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： 刘畅

学 号： 2054164

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 1](#_Toc91254795)

[1.1 背景分析 1](#_Toc91254796)

[1.2 功能分析 1](#_Toc91254797)

[2 设计 1](#_Toc91254798)

[2.1 结构设计 1](#_Toc91254799)

[2.2 运行时间计算 2](#_Toc91254800)

[2.3 排序算法设计 2](#_Toc91254801)

[2.3.1 冒泡排序 2](#_Toc91254802)

[2.3.2 选择排序 2](#_Toc91254803)

[2.3.3 插入排序 2](#_Toc91254804)

[2.3.4 希尔排序 3](#_Toc91254805)

[2.3.5 快速排序 3](#_Toc91254806)

[2.3.6 堆排序 4](#_Toc91254807)

[2.3.6.1 堆的思想 4](#_Toc91254808)

[2.3.6.1 成员函数的实现： 5](#_Toc91254809)

[2.3.7 归并排序 5](#_Toc91254810)

[2.3.8 基数排序 5](#_Toc91254811)

[2.4 算法分析 6](#_Toc91254812)

[2.4.1 时间复杂度 6](#_Toc91254813)

[2.4.2 空间复杂度 6](#_Toc91254814)

[2.4.3 稳定性 7](#_Toc91254815)

[2.4.4 最好情况与最坏情况 7](#_Toc91254816)

[3 总结 8](#_Toc91254817)

# 1 分析

## 1.1 背景分析

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，递增或递减地排列起来地操作。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的重视，尤其是在大量数据的处理方面，一个优秀的算法可以节省大量的资源。

## 1.2 功能分析

基本的排序算法有8种，分别是冒泡排序、选择排序、插入排序、希尔排序、快速排序、堆排序、归并排序、桶排序。为了比较这8种排序算法的性能，随机生成10000个整数，分别用这些排序算法进行排序，计算每种排序算法花费的排序时间和交换次数，进而比较它们的性能。

# 2 设计

## 2.1 结构设计

如上分析，在主函数main()中先随机生成10000个数字并使用数组存储，并在main.cpp中定义8个排序算法函数，返回值为该排序算法运行时产生的比较次数。而后定义一个计算时间的类TimeCalculating用于计算每种排序所用时间。

TimeCalculating类的构造函数需要传入一个数组和一个函数指针，函数指针对应前面定义的8种排序函数。在类的构造函数中，先将类内部的数组赋值循环赋值为传入的数组，因为要采用多次排序，不可避免地需要开辟8个一模一样的数组；而后获取当前时间，使用传入的函数指针进行排序，再获取当前时间，所用的时间即为两次获取的时间之差。将时间差保存至私有变量中即可。

TimeCalculating类中包含比较次数\_comparison\_times、运行时间\_running\_time、和目标数组\_target[]，设置了获取比较次数getComparisionTimes()和获取运行时间getRunningTime()两个成员变量，主要的逻辑运行在构造函数中运行。

## 2.2 运行时间计算

在头文件time.h中使用clock()函数获取当前时间，在排序算法运行之前与之后分别获取一次当前时间，取两者之差并除以1000.0，即得到了单位为秒的运行时间。

TimeCalculating(int target[], int (\*pfun)(int target[]) = NULL) {

for (int i = 0; i < MAX; i++) {

\_target[i] = target[i];

}

clock\_t starttime = clock();//开始计时

this->\_comparison\_times = (\*pfun)(\_target);//进行排序

clock\_t endtime = clock();//结束计时

this->\_running\_time = (double(endtime) - double(starttime)) / 1000.0;

}

## 2.3 排序算法设计

以下排序算法都是以升序为例，对于降序算法也是类似的。

### 2.3.1 冒泡排序

执行过程：从第一个元素开始，依次它与它右边的数的大小，若不满足升序关系，则交换它们，直到倒数第二个元素比较完成，这样就保证了最后一个元素是整个序列中最大的元素。依次这样操作：从第一个元素开始循环遍历到倒数第三个元素、倒数第四个元素......直至最后只比较第一个和第二个元素的大小后，结束程序。

复杂度分析：对于规模为的序列，第一轮比较次数为，第二轮为次......第轮为1次，故总比较次数为，如令，则总比较次数，复杂度为。

### 2.3.2 选择排序

执行过程：类似冒泡排序，也需要进行轮循环，第轮循环选出位置中的最小值，将其与位置的元素交换，即每轮循环确定了第i小的元素，当i取到n时也就确定了第n小的元素，即最大元素，此时就完成了整个序列的排序。

复杂度分析：对于规模为n的序列，总比较次数类似于冒泡排序，，复杂度为。

### 2.3.3 插入排序

这里的插入排序是最简单的直接插入排序。

执行过程：与上述两个排序类似，也需要轮循环，第i轮循环时已经做好了前面i-1个数的排序，对于第i个数，将其插入到前i-1个数中的合适位置。因为前i-1个数是有序的，只需不断将第i个数与前面的数比较，若前面的数大则交换位置，直至前面的数小于第i个数，则已经成功插入。

复杂度分析：该算法也需要n-1轮循环，但相较前面而言，每轮循环的平均比较次数有所下降，因为在第i轮循环时前面i-1个数是有序的，因此平均下来比较(i-1)/2次，总比较次数，复杂度仍为。

优化：折半插入排序法，当我们将第i个数字插入到前i-1个数字中时，由于前i-1个数字是有序的，因此可以采用折半搜索法寻找插入位置。因此在插入第i个元素时平均需要比较次，从此总比较次数，复杂度降为。当n较大时，总比较次数比直接插入排序的最坏情况要好得多，但比其最好情况要差。

### 2.3.4 希尔排序

希尔排序是优化的插入排序算法，又称缩小增量排序。

执行过程：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的元素放在同一个子序列中，在每个子序列中分别施行直接插入排序；然后以一定方式缩小间隔gap，直至gap=1后完成排序。显然若gap直接取1，排序过程等价于直接插入排序，因此该排序一定能将序列有序化。

复杂度分析：刚开始时的gap值比较大，子序列中的元素较少，排序速度较快，随着排序的进展，gap值逐渐变小，子序列中元素个数逐渐变多，但由于前面工作的基础，大多数元素已经基本有序，所以排序速度仍然很快。但是该算法的比较次数和元素移动次数和gap的递减方式存在依赖关系，不同的递减方法如、等等，不同的序列等等，对其运行速度都存在影响。目前仍未给出一个有效的复杂度分析方式，人们通过大量实验得出，当n很大时，关键码平均比较次数和元素平均移动次数大约在之间，时间复杂度近似可以认为。

### 2.3.5 快速排序

快速排序的基本思想是任取待排序元素序列中的某个元素作为基准，将整个序列分为小于它和大于它的两个子序列，并按此递归进行。

执行过程：对于长度为n的序列，任取一个基准，将整个序列划分为左右两个子序列，其中左子序列的元素全部小于基准，有子序列大于基准。然后对两个子序列重复执行该才做，直至所有元素都排在相应位置上位置。

复杂度分析：快速排序的速度取决于递归的深度。理想情况下，每次划分得到的两个子序列的长度大致相等，这样子序列长度缩小的速度最快，排序的效率也就越高；但最坏情况下每次递归处理时子序列的长度只减少了1，这样必须经过n-1次递归才能将所有元素排好序，而第i次递归时需要比较n-i次才能找到第i个元素的安放位置，总的比较次数将达到，比直接插入排序更差。但对于一个随机的序列，快速排序的时间复杂度为，并且通过大量实验证明，当n很大时，快速排序是所有内排序中最快的一个。

优化方法：快速排序的基准取值有一定讲究，若能合理地选择基准元素，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，可以加速排序速度，但由于初始序列的随机性，这个要求很难办到。我们可以通过取待排序列中第一个、中间一个、最后一个对象中的中位数为基准对象，一定程度上可能形成优化。

### 2.3.6 堆排序

堆排序可以不断将一个序列中的最小值弹出，不断将最小值弹出直至最后一个元素，这样就完成了排序。

执行过程：先将长度为n的序列依次插入堆，然后执行n次removeMin()算法，完成排序。

复杂度分析：设堆中有n个元素，且，则对应的完全二叉树有k层，在第一个形成初始堆的循环中对每一个非叶结点调用了依次堆调整算法siftDown()，因此该循环所用的时间为，在依次弹出最小值的过程中，调用了n-1次siftDown()函数，复杂度为，因此，堆排序总的时间复杂度为。

### 2.3.6.1 堆的思想

堆是一个半有序的树型结构，因此采用树的存储方法，这里使用顺序存储。将n个结点值存入0~n-1的数组中。由于堆是一个完全二叉树，因此结点序号和父子关系之间有良好的对应关系，即对于结点i，若则为根结点，不然则其父节点为，左子女为，右子女为。对于最小堆，每个结点的关键码均小于或等于它左右子女的关键码，因此堆顶元素即为最小值。

### 2.3.6.1 成员函数的实现：

插入元素及向上调整：若堆满则返回false，不然则将要插入的元素放在数组的最后，然后调用siftUp()向上调整堆。首先将新元素作为操作对象，将其与父结点比较，若大于或等于则停止，不然则将其与父结点交换，而后将交换后的新结点作为对象继续如上操作，直至其大于或等于父结点或成为堆顶元素为止。

2、删除并返回堆顶元素及向下调整：在该函数中传入一个引用类型的变量将其赋值为堆顶元素，从而实现堆顶元素的返回；而后将堆顶元素赋值为堆尾元素，再调用siftDown()向下调整堆。首先将堆顶元素作为操作对象，将其与较小的子女进行比较，若本身更大，交换两者，然后将交换后的新结点作为对象继续如上操作，直至结点本身较小或已经到达堆尾为止。

### 2.3.7 归并排序

归并排序的核心思想是将两个有序表合并成一个新的有序表，并采用二分的思想。

执行过程：初始序列为n，首先将其拆分为长度基本相等的两部分，这样就得到了两个子序列，对于每个子序列，继续进行拆分，直至每个子序列都只有一个元素，即将长度为n的序列拆分为n个长度为1的序列。归并时，先做两两归并，得到n/2个长度为2的归并项，再做两两归并......最后得到一个长度为n的有序序列。

归并时，对于长度为al和bl的两个序列a和b（它们已经有序），设合成为长度为cl=al+bl的序列c，定义两个指针pa和pb分别指向a和b序列的初始位置，依次比较pa和pb的值，将小的值加入到c中，并使该指针指向下一个元素，直至a或b序列中的元素已经全部完成归并，而后将剩余的序列元素全部加入到c中，这就完成了两个有序序列归并为另一个有序序列的操作。

复杂度分析：利用复杂度的递推，，第n次归并操作需要有两个已经归并好的子序列，需要时间，第n次归并操作是合并为一个长度为n的序列，因此需要次操作，对递推式进行通项公式的求解，则有。

### 2.3.8 基数排序

基数排序的原理是多关键码排序。

执行过程：初始序列为n，假设这n个数都是在1~99999之间的五位数（代码中rand()随机数是其子区间，不然则计算最大的数并得到其位数d，此处d=5），每一位都是一个关键码。从个位开始，先按照个位的大小对序列进行排序，而后十位、百位......直至第d位都排序完毕，便得到排序完毕的序列。对第i位的关键码排序，我们首先选出每个数的第i位数k，k共有0~9即10种可能，对应一个大小为10的count[]数组，count[k]表示所有数第i位中出现k的次数；我们将count[]依次累加，即count[j]+=count[j-1]()，此时count[k]存储的是第i位为k的数在当下关键码排序结束后的最后位置，如count[9]=MAX-1，即第i位为9的数的最后位置在MAX-1处（序列中数的总数为MAX，而9又是最大的，因此肯定存储与MAX-1处）；我们对整个序列进行倒序遍历，当遇到第i位为k的数时，我们将其存于temp的count[k]-1处，并将count[k]减1，用于放置下一个第i位为k的数。由于count[k]-count[k-1]即为第i位为k的数的个数，所以temp中第(count[k-1])~(count[k]-1)即为第i位为k的所有数所在的位置；采用倒序遍历的原因是为了保持上一位i-1的有序性，因为第1~i-1位已经按降序完成排列，而在放置第i位为k的数时，是采用从count[k]-1直至count[k-1]的顺序，因此倒序遍历能保持先前结果的有序性。

## 2.4 算法分析

### 2.4.1 时间复杂度

冒泡排序、选择排序、插入排序的时间复杂度为。

折半插入排序、快速排序、堆排序、归并排序的时间复杂度为。

基数排序的时间复杂度为，e为关键码个数。

### 2.4.2 空间复杂度

冒泡排序、选择排序、（折半）插入排序、希尔排序的空间复杂度为。

快速排序平均递归的次数是，因此空间复杂度为。

归并排序在每次归并过程中需要空间存储新的序列，因此总的需要一个长度为n的空间。

堆排序的空间复杂度为，因为每次拿出堆顶元素之后堆的有效数据减少了1，可以利用这个多余的空间来存储排序后的结果。

基数排序的空间复杂度为（e为关键码总数），因为需要一个长度为e的count数组存储每个关键码的元素个数，还需要长度为n的temp数组存储每次按照第i个关键码的排序结果，再将其赋值给原数组。

### 2.4.3 稳定性

稳定性：对于相等的两个元素，若排序前后其相对位置保持不变，则为稳定的排序；不然则是不稳定的排序。

冒泡排序：两两交换，不存在跨元素的元素交换，因此稳定。

选择排序：由于在选择最大/小元素时会存在跨元素的交换，因此不稳定。

插入排序：在每次插入元素时只进行两两交换，因此稳定。

希尔排序：由于在间距为gap的子序列进行直接插入排序时，会出现跨元素的交换，因此不稳定。

快速排序：选中基准后，将原序列划分为小于基准和大于基准的两部分时，会出现跨越元素的交换，因此不稳定。

堆排序：在堆的下调操作siftDown()时会出现跨越元素的交换（因为整个堆采用完全二叉树存储），因此不稳定。

归并排序：由于在归并的过程中，是逐个选择最小的元素进入新的序列，因此两个相等的元素不会交换位置，也不存在跨越元素的交换，因此稳定。

基数排序：两个相等的元素在每位关键码的排序过程中，原本在前面的元素总是会后于在之后的元素，而基数排序又采用倒序遍历赋值的方式，不会发生位置的改变，因此稳定。

### 2.4.4 最好情况与最坏情况

冒泡排序：最好情况下，原序列已经有序，因此一轮循环过后发现没有元素发生交换，直接结束排序过程，故复杂度为；最坏情况下仍为。

选择排序：无论原序列如何，每轮循环都需要选出当前序列的最大值，并且没有方法检测当前序列的排布情况（不像冒泡排序，在一轮循环的过程中若没有元素发生交换则可以判断当前序列是有序的），因此最好和最坏情况下都需要的复杂度。

插入排序：最好情况下原序列已经有序，遍历一遍就知道不存在后项小于前项；最坏情况下序列为逆序，需要交换次，因此复杂度为。

希尔排序：最好情况下原序列已经有序，因此复杂度为；最坏情况下，可以通过优化增量序列，使得其复杂度不超过。

快速排序：最好情况下，每次都能均匀地划分序列，此时复杂度为，最坏情况下每次用于划分序列地元素都为最大或最小，这样导致所有数都划分到一个序列去了，此时复杂度为。

堆排序：堆排序所涉及的“插入堆”以及“退出最小元素”两个操作不可避免地要做n次，而两者的复杂度皆为级别，因此无论最好情况或是最坏情况，堆排序的时间复杂度都为。

归并排序：归并排序所涉及的归并操作，无论原序列如何，都要将他们分成最小单位两两进行归并，归并次数为次，而每次归并时的操作次数与原序列无关，故无论最好最坏情况，归并排序的时间复杂度都为。

基数排序：基数排序比较的是关键码，无论原序列如何，若元素总数为n，关键码个数为k，对于每个关键码，总要将其分配入k个不同关键码的容器，然后再取出，故无论最好最坏情况，基数排序的时间复杂度都为。

# 3 总结

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 排序算法 | 时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定性 |
| 冒泡排序 |  |  |  |  | 稳定 |
| 选择排序 |  |  |  |  | 不稳定 |
| 插入排序 |  |  |  |  | 稳定 |
| 希尔排序 |  |  |  |  | 不稳定 |
| 快速排序 |  |  |  |  | 不稳定 |
| 堆排序 |  |  |  |  | 不稳定 |
| 归并排序 |  |  |  |  | 稳定 |
| 基数排序 |  |  |  |  | 稳定 |

# 4 运行截图

