**数据结构课程设计**

**项目说明文档**

**十一种排序算法的比较**

作 者 姓 名： 苏家铭

学 号： 2151299

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 项目分析 1](#_Toc32459)

[1.1 项目背景 1](#_Toc1034)

[1.2项目要求 1](#_Toc26071)

[1.2.1 功能要求 1](#_Toc17736)

[1.2.2 输入要求 1](#_Toc19480)

[1.2.3 输出要求 1](#_Toc27948)

[1.2.4 项目实例 1](#_Toc15000)

[2 项目设计及实现 2](#_Toc3274)

[2.1 数据结构设计思路 2](#_Toc16769)

[2.2 类设计 2](#_Toc20843)

[2.2.1 链表 2](#_Toc18353)

[2.3 项目算法 3](#_Toc8766)

[2.3.1 实现思路 3](#_Toc3110)

[2.3.2 代码实现 4](#_Toc3252)

[3 项目测试 4](#_Toc3943)

[3.1 一般情况 4](#_Toc31909)

[3.2 交集为空的情况 5](#_Toc3479)

[3.3 完全相交的情况 5](#_Toc26284)

[3.4 其中一个序列完全属于交集的情况 5](#_Toc13791)

[3.5 其中一个序列为空的情况 6](#_Toc32352)

[3.6 输入非法字符的情况（健壮性） 6](#_Toc29253)

[4 算法性能分析 7](#_Toc8015)

[4.1 正确性 7](#_Toc27442)

[4.2 可使用性 7](#_Toc1775)

[4.3 可读性 7](#_Toc3126)

[4.4 效率 7](#_Toc11991)

[4.5 健壮性 7](#_Toc15299)

# 1 项目分析

## 项目背景

排序问题在日常生活中无处不在，小到把一个班级的学生按学号排序，大到按国家实力将国家发展水平排序。虽然排序这一过程开始都是一个无序序列，而最终都会化为一个升序或者降序序列。但是不同排序的方法有不同的效果，所用时间，复杂程度也千差万别。

对于初学算法的代码新手来说，研究不同排序算法的优缺点具有重要意义。研究出最合适的排序算法可以极大提高工作效率，在大数据背景下焕发巨大的社会效益。

## 1.2项目要求

### 1.2.1 功能要求

随机函数产生一百，一千，一万和十万个随机数，用快速排序，直接插入排序，冒泡排序，选择排序的排序方法排序，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。

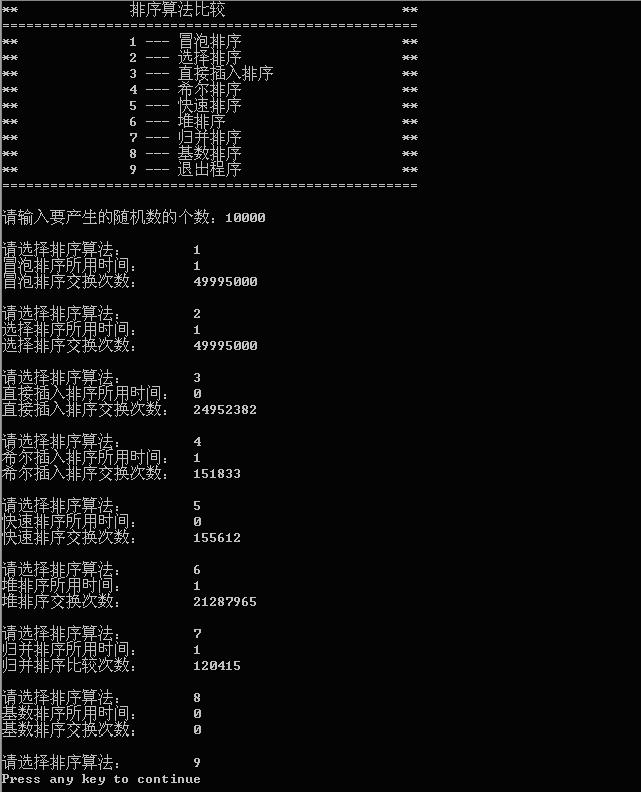
### 1.2.2 输入要求

产生随机数的个数。

### 1.2.3 **输出要求**

打印各排序算法的排序时间、交换次数和比较次数。

### 1.2.4 项目实例



# 2 **项目设计及实现**

本项目实现了11种排序算法：

1. **冒泡排序**
2. **冒泡排序（优化版）**
3. **选择排序**
4. **选择排序（优化版）**
5. **直接插入排序**
6. **希尔排序**
7. **快速排序**
8. **堆排序**
9. **归并排序**
10. **MSD基数排序**
11. **LSD基数排序**

该版块是这十一种排序算法的算法设计、代码实现已经算法分析。

## 2.1 冒泡排序

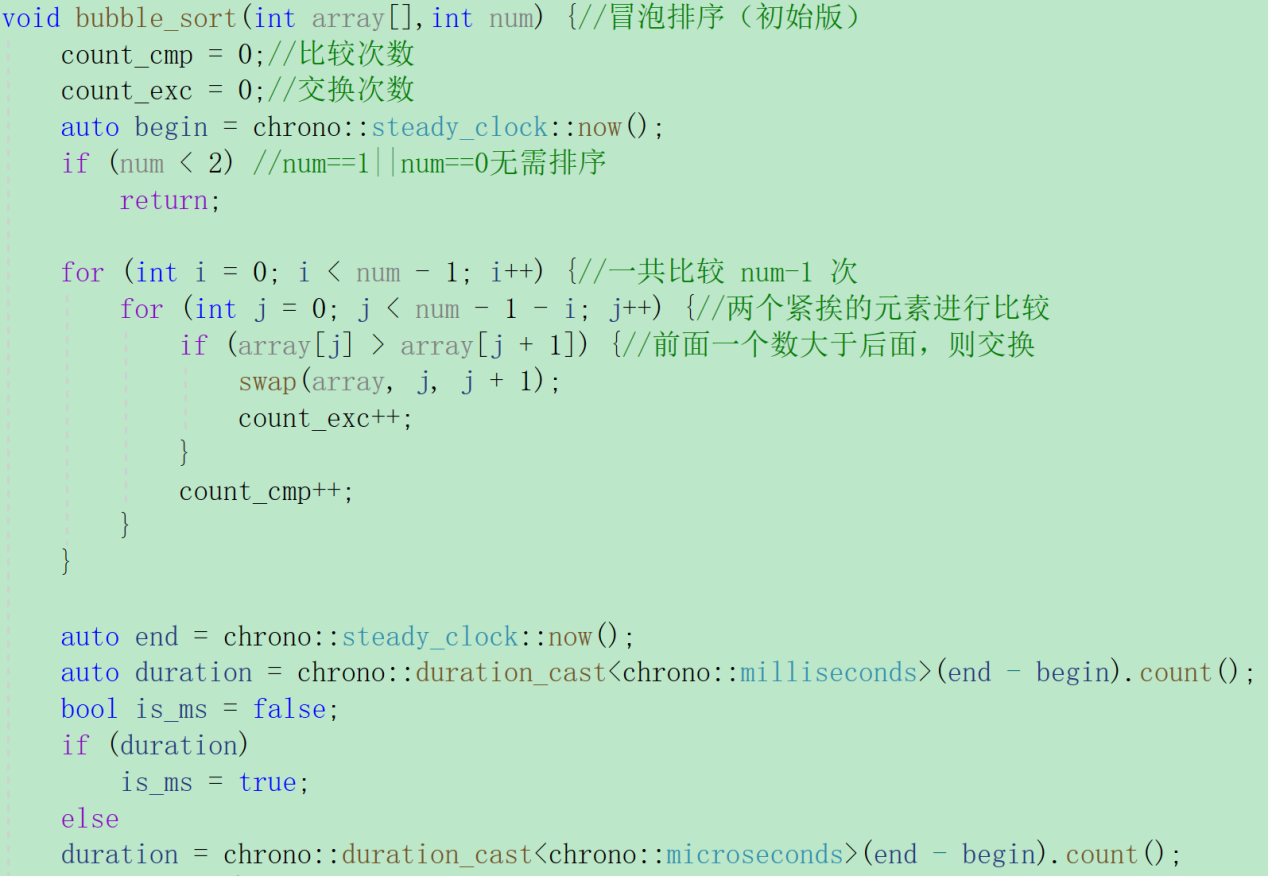
### 2.1.1 算法设计

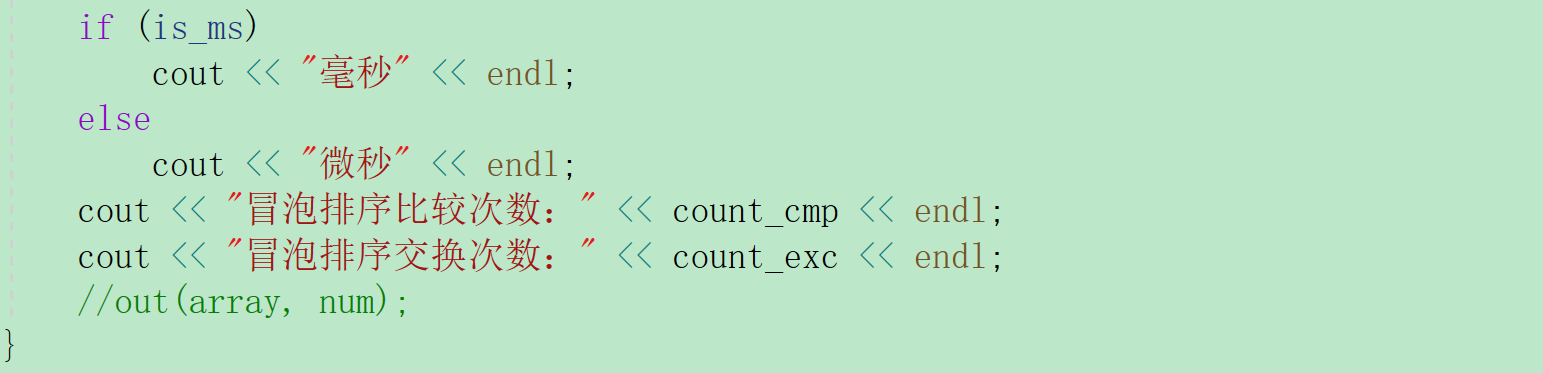
冒泡排序（Bubble Sort）是一种简单直观的排序算法。假设长度为n的数组arr[]，要按照从小到大排序。则冒泡排序的具体过程可以描述为：首先从数组的第一个元素开始到数组最后一个元素为止，对数组中相邻的两个元素进行比较，如果位于数组左端的元素大于数组右端的元素，则交换这两个元素在数组中的位置。这样操作后数组最右端的元素即为该数组中所有元素的最大值。接着对该数组除最右端的n-1个元素进行同样的操作，再接着对剩下的n-2个元素做同样的操作，直到整个数组有序排列。

冒泡排序算法的原理如下：

1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
2. 对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。在这一点，最后的元素应该会是最大的数。
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。

### 2.1.2 算法代码





### 2.1.3 算法分析

若初始序列是有序的，则比较次数为O(n2)，交换次数为0；

若初始系列是逆序的，则比较次数为O(n2),交换次数为O(n2)；

所以冒泡排序最好的时间复杂度和最坏的时间复杂度都是O(n2);

对于冒泡排序来说，每次交换都是相连的两个元素进行的，没有发生长距离的交换，所以不存在相同关键码元素会倒置的情况，即冒泡排序是一种稳定的算法。

冒泡排序适用于数据量很小的排序场景，因为冒泡的实现方式较为简单。

## 2.2 冒泡排序（优化版）

### 2.2.1 算法设计

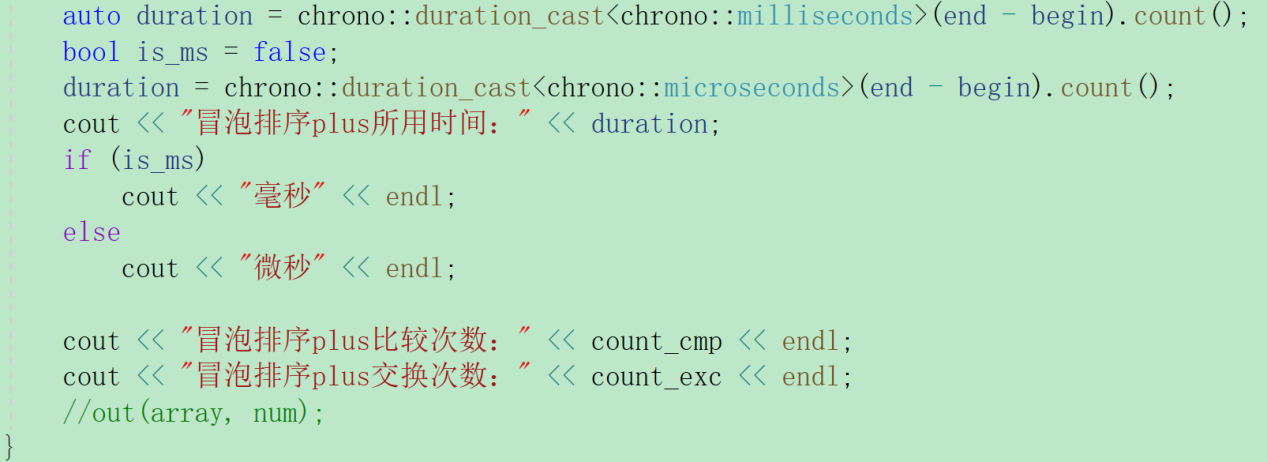
鉴于冒泡排序不管最好情况与最坏情况比较次数都是O(n2)，考虑到在某些情况下其实并不需要比较n-1趟起泡就可以完成排序，例如，在一个完全有序情况下，只需要比较n-1次即可完成排序。

不仅如此，普通冒泡排序还有一个缺点，对于每一趟比较时，很多情况下，只是前面一部分序列需要排序，而后面一部分序列已经处于有序的情况，不需要比较和交换。

针对上述两个缺点，我们对冒泡排序进行优化：

1. 定义变量sort\_border，用于指示无序序列的边界
2. 定义变量is\_order，用于指示本趟冒泡排序结果是否发生交换，如果没有则可以说明此事已经排好序。

### 2.2.2 算法代码



### 2.2.3 算法分析

若初始序列是部分有序甚至完全有序的，则比较次数可以降低为O(n)；

对于冒泡排序来说，每次交换都是相连的两个元素进行的，没有发生长距离的交换，所以不存在相同关键码元素会倒置的情况，即冒泡排序是一种稳定的算法。

## 2.3 选择排序

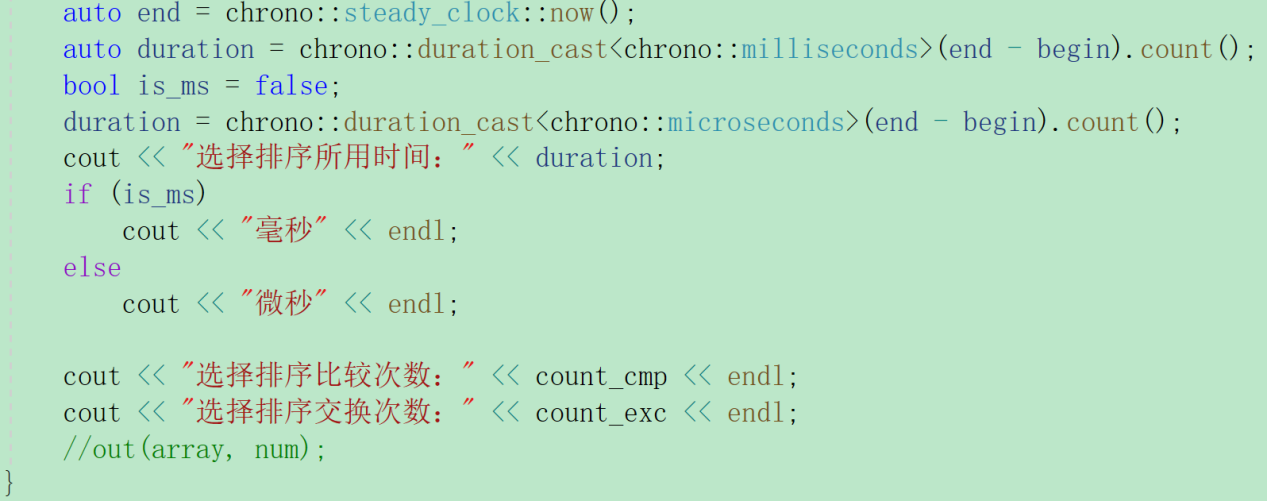
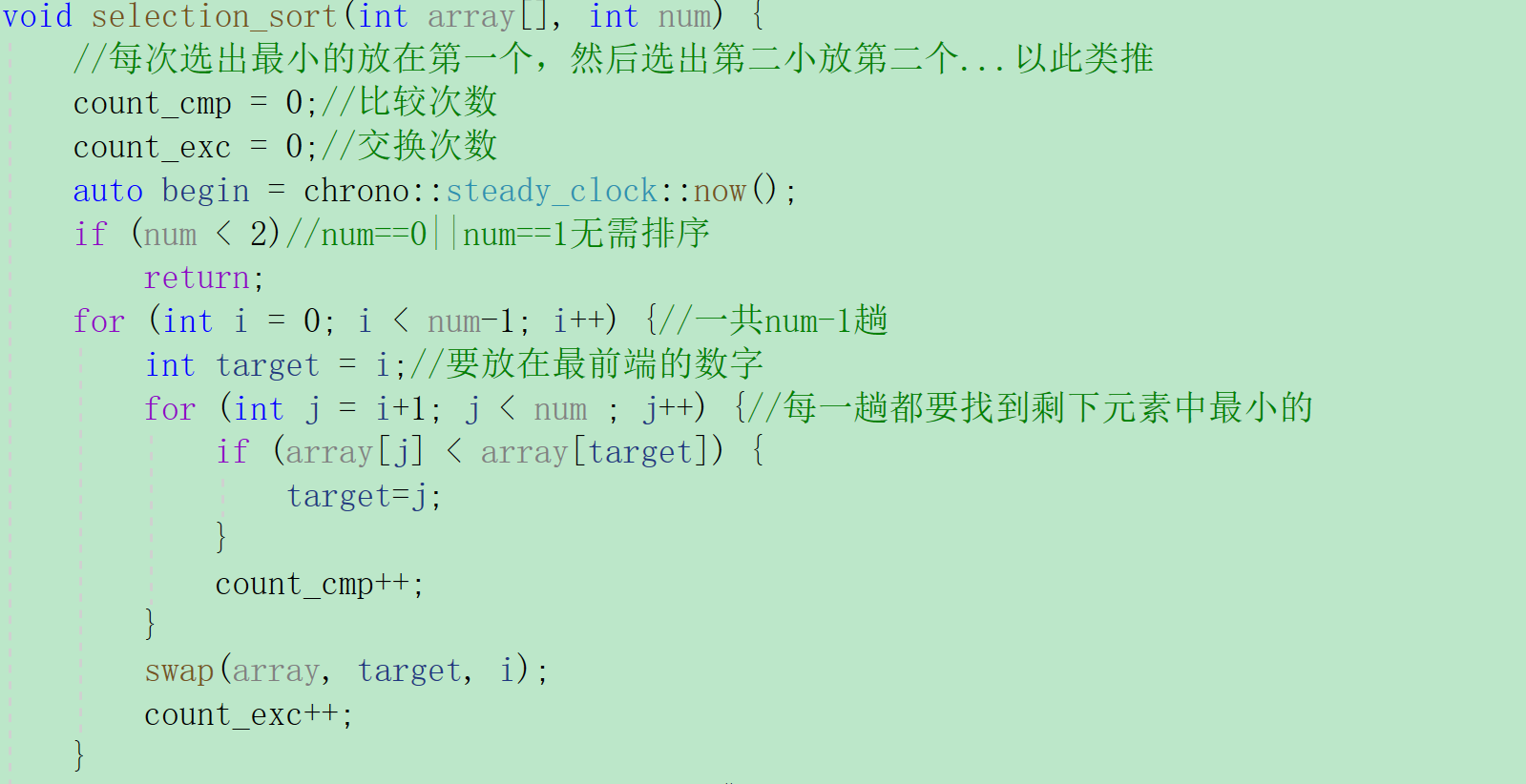
### 2.3.1 算法设计

选择排序（Selection sort）是一种简单直观的排序算法，简称为选排。它的工作原理是每一趟从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。

选择排序算法通过选择和交换来实现排序，其排序流程如下：

1. 首先从原始数组中选择最小的（或最大的）1个数据，将其和位于第1（n）个位置的数据交换。
2. 接着从剩下的n-1个数据中选择次小的1个元素，将其和第2（n）个位置的数据交换。
3. 然后，这样不断重复，直到最后两个数据完成交换。最后，便完成了对原始数组的从小到大的排序。

### 2.3.2 算法代码



### 2.3.3 算法分析

选择排序比较次数为O(n2），与关键字的初始状态无关，总的比较次数N=(n-1）+(n-2）+...+1=n\*(n-1）/2。交换次数O(n），最好情况是，已经有序，交换0次；最坏情况交换n-1次，逆序交换n/2次。交换次数比[冒泡排序](https://baike.baidu.com/item/%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F?fromModule=lemma_inlink" \t "https://baike.baidu.com/_blank)少，由于交换所需CPU时间比比较所需的CPU时间多，n值较小时，选择排序比冒泡排序快。

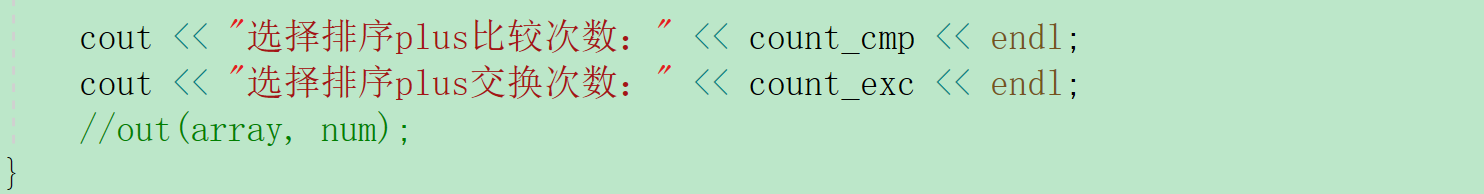
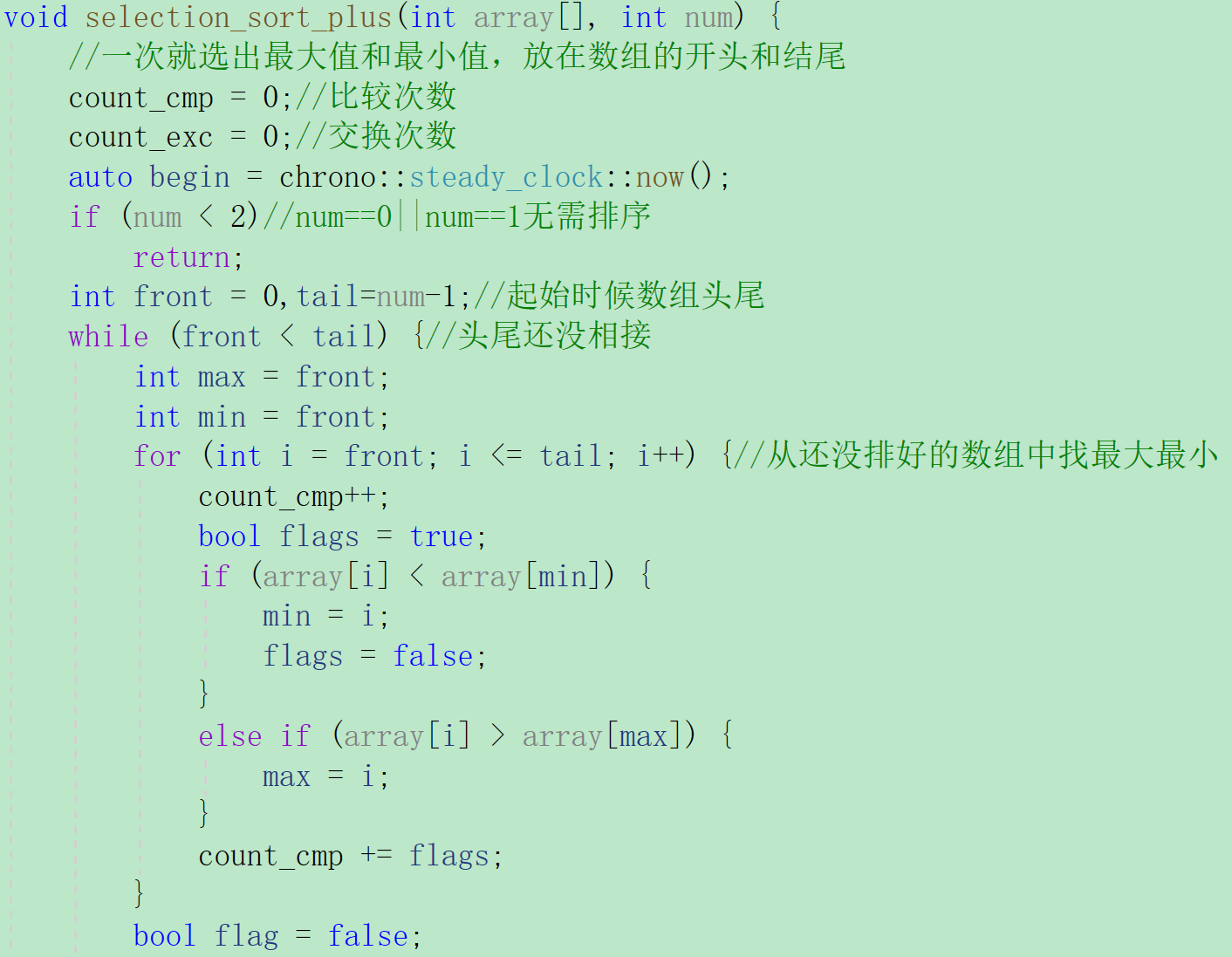
对于选择排序来说，在一趟选择，如果一个元素比当前元素小，而该小的元素又出现在一个和当前元素相等的元素后面，发生了长距离的交换，那么交换后稳定性就被破坏了，即选择排序是一种不稳定的算法。

## 2.4 选择排序（优化版）

### 2.4.1 算法设计

在一般选择排序时每趟遍历中只挑选了最小的元素，但其实可以在这一趟遍历中把最大值也选出来。选择排序的优化思路一般是在一趟遍历中，同时找出最大值与最小值，放到数组两端，这样就能将遍历的趟数减少一半。

### 2.4.2 算法代码



### 2.4.3 算法分析

利用这种优化方法后，降低了选择排序的时间复杂度系数，循环次数减半。选择排序适用于待排序列元素个数较少时。

## 2.5 直接插入排序

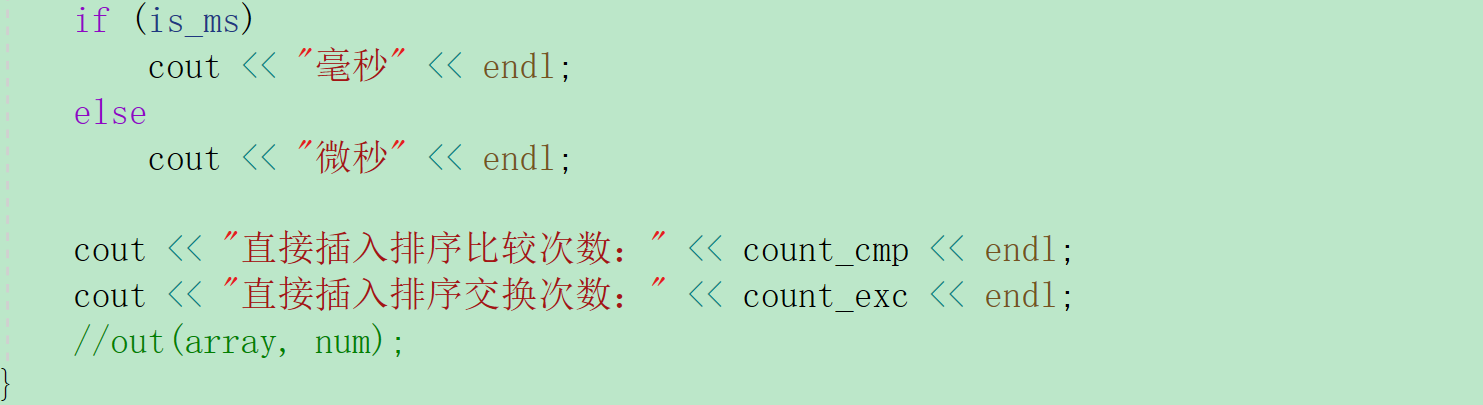
### 2.5.1 算法设计

直接插入排序也是一种常见的排序算法，插入排序的思想是：将初始数据分为有序和无序两个部分，每一次将无序部分的第一个数据插入到前面已经排好序的有序部分中，原来位置上的元素依次向后顺移，直到插完所有元素为止。

  插入排序的步骤如下：

1. 每次从无序部分中取出首元素，
2. 与有序部分中的元素从后向前依次进行比较，并找到合适的位置，
3. 将原本位置上的元素向后顺移，将该元素插到有序序列当中。
4. 重复以上三步，直到最后一个元素也插入有序序列。

### 2.5.2 算法代码



### 2.5.3 算法分析

* 当待排序序列是有序时，是最优的情况，只需当前数跟前一个数比较次，这时一共需要比较n- 1次，时间复杂度为O(n)。
* 当待排序数组是逆序的，此时需要比较次数最多，总次数记为：1+2+3+…+n-1，所以，插入排序最坏情况下的时间复杂度为O(n2)。

  综上，插入排序在平均情况运行时间是O(n2)。

直接插入排序是一种稳定的排序方式。适用于待排元素不多，且基本有序的情况。

## 2.6 希尔排序

### 2.6.1 算法设计

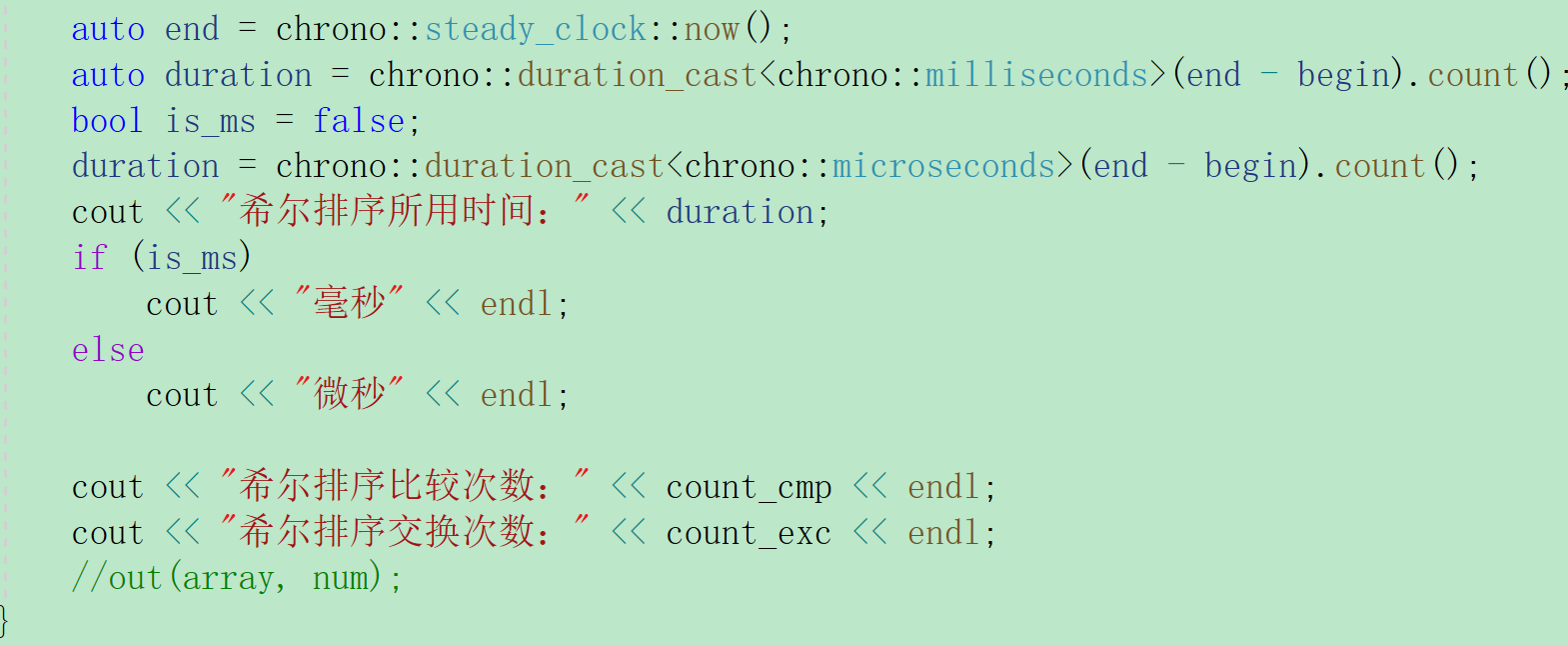
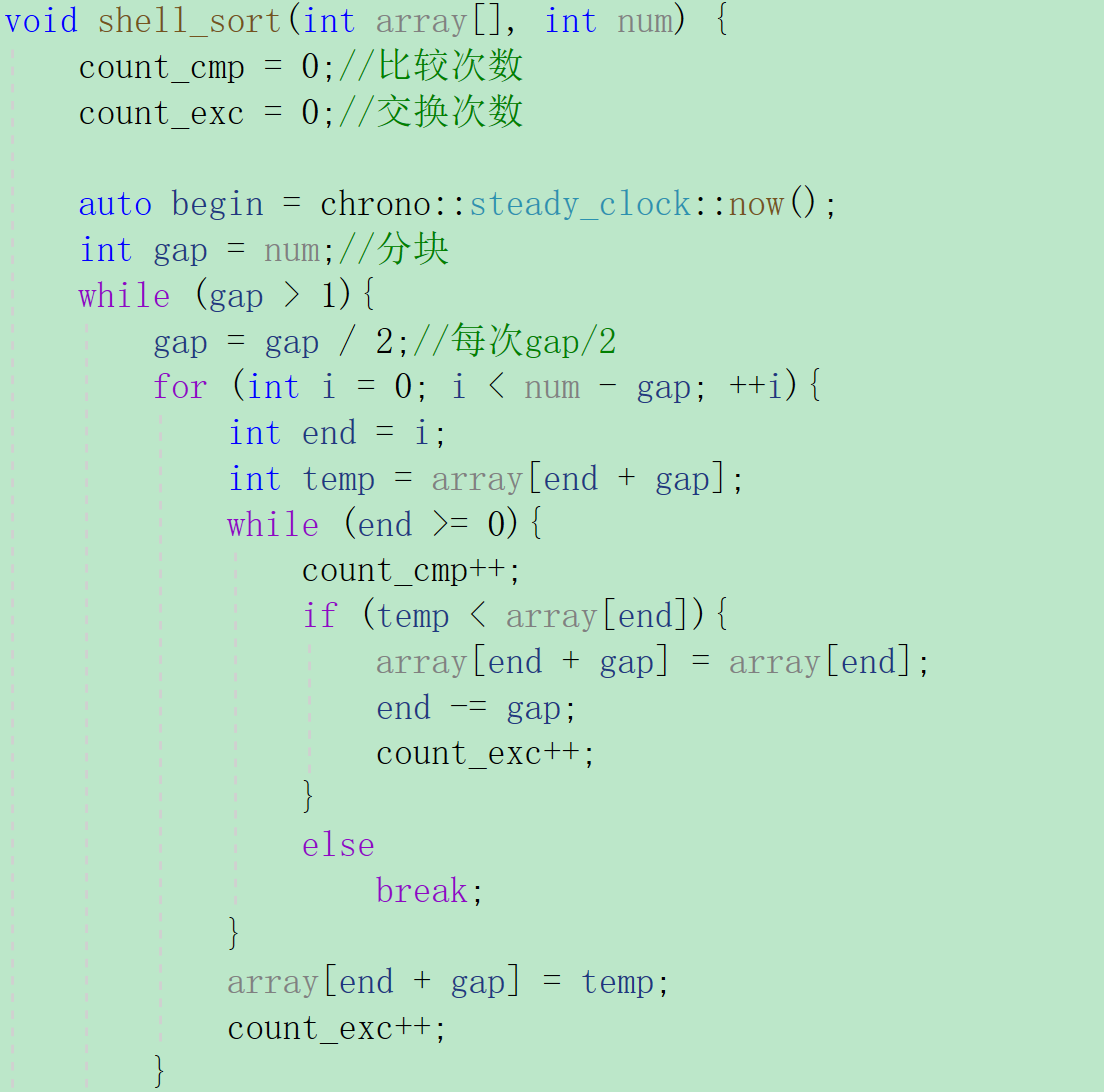
希尔排序(Shell sort)又称为缩小增量排序(diminishing-increment sort)，是对直接插入排序的一种升级迭代，优化之处在于：不用每次插入一个无序序列中的元素时，就和序列中前面有序部分中的所有元素进行比较，更加适用于直接插入排序的适用场景——元素较少且基本有序。

  该方法的基本思想是：设待排序元素序列有n个元素，首先取一个整数gap（gap<n）作为间隔，所有距离为gap的元素放在同一个子序列中（分成gap个子序列），在每一个逻辑数组中分别实行直接插入排序。然后缩小间隔gap，重复上述子序列划分和排序工作。直到最后取gap=1，将所有元素放在同一个序列中排序为止。

  综上所述，希尔排序的实现步骤如下：

1. 确定gap的值，划分子序列，子序列内进行直接插入排序。
2. 不断缩小gap，继续在每个子序列进行插入排序。
3. 重复第二步，直到gap==1，在包含所有元素的序列内进行直接插入排序。

### 2.6.2 算法代码



### 2.6.3 算法分析

本质上讲，[希尔排序算法](https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F/_blank)是直接插入排序算法的一种改进，减少了其复制的次数，速度要快很多,希尔排序的[时间](https://baike.baidu.com/item/%E6%97%B6%E9%97%B4?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F/_blank)的时间复杂度为O(n1.3)，希尔排序时间复杂度的下界是nlog2n。希尔排序对于中等大小规模表现良好，对规模非常大的[数据排序](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%8E%92%E5%BA%8F?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B8%8C%E5%B0%94%E6%8E%92%E5%BA%8F/_blank)不是最优选择。但是比O(n2)复杂度的算法快得多。并且希尔排序非常容易实现，算法代码短而简单。

## 2.7 快速排序

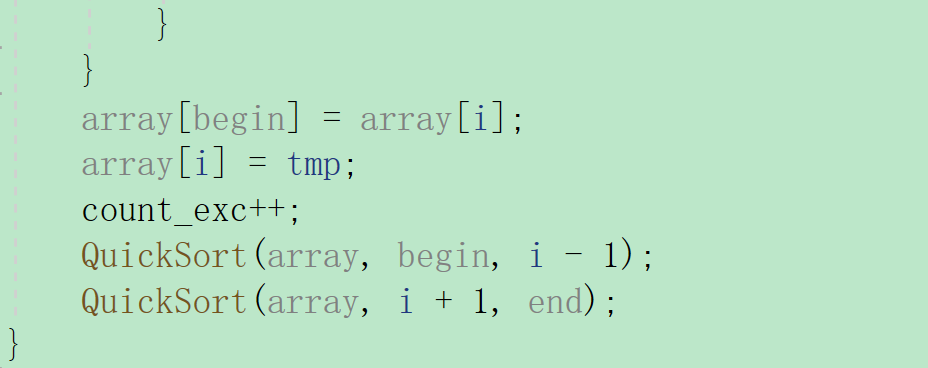
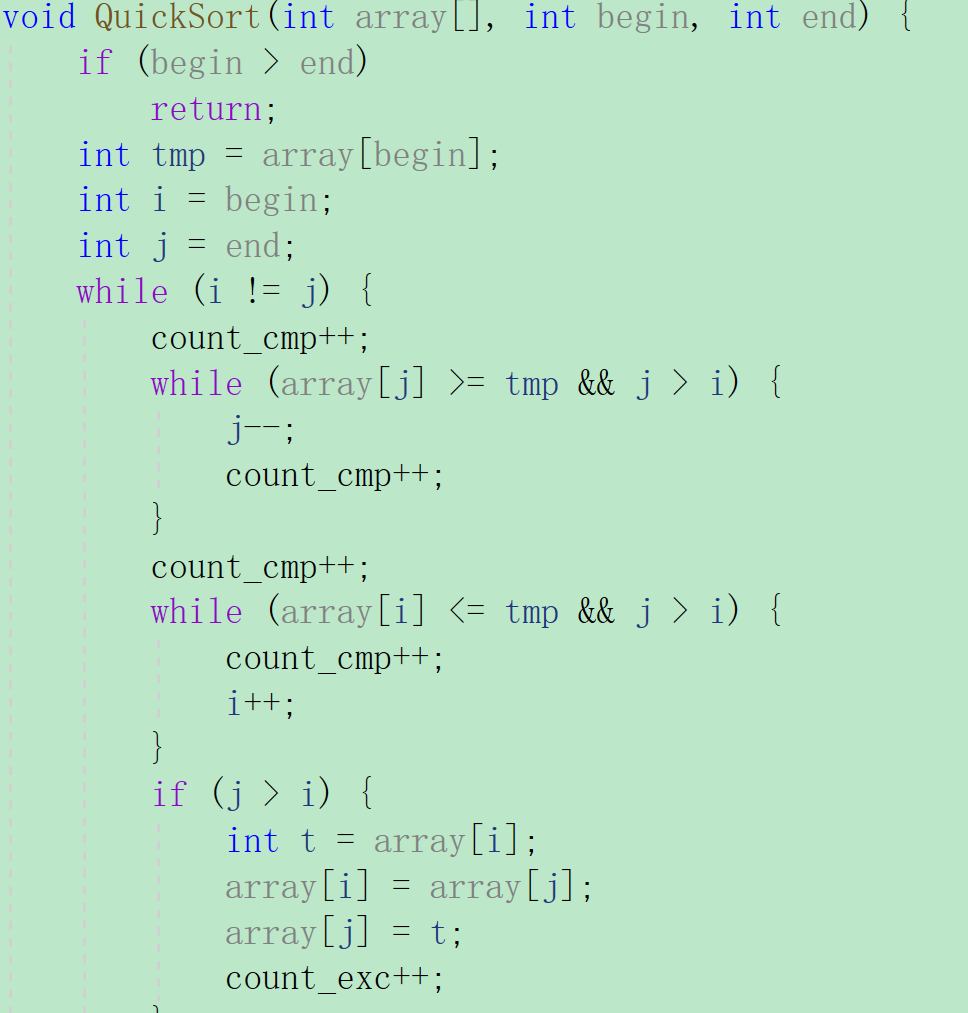
### 2.7.1 算法设计

[快速排序](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F/2084344?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F/_blank)(Quick sort)的基本思想是:通过一趟排序算法把所需要排序的序列的元素分割成两大块，其中，一部分的元素都要小于或等于另外一部分的序列元素，然后仍根据该种方法对划分后的这两块序列的元素分别再次实行[快速排序算法](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E6%8E%92%E5%BA%8F%E7%AE%97%E6%B3%95/369842?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%86%92%E6%B3%A1%E6%8E%92%E5%BA%8F/_blank)，排序实现的整个过程可以是递归的来进行调用，最终能够实现将所需排序的无序序列元素变为一个有序的序列。

本项目实现了左右指针法的快速排序，具体流程如下：

1. 选定一个最左边或是最右边的元素key。
2. 定义一个begin和一个end，begin从左向右走，end从右向左走。（需要注意的是：若选择最左边的数据作为key，则需要end先走；若选择最右边的数据作为key，则需要begin先走）。
3. 以最左边的值为key为例：在走的过程中，若end遇到小于key的数，则停下，begin开始走，直到begin遇到一个大于key的数时，将begin和right的内容交换，end再次开始走，如此进行下去，直到begin和end最终相遇，此时将相遇点的内容与key交换即可。
4. 此时key的左边都是小于key的数，key的右边都是大于key的数。
5. 将key的左序列和右序列再次进行这种单趟排序，如此反复操作下去，直到左右序列只有一个数据，或是左右序列不存在时，便停止操作，此时得到有序序列。

### 2.7.2 算法代码



### 2.7.3 算法分析

快速排序的平均时间复杂度为O(nlog2n)，由于快速排序是递归的需要有一个栈存放每层递归调用时的指针和参数，最大递归调用层数与递归树的深度一致，因此要求存储开销为O(log2n)。

快速排序适用于待排序序列元素较多，并且元素较无序的情况，并且当元素个数很小时，快速排序往往比其他简单排序方法还要慢。

## 2.8 堆排序

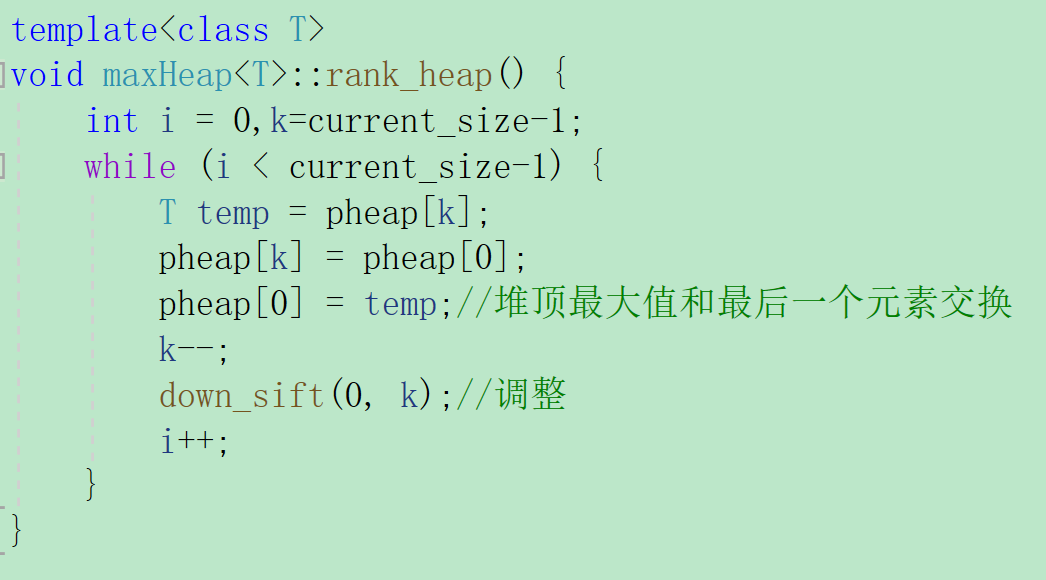
### 2.8.1 算法设计

[堆排序](https://so.csdn.net/so/search?q=%E5%A0%86%E6%8E%92%E5%BA%8F&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/lcshen1234/article/details/_blank)(heap sort)是一种重要的选择排序方法，它只需要一个记录大小的辅助存储空间，每个待排序的记录仅占用一个记录大小的存储空间，因此弥补了树形选择排序的弱点。一般升序采用大顶堆，降序采用小顶堆。

堆排序的基本流程如下：

1. 将这n条记录按关键字值的大小建立堆（称为初始堆）。
2. 然后将堆顶的元素，与最后一个元素交换，获得最大元素。
3. 接着将剩余无序序列，调整成小顶堆，重复步骤2，获得每一次调整后的最大值
4. 直到调整到根，从而获得排序序列

### 2.8.2 算法代码



### 2.8.3 算法分析

空间复杂度：需要一个记录辅助存储空间，O(1)

最好、最坏时间复杂度：O(nlog2n)

堆排序涉及长距离交换，是一种不稳定的排序算法。

## 2.9 归并排序

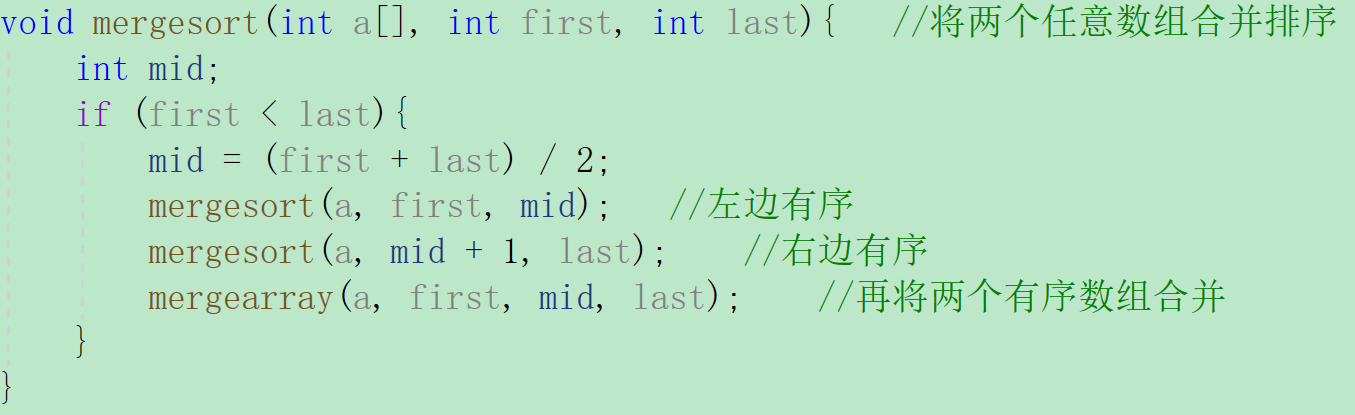
### 2.9.1 算法设计

归并排序（Merge Sort）是建立在归并操作上的一种有效、稳定的排序算法，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，称为二路归并。

首先将整个无序序列通过递归分为一个个子序列，以每个子序列只有一个元素或只有两个元素作为终点，在合并时将一个个有序的子序列按照一定规则合并成一个个有序的长序列。即将含有n个记录的初始序列中的每个记录均视为长度为1的子序列，再将这n个子序列两两合并得到n/2个长度为2的有序子序列；将上述步骤重复操作，直至得到1个长度为n的有序长序列。

在进行元素比较和交换时，若两个元素大小相等则不必交换位置，因此该算法不会破坏序列的稳定性，即归并排序也是稳定的排序算法。

### 2.9.2 算法代码



### 2.9.3 算法分析

归并排序速度仅次于快速排序。归并算法是一个不断递归的过程，假设数组的元素个数是n。时间复杂度是T(n)的函数: T(n) = 2\*T(n/2) + O(n)，整体的复杂度为O(nlog2n)。因为归并排序需要一个与原数组相同长度的数组做辅助来排序，所以空间复杂度为O(n)。

## 2.10 MSD基数排序

### 2.10.1 算法设计

基数排序又被称为桶排序。而基数排序首先将待排序数据元素依次“分配”到不同的桶里，然后再把各桶中的数据元素“收集”到一起。通过使用对多关键字进行排序的这种“分配”和“收集”的方法，基数排序实现对单关键字进行排序。

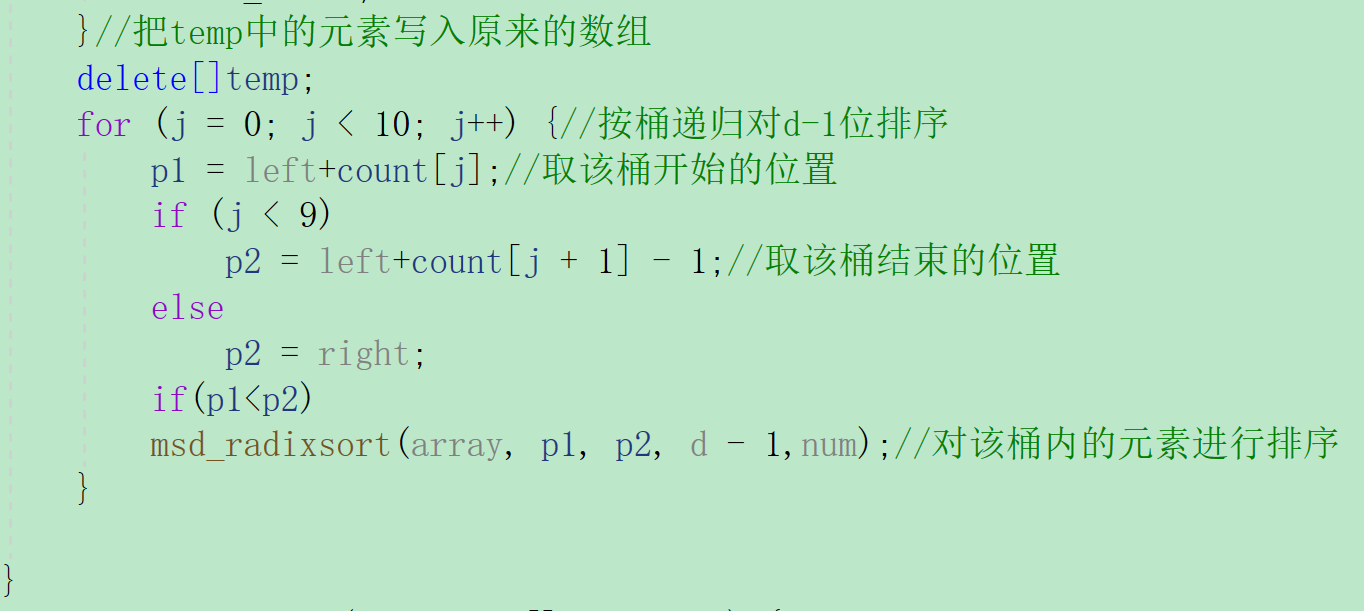
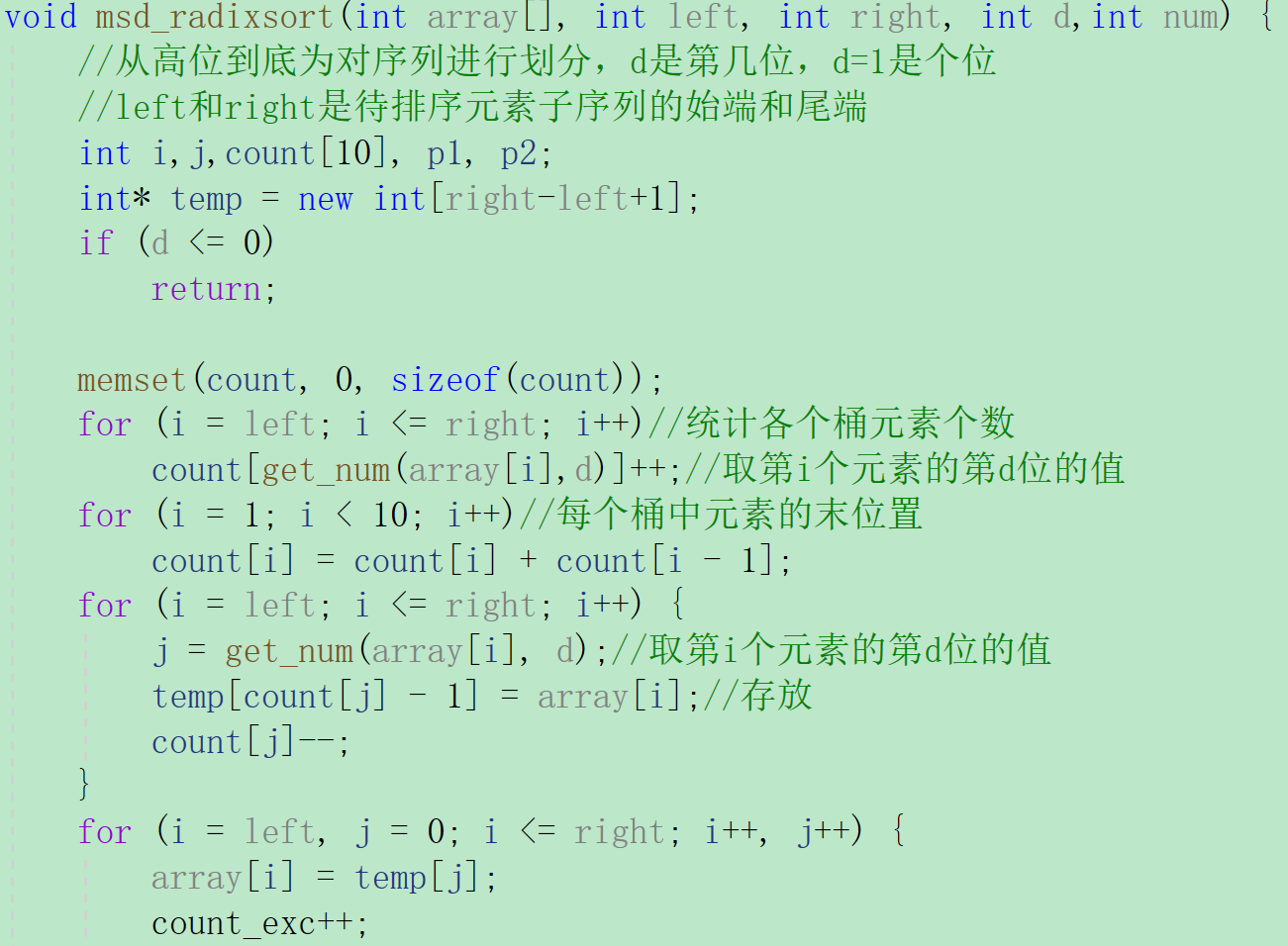
基数排序通常分为最高位优先法和最低位优先法。

最高位优先法MSD (Most Significant Digit first)通常是一个递归的过程：

设每个元素的关键字为k1k2k3k4...kd

1. 首先根据最高位关键字k1进行排序，按k1值的不同，将整个数据表分成若干个子表，每个子表中的数据元素具有相同的关键字k1。
2. 然后分别对每一个每个子表中的数据元素根据关键字(k2，…，kd)用最高位优先法进行排序。
3. 不断递归，直到对关键字kd完成排序为止。

### 2.10.2 算法代码



### 2.10.3 算法分析

* 平均时间复杂度：O(d\*n) (d为整数的最高位数)
* 空间复杂度:O(10n)
* MSD基数排序是一种稳定的算法，适用于位数比较多的序列。

## 2.11 LSD基数排序

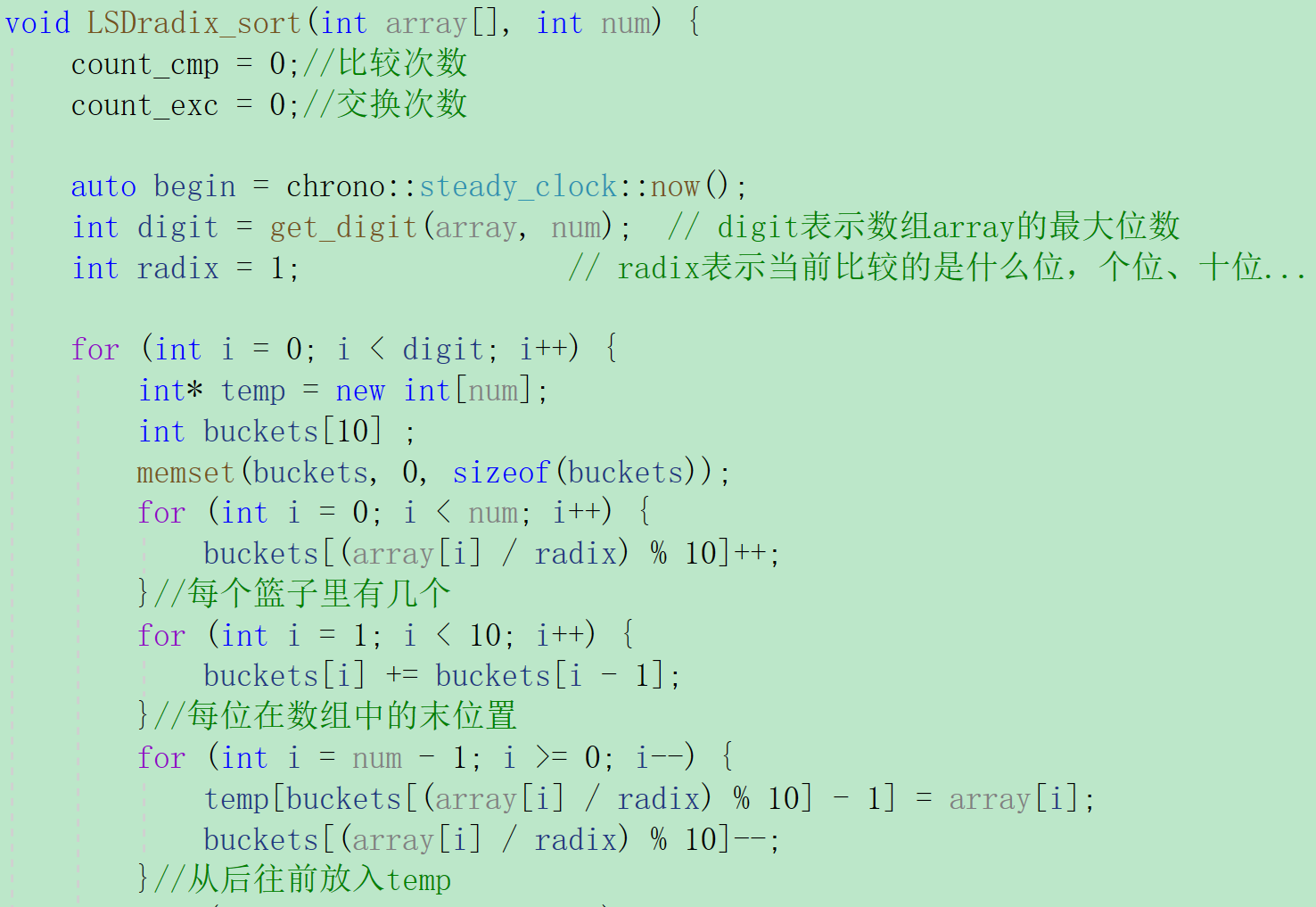
### 2.11.1 算法设计

最低位优先法LSD (Most Significant Digit first)流程：

设每个元素的关键字为k1k2k3k4...kd

1. 首先依据最低位关键字kd对数据表中所有数据元素进行一趟排序;
2. 然后依据次低位关键字kd-1对上一趟排序的结果再排序。
3. 依次重复，直到按关键字k1完成最后一趟排序。

### 2.11.2 算法代码



### 2.11.3 算法分析

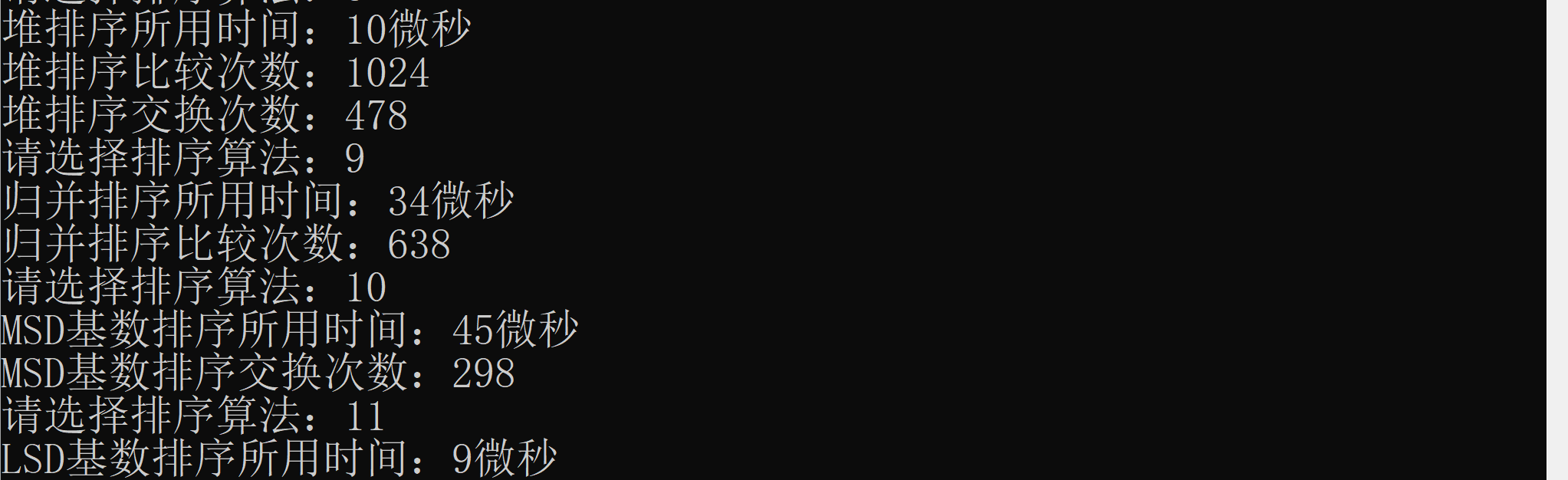
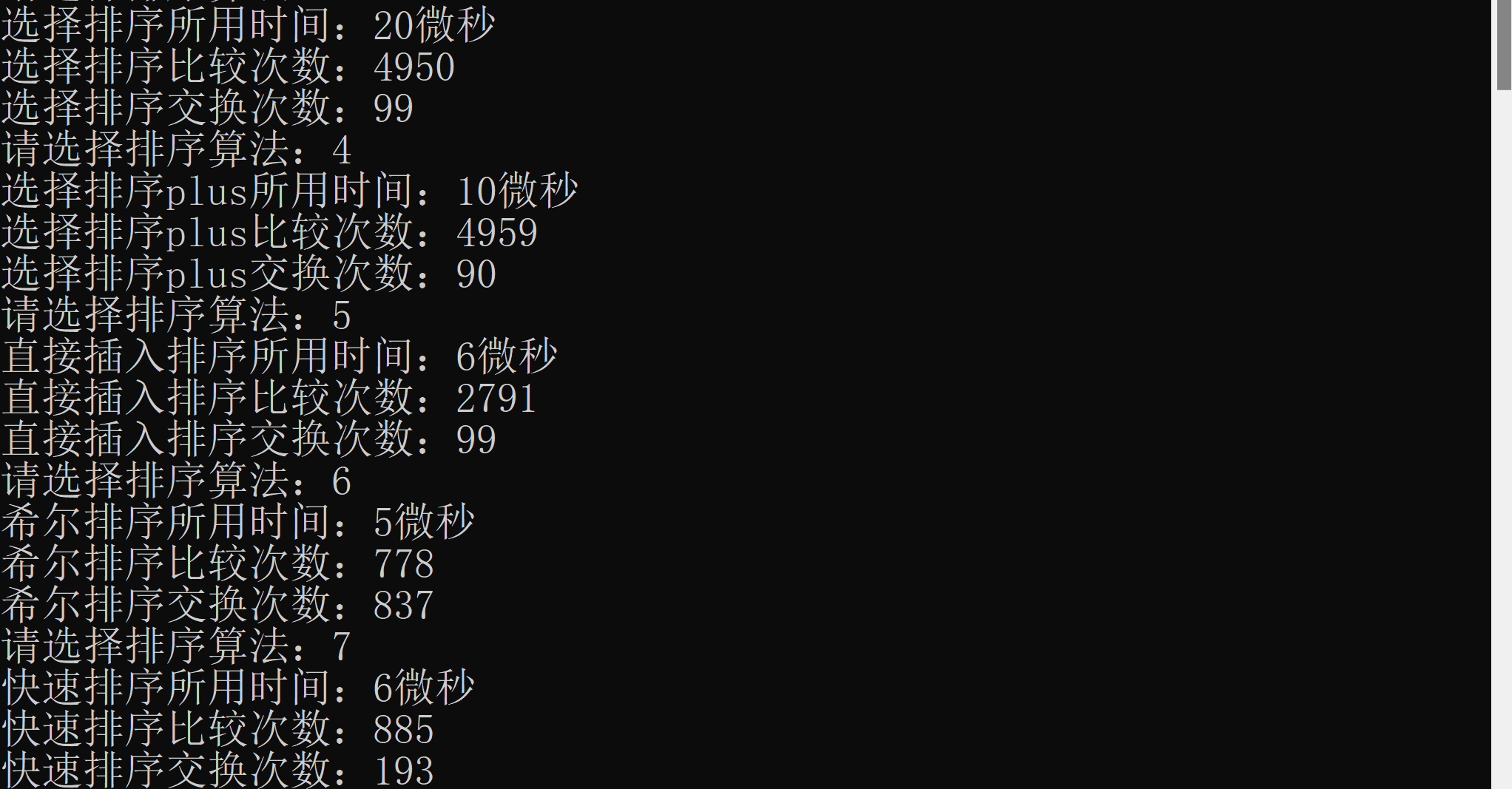
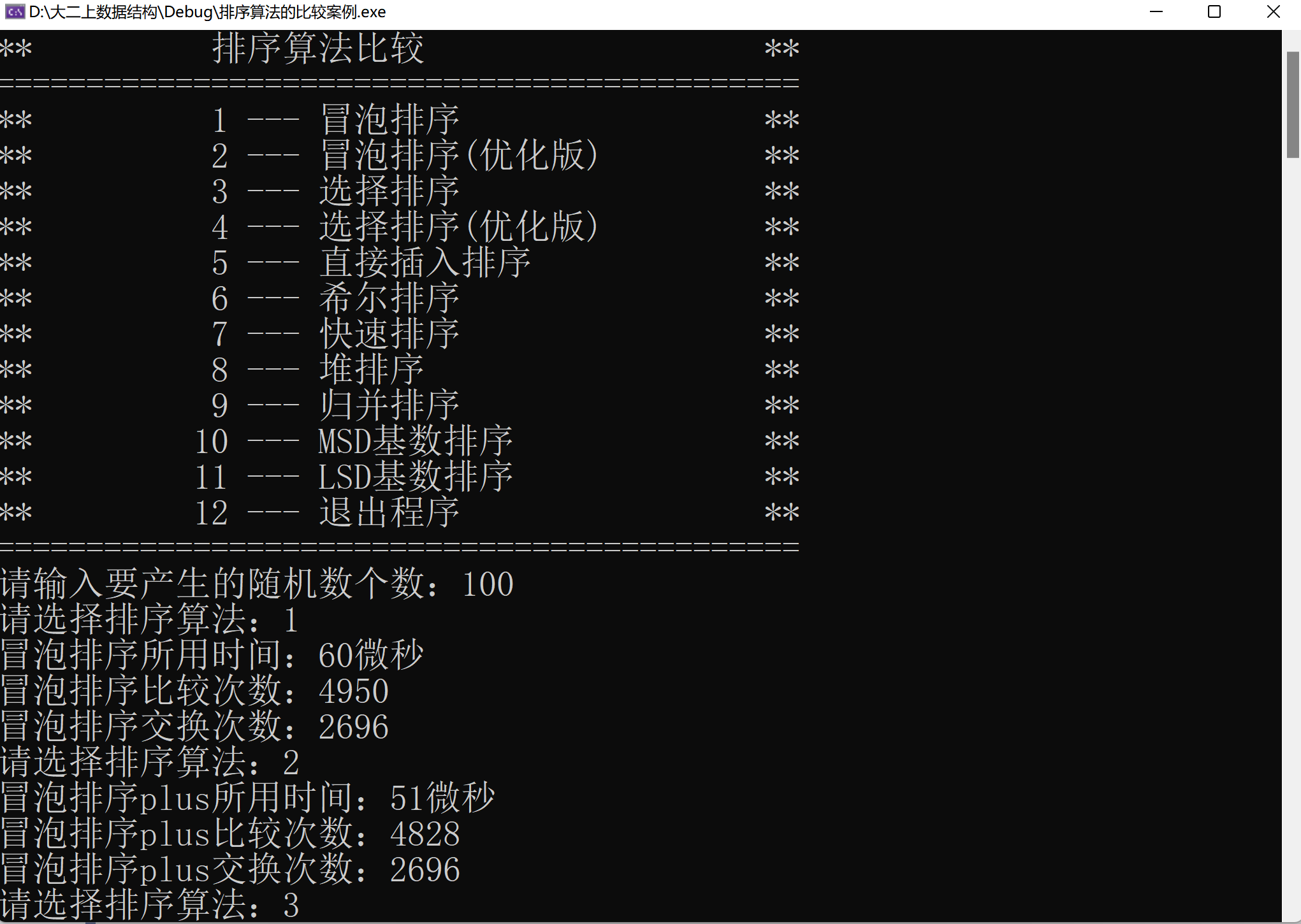
* 平均时间复杂度：O(d\*(n+r)) (d为整数的最高位数)
* 空间复杂度:O(n+r)
* LSD基数排序是一种稳定的算法，适用于位数比较少的序列。

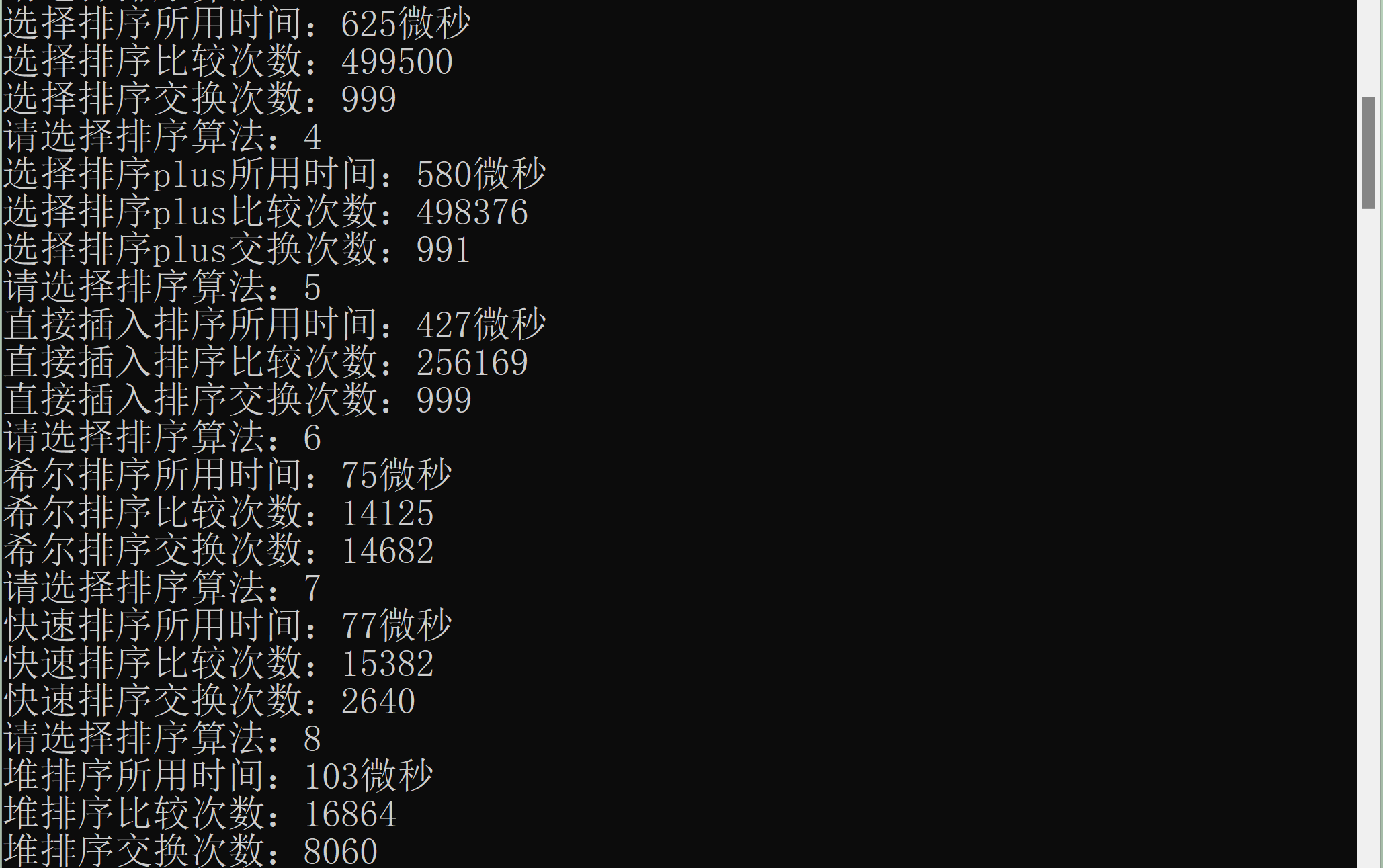
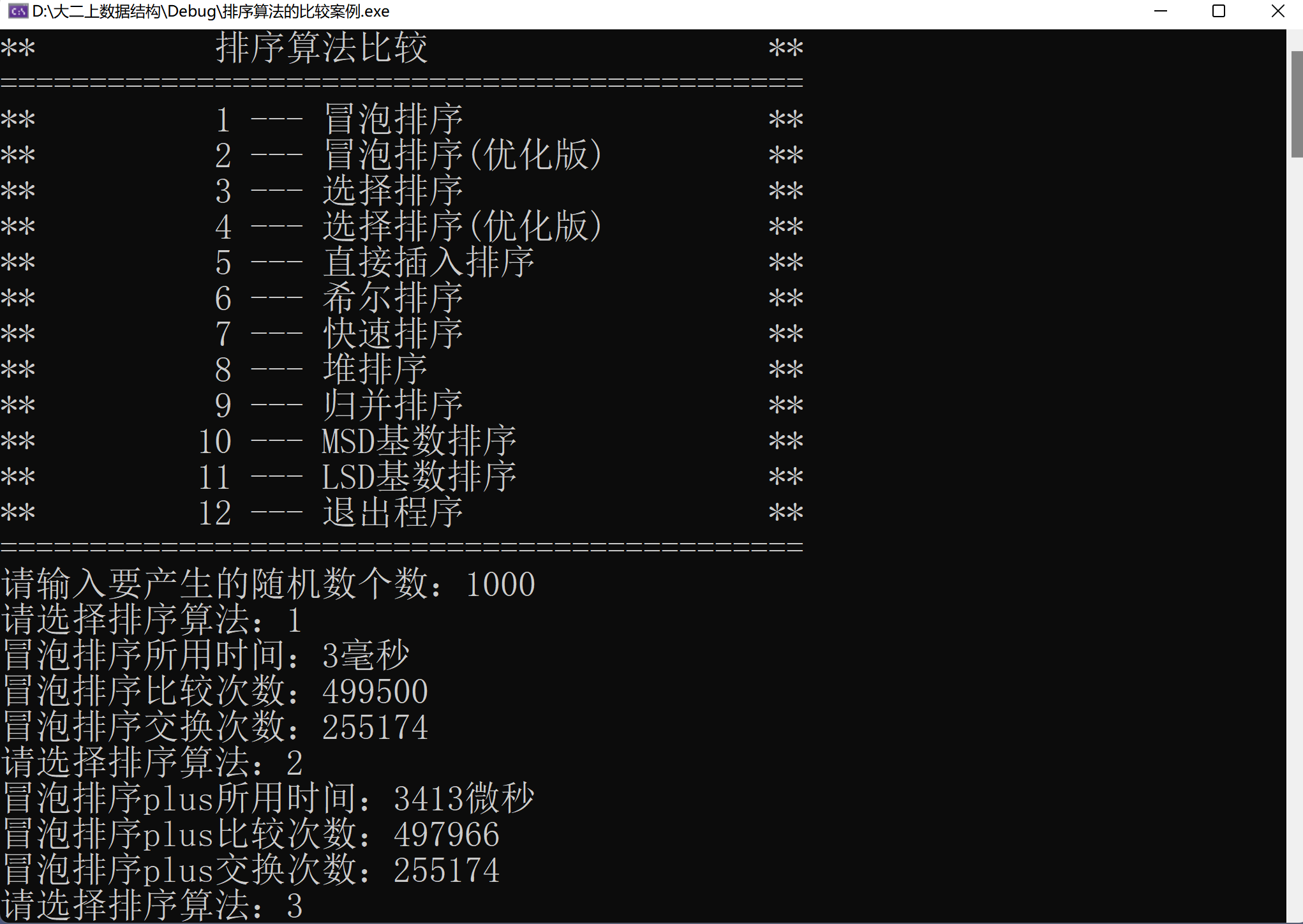
# 3十一种排序算法的对比

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **算法** | **时间复杂度** | | | **空间复杂度** | **稳定性** |
| **平均情况** | **最怀情况** | **最好情况** |
| 冒泡排序 | O(n²) | O(n2) | O(n) | O(1) | 是 |
| 选择排序 | O(n²) | O(n2) | O(n2) | O(1) | 否 |
| 直接插入排序 | O(n²) | O(n2) | O(n) | O(1) | 是 |
| 希尔排序 |  | O(n2) | O(n) | O(1) | 否 |
| 堆排序 | O(nlog2n) | O(nlog2n) | O(nlog2n) | O(1) | 否 |
| 快速排序 | O(nlog2n) | O(n2) | O(nlog2n) | O(log2n) | 否 |
| 归并排序 | O(nlog2n) | O(nlog2n) | O(nlog2n) | O(n) | 是 |
| LSD基数排序 | O(d(n+r)) | O(d(n+r)) | O(d(n+r)) | O(n+r) | 是 |
| MSD基数排序 | O(d(n+r)) | O(d(n+r)) | O(d(n+r)) | O(n+r) | 是 |

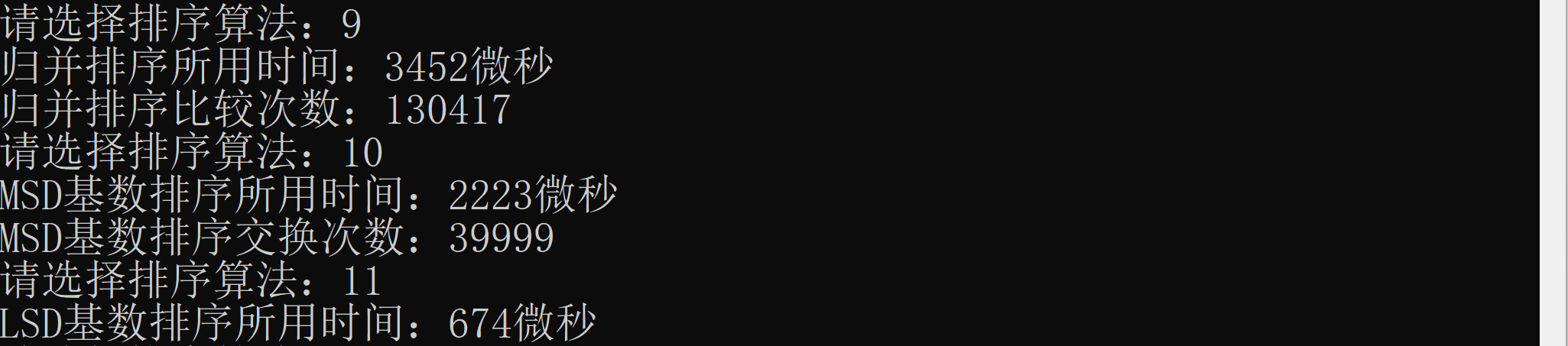
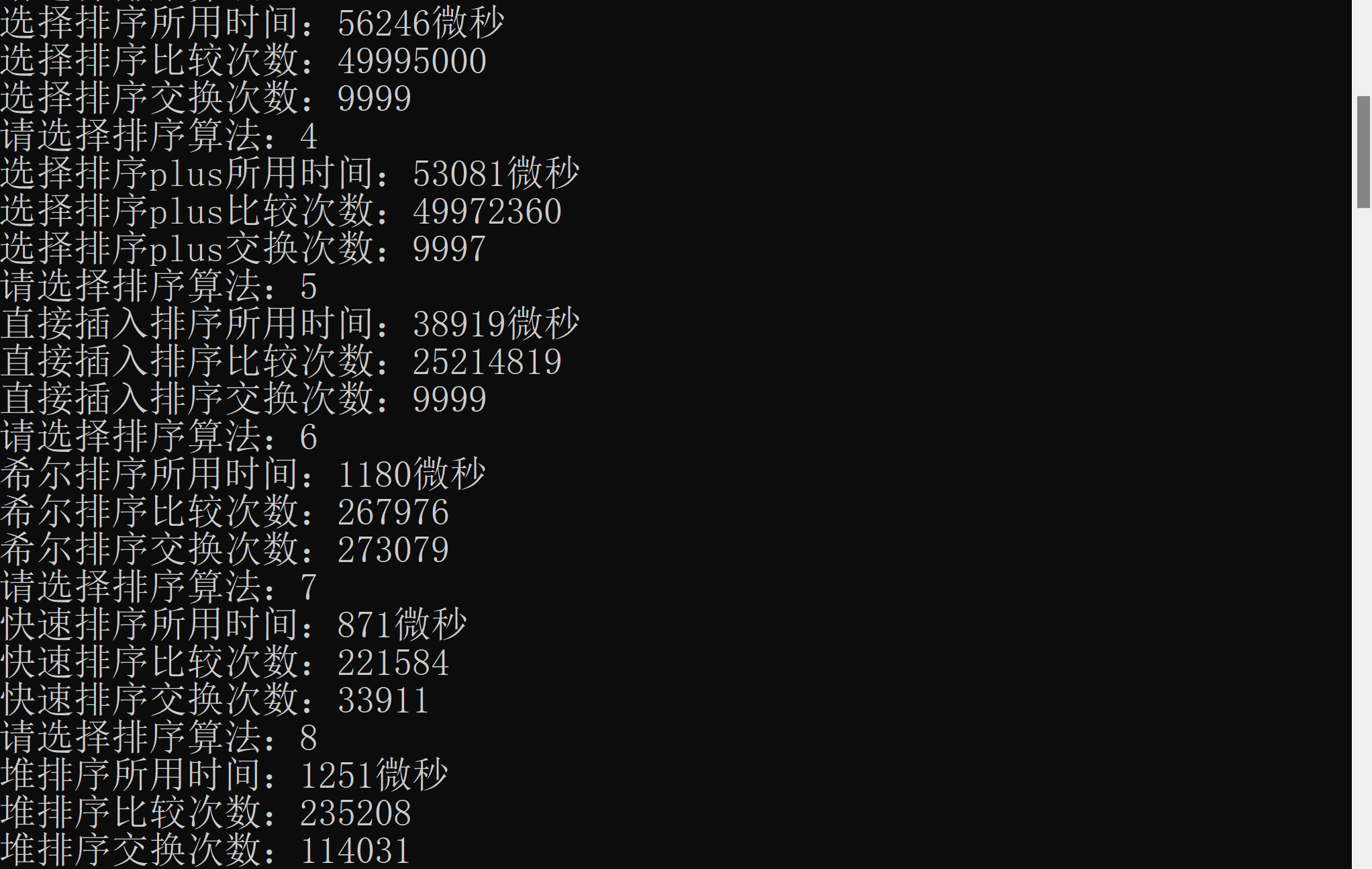
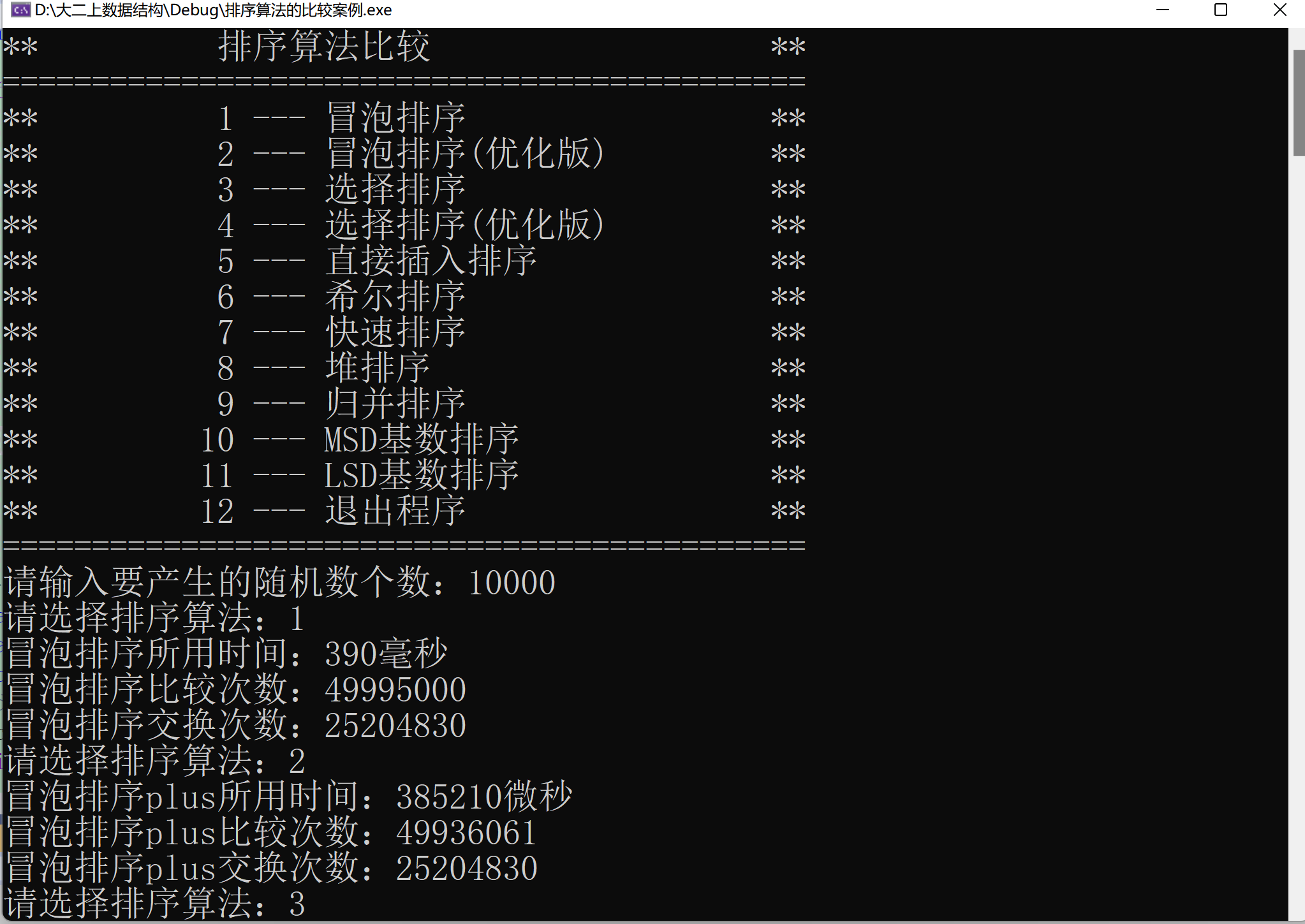
# 4 项目测试

## 4.1 100个随机数测试

**4.2 1000个随机数测试**



## 4.3 10000个随机数测试



## 4.4 100000个随机数测试

