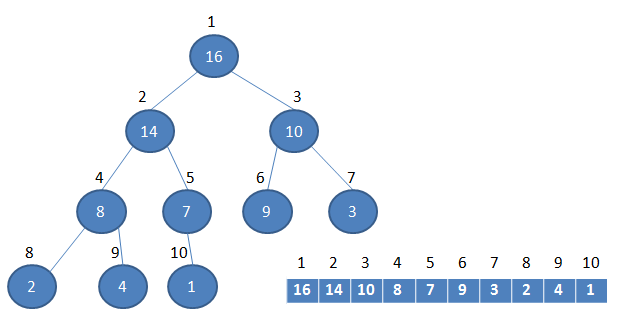
**第三块,堆排序介绍**

## ****什么是堆****

这里的堆(二叉堆),指得不是堆栈的那个堆,而是一种数据结构。

堆可以视为一棵完全的二叉树,完全二叉树的一个“优秀”的性质是,除了最底层之外,每一层都是满的,这使得堆可以利用数组来表示,每一个结点对应数组中的一个元素.

**数组与堆之间的关系**

****

**二叉堆一般分为两种：最大堆和最小堆**

**最大堆:** 根结点的键值是所有堆结点键值中最大者

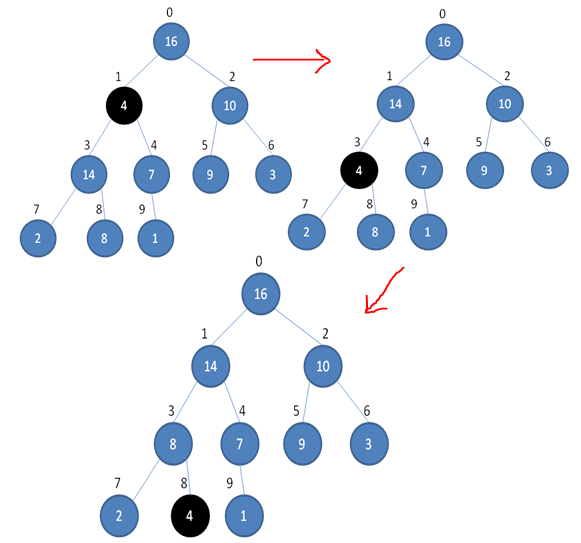
**最小堆:** 根结点的键值是所有堆结点键值中最小者

## 怎么将堆调整为最大堆

**下图展示了一趟调整的过程,这个过程递归实现,直到调整为最大堆为止**

在4,14,7这个小堆里边,父节点4小于左孩子14,所以两者交换

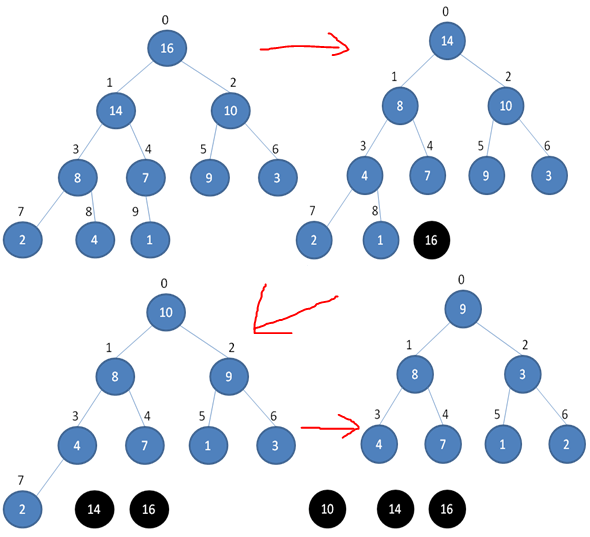
在4,2,8这个小堆里边,父节点4小于右孩子8,所以两者交换

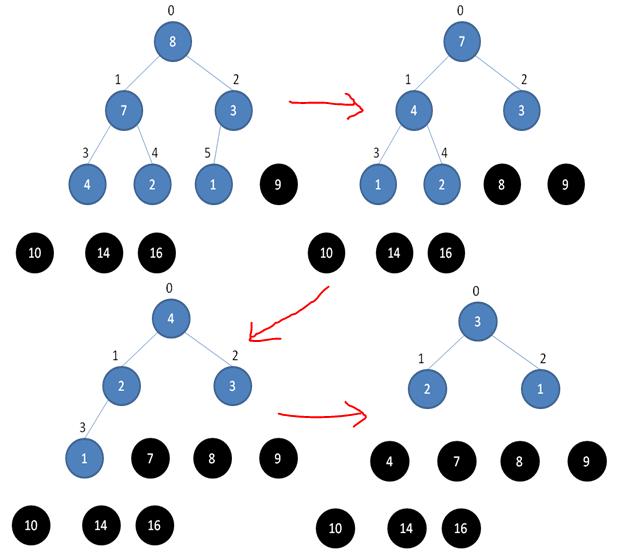
****

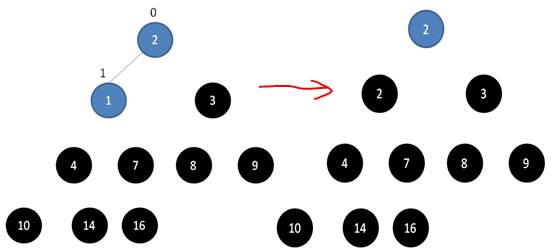
## 堆排序介绍

**堆排序就是把堆顶的最大数取出，将剩余的堆继续调整为最大堆,具体过程在第二块有介绍,以递归实现，剩余部分调整为最大堆后,再次将堆顶的最大数取出,再将剩余部分调整为最大堆,这个过程持续到剩余数只有一个时结束**

**下边三张图详细描述了整个过程**

****

****

****

## 算法稳定性

堆排序是一种不稳定的排序方法。

因为在堆的调整过程中，关键字进行比较和交换所走的是该结点到叶子结点的一条路径，

因此对于相同的关键字就可能出现排在后面的关键字被交换到前面来的情况

## 代码实现

# 归并排序

归并排序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

## 基本思想

将待排序序列R[0...n-1]看成是n个长度为1的有序序列，将相邻的有序表成对归并，得到n/2个长度为2的有序表；将这些有序序列再次归并，得到n/4个长度为4的有序序列；如此反复进行下去，最后得到一个长度为n的有序序列。

综上可知：

归并排序其实要做两件事：

（1）“分解”——将序列每次折半划分。

（2）“合并”——将划分后的序列段两两合并后排序。

## 如何合并

在每次合并过程中，都是对两个有序的序列段进行合并，然后排序。

这两个有序序列段分别为 R[low, mid] 和 R[mid+1, high]。

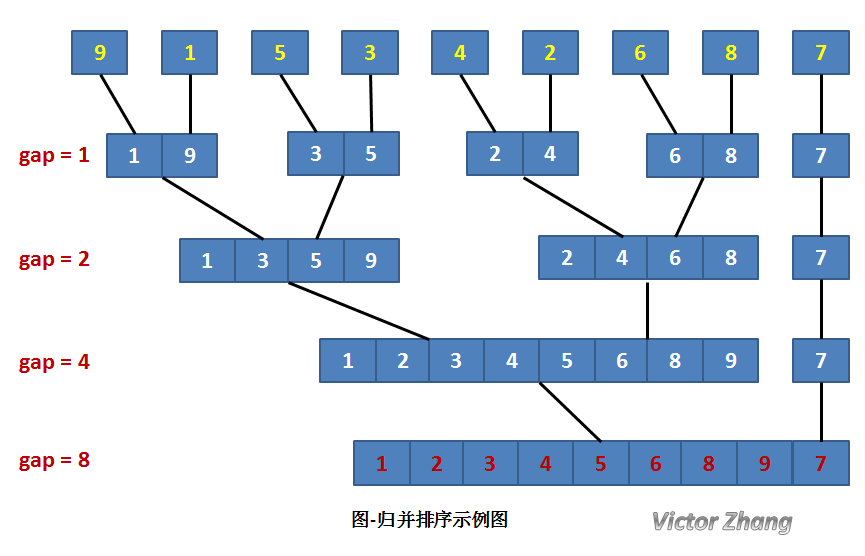
先将他们合并到一个局部的暂存数组R2中，带合并完成后再将R2复制回R中。

为了方便描述，我们称 R[low, mid] 第一段，R[mid+1, high] 为第二段。

每次从两个段中取出一个记录进行关键字的比较，将较小者放入R2中。最后将各段中余下的部分直接复制到R2中。

经过这样的过程，R2已经是一个有序的序列，再将其复制回R中，一次合并排序就完成了。

## 如何分解



在某趟归并中，设各子表的长度为gap，则归并前R[0...n-1]中共有n/gap个有序的子表：R[0...gap-1], R[gap...2\*gap-1], ... , R[(n/gap)\*gap ... n-1]。

调用Merge将相邻的子表归并时，必须对表的特殊情况进行特殊处理。

若子表个数为奇数，则最后一个子表无须和其他子表归并（即本趟处理轮空）：若子表个数为偶数，则要注意到最后一对子表中后一个子表区间的上限为n-1。

## 算法稳定性

在归并排序中，相等的元素的顺序不会改变，所以它是稳定的算法。

## 归并排序和堆排序、快速排序的比较

若从空间复杂度来考虑：首选堆排序，其次是快速排序，最后是归并排序。

若从稳定性来考虑，应选取归并排序，因为堆排序和快速排序都是不稳定的。

若从平均情况下的排序速度考虑，应该选择快速排序。