项目说明文档

数据结构课程设计

——8种排序算法的比较案例

作 者 姓 名： 欧明锋

学 号： 1651822

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

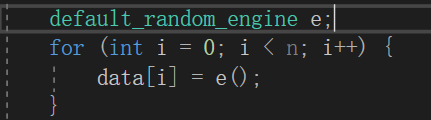
同济大学

Tongji University

1. 功能分析
2. 本项目是为了比较冒泡排序、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、基数排序8种排序算法在一百，一千，一万和十万这四个数量级的随机数下进行排序的性能表现，其中性能评估包括三个方面：所用的现实时间、比较交换次数、以及CPU的clock时钟差，CPU的clock时钟差在不同算法中的差别相对与现实时间更加明显直观，能帮助我们更准确地分析算法性能。
3. 代码实现

说明：该项目的所有排序算法都是将数按从小到大排列，比较次数的存储是使用unsigned long(最大值为4294967295)以保证在十万数量级下，记录的比较次数能够不溢出，所有的排序函数都是使用函数模板，以提升排序元素类型的可扩展性。

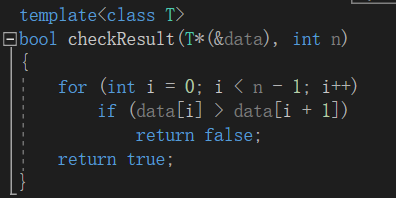
1. 随机数产生



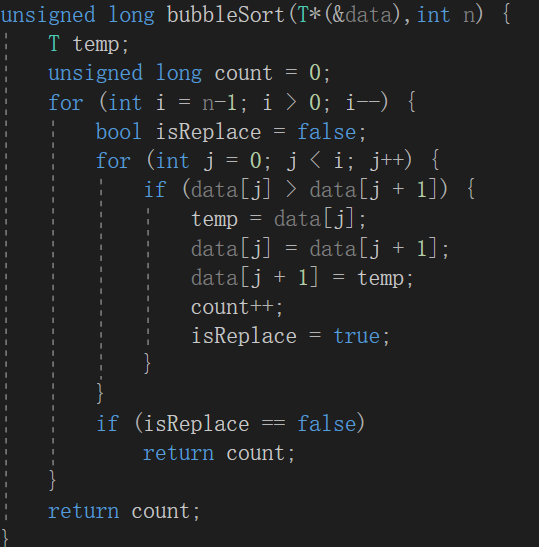
随机数的个数是由用户指定，类型为int,其产生是使用c++的random库中的随机数引擎产生的，其中并没有使用时间种子，以保证每个排序算法都是排序同样的数组

1. 排序算法正确性验证

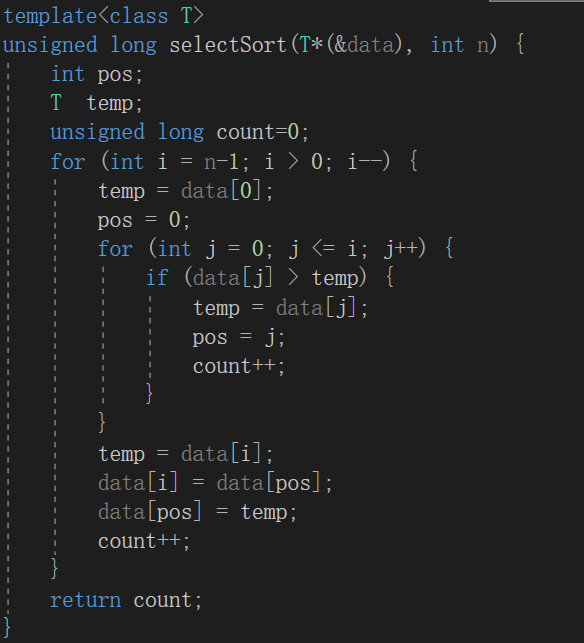
在排序结束后，都会经过checkResult函数验证数组是否从小到大排序，若是则有排序时间和交换次数的结果输出，若不是则不输出。



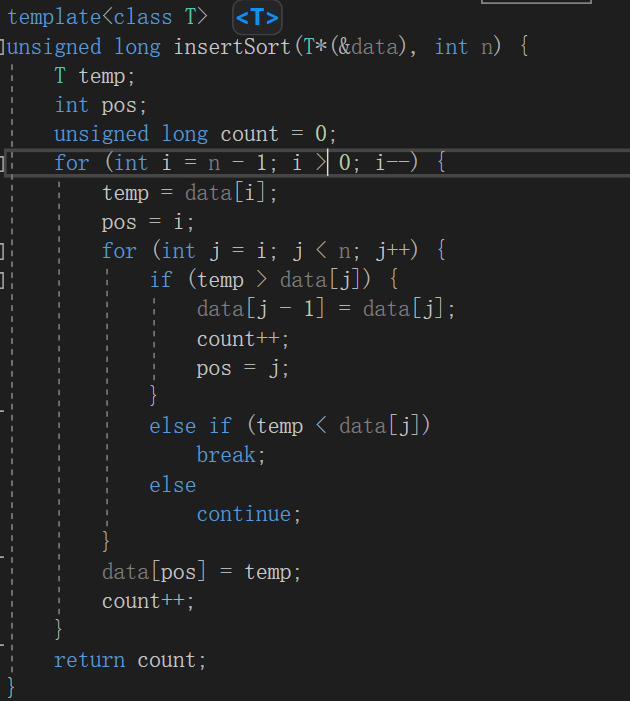
1. 冒泡排序
2. 该项目的冒泡排序是从数组的头部向尾部冒泡，每轮从底向上的冒泡确定一个数的位置
3. 每轮冒泡的结束位置比上一轮更靠近头部一位，最多进行n-1轮。
4. 其中使用了isReplace来标志每轮冒泡过程是否有数进行交换，若没有则说明排序结束则可返回比较次数，从而来优化冒泡排序算法。
5. 核心代码如下



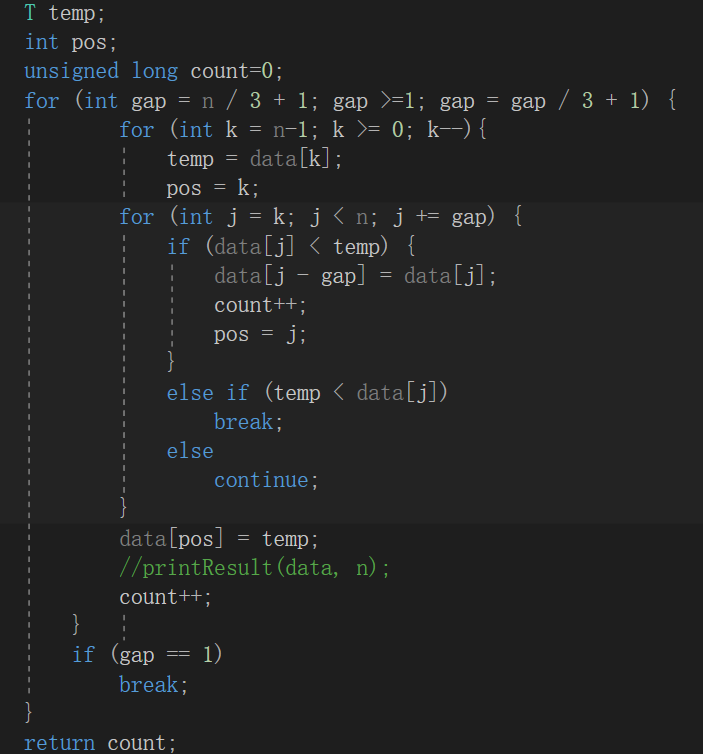
1. 选择排序
2. 首先将扫描范围定为整个数组，在排序数组中选择最大的数，若它不在最后一位则与扫描区间中最后以为调换，
3. 然后将上次扫描范围的尾部减一作为新的扫描范围，重复上述（选择最大数、调换）操作。
4. 总共n-1次扫描。
5. 核心代码如下：



1. 直接插入排序
2. 把n个待排序的元素看成为一个有序表和一个无序表。开始时有序表中只包含1个元素即数组最后一个元素，无序表中包含有n-1个元素即数组前n-1个数
3. 排序过程中每次从无序表中取出最后一个元素，将它插入到有序表中的正确位置(插入前使其他元素向上顺移一位)，使之成为新的有序表。
4. 这样重复n-1次即可完成排序。
5. 核心代码如下

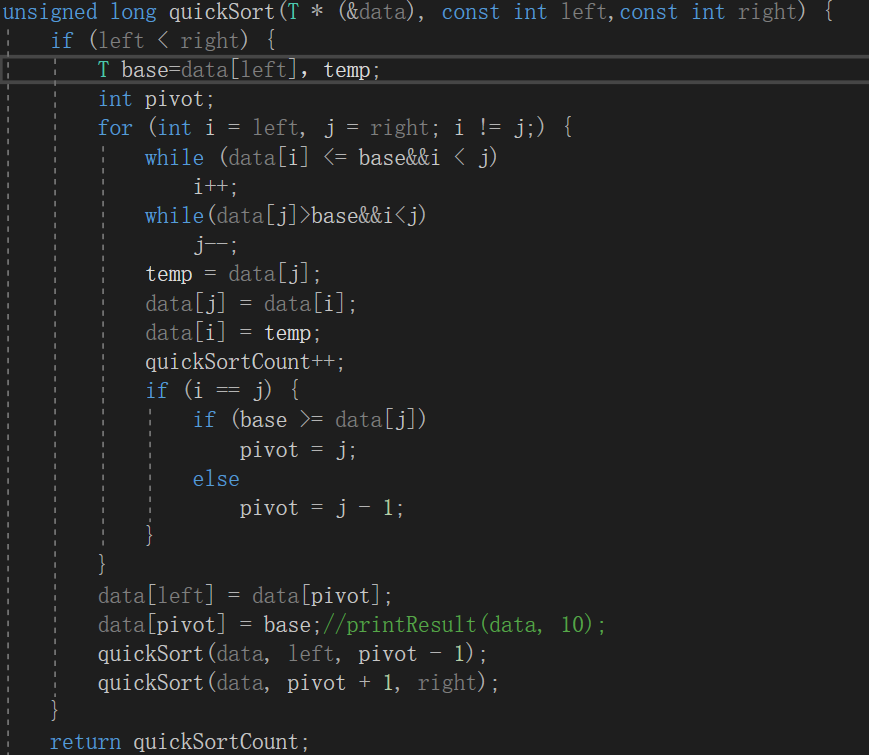


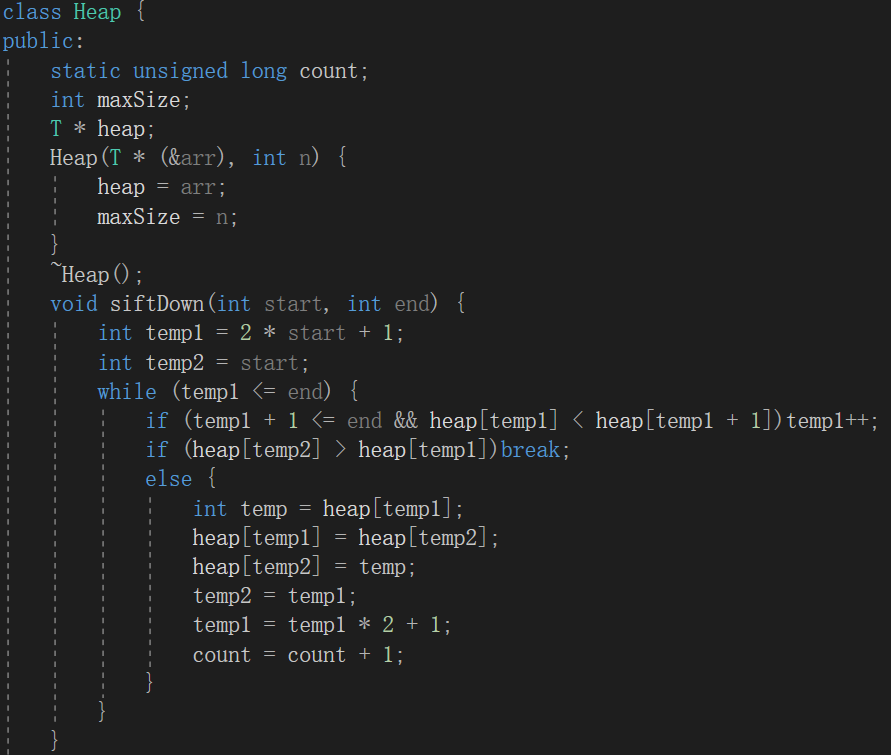
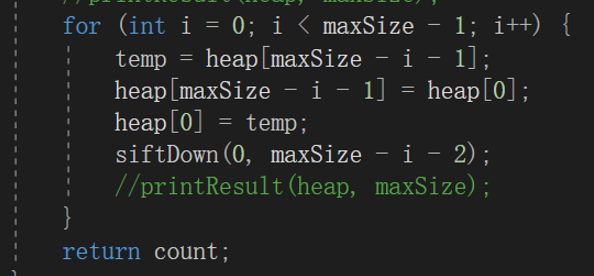
1. 希尔排序
2. 可以说是直接插入排序的优化，首先取gap=n/3+1作为间隔,将全部元素分为gap个子序列
3. 所有间距为gap的元素视为一个新子序列依次进行插入排序
4. 然后缩小gap = gap/3+1重复上述子序列划分插入排序操作
5. 直到gap为1时，将所有元素排序，此时大多数元素已经有序，所以能快速完成。
6. 核心代码如下



1. 快速排序
2. 快排使用的是分治的思想，首先数组的首元素作为基准，将数组分为左右两个子序列，左边是小于等于该基准元素的所有元素构成的子序列，右边是大于该基准元素的所有元素构成的子序列，基准元素排在左右两序列中间。
3. 使用递归对左右两序列进行相同操作，直到每个子序列的left都不小于right即每个子序列都不可再分，就完成排序。
4. 其中划分左右两子序列的具体操作为：
   1. 设置两个标志i，j分别从未划分的原序列左右两端开始扫描
   2. i从左往右，j从右往左，当i遇到比基准元素大的元素时停下来，j遇到比基准元素小或等的元素时停止
   3. i、j位置上的元素交换位置，i、j继续扫描
   4. 直到i==j为止，将基准元素与i或i-1位置的元素调换，这样就完成了一次左右子序列的划分

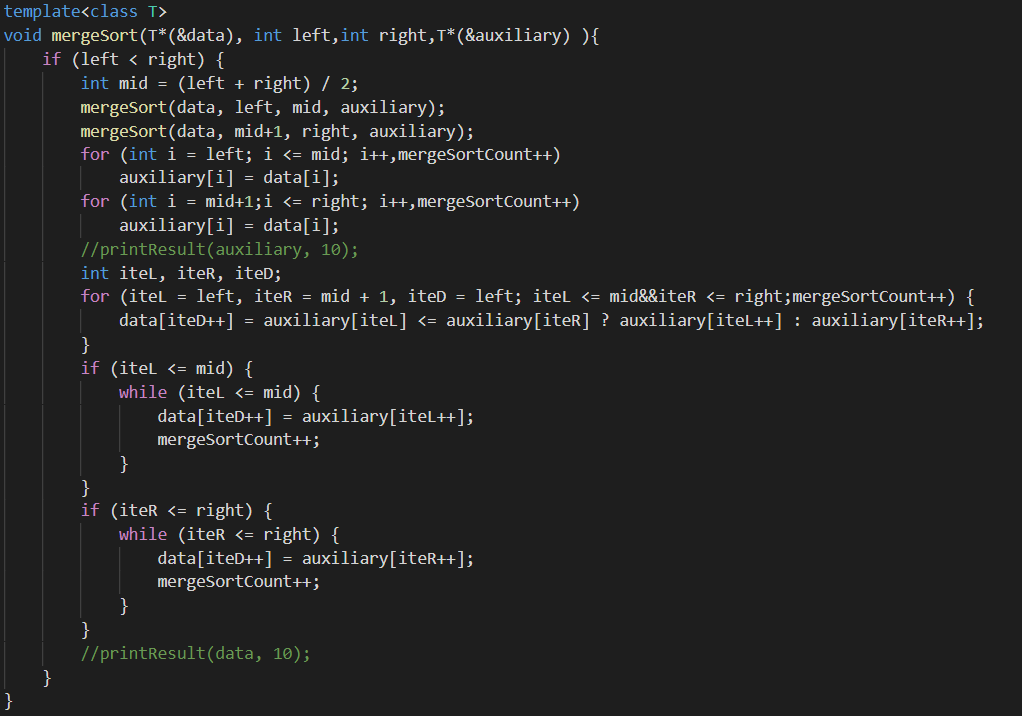
核心代码如下：



1. 堆排序
2. 首先将数组data作为输入数据来初始化一个堆，然后调用sort函数，先将其初始化为一个最小堆
3. 接着取出堆顶元素与第n个元素交换，对0~n-1使用siftdown再次将其调整为最小堆，将堆顶元素与第n-1个元素交换，对0~n-2使用siftdown将其调整为最小堆
4. 不断重复上述操作直到堆中只剩下一个元素，此时便完成排序。
5. 最小堆数据结构：
6. 排序函数核心代码：
7. 归并排序

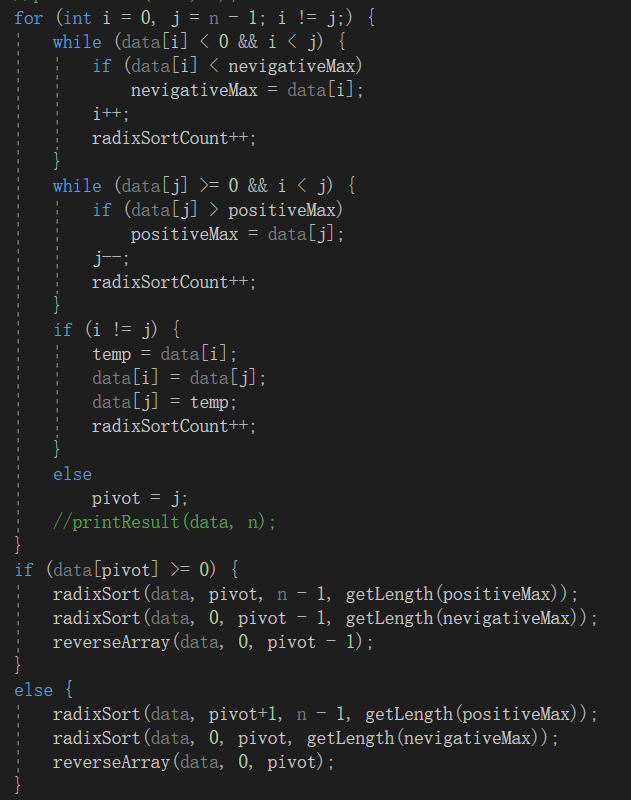
归并排序(Merge-Sort)是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。将已有序的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为2-路归并。

1. 把长度为n的输入序列分成两个长度为n/2的子序列；
2. 对这两个子序列分别采用归并排序；
3. 将两个排序好的子序列合并成一个最终的排序序列。

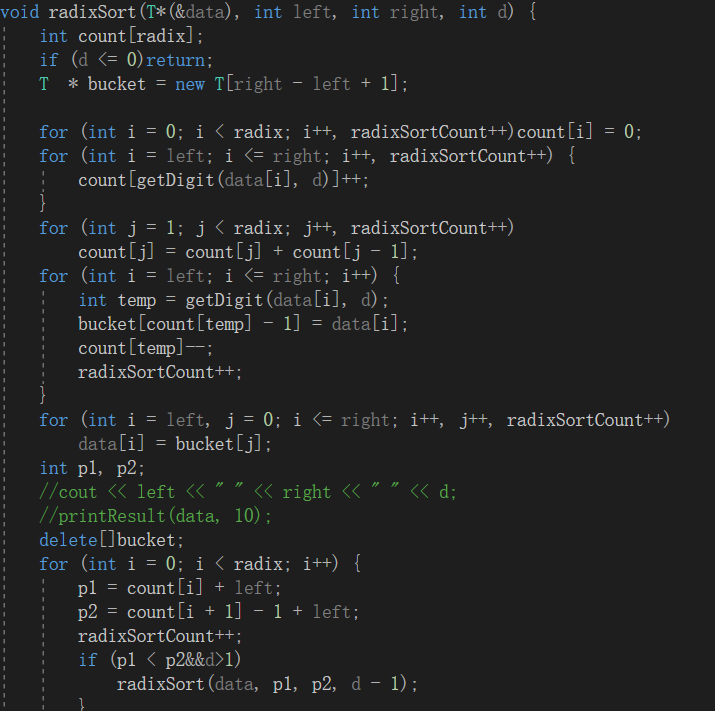


1. 基数排序
2. 基数排序是一种比较特别的排序，它不是直接比较两个数之间的大小，而是通过将数分割为不同的位数，然后每个位数进行比较，因此基数排序只适合整数。
3. 首先使用radixSortPre函数对数组进行预处理，以0为基准将划分为左右子序列，左子序列为负整数，右子序列为非负整数，这个划分方式与快速排序的左右序列划分方式相同，这里就不赘述了。

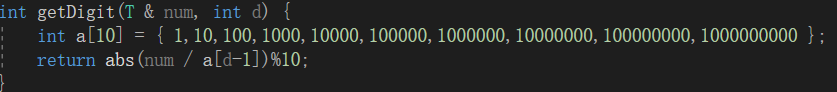
radixSortPre函数的核心代码如下



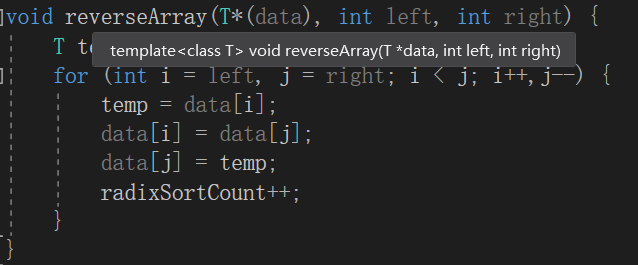
1. 然后将左右序列分别进行MSD基数排序。(即最高位优先)具体操作为：
   1. 首先将获得所有元素中最长位数（在radixSortPre函数中就已经获得），创建一个count[10]的辅助数组和一个bucket辅助数组
   2. 从这个最长位（假设为y）开始统计每个元素y位置上的数x(没有就是0)，将count[x]++
   3. 统计完后就更新y位上数字为x的元素应该存放的起始位置，然后将元素根据自己y位上的数字存放进bucket数组，
   4. 再将bucket中数来替换原数组，最后对每个y位上数字相同的子序列进行递归基数排序（最高位-1）。
   5. 当排序位数为0时就结束。
   6. 核心代码如下图所示：



* 1. 获得每个元素第d位上的数字的代码如下

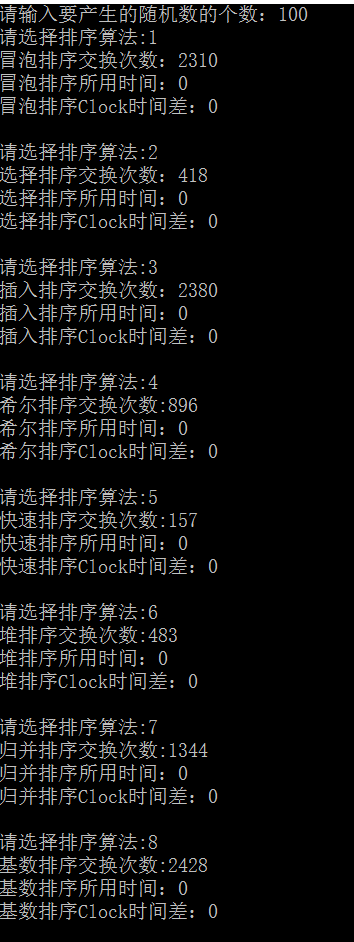


1. 因为数组中的负整数经过基数排序后的顺序是从大到小，所以最后要将负整数部分序列反转，完成从小到大的排序。
   1. 反转部分核心代码如下

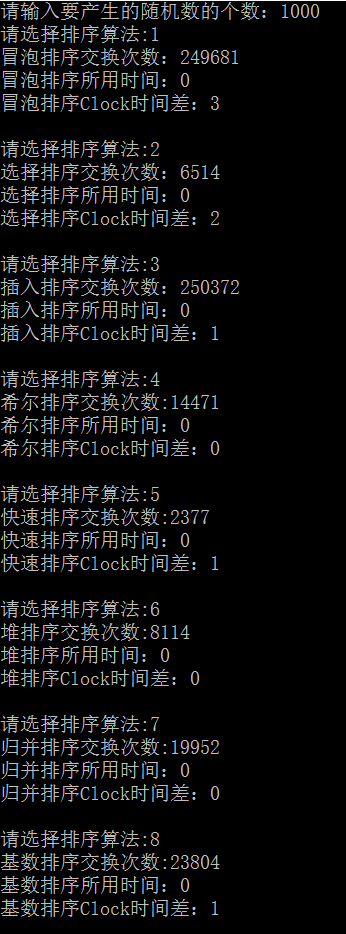


1. 测试结果

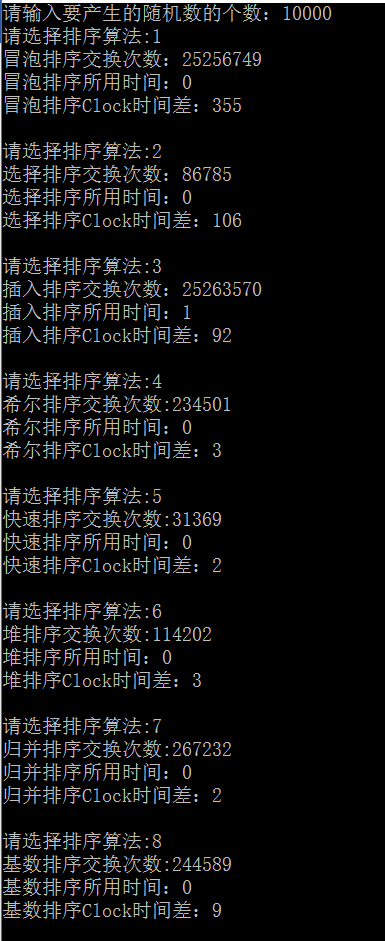
1.100个随机数



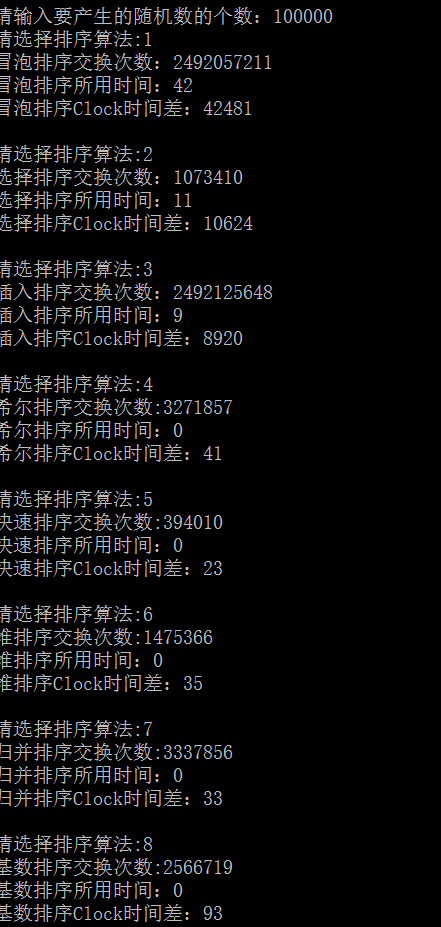
2.1000个随机数



3.10000个随机数



4.100000个随机数



1. 结果分析
2. 当排序数组的大小为100时
   1. 所有排序的现实时间和clock时钟周期都为0，
3. 当排序数组的大小为1000时，
   1. 所有排序的现实时间都为0，clock时钟差都处于一个量级(10以内)，
4. 当排序数组的大小为10000时，
   1. 所有排序的现实时间都为0，
   2. 前三个排序的clock时钟周期都处于一个量级（100），后5个排序的clock时钟周期都属于一个量级（10），
5. 当排序数组的大小为100000时，
   1. 后5个排序算法的现实时间依旧都为0，冒泡排序的时间最长42s,直接插入和选择排序的时间处于一个量级10s;
   2. 希尔排序、快速排序、堆排序、归并排序的clock时钟的量级相同，其中快速排序所用时间最少；基数排序，由于负整数存在的原因，其的clock时钟就比前四个大2倍。
6. 算法复杂度分析表

