**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：香宁雨

学号：1954098

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.题目要求及功能分析 2](#_Toc58805954)

[2.设计 2](#_Toc58805955)

[2.1算法设计 2](#_Toc58805956)

[2.2分析术语 2](#_Toc58805957)

[2.3测试函数 4](#_Toc58805958)

[3.实现 7](#_Toc58805959)

[3.1随机数组生成 7](#_Toc58805960)

[3.2冒泡排序 7](#_Toc58805961)

[3.2.1流程图 7](#_Toc58805962)

[3.2.2代码实现 8](#_Toc58805963)

[3.2.3复杂度分析 9](#_Toc58805964)

[3.3插入排序 10](#_Toc58805965)

[3.3.1流程图 10](#_Toc58805966)

[3.3.2代码实现 11](#_Toc58805967)

[3.3.3复杂度分析 12](#_Toc58805968)

[3.4选择排序 13](#_Toc58805969)

[3.4.1流程图 13](#_Toc58805970)

[3.4.2代码实现 14](#_Toc58805971)

[3.4.3复杂度分析 15](#_Toc58805972)

[3.5快速排序 16](#_Toc58805973)

[3.5.1流程图 16](#_Toc58805974)

[3.5.2代码实现 17](#_Toc58805975)

[3.5.3复杂度分析 18](#_Toc58805976)

[3.6希尔排序 19](#_Toc58805977)

[3.6.1流程图 19](#_Toc58805978)

[3.6.2代码实现 20](#_Toc58805979)

[3.6.3复杂度分析 21](#_Toc58805980)

[3.7归并排序 22](#_Toc58805981)

[3.7.1流程图 22](#_Toc58805982)

[3.7.2代码实现 23](#_Toc58805983)

[3.7.3复杂度分析 25](#_Toc58805984)

[3.8桶排序 26](#_Toc58805985)

[3.8.1流程图 26](#_Toc58805986)

[3.8.2代码实现 27](#_Toc58805987)

[3.8.3复杂度分析 28](#_Toc58805988)

[3.9堆排序 29](#_Toc58805989)

[3.9.1流程图 29](#_Toc58805990)

[3.9.2代码实现 30](#_Toc58805991)

[3.9.3复杂度分析 32](#_Toc58805992)

[4.输出测试 33](#_Toc58805993)

[4.1随机数组大小为100 33](#_Toc58805994)

[4.2随机数组大小为1，000 34](#_Toc58805995)

[4.3随机数组大小为10，000 35](#_Toc58805996)

# 1.题目要求及功能分析

项目简介：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。

功能分析：实现8种排序方法的排序功能并显示其排序的复杂度。

# 2.设计

## 2.1算法设计

共需要实现8种排序方法，本项目采用了冒泡排序、插入排序、选择排序、快速排序、希尔排序、归并排序、桶排序以及堆排序并分析他们的优劣。

## 2.2分析术语

稳定：如果a原本在b前面，而a=b时，排序之后a仍然在b的前面。

不稳定：如果a原本在b的前面，而a=b时，排序之后a可能出现在b的后面。

内排序：所有排序操作都在内存中完成。

外排序：通常是由于数据太大，不能同时存放在内存中，根据排序过程的需要而在外存与内存之间 数据传输才能进行。

时间复杂度：时间频度，一个算法执行所耗费的时间。算法中通常用数据比较次数与数据移动次数 进行衡量。

空间复杂度：算法执行所需要的内存大小。

n：数据规模

k：”桶”的个数

In-place：占用常数内存，不占用额外内存

Out-place：占用额外内存

## 2.3测试函数





# 3.实现

## 3.1随机数组生成

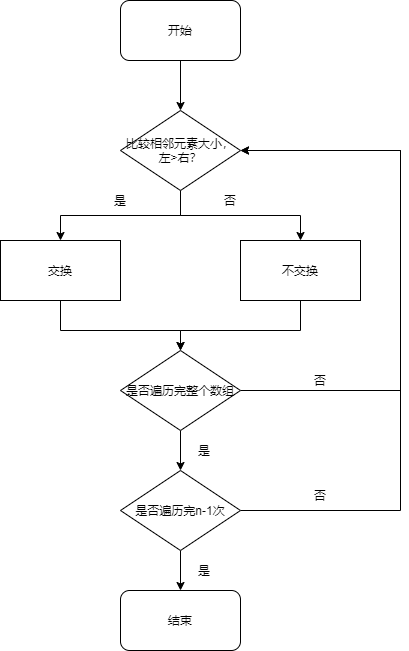


## 3.2冒泡排序

冒泡排序是一种交换排序，核心是冒泡，把数组中最小的那个往上冒，冒的过程就是和他相邻的元素交换。

重复走访要排序的数列，通过两两比较相邻记录的排序码。排序过程中每次从后往前冒一个最小值，且每次能确定一个数在序列中的最终位置。若发生逆序，则交换；有两种方式进行冒泡，一种是先把小的冒泡到前边去，另一种是把大的元素冒泡到后边。

### 3.2.1流程图



### 3.2.2代码实现



### 3.2.3复杂度分析

平均时间复杂度：O(N^2)

最佳时间复杂度：O(N)

最差时间复杂度：O(N^2)

空间复杂度：O(1)

排序方式：In-place

稳定性：稳定

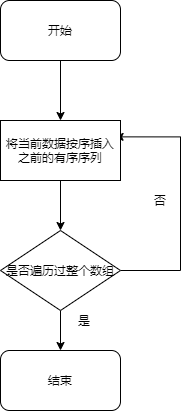
冒泡排序涉及相邻两两数据的比较，故需要嵌套两层 for 循环来控制。外层循环 n 次，内层最多时循环 n – 1次、最少循环 0 次，平均循环(n-1)/2。所以循环体内总的比较交换次数为：n\*(n-1) / 2 = (n^2-n)/2 ;按照计算时间复杂度的规则，去掉常数、去掉最高项系数，其复杂度为O(N^2) 。

最优的空间复杂度为开始元素已排序，则空间复杂度为 0，最差的空间复杂度为开始元素为逆排序，则空间复杂度为 O(N)，平均的空间复杂度为O(1) .

## 3.3插入排序

插入排序的工作原理是通过构建有序序列，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。插入排序在实现上，通常采用in-place排序（即只需用到O(1)的额外空间的排序），因而在从后向前扫描过程中，需要反复把已排序元素逐步向后挪位，为最新元素提供插入空间。

### 3.3.1流程图



### 3.3.2代码实现



### 3.3.3复杂度分析

平均时间复杂度：O(N^2)

最差时间复杂度：O(N^2)

空间复杂度：O(1)

排序方式：In-place

稳定性：稳定

如果插入排序的目标是把n个元素的序列升序排列，那么采用插入排序存在最好情况和最坏情况：

(1) 最好情况：序列已经是升序排列，在这种情况下，需要进行的比较操作需(n-1)次即可。

(2) 最坏情况：序列是降序排列，那么此时需要进行的比较共有n(n-1)/2次。

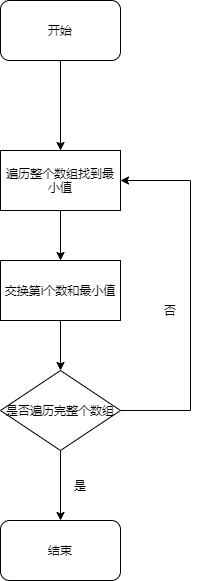
插入排序的赋值操作是比较操作的次数减去(n-1)次。平均来说插入排序算法复杂度为O(N^2)。

最优的空间复杂度为开始元素已排序，则空间复杂度为 0；最差的空间复杂度为开始元素为逆排序，则空间复杂度最坏时为 O(N);平均的空间复杂度为O(1)

## 3.4选择排序

首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。

### 3.4.1流程图



### 3.4.2代码实现



### 3.4.3复杂度分析

平均时间复杂度：O(N^2)

最佳时间复杂度：O(N^2)

最差时间复杂度：O(N^2)

空间复杂度：O(1)

排序方式：In-place

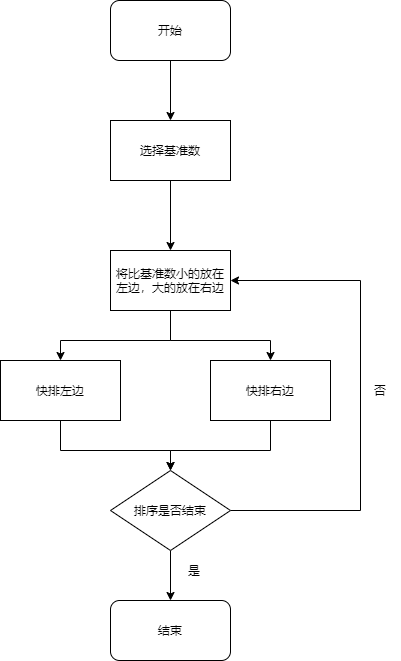
稳定性：不稳定

选择排序的交换操作介于和(n-1)次之间。选择排序的比较操作为n(n-1)/2次之间。选择排序的赋值操作介于0和3(n-1)次之间。比较次数O(n^2)，比较次数与关键字的初始状态无关，总的比较次数N = (n-1) + (n-2) +…+ 1 = n x (n-1)/2。交换次数O(n)，最好情况是，已经有序，交换0次；最坏情况是，逆序，交换n-1次。

## 3.5快速排序

通过一趟排序将待排记录分隔成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

### 3.5.1流程图



### 3.5.2代码实现



### 3.5.3复杂度分析

平均时间复杂度：O(NlogN)

最佳时间复杂度：O(NlogN)

最差时间复杂度：O(N^2)

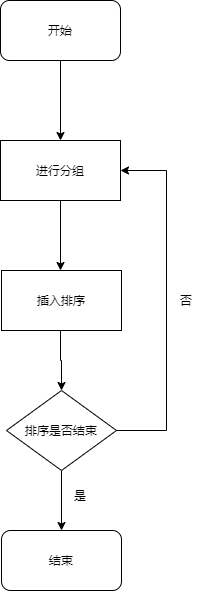
空间复杂度：根据实现方式的不同而不同

## 3.6希尔排序

先将整个待排元素序列分割成若干个子序列（由相隔某个“增量”的元素组成的）分别进行直接插入排序，然后依次缩减增量再进行排序，待整个序列中的元素基本有序（增量足够小）时，再对全体元素进行一次直接插入排序。

因为直接插入排序在元素基本有序的情况下（接近最好情况），效率是很高的，因此希尔排序在时间效率上比前两种方法有较大提高。

### 3.6.1流程图



### 3.6.2代码实现



### 3.6.3复杂度分析

平均时间复杂度：O(Nlog2N)

最佳时间复杂度：

最差时间复杂度：O(N^2)

空间复杂度：O(1)

稳定性：不稳定

复杂性：较复杂

希尔排序的效率取决于增量值gap的选取，时间复杂度并不是一个定值。

开始时，gap取值较大，子序列中的元素较少，排序速度快，克服了直接插入排序的缺点；其次，gap值逐渐变小后，虽然子序列的元素逐渐变多，但大多元素已基本有序，所以继承了直接插入排序的优点，能以近线性的速度排好序。

最优的空间复杂度为开始元素已排序，则空间复杂度为 0；最差的空间复杂度为开始元素为逆排序，则空间复杂度为 O(N);平均的空间复杂度为O(1)希尔排序并不只是相邻元素的比较，有许多跳跃式的比较，难免会出现相同元素之间的相对位置发生变化。

## 3.7归并排序

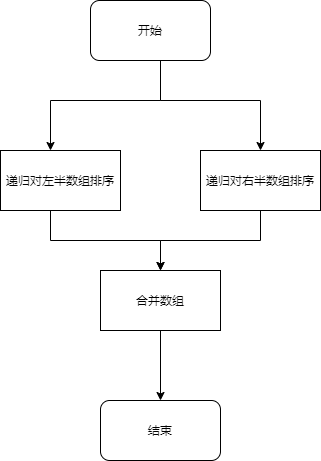
归并排序是用分治思想，分治模式在每一层递归上有三个步骤：

（1）分解（Divide）：将n个元素分成个含n/2个元素的子序列。

（2）解决（Conquer）：用合并排序法对两个子序列递归的排序。

（3）合并（Combine）：合并两个已排序的子序列已得到排序结果。

### 3.7.1流程图



### 3.7.2代码实现





### 3.7.3复杂度分析

平均时间复杂度：O(nlogn)

最佳时间复杂度：O(n)

最差时间复杂度：O(nlogn)

空间复杂度：O(n)

排序方式：In-place

稳定性：稳定

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。

## 3.8桶排序

桶排序（Bucket sort）或所谓的箱排序，是一个排序算法，工作的原理是将数组分到有限数量的桶里。每个桶再个别排序（有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排序），最后依次把各个桶中的记录列出来记得到有序序列。桶排序是鸽巢排序的一种归纳结果。当要被排序的数组内的数值是均匀分配的时候，桶排序使用线性时间（Θ(n)）。但桶排序并不是比较排序，他不受到O(n log n)下限的影响。

本项目中的桶排序为简单桶排序，即采用数组下标作为数的大小进行排序。

### 3.8.1流程图



### 3.8.2代码实现



### 3.8.3复杂度分析

平均时间复杂度：O(n + k)

最佳时间复杂度：O(n + k)

最差时间复杂度：O(n ^ 2)

空间复杂度：O(n \* k)

稳定性：稳定

桶排序最好情况下使用线性时间O(n)，桶排序的时间复杂度，取决与对各个桶之间数据进行排序的时间复杂度，因为其它部分的时间复杂度都为O(n)。很显然，桶划分的越小，各个桶之间的数据越少，排序所用的时间也会越少。但相应的空间消耗就会增大。

## 3.9堆排序

堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积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。

### 3.9.1流程图



### 3.9.2代码实现





### 3.9.3复杂度分析

平均时间复杂度：O(nlogn)

最佳时间复杂度：O(nlogn)

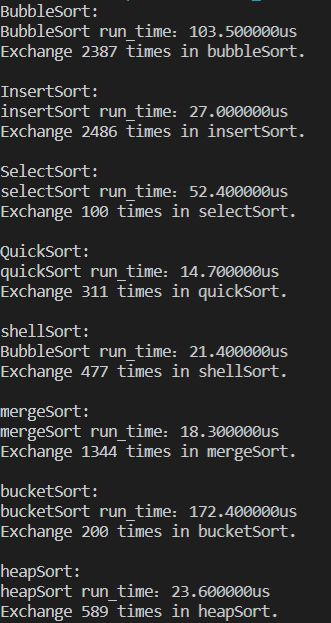
最差时间复杂度：O(nlogn)

稳定性：不稳定

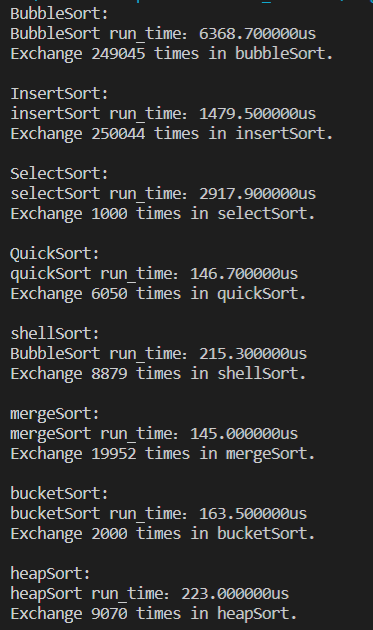
堆排序其实也是一种选择排序，是一种树形选择排序。只不过直接选择排序中，为了从R[1…n]中选择最大记录，需比较n-1次，然后从R[1…n-2]中选择最大记录需比较n-2次。事实上这n-2次比较中有很多已经在前面的n-1次比较中已经做过，而树形选择排序恰好利用树形的特点保存了部分前面的比较结果，因此可以减少比较次数。对于n个关键字序列，最坏情况下每个节点需比较log2(n)次，因此其最坏情况下时间复杂度为nlogn。堆排序为不稳定排序，不适合记录较少的排序。

# 4.输出测试

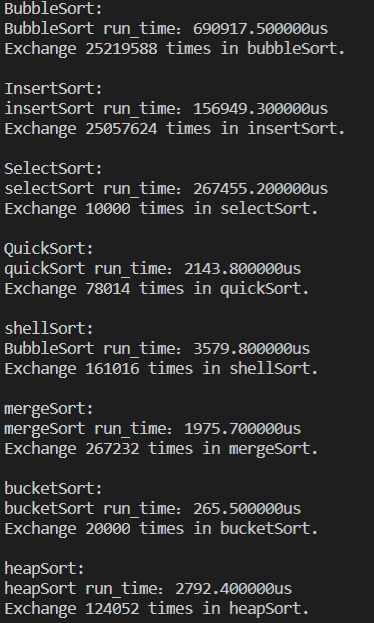
## 4.1随机数组大小为100



## 4.2随机数组大小为1，000



## 4.3随机数组大小为10，000



4.4随机数组大小为100，000

