课程设计报告

课程名称：数据结构

题目名称：排序算法效率比较

学生学院：计算机与信息科学学院

专业班级：计算机科学与技术 2020 级 4 班

小组组长：胡奉民

小组成员：胡奉民 谭恩遥 文海东

指导老师：唐万梅　　职称　教　授

2022 年 6 月 28 日

目录

排序算法效率比较

1 概述分析

（1）设计目的

由于学习了很多不同的排序算法，每个排序算法的效率各有高低，而且各个排序算法的时间复杂度及执行效率比较不够直观，所以我们设计开发了此程序，以对各个排序算法进行时间复杂度及执行效率比较，让排序算法的效率能够更直观的体现出来。

（2）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。

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。

（3）功能分析

首先，得在程序中实现了八种排序算法，其次，要有两个辅助函数，分别计算排序前后所花费的时间与排序的比较次数。

最后，在主程序中，给用户提供界面，选择某种排序算法后，展示排序所需要的时间和比较次数。用户可以不断选择排序算法直到退出。用户也可以任意选择想排序的规模。

（4）开发环境

系统环境：

Windows 11

硬件环境：

CPU：AMD 移动端 4600H

内存：16GB

软件环境：

JetBrains CLion 2022.1.1

2 工作分配

项目工作分配如下所示：

（1）框架设计：

由组长胡奉民和组员谭恩遥，文海东，一起进行构思。

（2）程序设计：

由文海东负责冒泡排序、插入排序、堆排序的子功能设计；谭恩遥负责选择排序、希尔排序、基数排序的子功能设计；胡奉民负责快速排序、归并排序的子功能设计，以及辅助函数和主界面调用功能设计。最后全员进行系统测试。

（3）文档制作：

由胡奉民辅助 Word 文档制作，文海东负责 PPT 设计，谭恩遥负责发言稿及讲述。

（4）视频录制：

由谭恩遥负责讲解视频的录制。

3 设计思路

我们将从以下三个方面讲述我们的程序设计思路：

（1）排序算法设计

八种排序算法之中，都会传入一个引用参数 exchangeTime 代表交换次数，当交换时让 exchangeTime 值+1。每个排序算法都会将原来的随机数组复制一遍，以免把原有的随机数组更改了。

同时为了规范，数组在进行交换数据时，都会调用同一个辅助函数 exch 进行交换，控制了变量。为了确保已经排序，也会调用 isSorted 确保已经排序。

（2）辅助函数设计

首先，在 SortCompare 类中，构造函数将会为它的成员变量 vec 赋予 N 个随机数，然后在 test 类中，根据用户输入的参数，选择适当的算法进行排序。且在排序的前后计时，两者相减，得到了该排序算法的用时。

然后，exch()通过传入数组与要交换的两个位置把元素交换，isSorted()通过一次循环比较后方元素是否大于前方元素来确保排序，否则返回 false。

（3）主程序设计

首先，用一个数组将数字与排序算法的名字关联起来，便于查看。输出窗口打印出表格以给用户进行提示，让用户输入排序的规模 N。随后进入循环，直到用户按 9 退出程序。循环中，根据用户输入，调用相应的函数，执行相应的操作， 完成排序，并打印排序的时间与交换次数。将交换次数清零，进行下一次选择，直到用户退出。

（4）结构展示

程序结构展示如下图所示：

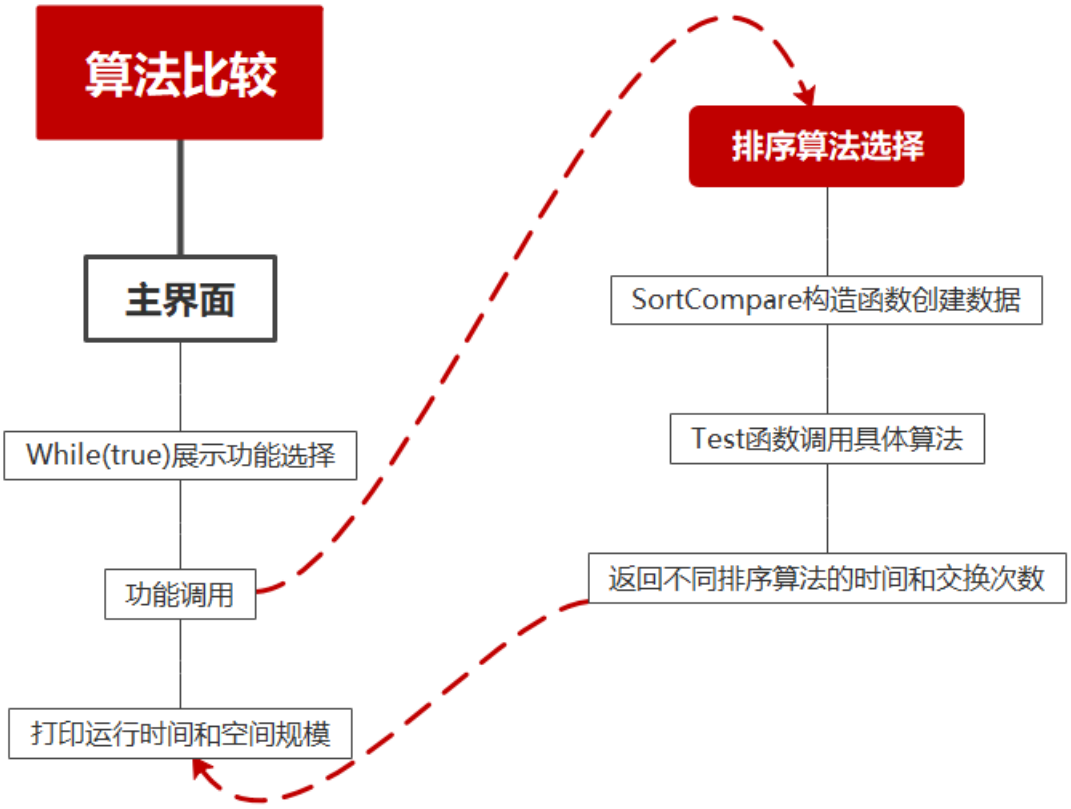

图 3-1 结构展示

4 功能实现

4.1 八个排序方法的实现

程序功能主体由八个排序算法组成，下面结束这些排序算法的设计实现思路：

（1）冒泡排序：

两遍循环，外层循环每次将最小的元素排定，直到整个数组有序。内层循环从数组最后一个元素分别与前一个元素对比，如果小于前一个元素，将两者交换，并且使交换次数+1，一直对比到排定的元素之前，因为排定的元素一定是小于后面的所有的元素的。

值得注意的是，每个排序都通过将原数组复制，并将复制后的数组 temp 进行排序来进行测试，并且每次排序完，都调用测试函数 isSorted 测试是否排序，所有排序都用了这个模式，所以之后不再赘述。

代码展示如下：

```
void bubbleSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    //每次都把最小的排定
    for (int i = 1; i < temp.size(); ++i)
    {
        for (int j = temp.size() - 1; j >= i; --j)
        {
            if (temp[j - 1] > temp[j])
            {
                exch(temp, j, j - 1);
                ++exchangeTimes;
            }
        }
    }
    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（2）选择排序：

同样是两层循环，外层循环先让 i 做为最小的元素，并且通过右层循环找出数组右边最小元素的索引，然后再交换 i 与内层循环找到的最小元素 min，并且让交换次数+1。

代码展示如下：

```
void chooseSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    for (int i = 0; i < temp.size() - 1; ++i)
    {
```

```
            int min = i;
            for (int j = i + 1; j < temp.size(); ++j)
            {
                if (temp[min] > temp[j])
                {
                    min = j;
                }
            }
            exch(temp, min, i);
            ++exchangeTimes;
        }
        if (!isSorted(temp))
        {
            cout << "没排序!" << endl;
        }
    }
```

（3）插入排序：

两层循环，外层循环从一开始到数组末尾。内层循环不断的将该元素 j 与它前面的一个元素 j-1 的值进行对比，如果小与索引 j-1 的值则交换，直到 j-1<0 了或者它的值大于索引 j-1 的值。算法本质就是左边一开始是一个元素，然后不断的让数组后面的元素插入到合适的位置，直到所有元素都插入，而它们也就是一个有序的数组了。

代码展示如下：

```
void insertSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    for (int i = 1; i < temp.size(); ++i)
    {
        for (int j = i; j - 1 >= 0 && temp[j] < temp[j - 1]; --j)
        {
            exch(temp, j - 1, j);
            ++exchangeTimes;
        }
    }
    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（4）希尔排序：

本质上希尔排序就是插入排序的改进。先用 h 不断*3+1，直到到数组大小的三分之一，然后和插入排序的循环类似，只不过外层循环是从 h 开始，且内层循环每一次对比的都是 j 和 j-h 的元素，这样一次排序就形成了 h 有序数组，即每相隔 h 位置的元素形

成了一个有序数组，然后最外面的循环负责不断的缩小 h 的值，使得形成的 h 越来越小，而当 h=1 时，即为插入排序，并且此次排序结束后循环终止。

之所以要这么麻烦，是为了刚刚开始时，h 大概是数组的三分之一左右，然后每一次交换都能让相对很小的元素放在很前面，不必像是插入排序一样，如果碰到最小的元素，就要一个个的比较交换，直到它到最左边的位置。

代码展示如下：

```
void shellSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    int h = 1;
    while (h < temp.size() / 3)
    {
        h = 3 * h + 1;
    }
    while (h >= 1)
    {
        for (int i = h; i < temp.size(); ++i)
        {
            for (int j = i; j - h >= 0 && temp[j] < temp[j - h]; j -= h)
            {
                exch(temp, j - h, j);
                ++exchangeTimes;
            }
        }
        h /= 3;
    }
    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（5）快速排序：

用了额外的两个辅助函数，每次排序，先调用辅助函数 partition，它将返回一个索引 j，且保证了 j 的左边元素都小于它，而它的右边元素都大于或等于它。然后继续递归地将左边的数组排序，将右边的数组排序，直到只有一个元素时返回。因为每次调用，j 的位置都是排定的，所以通过归纳法可知最后数组一定排序好了。

Partition 函数具体实现则是：先将传入的第一个元素（传入的元素不一定就是数组的首元素，而是要排序的子数组的第一个元素，用 lo 标定）定为 v，用 i，j 指向该要排序的数组，然后 i 不断的向右扫描，j 不断的向左扫描，如果 i 找到了大于等于 v 的元素，j 找到了小于等于它的元素，则将他们两个交换。重复扫描直到 i>=j 或者 i 到了数组末尾，或者 j 到达数组开头。此时可知索引 j 的元素肯定是小于 v 的，而 i 一定是大于 v

的，此时只要交换索引 j 和首元素，再返回索引 j，即实现了 j 左边都是小于它的而右边都是大于等于它的。

代码展示如下：

```
void quickSort(vector<int> &temp, int lo, int hi, long long &exchangeTimes)
{
    if (hi <= lo)
    {
        return;
    }
    int j = partition(temp, lo, hi, exchangeTimes);
    quickSort(temp, lo, j - 1, exchangeTimes);
    quickSort(temp, j + 1, hi, exchangeTimes);
}

void quickSort(long long &exchangeTimes)
{
    vector<int> temp(vec);

    //因为本来就是乱的,所以不用打乱了
    quickSort(temp, 0, temp.size() - 1, exchangeTimes);

    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（6）堆排序：

首先构造一个堆，从数组长度/2 的位置直到第一个元素进行构建，因为数组长度/2 到数组末尾的可以直接作为堆底，不用构建了，所谓构建，就是对每一个索引 i 调用 sink 函数，直觉上来说就是把大的元素放上面，小的放下面。然后排序阶段，将数组最后一个元素与第一个交换，即将堆最大的元素放再数组末端，然后对数组首元素进行 sink 操作，来保证堆的有序性。N--，不断这样操作直到 N==1，意味着数组已经排序。

Sink 函数主要是将元素下沉，即通过对比该元素 k 与它的下一层 2*k 与 2*k+1 中较大的一个，如果 k 小于它的下一层，则交换，且让它等于 2*k 或者 2*k+1，不断比较交换，直到到堆末尾或者它大于它下一层的较大的元素。

代码展示如下：

```
void sink(vector<int> &temp, int k, int N, long long &exchangeTimes)
{
    while(2*k<=N)
    {
        int j = 2 * k;
        if(j+1<=N && temp[j+1]>temp[j])
        {
            ++j;
```

```
        }
        if(temp[k]>temp[j])
        {
            break;
        }
        exch(temp, k, j);
        k = j;
        ++exchangeTimes;
    }
}

void heapSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    int N = temp.size();
    temp.insert(temp.begin(), -1);//temp[0]不用
    //构造堆
    for (int k = N/2; k >= 1 ; --k)
    {
        sink(temp, k, N,exchangeTimes);
    }
    //排序
    while(N>1)
    {
        exch(temp, 1, N--);
        ++exchangeTimes;
        sink(temp, 1, N,exchangeTimes);
    }
    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（7）归并排序：

总的来说，是用了分治法的思路，即，首先将左边一半数组排序，右一半数组排序。然后将两个排序的数组进行归并。而将左右两边数组排序是采用递归的思路，例如将左边的一半数组排序，则为将左边的数组的左一半数组排序，右一半数组排序，再将之合并。归并到最后的结果就是，如果有一个子数组只有一个或者没有元素就返回，这时算是已经排序了的。然后将之不断合并。一直到整个数组合并为一个有序数组。

合并两个有序算法调用了一个辅助函数 merge，思路很简单，即为用两个索引分别指向他们两个数组的开头，然后比较这两个索引的元素，存储结果的数组存储较小的那个元素，然后对于的索引+1，一直对比到其中的一个索引指向了那个数组的尽头。此时，结果数组直接全部存储剩余那个数组的剩余元素即可。

代码展示如下：

```
void merge(vector<int> &temp, vector<int> &aux, int lo, int mid, int hi,
long long &exchangeTimes)
{
    //mid 左右两边已经拍好了序
    int i = lo, j = mid + 1;
    for (int k = lo; k <= hi; ++k){
        aux[k] = temp[k];
    }
    for (int k = lo; k <= hi; ++k){
        //上面两个不进行比较
        if (i > mid) temp[k] = aux[j++];
        else if (j > hi) temp[k] = aux[i++];
        else if (aux[j] < aux[i])
{
            temp[k] = aux[j++];
            ++exchangeTimes;
        }
        else
        {
            temp[k] = aux[i++];
            ++exchangeTimes;
        }
    }
}

void mergeSort(vector<int> &temp, vector<int> &aux, int lo, int hi, long
long &exchangeTimes)
{
    if (hi <= lo){
        return;
    }
    int mid = lo + (hi - lo) / 2;
    mergeSort(temp, aux, lo, mid, exchangeTimes);
    mergeSort(temp, aux, mid + 1, hi, exchangeTimes);

    merge(temp, aux, lo, mid, hi, exchangeTimes);

}

void mergeSort(long long &exchangeTimes)
{
    vector<int> temp(vec);
    vector<int> aux(vec.size());//辅助数组

    mergeSort(temp, aux, 0, temp.size() - 1, exchangeTimes);

    if (!isSorted(temp))
    {
        cout << "没排序!" << endl;
    }
}
```

（8）基数排序：

本质上是低位优先的字符串排序，所以首先得到数组的元素中最大元素的位数。然后对每一位进行一次排序，因为如果前一位的不同，则它们肯定能排序，而前一位的相同，它是稳定的放入，而后一位已经排序好了，所以可以直到从低位往高位不断排序直到最后一定能排序好。

外层循环对每一位进行排序，而排序具体实现是：首先统计每个元素在这一位中 0-9 出现的个数，然后通过刚刚统计的元素个数，计算得到每一个元素放入辅助数组的开始的索引，然后对于每一个元素放入辅助数组相应的位置，就是通过查看元素这一位的值，在辅助数组找到相应的索引位置放入，并让该索引位置+1。直到把所有元素放入，即为将这一位的所有元素进行排序了。

代码展示如下：

```
//实际上就是低位优先的字符串排序
void radixSort(long long &exchangeTimes){
    vector<int> temp(vec);
    vector<int> aux(temp);
    int N = temp.size();
    int R = 10;//一共只有 0-9
    int digit = getDigit(temp);
    int base = 1;//目的是得到一个数的某一位的数,如得到 123 中的 2
    //0 表示个位
    for (int d = 0; d < digit; ++d){
        vector<int> count(R + 1);//计算出现的频率
        for (int i = 0; i < N; ++i){
            //就算有一些的位数不够也没有关系,因为这样会是 0,还是会排序
            int index = temp[i] / base % 10;//第一次得到是个位,第二次是十位..
            count[index + 1]++;
        }
        for (int r = 0; r < R; ++r){ //将频率转换为索引
            count[r + 1] += count[r];
        }
        for (int i = 0; i < N; ++i){//将元素分类
            int index = temp[i] / base % 10;
            aux[count[index]++] = temp[i];
        }
        for (int i = 0; i < N; ++i)//回写
        {
            temp[i] = aux[i];
        }
        base *= 10;//不断处理高位
    }

    if (!isSorted(temp)){
        cout << "没排序!" << endl;
    }
}
```

4.2 辅助函数功能实现

exch()通过传入数组与要交换的两个位置把元素交换，isSorted()通过一次循环比较后方元素是否大于前方元素来确保排序，否则返回 false。

（1）exch()函数：

代码展示如下：

```
void exch(vector<int> &cur, int i, int j)
{
    int temp = cur[i];
    cur[i] = cur[j];
    cur[j] = temp;
}
```

（2）isSorted()函数：

代码展示如下：

```
bool isSorted(vector<int> &cur){
    for (int i = 0; i < cur.size() - 1; ++i)
    {
        if (cur[i + 1] < cur[i])
        {
            return false;
        }
    }
    return true;
}
```

4.3 主界面实现

进入一个循环，根据用户输入，调用不同的排序算法进行排序测试，并且打印测试时间与比较次数，随后重置排序时间并且进行下一次比较。

代码展示如下：

```
while(true){
    cout << "请选择排序算法:";
    cin >> whichSort;
    if(whichSort==9){
        break;
    }
    double start = GetTickCount();
    cout << "排序前时间: " << (long)start <<endl;
    sortCompare.test(whichSort, exchangeTimes);
    double  end=GetTickCount();
    cout << "排序后时间: " << (long)end <<endl;
    sortTime = end - start;
    cout<<changeIndex[whichSort]<<"所用时间:"<<sortTime<<"ms"<<endl;
```

```
        //归并排序不进行两两交换,所以看比较次数
        if(whichSort==7)
            cout << changeIndex[whichSort] << "比较次数:" << exchangeTimes << endl;
        else
            cout << changeIndex[whichSort] << "交换次数:" << exchangeTimes << endl;
        //重置
        sortTime = 0.0;
        exchangeTimes = 0;
    }
```

5 功能测试

程序打印出简单界面做信息提示，用户输入数据量后选择要测试的排序算法，代码运行截图如图 5-1、5-2、5-3 所示：

```
D:\C和C++\数据结构课程设计\SortCompare\cmake-build-debug\SortCompare.exe
**              排序算法比较              **
========================================
**              1---冒泡排序              **
**              2---选择排序              **
**              3---直接插入排序          **
**              4---希尔排序              **
**              5---快速排序              **
**              6---堆排序                **
**              7---归并排序              **
**              8---基数排序              **
**              9---退出程序              **
========================================
请输入要产生的随机数的个数：100000
请选择排序算法:1
排序前时间：509276875
排序后时间：509328843
冒泡排序所用时间:51968ms
冒泡排序交换次数：2506044511
```

图 5-1 测试结果截图 1

```
请选择排序算法:3
排序前时间：509657109
排序后时间：509685906
直接插入排序所用时间:28797ms
直接插入排序交换次数：2506044511
请选择排序算法:4
排序前时间：509726312
排序后时间：509726375
希尔排序所用时间:63ms
希尔排序交换次数：3071552
请选择排序算法:5
排序前时间：509730390
排序后时间：509730421
快速排序所用时间:31ms
```

图 5-2 测试结果截图 2

```
请选择排序算法:6
排序前时间：509731156
排序后时间：509731187
堆排序所用时间:31ms
堆排序交换次数：1574669
请选择排序算法:7
排序前时间：509731953
排序后时间：509731984
归并排序所用时间:31ms
归并排序比较次数：1536134
请选择排序算法:8
排序前时间：509732734
排序后时间：509732750
基数排序所用时间:16ms
基数排序交换次数：0
请选择排序算法:9
```

图 5-3 测试结果截图 3

6 项目总结

在本次课程设计中，我们遇到了不少问题，但在组员的积极配合下，我们最终成功达成了目标，并且受益颇多，我们将从以下两个方面进行总结：

（1）不足：

经过小组同学的努力，我们也终于结束了这次的课程设计，虽然最后的检验已经通过了，但是其中仍些许不足。由于要通过控制变量来比较不同排序算法的效率，所以每个排序算法必须使用相同的数据，然而当数据量小于 100000 时，由于希尔排序、快速排序、堆排序、归并排序以及基数排序的时间复杂度较其他排序算法来说要简单很多，所以以毫秒为单位的计时功能就不能记录到这几个算法的运行时间；当数据量大于 10000 时，由选择排序、冒泡排序以及直接插入排序的时间复杂度较其他排序算法而言过于复杂，又会导致程序运行等待时间过长。

（2）心得体会：

一个 c ++语言程序从编辑、编译、连接到运行，都要在一定的外部操作环境下才能进行。所谓 "环境" 就是所用的计算机系统硬件、软件条件，只有学会使用这些环境，才能进行程序开发工作。通过大一/大二的 c/c++的学习，以及数据结构理论知识， 熟练地掌握语言开发环境，为此次编写计算机程序解决实际问题打下基础。同时，在今后遇到其它开发环境时就会触类旁通，很快掌握新系统的使用。

此次小组作业在实操过程中，不断出现麻烦。如编译程序检测出一大堆错误。有时程序本身不存在语法错误，也能够顺利运行，但是运行结果显然是错误的。开发环境所提供的编译系统无法发现这种程序逻辑错误，只能靠自己的上机经验分析判断错误所在。但通过大家的努力，成功调试出程序，这也增加了团队协作能力，对未来发展也有极大的帮助。

参考文献：

1. 严蔚敏，李冬梅，吴伟民 数据结构：C 语言版[M] 第二版 北京：人民邮电出版社，2015.

2. 史蒂芬·普拉达（Stephen Prata） 著，姜佑 译 C Primer Plus[M] 第六版 中文版 北京：人民邮电出版社，2019.

3. Stanley B. Lippman，Josée Lajoie，Barbara E. Moo 著，王刚，杨巨峰 译 C++ Primer[M] 第五版 中文版 北京：人民邮电出版社，2013.

4. 乔恩·克莱因伯格（Jon Kleinberg） 著，王海鹏 译 算法设计[M] 北京：人民邮电出版社，2021.