8种排序算法的比较案例

#### 软件工程20级2班 2053300 胡锦晖

# 一、项目概述

## 1.1 项目背景

排序算法可以说是数据结构与算法中最基础的算法之一了，其定义如下：

假设含有个记录的序列为，其相应的关键字分别为，需确定的一种排列，使其相应的关键字满足(非递减或非递增) 关系，即使得序列成为一个按关键字有序的序列， 这样的操作就称为排序。

## 1.2 项目目标

本项目通过获取用户输入的数据个数产生相应数量的随机数，并分别用冒泡排序、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、基数排序共八种排序方式对其进行排序，并且输出通过排序所用的时间和交换次数对这八种排序算法进行比较和性能分析。

# 二、实现思路

## 2.1 数据的存储结构

本项目的数据存储结构十分简单，用一个int类型的数组存储待排序的数据即可。

此外，还需要一个char型变量存储用户选择的排序算法种类；一个int型变量存储排序所用的交换操作次数；用两个clock\_t型变量记录排序起止的机器时间并用一个double型变量存储单次排序的执行时间；一个string类型数组存储各排序的名称。

如下图所示：

## 2.2 类的设计

设计思路：将不同排序都要进行的相同操作都进行了封装统一调用，尽可能提高代码的复用性。

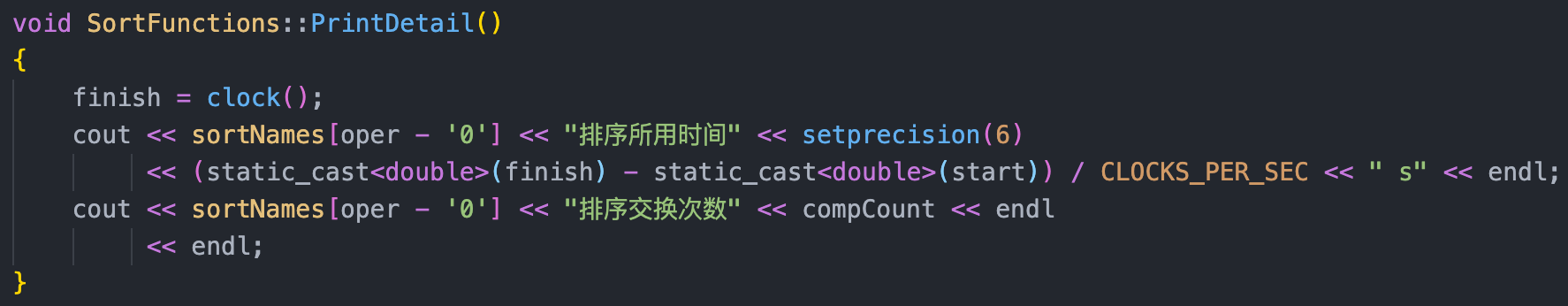
## 2.3 功能实现

### 2.3.1 功能菜单

与之前的许多项目一样，SortFunction也是一个单例，所以将菜单的打印加入到构造函数当中，此外，该部分还包括了所涉及到数据的初始化，并获取随机数种子。

### 2.3.2 排序用时的比较

在每次排序前重置完随机数之后，start记录当前程序运行的机器时间：

在每次排序结束输出本次排序的信息之前，finish记录当前程序运行的机器时间：

则start与finish的差值即为排序用时（单位：ms），将该值强制转换为double类型并进行进制转换之后格式化为六位小数输出。

### 2.3.3 错误处理

手机的屏幕

描述已自动生成输入的随机数个数： 选择的操作编号：

# 三、性能分析

## 3.1 冒泡排序

### 3.1.1 基本思想

冒泡排序的基本思想是两两比较相邻记录的关键字，如果前边比后边大，则交换它们，直到序列的最后的两个关键字进行比较，这样最后的关键字就是最大的，之后再进行第二轮、第三轮遍历，直到剩下序列的最前的关键字。

### 3.1.2 优化方向

这里通过一个bool型变量flag来标记一轮遍历之中是否发生了关键字的交换，若某一轮遍历未发生交换，则说明序列已经有序，避免了有序情况下无意义的循环判断，可以在性能上获得一些提升。

### 3.1.3 复杂度及稳定性分析

**空间复杂度**：冒泡排序只需要一个值的空间用于交换节点，所以空间复杂度为。

**时间复杂度**：

最坏情况：序列为逆序状态，每一轮遍历都需要n次交换位置，时间复杂度为。

最好情况：序列为正序状态，每一轮遍历不需要交换位置，时间复杂度为。

平均情况：每一轮遍历需要次交换位置，时间复杂度依然为。

**稳定性**：冒泡排序就是把小的元素往前调或者把大的元素往后调。比较是相邻的两个元素比较，交换也发生在这两个元素之间。所以，如果两个元素相等，不会发生交换。如果两个相等的元素没有相邻，那么即使通过前面的两两交换把两个相邻起来，也不会发生交换，所以相同元素的前后顺序并没有改变，所以冒泡排序是一种稳定排序算法。

## 文本 描述已自动生成3.2 选择排序

### 3.2.1 基本思路

直接选择排序的基本思想是在待排序的序列中选取最小关键字的记录放在序列的第一个位置。遍历整个序列，首先选取第一位置的关键字分别与之后所有的关键字比较，如果后边值更小则与之交换，直到第一轮遍历结束时，序列第一个位置记录的关键字就是最小的，接下来继续从第二个、第三个做同样的操作，直到整个序列有序。

### 3.2.2 复杂度及稳定性分析

**空间复杂度**：需要一个额外的空间保存最小关键字用于交换，所以空间复杂度为

**时间复杂度**：从选择排序的过程来看，它最大的特点就是交换移动数据次数相当少，这样也就节约了相应的时间。分析它的时间复杂度发现，无论最好最差的情况，其比较次数都是一样的多，第趟排序需要进行次关键字的比较，此时比较次数为：

对于交换次数来说，最好情况：即初始升序时为0次，最坏情况：即初始降序时为次。基于最终排序时间为比较次数与交换次数的总和，时间复杂度仍为

**稳定性**：从选择排序的过程来看，排序可能会进行远距离的交换，所以说本项目中选择排序是不稳定的。

## 文本 描述已自动生成3.3 直接插入排序

### 3.3.1 基本思路

直接插入排序的基本思想是将未排好序的序列中的记录一个个地插入到已排好序的序列中，每次插入时，插入记录的关键字需要与已排好序的序列中的关键字进行多次比较，直到找到合适的位置插入，而原来已排好序的部分记录可能需要进行后移操作。

### 3.3.2 复杂度与稳定性分析

**空间复杂度**：需要一个额外的空间保存一个关键字用于交换，所以空间复杂度为。

**时间复杂度**：

最坏情况：当待排序序列正好为逆序状态，首先遍历整个序列，之后一个个地将待插入元素放在已排序的序列最前面，之后的所有元素都需要向后移动一位，所以比较和移动的时间复杂度都是，再加上遍历整个序列的复杂度，时间复杂度为。

最好情况：当待排序序列正好为正序状态，则遍历完整个序列，当插入元素时，只比较一次就够了，所以时间复杂度为。

平均情况：当被插入的元素放在已排序的序列中间位置时，为平均情况，比较和移动的时间复杂度为，所以总的时间复杂度依然为。

**稳定性**：若当前记录的关键字碰见一个和要插入记录的关键字相等的，则要插入记录会插入到当前记录之后。所以，相等记录的前后顺序没有改变，从原无序序列出去的顺序就是排好序后的顺序，即插入排序是稳定的。

## 3.4 希尔排序

### 屏幕上有字 描述已自动生成3.4.1 基本思路

希尔排序的基本思想是加大插入排序中元素的间隔，并在这些有间隔的元素中进行插入排序，从而使数据进行大幅度的移动，当进行过依次排序后，再减小间隔再一次进行插入排序，直到间隔缩小为1，这里的间隔称为“增量”。本项目中的增量序列为：

### 3.4.2 复杂度及稳定性分析

**空间复杂度**：需要一个额外的空间保存一个关键字用于交换，所以空间复杂度为。

**时间复杂度**：时间复杂度与增量的选取有关。迄今为止还没有找到一种最好的增量序列。不过大量的研究明，当增量序列为：

时，可以获得不错的效果，时间复杂度为。

**稳定性**：希尔排序作为直接插入排序的优化，进行多次插入排序。虽然每一次插入排序是稳定的，不会改变相同元素的相对顺序，但在不同的插入排序过程中，相同的元素可能在各自的插入排序中移动，最后其稳定性就会被打乱，所以希尔排序是不稳定的。

## 3.5 快速排序

### 文本 描述已自动生成3.5.1 基本思路

快速排序的基本思想是选择一个基准元素（本项目选择最后一个元素），通过一趟扫描，将待排序列分成两部分，一部分的关键字比基准元素小，一部分大于等于基准元素，此时基准元素在其排好序后的正确位置，然后再用同样的方法递归地排序划分的两部分，以达到使整个序列有序的目的。

### 3.5.2 复杂度及稳定性分析

**空间复杂度**：快速排序使用的辅助数据存储空间的复杂度为，但由于是递归工作，真正消耗空间的是递归调用。每一次都平分数组的情况下空间复杂度为 ，最差的情况下空间复杂度为。

**时间复杂度**：在最优情况下，基准元素pivot每次都划分得很均匀，如果排序n个关键字，其递归树 的深度就为，即仅需递归次，需要时间为的话，第一次划分应该是需要对整个数组扫描一遍，做次比较。然后，获得的pivot将数组一分为二，那么各自还需要的时间（因为是最优情况，所以平分两半）。于是不断地划分下去，我们就有了下面的不等式推断：

也就是说，在最优的情况下，快速排序算法的时间复杂度为

在最坏情况下待排序的序列为正序或者逆序，每次划分只得到一个比上一次划分少一个记录的子序列。递归树为一棵斜树。此时需要执行次递归调用，且第 次划分需要经过次关键字的比较才能找到第 个记录，也就是pivot的位置，因此比较次数为：

则最终的时间复杂度为。

平均情况，设pivot的关键字在第 个位置，则有：

通过数学归纳法可以证明，平均情况的时间复杂度为。

**稳定性**：由于在进行pivot的交换是远距离交换，则快速排序是一种不稳定的排序算法。

## 3.6 堆排序

### 3.6.1 基本思路

堆排序的基本思想就是将待排序的序列构造成一个大顶堆。此时，整个序列的最大值就是堆顶的根结点。将其移走（即将其与堆数组的末尾元素交换，此时末尾元素就是最大值) 。然后将剩余的个序列重新构造成一个堆，这样就会得到 个元素中的次小值。如此反复执行，则可以得到一个有序序列。

### 文本 描述已自动生成3.6.2 复杂度及稳定性分析

**空间复杂度**：需要一个额外的空间保存一个关键字用于交换，所以空间复杂度为。

**时间复杂度**：程序的运行时间主要消耗在初始构建大根堆以及后续对大根堆的维护上，在构建堆的过程中，因为我们是完全二叉树从最下层最右边的非终端结点开始构将它与其孩子进行比较和若有必要的互换，对于每个非终端结点来说，最多进行两次比较和互换操作，因此整个构建堆的时间复杂度为。

由于完全二叉树的某个结点到根结点的距离为，所以在正式排序时，第 次取堆顶记录重建堆需要用的时间，并且需要取次堆顶记录，因此，重建堆的时间复杂度为。

所以总体来说，堆排序的时间复杂度为。由于堆排序对原始记录的排序状态并不敏感，因此它无论是最好、最坏和平均时间复杂度均为。

**稳定性**：由于记录的比较与交换是跳跃式进行，因此堆排序也是一种不稳定的排序方法。

## 3.7 归并排序

### 3.7.1 基本思路

本项目使用的是简单的二路归并排序，基本思想是：假设初始序列含有个记录，则可以看成是个有序的子序列，每个子序列的长度为1，然后两两归并，得到个长度为2或1的有序子序列。重复上述操作，直至得到一个长度为的有序序列为止。

文本

描述已自动生成归并操作：

递归进行排序：

### 文本 描述已自动生成3.7.2 复杂度及稳定性分析

**空间复杂度**：由于归并排序在归并过程中需要与原始记录序列同样数量的存储空间存放归并结果，以及递归时深度为的栈空间，因此空间复杂度为。

**时间复杂度**：归并排序的时间主要花在了划分序列和合并序列上，由于是采用递归的方式进行合并，树的每层元元素的个数最多是，也就代表着每层最多进行次比较，而递归树最多只有层，而且不管元素在什么情况下都要做这些步骤，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，都是。

**稳定性**：归并排序在进行序列合并的时候需要进行两两比较，不会出现跳跃，即归并排序算法是稳定的。

## 3.8 基数排序

### 3.8.1 基本思路

基数排序根据对多关键字的排序顺序分为最高位优先（MSD）和最低位优先（LSD）的基数排序，本项目使用了LSD方法。

基数排序的基本思想是对所要排序的序列进行分配和收集，记为记录的关键字个数（在本问题中为数据的位数）。分配时按照每个关键字的有序序列对其进行排序（本项目中使用的是计数排序），收集时根据分配时对于每个关键字的排序进行收集。如此经过趟分配与收集，即可得到一个有序序列，在本项目中是对十进制数据进行排序，即。

### 3.8.2 复杂度及稳定性分析

**空间复杂度**：需要个存储空间暂存对于每个关键字的有序序列，空间复杂度为。

**时间复杂度**：由于分配时使用线性时间复杂度的计数排序对于每个关键字进行排序，时间复杂度为，而收集时对于个序列进行收集，时间复杂度为，共进行趟分配与收集，时间复杂度为。

**稳定性**：由于基数排序在分配和收集时的依据都是关键字的主次（本项目中为按位），不存在交换，所以基数排序算法是稳定的。

*文本

描述已自动生成*

# 四、测试结果

## 4.1 Windows平台

操作系统：Windows11

编译器：g++ (GCC) 8.1.0

C++标准：c++17

## 4.2 Linux平台

操作系统：CentOS Linux release 8.5.2111

编译器：g++ (GCC) 8.5.0-4

C++标准：c++17

## 4.3 Mac平台

操作系统：macOS Monterey 12.0.1

编译器：clang++ 13.0.0

C++标准：c++17

文本

描述已自动生成