****

**Tongji University**

**数据结构课程设计实验报告**

**010-排序算法比较**

专 业： 软 件 工 程

指导教师： 张 颖

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **项目概览**
   1. **项目背景**

排序即是将一组记录数据以特定特征进行排列，而排序算法则是用来程序实现这种操作。由于排序在日常生活中具有极广泛地应用，同时被排序的数据也从几个乃至成千上万的大数据不等，故而需要对排序算法进行不断的优化与更迭。通过使用性能优秀的排序算法，能够节省大量资源。

* 1. **项目要求**
     1. **项目整体要求**

随机函数产生一百，一千，一万和十万个随机数，用冒泡排序，选择排序，直接插入排序，希尔排序，快速排序，堆排序，归并排序和基数排序的八种排序方法排序，并统计每种排序所花费的排序时间和交换次数。

其中，随机数的个数由用户定义，系统产生随机数。并且显示他们的比较次数。

请在文档中记录上述数据量下，各种排序的计算时间和存储开销，并且根据实验结果说 明这些方法的优缺点。

* + 1. **项目输入要求**

项目输入要求实现客户手动键入想产生的随机数个数，程序得到命令后生成随机数并存储在数组单元中，用户键入选择的排序算法代号即可。

* + 1. **项目输出要求**

要求项目输出排序算法完成任务的排序时间和交换次数。

* 1. **项目需求分析**

根据项目要求可以得到，本次项目需要实现八种排序算法，并实现相应功能。

* + 1. **项目功能完善**

项目将排序算法及其内核数据存储进行类的封装，使调用简洁明了，功能实现周全。

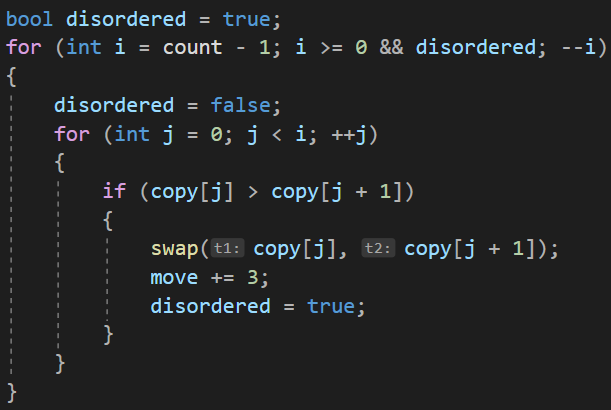
* + 1. **代码分块书写**

由于本次项目实现代码功能较多，故将代码进行分区分块。公有函数作为功能标签负责实现功能或调用辅助函数（多为需要递归操作或辅助函数的排序算法），而私有函数则作为辅助对算法进行实现。

* + 1. **代码的健壮性**

为了防止程序崩溃退出或时间过长无法求解，故为随机值设定上下限，当用户输入太大或太小的随机数个数时，程序会提示错误并要求重新输入。

1. **算法实现**
   1. **冒泡排序**
      1. **算法设计与实现**
2. 冒泡排序设元素序列中的元素个数为n，首先比较第n-2个元素和第n-1个元素，如果发生逆序（即前一个大于后一个），则将这两个元素交换；
3. 然后对第n-3个和第n-2个元素做同样处理，重复此过程直到处理完第0个和第1个元素。
4. 重复过程直到数据有序为止。
   * 1. **算法性能分析**
5. 冒泡排序优点：算法逻辑简单，且空间复杂度较低，只有O(1)，是一种稳定的排序算法。
6. 冒泡排序缺点：冒泡排序涉及到多重循环嵌套和数据遍历，算法时间开销大，时间复杂度高，为O(N2),执行效率低。
   * 1. **算法代码实现**

****

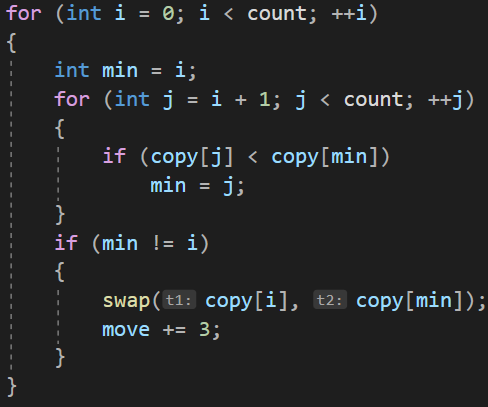
* 1. **选择排序**
     1. **算法设计与实现**

选择排序的基本思路是：

1. 每一趟在后面n-i个待排序的元素中选出排序码最小的元素，作为有序元素序列的第i个元素。
2. 待到n-2趟做完，待排序元素就只剩一个，不必再选择。

本次项目中选择直接选择排序的方法：

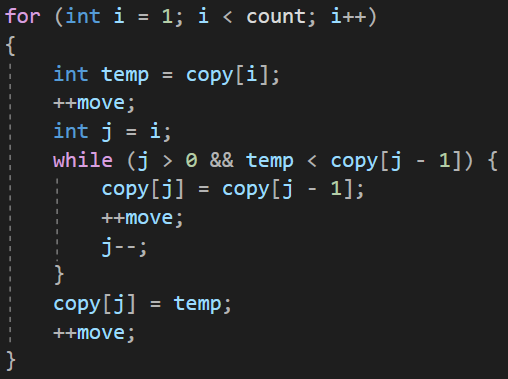
1. 在一组元素V[i]-V[n-1]中选择具有最小排序码的元素；
2. 若它不是这组元素中的第一个元素，则将它与这组元素中的第一个元素对调；
3. 在这组元素中剔除这个具有最小排序码的元素，在剩下的V[i+1]-V[n-1]中重复执行第（1）（2）步，直到剩余元素只有一个为止。
   * 1. **算法性能分析**
4. 选择排序优点：选择排序每一轮只需要交换一个元素，其交换次数是已知的。空间复杂度为O(1)。
5. 选择排序缺点：选择排序交换次数过多，时间开销大，时间复杂度达到O(N2),而且是一种极不稳定的排序算法。
   * 1. **算法代码实现**

****

* 1. **直接插入排序**
     1. **算法设计与实现**

直接插入排序的基本思想是：

1. 当插入第i（i>=1）个元素时，前面的V[0],V[1],…,V[i-1]已经排好序。
2. 用V[i]的排序码与V[i-1],V[i-2],…的排序码顺序进行比较；
3. 找到插入位置即将V[i]插入，原来位置上的元素向后顺移。
   * 1. **算法性能分析**
4. 直接插入排序优点：空间复杂度较低，为O(1)，是一种稳定的排序算法。
5. 直接插入排序缺点：直接插入排序最好情况下比较（n-1）次，最坏情况下比较n（n-1）/2次。平均时间复杂度达到O(N2)，时间开销大，时间复杂度高。
   * 1. **算法代码实现**



* 1. **希尔排序**
     1. **算法设计与实现**

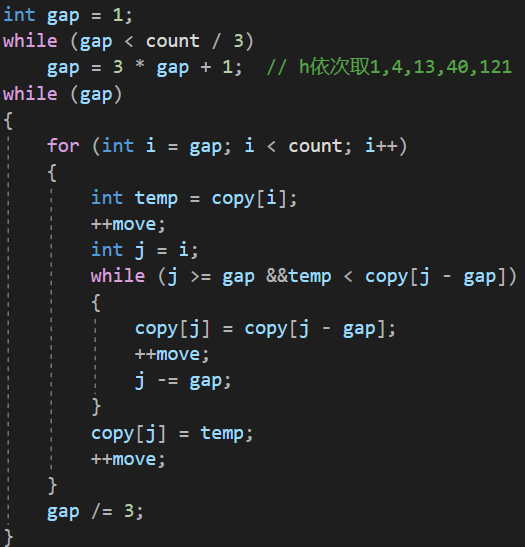
希尔排序又称缩小增量排序（diminishing-increment sort）。该算法的基本思想是：

1. 设待排序元素序列有n个元素，首先取一个整数gap<n作为间隔，将全部元素分为gap个子序列，将所有距离为gap的元素放在同一个子序列中。
2. 在每一个子序列中施行直接插入排序。
3. 缩小间隔gap，重复上述的子序列划分和排序工作。
4. 最后取gap==1，将所有元素放在同一个序列中排序为止。
   * 1. **算法性能分析**

由于开始时gap取值较大，每个子序列中的元素较少，排序速度较快；

排序后期gap取值逐渐变小，子序列中元素个数逐渐变多，但由于前面工作的基础，大多数元素已经基本有序，故排序速度仍然很快。

1. 希尔排序优点：空间复杂度好，O(1);是一种相对高效的基于交换元素的排序方法；
2. 希尔排序缺点：希尔排序在交换过程中会改变元素的相对次序，是一种不稳定的排序算法。排序算法时间复杂度依赖于增量序列函数，当n在某个特定范围时，希尔排序的时间复杂度约为O(N1.3)
   * 1. **算法代码实现**

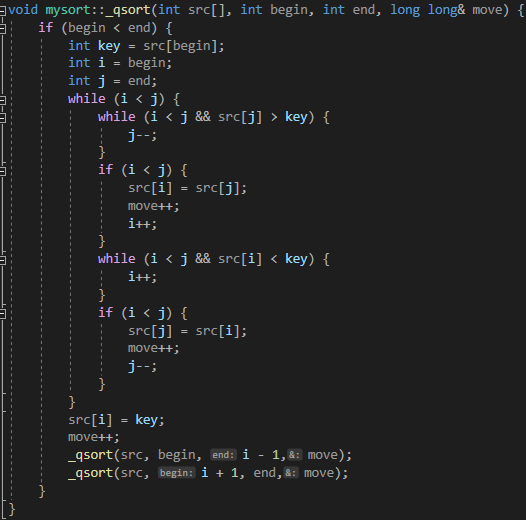


* 1. **快速排序**
     1. **算法设计与实现**

快速排序（Quick Sort）也叫分区排序，是目前应用最广泛的排序算法。

该算法的基本思路是：

1. 先从数列中取出一个数作为基准数。
2. 分区过程，将比这个数大的数全放到它的右边，小于或等于它的数全放到它的左边。
3. 再对左右区间重复第二步，直到各区间只有一个数。
   * 1. **算法性能分析**
4. 快速排序优点：快速排序平均性能好，时间复杂度仅为O(nlog2n)；空间复杂度为O(logN)；
5. 快速排序缺点：快速排序涉及远距离交换，是一种不稳定的排序算法。当初始序列有序或者基本有序时，时间复杂度变为O(N2)
   * 1. **算法代码实现**

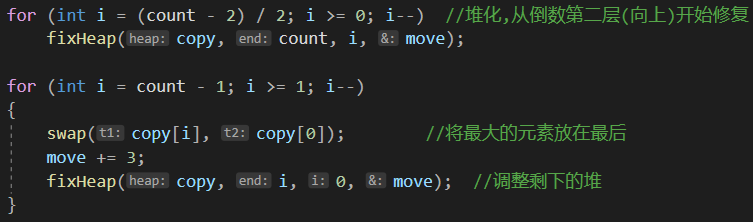


* 1. **堆排序**
     1. **算法设计与实现**

之前已经实现了堆结构和基本的堆形成算法。通过堆运算可以实现选择排序思路。

堆排序（heap sort）分为两个步骤：

1. 根据初始输入数据利用堆调整算法siftDown（）形成初始堆；
2. 通过一系列元素交换和重新调整堆进行排序。
   * 1. **算法性能分析**
3. 堆排序优点：堆排序效率相对稳定，无论待排序序列是否有序，堆排序效率都是O(nlogn)不变。空间复杂度小，仅为O(1)；
4. 堆排序缺点：堆在实际场景中数据频繁变动，待排序序列每次更新都需要进行堆的维护。
   * 1. **算法代码实现**

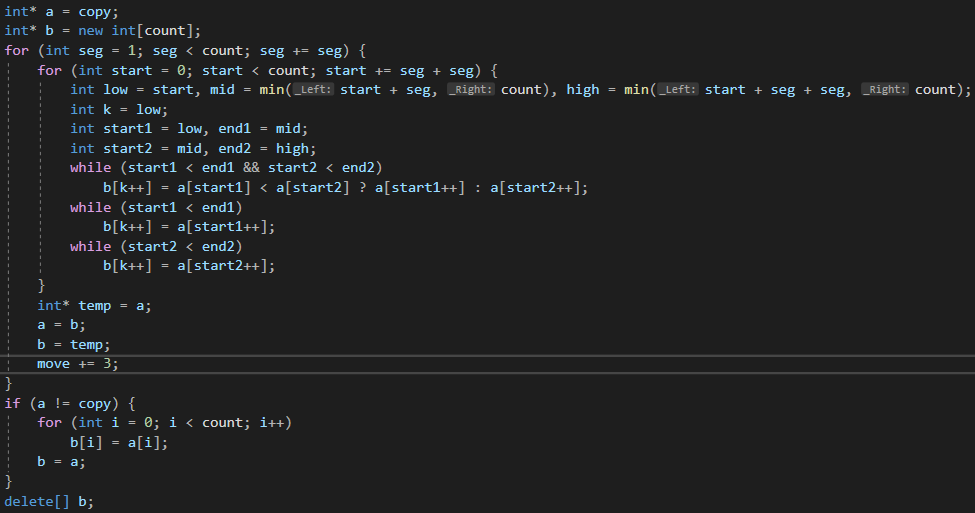


* 1. **归并排序**
     1. **算法设计与实现**

归并排序（merge sort）是一种概念商最为简单的排序算法。与快速排序一样，归并排序属于分治法的范畴。归并排序将待排序的元素序列分成两个长度相等的子序列，为每一个子序列排序，然后将它们合并成同一个序列。

* + 1. **算法性能分析**

1. 归并排序优点：归并排序的平均时间复杂度仅为O(nlogn)，是基于比较的排序算法所能达到的巅峰。且归并排序是一种稳定的算法；
2. 归并排序缺点：归并排序空间开销较大，需要O(n)的辅助空间，同类归并排序空间复杂度较高。
   * 1. **算法代码实现**

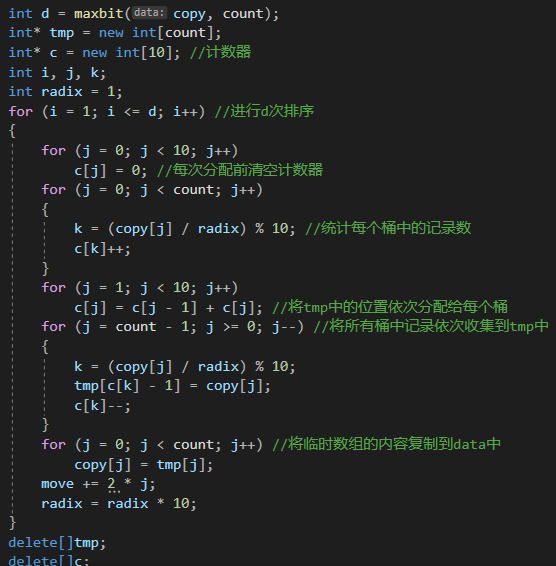


* 1. **基数排序**
     1. **算法设计与实现**

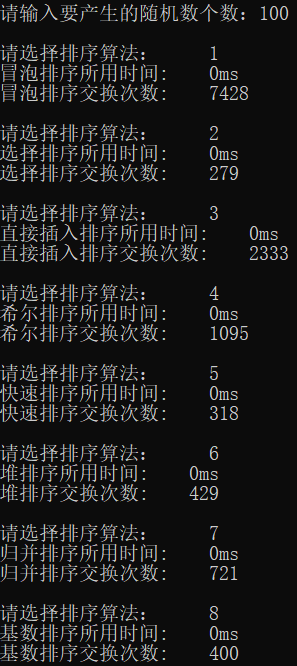
基数排序（radix sort）是一种非比较型整数排序算法。其基本思想是将证书按位数切割成不同的数字，然后按每个位数分别比较。基数排序的方式可以采用LSD（Least significant digital）或MSD（Most significant digital）。LSD的排序方式由键值的最右边开始，而MSD则相反，从键值最左边开始。

* 1. 将所有待比较数值统一为同样的数位长度，数位短的数前面补零；
  2. 从最低位开始，依次进行排序；
  3. 从最低位一直到最高位排序完成以后，数列成为有序序列。
     1. **算法性能分析**

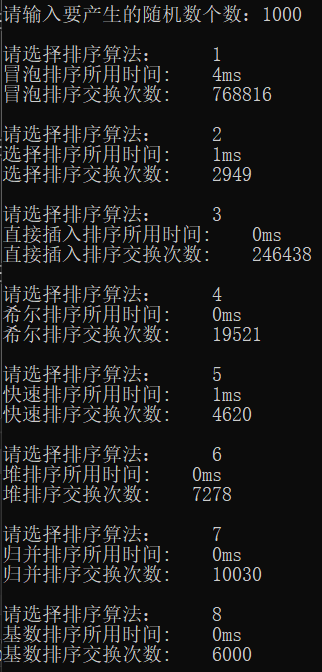
1. 基数排序优点：基数排序利用桶排序的方法，并不需要交换位置，故其是一种稳定的算法。基数排序效率与初始序列是否有序没有关联，其时间复杂度为O(NlogM)。
2. 基数排序缺点：基数排序需要将元素进行“装桶”操作，故需要任何位数都n+r个临时空间，空间开销比较大，空间复杂度O(N);
   * 1. **算法代码实现**



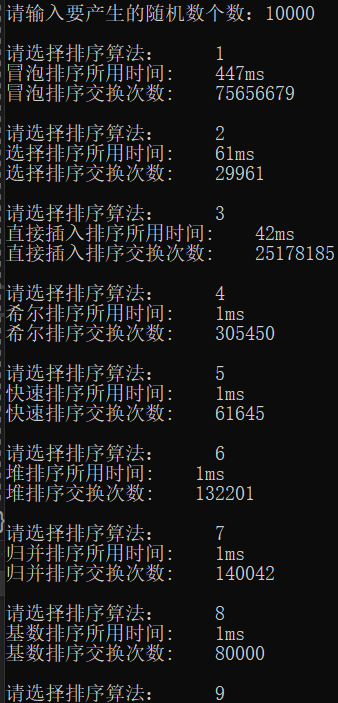
1. **程序测试**
   1. **100个随机数测试**

****

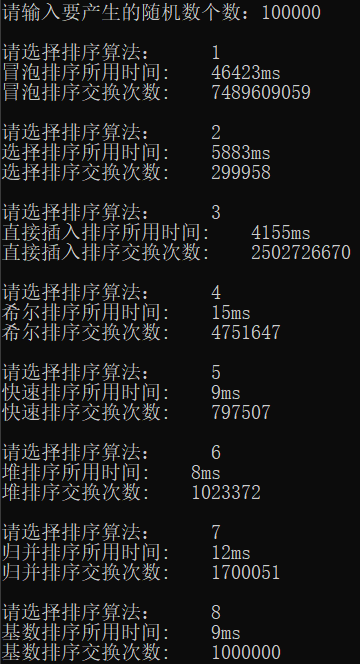
* 1. **1000个随机数测试**

****

* 1. **10000个随机数测试**

****

* 1. **100000个随机数测试**

****