**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： 戴仁杰

学 号： 1951650

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 - 3 -](#_Toc91237407)

[1.1 背景分析 - 3 -](#_Toc91237408)

[1.2 功能分析 - 3 -](#_Toc91237409)

[2 设计 - 3 -](#_Toc91237410)

[2.1 辅助函数设计 - 3 -](#_Toc91237411)

[2.2 系统设计 - 3 -](#_Toc91237412)

[3 算法分析与实现 - 4 -](#_Toc91237413)

[3.1 冒泡排序分析 - 4 -](#_Toc91237414)

[3.2 选择排序分析 - 4 -](#_Toc91237415)

[3.3 直接插入排序分析 - 5 -](#_Toc91237416)

[3.4 希尔排序分析 - 5 -](#_Toc91237417)

[3.5 快速排序分析 - 6 -](#_Toc91237418)

[3.6 堆排序分析 - 6 -](#_Toc91237419)

[3.7 归并排序分析 - 7 -](#_Toc91237420)

[3.8 基数排序分析 - 7 -](#_Toc91237421)

[3.9 总体功能的实现 - 8 -](#_Toc91237422)

[3.9.1 总体功能流程图 - 8 -](#_Toc91237423)

[3.9.2 总体功能核心代码 - 9 -](#_Toc91237424)

[3.9.3 总体功能截屏示例 - 10 -](#_Toc91237425)

[4 测试 - 11 -](#_Toc91237426)

[4.1 小范围测试 - 11 -](#_Toc91237427)

[4.2 大范围测试 - 12 -](#_Toc91237428)

[4.3 已有序数据测试 - 13 -](#_Toc91237429)

[5 案例总结 - 14 -](#_Toc91237430)

1 分析

1.1 背景分析

排序问题也是我们在日常生活中经常会遇到的问题。有时候我们对人进行排序，有时候我们对物品进行排序，还有的时候我们仅仅是对数字进行排序。不管对什么进行排序，它总要有一个用来排序的依据，例如根据人的体重进行排序，此时排序的依据就是体重，按人的体重值将人依次排列。

在计算机中，也有排序的存在，计算机中的排序是指将杂乱无章的数据元素，通过一定的方法按关键字顺序排列的过程。排序算法就是如何使得记录按照要求排列的方法。计算机中的排序算法有很多种，每一种算法都有它们独特的实现方式与优化方法。在本案例中我们就来讨论其中使用频率比较高的几种排序算法。

1.2 功能分析

在本案例中，我们选择了冒泡排序，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，基数排序八种排序算法来进行分析。

随机函数产生一百，一千，一万……随机数，用各种排序方法排序，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。同时，分析每一种排序算法是否具有稳定性，即在排序的过程中会不会改变元素彼此的位置的相对次序。在文档中记录每个数据量下，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。

2 设计

2.1 辅助函数设计

由于项目中要求随机数数目较多，但c++中的rand()范围很小，这样的话生成的随机数质量不高，且速度也较慢。为了简化操作，产生0-1000000000之间的随机数使用按位生成再拼接的方式。系统中设计了makeRandomNumber函数，来生成指定数量的随机数。此外，对大量的交换操作，封装了swapNumber函数用来将两个数据进行位置上的交换。

2.2 系统设计

本系统封装了一个排序比较系统（SortCompareSystem）类，采用了类模板提高泛用性。类内封装了runSortCompareSystem()函数，用于启动程序，方便外部调用。系统在使用时，会生成菜单，用户根据菜单中的指示。通过输入操作码来完成对应的排序算法。同时系统也会记录本次算法所使用的时间，数据交换次数等信息并显示在屏幕上。

3 算法分析与实现

3.1 冒泡排序分析

**排序过程：**

有一个待排序的元素序列中的元素个数为n，第一次的排序从第0个元素开始，首先比较第0个元素与第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；然后再按此规律比较第1个元素与第2个元素，第2个元素与第3个元素……直到比较完第n-2个元素与第n-1个元素。一次排序会将序列中的最大的元素交换到序列的最后。然后再进行一次排序，仍然是从第0个元素开始，不过到n-2个元素就结束，这会使得序列中第二大的元素交换到序列的倒数第二个位置……进行n-1次排序就可以将所有元素排好序。

**算法分析：**

1. 冒泡排序的时间复杂度为O()
2. 冒泡排序的空间复杂度为O(1)
3. 冒泡排序是一种稳定的排序算法。

**优化方法：**

不一定每个序列都需要n-1次排序才能排好序。可以在冒泡排序中增加一个判断标志flag，如果在一次排列中发生了元素交换，flag就为true。如果执行完一次排序后flag为false，就说明没有发生元素交换，此时的元素已经排好序，不需要再进行之后的排序。

3.2 选择排序分析

**排序过程：**

传统的选择排序过程即先找到整个待排序序列中最小的一个元素，如果它不是序列中的第一个元素，就把它与第一个元素互换位置。然后在剩下的元素中再寻找最小的元素，将它放在第二个元素的位置……重新进行，直到整个序列排好序。

**算法分析：**

1. 选择排序的时间复杂度为O()
2. 选择排序的空间复杂度为O(1)
3. 选择排序是一种不稳定的排序算法。

**优化方法：**

可以一次遍历序列同时找出其中的最小元素和最大元素，将最小元素放在起始位置，最大元素放在末尾位置，这样可以减少整体排序次数，改进后对n个数据进行排序，最多只需进行n/2趟排序即可。在该优化方法中要特别注意如果最大值就在起始位置，那么将最小元素交换完后，最大元素的位置跑到了之前最小元素所在的位置，要将此处的元素与末尾位置进行交换。

3.3 直接插入排序分析

**排序过程：**

从第1个元素开始，将第1个元素取出，找到前面该插入的位置（即插入位置的前一个元素比其小，后一个元素比其大）并插入，随后再取出第2个元素，按照此方式插入……以此类推，这样的话，当要插入第i个元素时，前面的第0，1，2……i-1个元素已经排好序，只需要将第i个元素插入，原来插入位置上及之后的元素向后顺延即可，保证插入后第0，1，2……i-1，i个元素仍保持有序。将所有的元素都插入一遍，排序就完成了。注意在排序过程中，如果之前的元素中有与待插入元素相同的元素，就要把待插入元素插入到该元素后面（保持稳定性）。

**算法分析：**

1. 直接插入排序的时间复杂度为O()
2. 直接插入排序的空间复杂度为O(1)
3. 直接插入排序是一种稳定的排序算法。

**优化方法：**

由于本排序方法题目中要求为直接插入排序，因此未加优化，如果要优化方法的话，可以使用折半插入的方法。即在寻找一个元素的插入位置时，使用折半查找的方法，每次与当前范围的中间值作比较，如果大于的话将范围缩小到后一半，如果小于的话将范围缩小到前一半，直至找到插入位置。这样优化的话可以减少排序次数，但是元素的移动次数不会改变。

插入排序的另一种优化就是之后会提到的希尔排序。

3.4 希尔排序分析

**排序过程：**

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，所有距离为gap的元素放置在同一个子序列中，在每一个子序列中分别进行直接插入排序。然后缩小间隔gap，重复上述的子序列划分与排序工作。直到最后将所有元素放在同一个序列中排序为止。

**算法分析：**

1. 应用不同的序列会使希尔排序的性能存在有很大差异，本系统中选择的是折半序列，在n处于一定范围时，其平均时间复杂度为O()，最坏时间复杂度为O（）。
2. 希尔排序是一种不稳定的排序算法。

**优化方法：**

希尔排序最主要的优化方法是使用不同的gap序列，最初Shell提出每次将gap除以2的序列，这样的话希尔排序的最坏时间复杂度可以达到O()。之后的人们都提出了不同的增量序列，例如Hibbard序列，其平均时间复杂度为O()，最坏时间复杂度为O（）。在不同序列中，Sedgewick序列是已知比较好的增量序列，其平均时间复杂度为O()，最坏时间复杂度为O（）。

3.5 快速排序分析

**排序过程：**

待排序元素序列有n个元素，首先找到当前第left个元素到第right个元素排序范围内的一个基准元素，对整个元素序列进行划分，把排序范围内比基准元素小的都放在左边，比基准元素大的都放在右边，基准元素放在中间。这样一次下来就得到左右两个子序列，再将子序列按照上述操作进行排列，直至所有元素排列完成为止。

**算法分析：**

1. 快速排序的平均时间复杂度为O()。
2. 快速排序的空间复杂度为O()。
3. 快速排序是一种不稳定的排序算法。

**优化方法：**

在快速排序算法中，每次划分时用于比较的基准元素的选择对于算法的性能有很大影响。如果基准元素选择不合适，会导致算法性能的大幅度退化。如果要可以合理选择基准元素，使得每次划分所得的两个子序列中的元素个数尽可能地接近，就可以加速排序速度。因此，我们可以使用三者取中的算法对快速排序进行优化。

所谓三者取中，就是在取基准元素时，将序列左端点left，序列右端点right与序列中间位置（left+right）/2相互进行比较，然后取中间值作为基准元素，对整个序列进行划分。

还有其他的方法，比如在划分过程中对小规模子序列不进行排序而跳过，这样在划分之后得到的是一个整体上几乎已经排好序的元素序列，随后在进行一遍插入排序将序列排好。有研究表明，三者取中和小规模序列的中止两种措施结合起来，可以将用递归实现的快速排序算法效率提高20%—25%。

3.6 堆排序分析

**排序过程：**

待排序元素序列有n个元素，首先先用调整算法siftDown()将堆初始化为最大堆，siftDown()的作用是从结点i开始到n-1为止，自上而下比较，如果子女的值小于双亲结点的值，就把它们相互交换，这样就可以将一个局部调整为最大堆，当i取遍从0到（n-2）/2的所有整数后，整个堆就被初始化为最大堆。随后将堆中最大元素，即第0个元素与第n-1个元素进行交换，之后使用算法siftDown()将从第0个元素到第n-2个元素重新调整为最大堆，此时最大元素就被成功安排在了序列的最后一位；再交换第0个元素与第n-2个元素，之后使用算法siftDown()将从第0个元素到第n-3个元素重新调整为最大堆……反复执行，最后就会得到排序完成的元素序列。

**算法分析：**

1. 堆排序的平均时间复杂度为O()。
2. 堆排序的空间复杂度为O(1)。
3. 堆排序是一种不稳定的排序算法。

3.7 归并排序分析

**排序过程：**

待排序元素序列有n个元素，首先把长度为n的输入序列分成两个长度为n/2的子序列，再将n/2的子序列分为两个长度为n/4的子序列……不断进行递归，直到子序列的长度为1，然后递归开始回升进行排序，每层递归将两个子序列合并排好序后返回上一层递归继续此操作，这样就使得每一层在进行子序列合并时，这两个子序列都是已经排好序的序列。当递归结束，序列就已经排好序。

**算法分析：**

1. 归并排序的平均时间复杂度为O()。
2. 归并排序的空间复杂度为O()。
3. 归并排序是一种稳定的排序算法。

**优化方法：**

归并排序有数种优化方法。

首先可以改进两路归并算法，把元素序列复制给辅助数组时，把第二个有序表的元素顺序翻转，这样的话两个待归并的表就会从两端开始处理，向中间归并。这样的话两个待归并的表的尾端就可以互成“监视哨”，就可以避免检查两个表是否已经复制结束的判断，提高程序效率。

其次，我们可以在合并的时候检查一下左边子序列的尾端是否小于右边子序列的首端，如果小于的话就说明此时的序列左右半边已经是有序序列，不需要再进行合并操作，直接复制即可。从而省去一定的时间。

最后，我们在每次执行归并的时候，都需要将待归并序列复制到辅助数组上，再从辅助数组归并到序列里。但我们可以把这个复制的过程给去除掉。方法就是要在递归调用的每个层次交换输入数组和输出数组的角色，从而不断地把输入数组排序到辅助数组，再将数据从辅助数组排序到输入数组。我们在进入递归前就将初始序列复制到一个辅助数组中，然后将它们两个一起传入递归，在递归调用的每个层次交换输入数组和输出数组的角色。

例如：5 6 4 7 1 3 8 2

我们在最后一层递归中利用辅助数组进行合并，结果进入原数组，为5 6 4 7 1 3 2 8，随后进入上一层，此时刚刚的辅助数组是这一层的原数组，刚刚的原数组是这一层的辅助数组，所以在这一层中是利用的辅助数组进行合并，原数组中结果为4 5 6 7 1 2 3 8……不断向前回升，最后就可以得到有序序列。

除此之外，对小规模子数组使用插入排序等方法也可以使归并排序的运行时间缩短。

3.8 基数排序分析

**排序过程：**

待排序元素序列有n个元素，首先选择基数，将序列中的所有元素分配到对应的“桶”中，并对每个桶里的元素个数进行计数，随后依次收集桶中元素并将其放在对应位置，重复执行“分配”与“收集”操作，直到所有元素已经排好序。

**算法分析：**

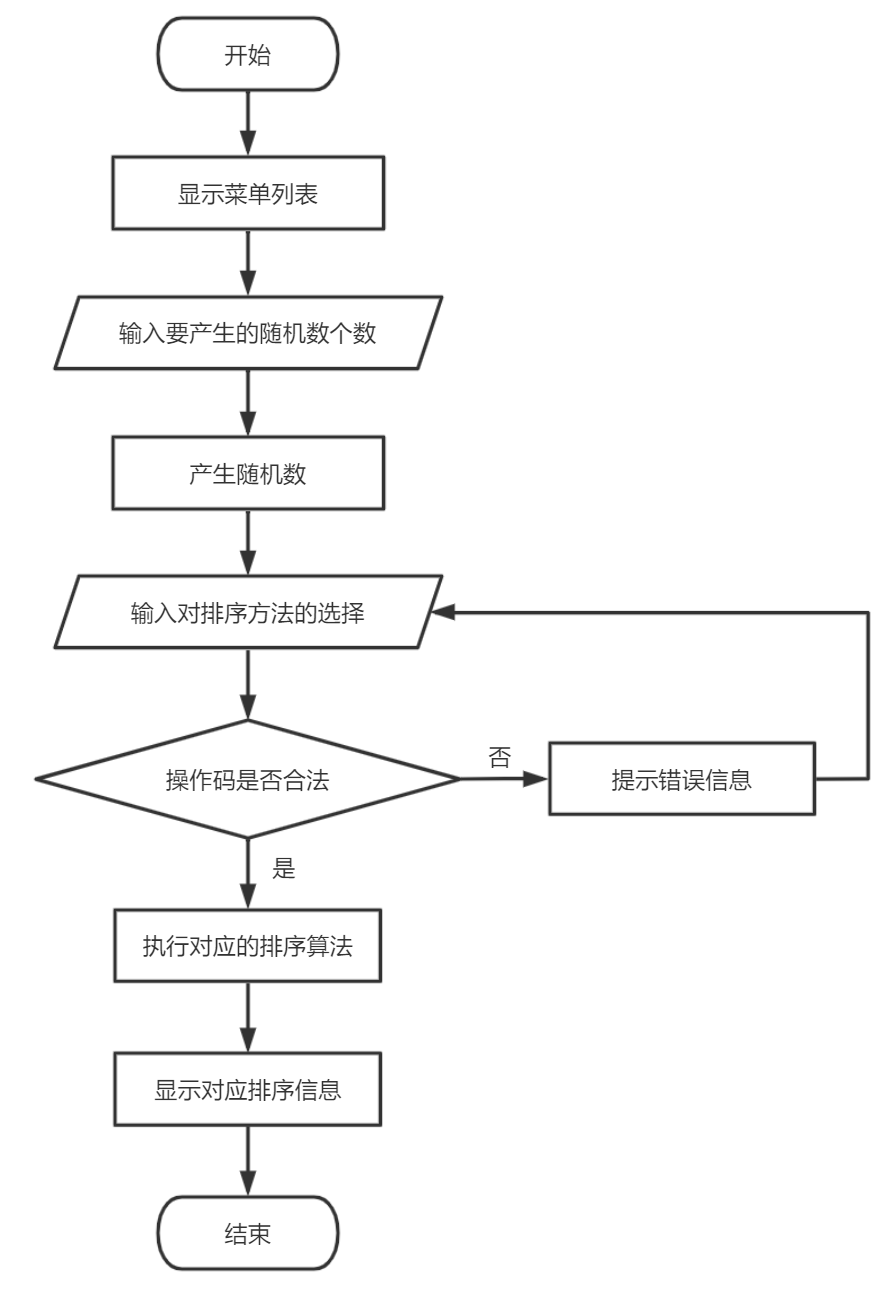
1. 基数排序的平均时间复杂度为O()，d为进行“分配”与“收集”的趟数。
2. 基数排序的空间复杂度为O()
3. 基数排序是一种稳定的排序算法。

**优化方法：**

在选择基数时我们总会习惯性的以10的幂作为基数，这与我们平时多用10进制运算相符合。但是，计算机是以二进制存储数据的，如果选择2的幂作为基数，可以通过位运算来简化代码。比如在本项目中，由于int类型的范围为-2^31——2^31-1，因此采用基数为2048，这样的话10进制中的除法操作与取模操作就会变成右移操作和与操作。由于2047的二进制表示为11111111111共11位，因此只需要进行三趟就可以覆盖所有的int类型数据。三次分别取元素的低11位，中间11位与高11位进行排序操作，实现方法为将元素进行右移操作后与2047做与运算，这样的话输出的结果即为需要的位数。

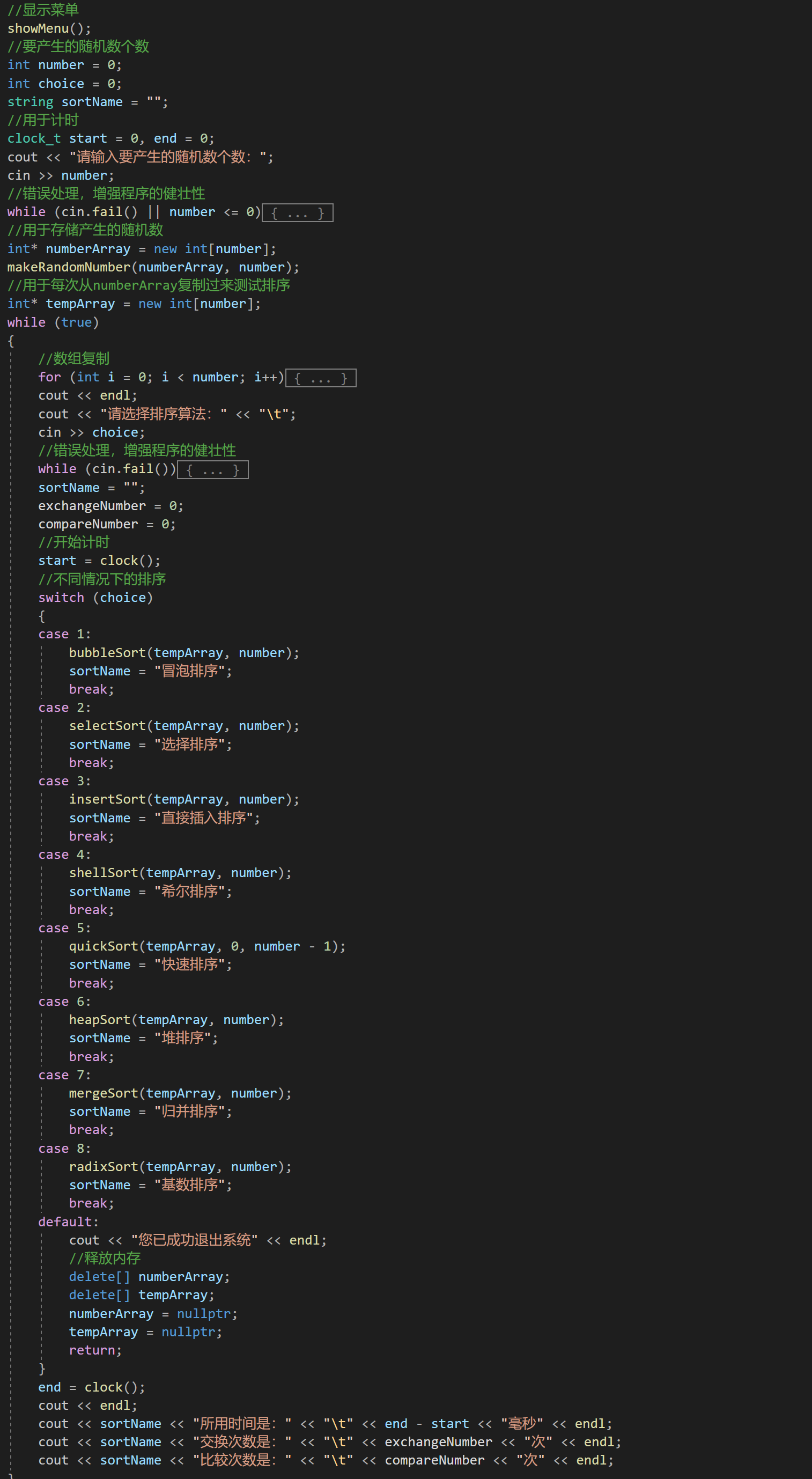
3.9 总体功能的实现

3.9.1 总体功能流程图

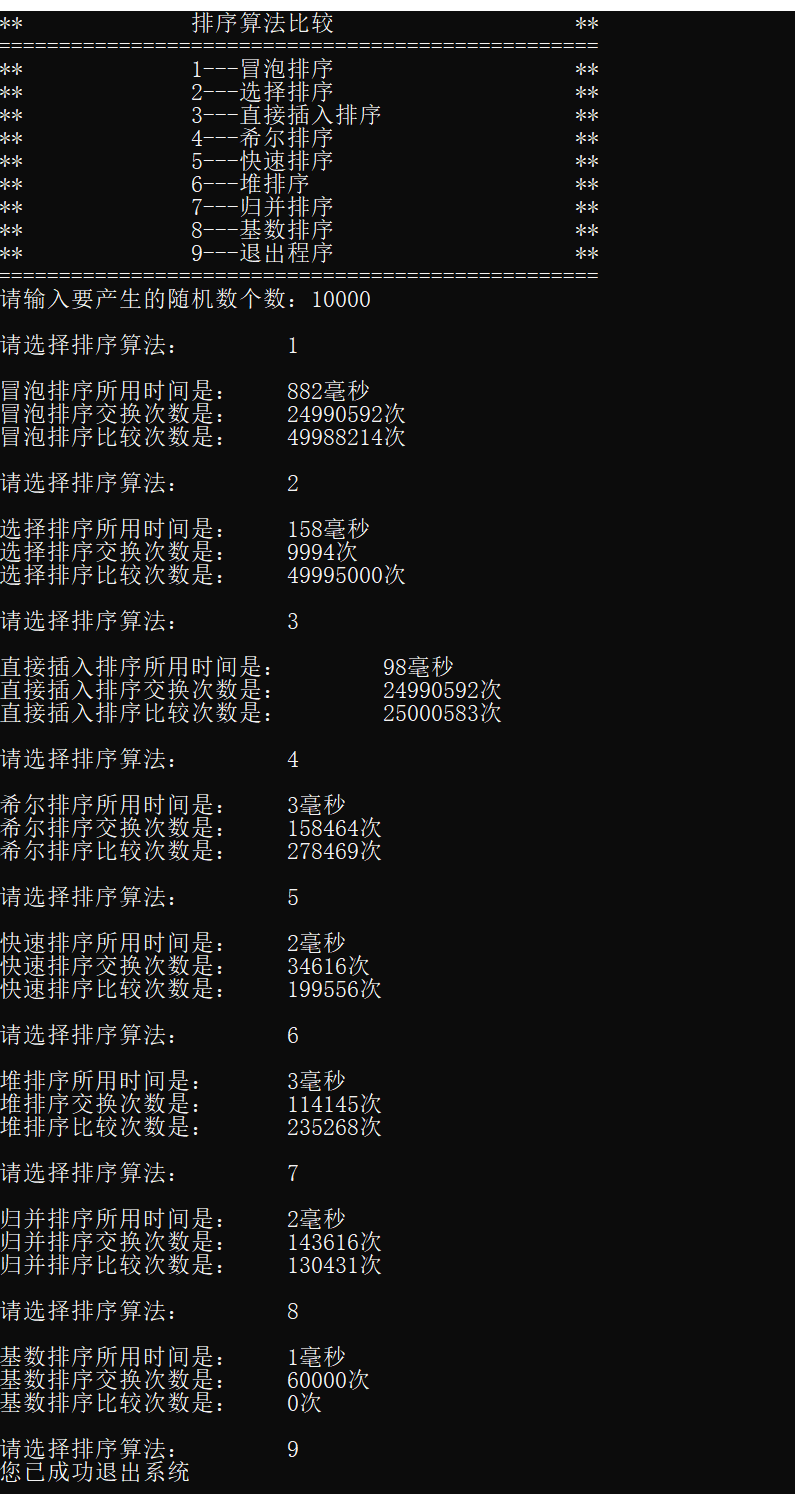


3.9.2 总体功能核心代码

SortCompareSystem类的runSortCompareSystem ()函数（仅截取部分）



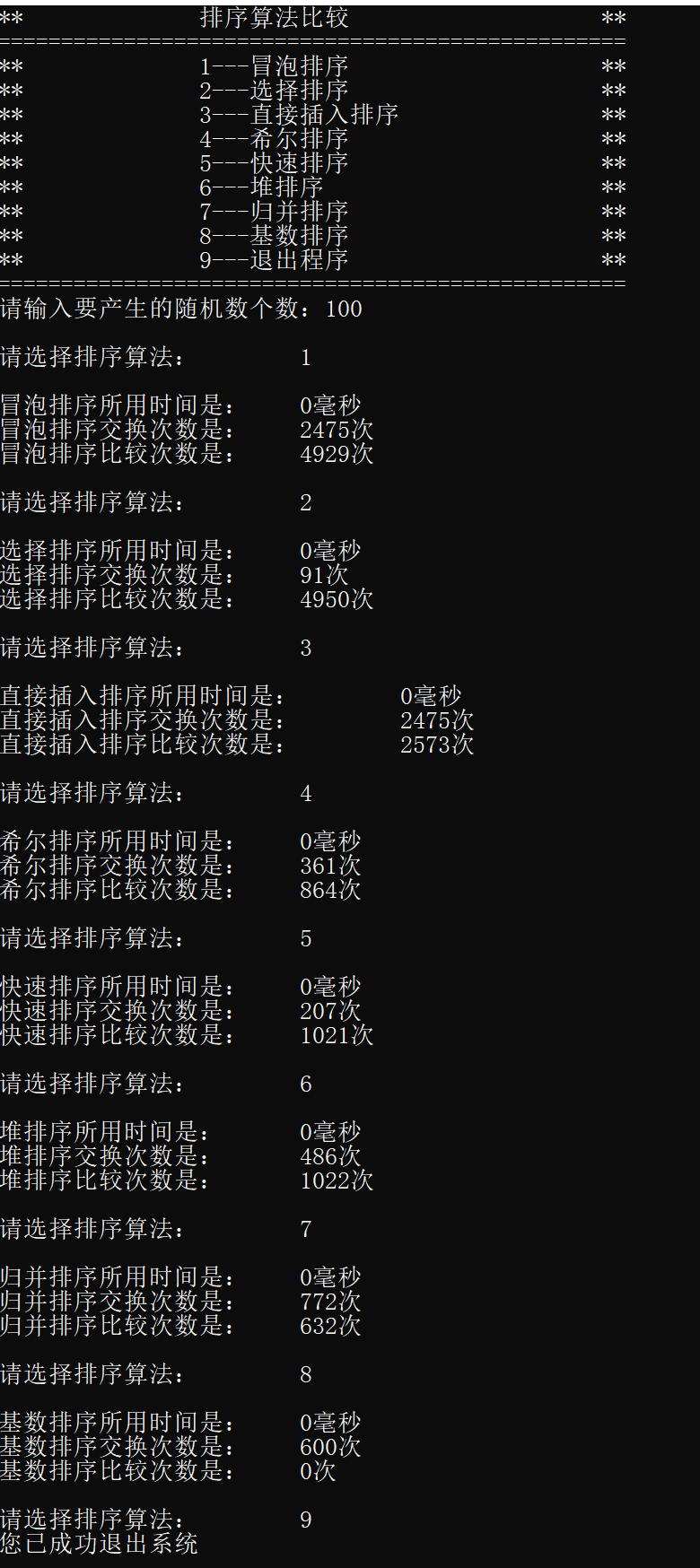
3.9.3 总体功能截屏示例



4 测试

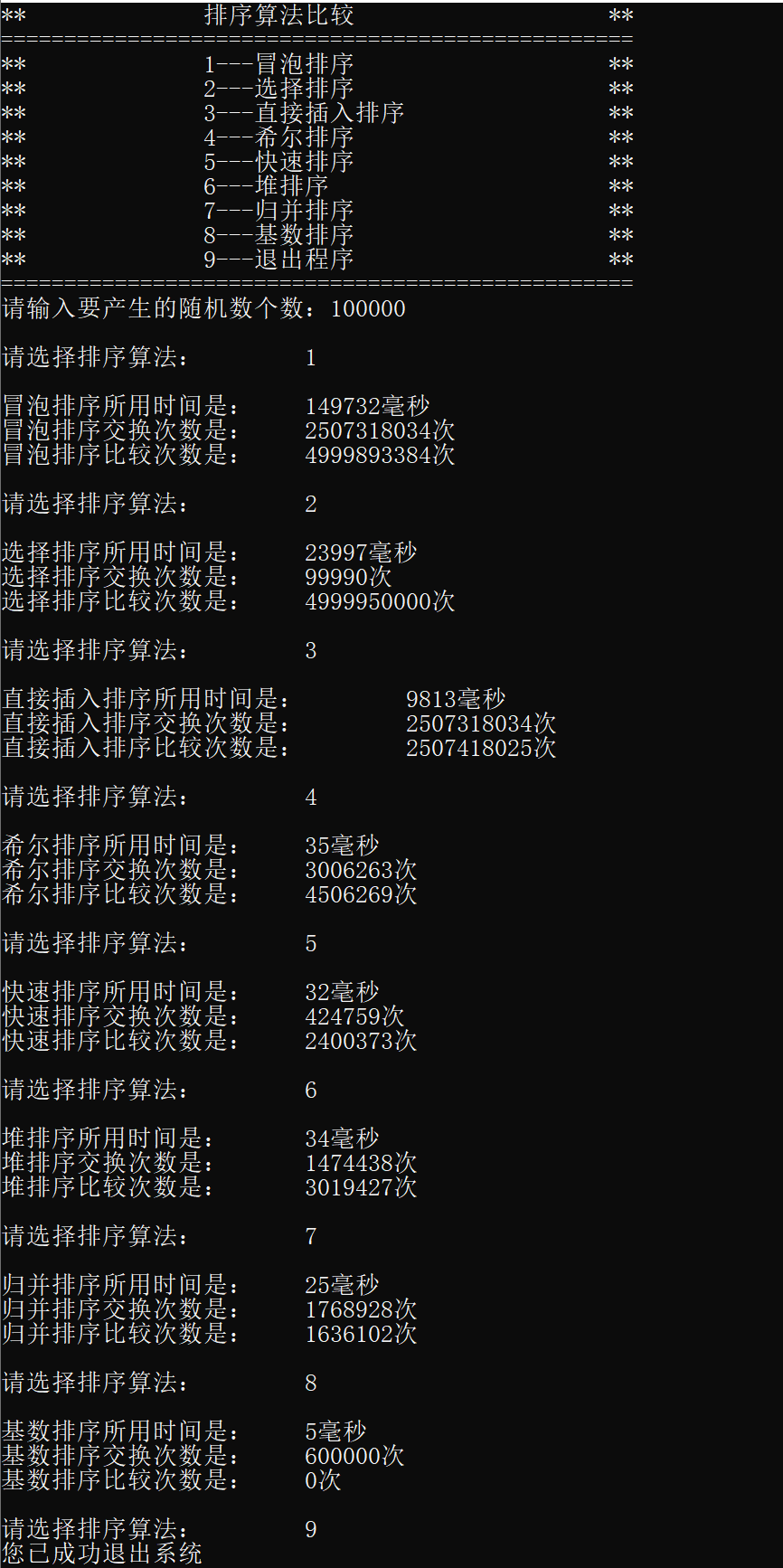
4.1 小范围测试

**测试用例**：100个随机数排序



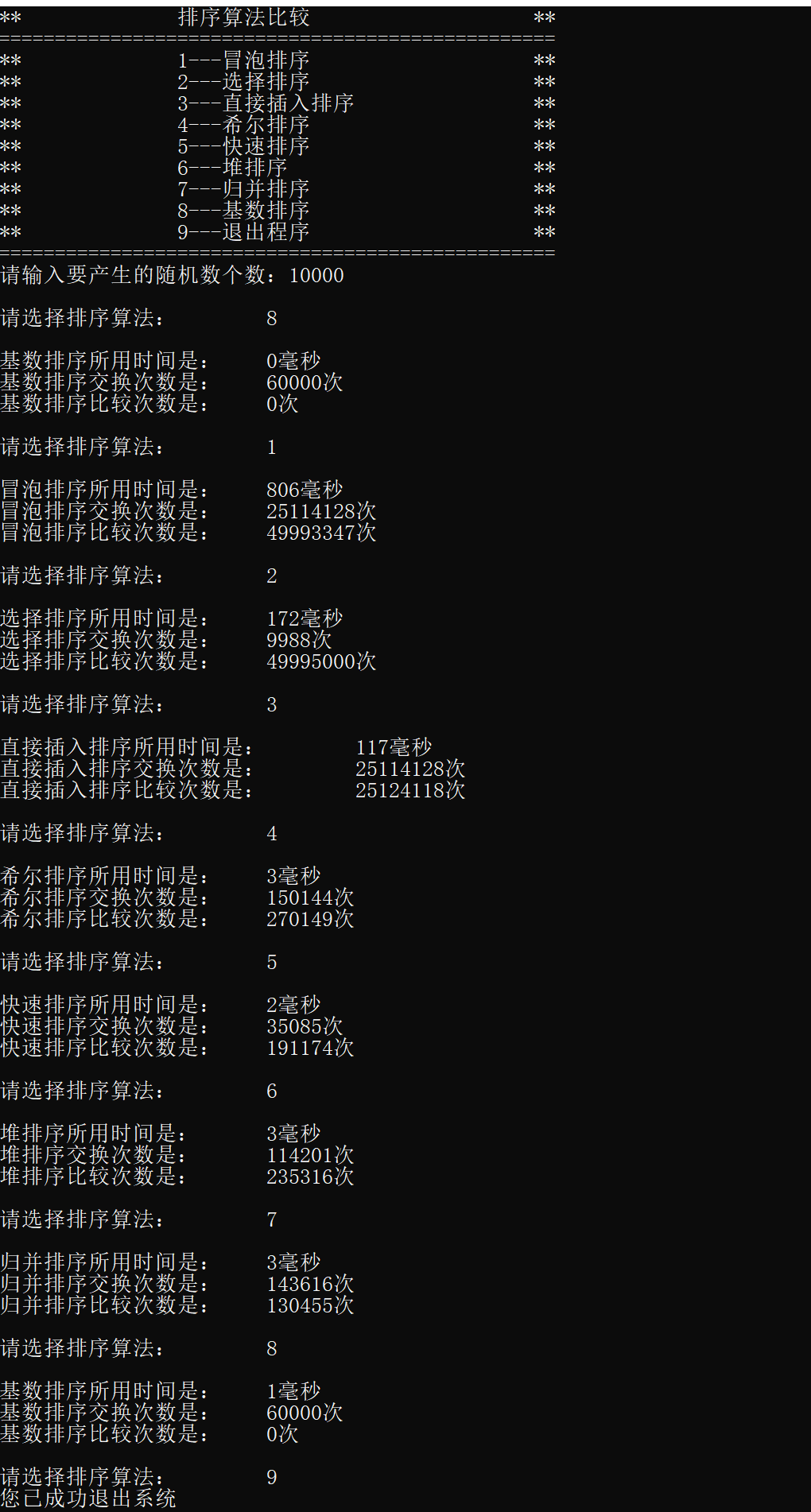
4.2 大范围测试

**测试用例**：100000个随机数排序



4.3 已有序数据测试

**测试用例**：10000个随机数先使用基数排序排成有序，然后再依次使用8种方法排序，对比所用时间与比较、交换次数。



5 案例总结

1. 排序算法的主要性能评价标准为排序交换次数与排序比较次数，此外排序的稳定性也是评价排序算法的一项重要因素。
2. 不同排序算法会因数据量大小和数据有序程度不同，性能表现发生差异。针对于不同的应用场景，应当选择不同的排序方案。对于数据量较小的情形，可以使用直接插入排序，冒泡排序，选择排序等基本排序算法，他们只要一个辅助元素即可完成排序。对于数据量较大的情形，则可以考虑使用希尔排序、快速排序、堆排序、归并排序和基数排序等方法。
3. 在排序时，可以结合不同的排序算法，使得排序对多种情形有更好的适应性。例如在归并排序中对小规模子数组使用插入排序可以使得运行时间缩短。
4. 对于特定排序算法，可以在原有排序算法的基础上，改进内部细节，实现对算法的优化。例如希尔排序中使用不同的gap序列对性能有较大影响，可以分析不同的gap序列，对希尔排序进行优化。