数据结构课程设计

项目说明文档

八种排序算法的比较案例

|  |  |
| --- | --- |
| 作者姓名： | 罗斌江 |
| 学 号： | 2053285 |
| 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 4](#_Toc90815383)

[1.1项目背景 4](#_Toc90815384)

[1.2 项目需求分析 4](#_Toc90815385)

[1.3 项目要求 5](#_Toc90815386)

[1.3.1 功能要求 5](#_Toc90815387)

[1.3.2 输入格式 5](#_Toc90815388)

[1.3.3 输出格式 5](#_Toc90815389)

[1.3.4 项目示例 5](#_Toc90815390)

[2 项目实现 6](#_Toc90815391)

[2.1 冒泡排序 6](#_Toc90815392)

[2.1.1 算法逻辑 6](#_Toc90815393)

[2.1.2 算法分析 6](#_Toc90815394)

[2.2 选择排序 7](#_Toc90815395)

[2.2.1 算法逻辑 7](#_Toc90815396)

[2.2.2 算法分析 8](#_Toc90815397)

[2.3 直接插入排序 8](#_Toc90815398)

[2.3.1 算法逻辑 8](#_Toc90815399)

[2.3.2 算法分析 8](#_Toc90815400)

[2.4 折半插入排序 9](#_Toc90815401)

[2.4.1 算法逻辑 9](#_Toc90815402)

[2.4.2 算法分析 9](#_Toc90815403)

[2.5 希尔排序 9](#_Toc90815404)

[2.5.1 算法逻辑 9](#_Toc90815405)

[2.5.2 算法分析 9](#_Toc90815406)

[2.6 堆排序 10](#_Toc90815407)

[2.6.1 算法逻辑 10](#_Toc90815408)

[2.6.2 算法逻辑 10](#_Toc90815409)

[2.7 快速排序 10](#_Toc90815410)

[2.7.1 算法逻辑 10](#_Toc90815411)

[2.7.2 算法分析 11](#_Toc90815412)

[2.8 归并排序 11](#_Toc90815413)

[2.8.1 算法逻辑 11](#_Toc90815414)

[2.8.2 算法分析 11](#_Toc90815415)

[2.9 基数排序 12](#_Toc90815416)

[2.9.1 算法逻辑 12](#_Toc90815417)

[2.9.2 算法分析 12](#_Toc90815418)

[3 各排序算法对比 12](#_Toc90815419)

[4 项目测试 14](#_Toc90815420)

[4.1 100个随机数测试 14](#_Toc90815421)

[4.2 1000个随机数测试 14](#_Toc90815422)

[4.3 10000个随机数测试 15](#_Toc90815423)

[4.4 100000个随机数测试 15](#_Toc90815424)

[4.5 1000000个随机数测试 15](#_Toc90815425)

# 1项目分析

## 1.1项目背景

所谓排序，就是使一串记录，按照其中的某个或某些关键字的大小，递增或递减的排列起来的操作。排序算法，就是如何使得记录按照要求排列的方法。排序算法在很多领域得到相当地重视，尤其是在大量数据的处理方面。一个优秀的排序算法可以节省大量的时间以及空间资源。在各个领域中考虑到数据的各种限制和规范，要得到一个符合实际的优秀算法，得经过大量的推理和分析。

本项目通过对8种排序算法性能的探究，能够为未来各个算法的使用途径提供一些帮助。

## 1.2 项目需求分析

针对于8种排序算法的比较案例，本项目在实现的过程中，考虑并且满足了以下的需求：

* 代码可读性强

本项目在实现过程中，将代码根据功能的不同划分为了不同的代码块，同时进行了合理封装。

* 健壮性

当用户输入的数据不合理时，系统应当给予相应的提示而非直接报错。

* 可视化

该系统通过输出当前正在执行的操作内容，使得用户可以直观的了解到当前操作内容。

* 对比性

为了让数据之间更加具有对比性，本项目考虑了100-1000000个随机数的情况。

## 1.3 项目要求

### 1.3.1 功能要求

随机函数产生10000个随机数，用快速排序，直接插入排序，冒泡排序，选择排序，希尔排序，堆排序，归并排序和基数排序对随机数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。

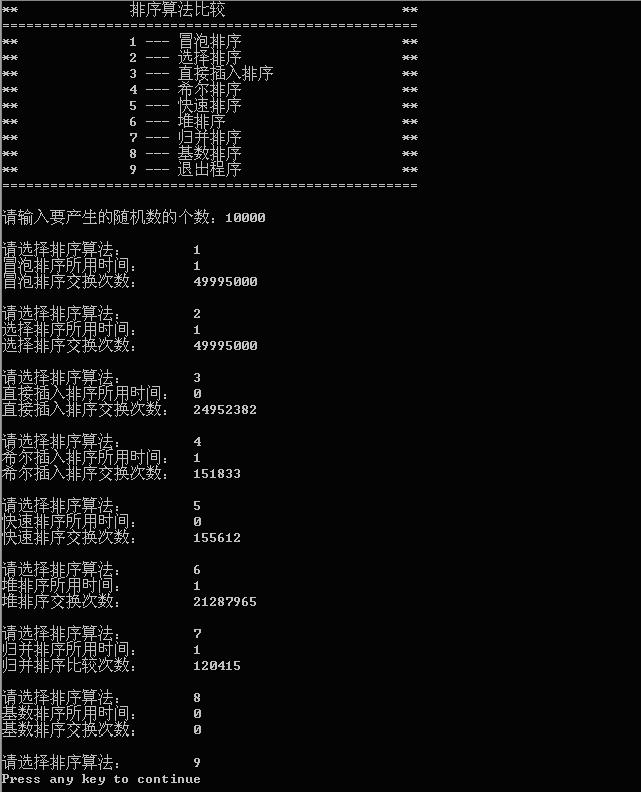
### 1.3.2 输入格式

生成随机数的个数，选中的算法类型。

### 1.3.3 输出格式

打印各排序算法的排序时间和交换次数等。

### 1.3.4 项目示例



# 2 项目实现

本项目共实现了以下9种排序算法：

* 冒泡排序
* 选择排序
* 直接插入排序
* 折半插入排序
* 希尔排序
* 堆排序
* 快速排序
* 归并排序
* 基数排序

下面将依次对本项目实现的这9种排序算法进行介绍。

# 2.1 冒泡排序

### 2.1.1 算法逻辑

以升序排序为例，冒泡算法是将按从头到尾遍历，将两两相邻的数据进行比较，如果前者大于后者，则交换二者位置，一次遍历结束后，最大的数会被送到数据的最后位置，最后一位变得有序，进行n-1趟后，从数据尾部开始的第1，2，···，n-1个数据变得有序，最终完成升序排序。需要特别注意的是，当相邻两数据的关键字大小一致时，二者不需要交换顺序，因此也说明冒泡排序是一种严格的稳定排序算法，不改变序列中相同元素之间的相同位置关系。

### 2.1.2 算法分析

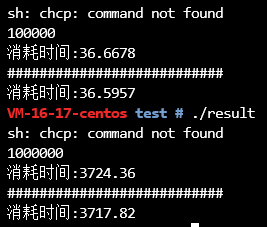
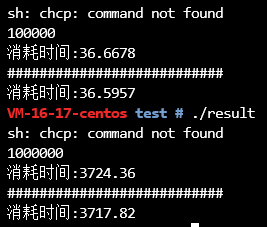
最佳情况下，待排序数据已经有序，只需一次起泡，进行n-1次关键码比较，不移动对象就可以完成排序；最坏情况下，待排序数据为目标数据序列的反转序列，执行n-1次起泡，第i趟进行n-i次关键码比较与n-i次对象互换，这样在最坏情况下总的关键码比较次数KCN和对象移动次数RMN为：

**KCN==**

**RMN==**

对于冒泡排序优化的主要思想是减少或者消除未交换数据的遍历。有三种情况：

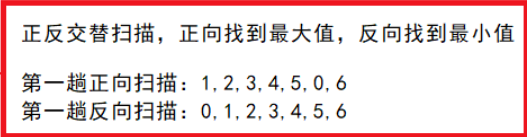
（1）可以发现，当数据序列{1，2，3，4，5，6，7，9，8}第一趟排序后，8和9交换位置，数据已经有序，后面的7次遍历都没有必要，所以在外循环添加exchange变量控制循环是否进行，一趟冒泡后没有发生exchange时边停止循环，在进行10000个和1000000个随机数排序时优化前后差距如下：



（2）第i次冒泡后，从第n-i个数开始的后续数据变得有序，不需要进行冒泡，在不进行优化（1）的情况下，减少了一半的比较，但当数据在第n-i个数之前已经有一部分有序时，存在不必要的比较，因此加入lastpos变量记录最后一次交换的位置，如下：



（3）双向冒泡，同时将未排序数据中的最大值与最小值排序到数据前后已经有序的位置：



## 2.2 选择排序

### 2.2.1 算法逻辑

选择排序算法的基本思路是为每一个位置选择当前最小的元素。选择排序的基本思想是，基于直接选择排序和堆排序这两种基本的简单排序方法。首先从第1个位置开始对全部元素进行选择，选出全部元素中最小的给该位置，再对第2个位置进行选择，在剩余元素中选择最小的给该位置即可；以此类推，重复进行“最小元素”的选择，直至完成第(n-1)个位置的元素选择，则第n个位置就只剩唯一的最大元素，此时不需再进行选择。

### 2.2.2 算法分析

以升序排序为例，选择排序是给每个位置选择当前未排序元素中最小的，比如给第一个位置选择最小的，在剩余元素里面给第二个元素选择第二小的，依次类推，直到第n-1个元素，第n个元素不用选择了，因为只剩下它一个最大的元素了。那么，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，那么交换后稳定性就被破坏了。因此，选择排序是一个不稳定的排序算法。相较于冒泡排序，选择排序的优点是效率高，在不增加比较次数的情况下极大程度上减少了数据元素交换的次数。最优情况下，只需要进行n-1次比较；最坏情况下，需要进行n-1次交换，比较次数小于n(n-1)/2；

## 2.3 直接插入排序

### 2.3.1 算法逻辑

直接插入排序通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入，如此重复，直至完成序列排序。

### 2.3.2 算法分析

直接插入排序是稳定的算法，它满足稳定算法的定义，即：假设在数列中存在a[i]=a[j]，若在排序之前，a[i]在a[j]前面；并且排序之后，a[i]仍然在a[j]前面。数据长度为n，该算法需要插入n-1趟，最好下情况，只需要比较n-1次；最坏情况下，关键码比较次数KCN和对象移动次数RMN为：

**KCN==**

**RMN==**

时间复杂度为O(n^2)。

## 2.4 折半插入排序

### 2.4.1 算法逻辑

二分查找插入排序是直接插入排序的一个变种，区别是在有序区中查找新元素插入位置时，为了减少元素比较次数提高效率，采用二分查找算法进行插入位置的确定。

### 2.4.2 算法分析

折半插入排序算法实际上是直接插入排序的改进，是一种稳定的排序算法，比直接插入算法明显减少了关键字之间比较的次数，因此速度比直接插入排序算法快，但记录移动的次数没有变，所以折半插入排序算法的时间复杂度仍然为O(n²)，与直接插入排序算法相同。附加空间O(1)。

## 2.5 希尔排序

### 2.5.1 算法逻辑

希尔排序实质上是一种分组插入方法。它的基本思想是：对于n个待排序的数列，取一个小于n的整数gap将待排序元素分成若干个组子序列，所有距离为gap的倍数的记录放在同一个组中；然后，对各组内的元素进行直接插入排序。 这一趟排序完成之后，每一个组的元素都是有序的。然后减小gap的值，并重复执行上述的分组和排序。重复这样的操作，当gap=1时，整个数列就是有序的。

### 2.5.2 算法分析

希尔排序是按照不同步长对元素进行插入排序，当刚开始元素很无序的时候，步长最大，所以插入排序的元素个数很少，速度很快；当元素基本有序了，步长很小，插入排序对于有序的序列效率很高。希尔排序的时间复杂度会比O(n²)好一些。同时，希尔排序是不稳定的排序算法。

## 2.6 堆排序

### 2.6.1 算法逻辑

堆排序是指利用堆（Heap）这种数据结构所设计的一种排序算法。堆是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。

堆是一类特殊数据结构的简称，通常是一个可以被看做一棵树的的数组对象。堆满足下述两个性质：

1. 堆中某个节点的值总是不大于或者不小于其父节点的值；
2. 堆总是一棵完全二叉树。

将根节点最大的堆叫做最大堆或者大根堆，根节点最小的堆叫做最小堆小根堆。常见的堆有二叉堆、斐波拉契堆等。主要函数如下所示：

### 2.6.2 算法逻辑

堆排序是一种选择排序，整体主要由构建初始堆+交换堆顶元素和末尾元素并重建堆两部分组成。其中构建初始堆经推导复杂度为O(n)，在交换并重建堆的过程中，需交换n-1次，而重建堆的过程中，根据完全二叉树的性质，[log2(n-1),log2(n-2)...1]逐步递减，近似为nlogn。所以堆排序时间复杂度一般认为就是O(nlogn)级。堆排序是一种不稳定的排序算法。

## 2.7 快速排序

### 2.7.1 算法逻辑

快速排序的基本思想是: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，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。

### 2.7.2 算法分析

快速排序的平均时间复杂度也是O(nlog2n)。因此，该排序方法被认为是目前最好的一种内部排序方法。

从空间性能上看，尽管快速排序只需要一个元素的辅助空间，但快速排序需要一个栈空间来实现递归。最好的情况下，即快速排序的每一趟排序都将元素序列均匀地分割成长度相近的两个子表，所需栈的最大深度为log2(n+1)；但最坏的情况下，栈的最大深度为n。这样，快速排序的空间复杂度为O(log2n)。

同时，快速排序是一个不稳定的排序算法。

## 2.8 归并排序

### 2.8.1 算法逻辑

归并排序算法就是把序列递归划分成为一个个短序列，以其中只有1个元素的直接序列或者只有2个元素的序列作为短序列的递归出口，再将全部有序的短序列按照一定的规则进行排序为长序列。归并排序融合了分治策略，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(当凡为奇数时会出现长度为l的情况)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。需要注意的是，在进行元素比较和交换时，若两个元素大小相等则不必刻意交换位置，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。

### 2.8.2 算法分析

不管元素在什么情况下都要做这些步骤，所以花销的时间是不变的，所以该算法的最优时间复杂度和最差时间复杂度及平均时间复杂度都是一样的为：O( nlogn)

归并的空间复杂度就是那个临时的数组和递归时压入栈的数据占用的空间：n + logn；所以空间复杂度为: O(n)。

归并排序算法中，归并最后到底都是相邻元素之间的比较交换，并不会发生相同元素的相对位置发生变化，故是稳定性算法。

## 2.9 基数排序

### 2.9.1 算法逻辑

将所有待比较数值（这里以正整数为例）统一为同样的数位长度，数位较短的数前面补零。设一整数的十进制位数表示为k1,k2...kd，其中k1为高位，kd为低位。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后, 数列就变成一个有序序列。即先从kd开始排序，再对kd-1进行排序，依次重复，直到对k1排序后便得到一个有序序列。

### 2.9.2 算法分析

基数排序基于计数排序，是经典的空间换时间的方式，内存占用很大，当数据中出现负数时，一般不考虑基数排序，但也可以通过分正负队列分别排序然后合并的方法来解决问题。

# 3 各排序算法对比

通过对各种排序算法实现方法的过程分析，可以得出它们的时间复杂度、空间复杂度和稳定性，如下所示：

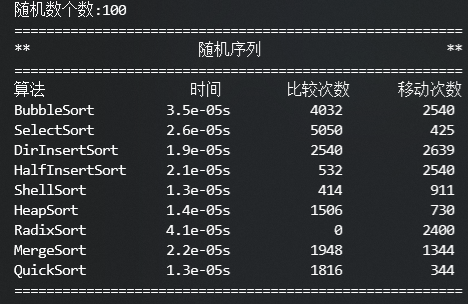
|  |  |  |  |
| --- | --- | --- | --- |
| **算法** | **时间复杂度** | **空间复杂度** | **稳定性** |
| 冒泡排序 | O(N²) | O(1) | 是 |
| 选择排序 | O(N²) | O(1) | 否 |
| 直接插入排序 | O(N²) | O(1) | 是 |
| 折半插入排序 | O(N²) | O(1) | 是 |
| 希尔排序 |  | O(1) | 否 |
| 堆排序 | O(NlogN) | O(1) | 否 |
| 快速排序 | O(NlogN) | O(logN) | 否 |
| 归并排序 | O(NlogN) | O(N) | 是 |
| 基数排序 | O(NlogM) | O(N) | 是 |

排序时间与比较次数和交换次数直接相关，冒泡排序、选择排序和直接插入排序的比较次数高出其他排序算法四个数量级，冒泡排序与选择排序的比较不可避免，不同的是，选择排序会在找到合适的位置后才开始移动位置，所以选择排序的移动次数是十分优秀的，在同一数量级的比较次数下二者由于交换次数的差距排序时间也有很大差距；折半插入相对于直接插入而言减少了很多不必要的比较次数，但由于移动次数不发生改变，时间优化的空间不大；希尔排序、堆排序、快速排序、归并排序、基数排序的速度都很快，希尔排序与堆排序的空间复杂度占优，但二者与快速排序一样，都不稳定；基数排序和普通的排序不同，充分借助了数字本身的特征来完成排序。

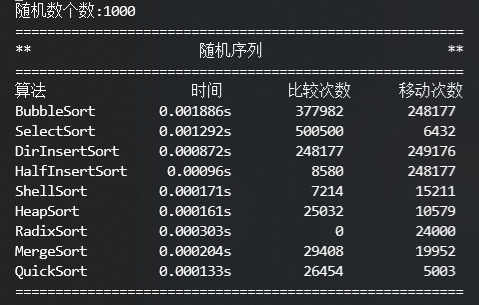
事实上，当待排序的数据数量级达到10^7时，时间复杂度低的算法优势将大大体现，我们应该像计算机硬件一样把算法看作一种技术。整个系统的性能不仅依赖于选择快速高级的硬件而且还依赖于选择有效的算法。

# 4 项目测试

## 4.1 100个随机数测试



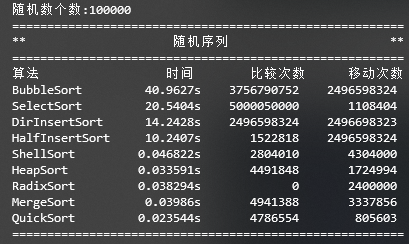
## 4.2 1000个随机数测试



## 4.3 10000个随机数测试



## 4.4 100000个随机数测试



## 4.5 1000000个随机数测试

