

数据结构

实验报告

专业班级：软件工程 18-4

姓名：刘铭源

学号：2018214937

日期：2019/6/24

一、实验目的和要求

熟练掌握几种排序方法，完成对排序的复习和温习

会分析各种排序的时间和空间开销。

二、实验环境

Dev-C++

三、实验内容

1. 直接插入排序
2. 希尔排序
3. 冒泡排序
4. 选择排序
5. 堆排序
6. 快速排序

四、实验过程

4.1 任务定义和问题分析

任务定义：对六种排序算法进行时间开销的比较

问题分析：（1）如何使得数据足够大

（2）如何保证六种排序算法的实验数据一样

（3）如何直观表示时间变化趋势

解决方案：（1）用随机数产生足够大的实验数据

（2）用主函数调用六种排序算法对同一组随机数进行排序

（3）用折线图表示时间变化趋势

4.2 数据结构的选择和概要设计

1. 直接插入排序

2. 希尔排序

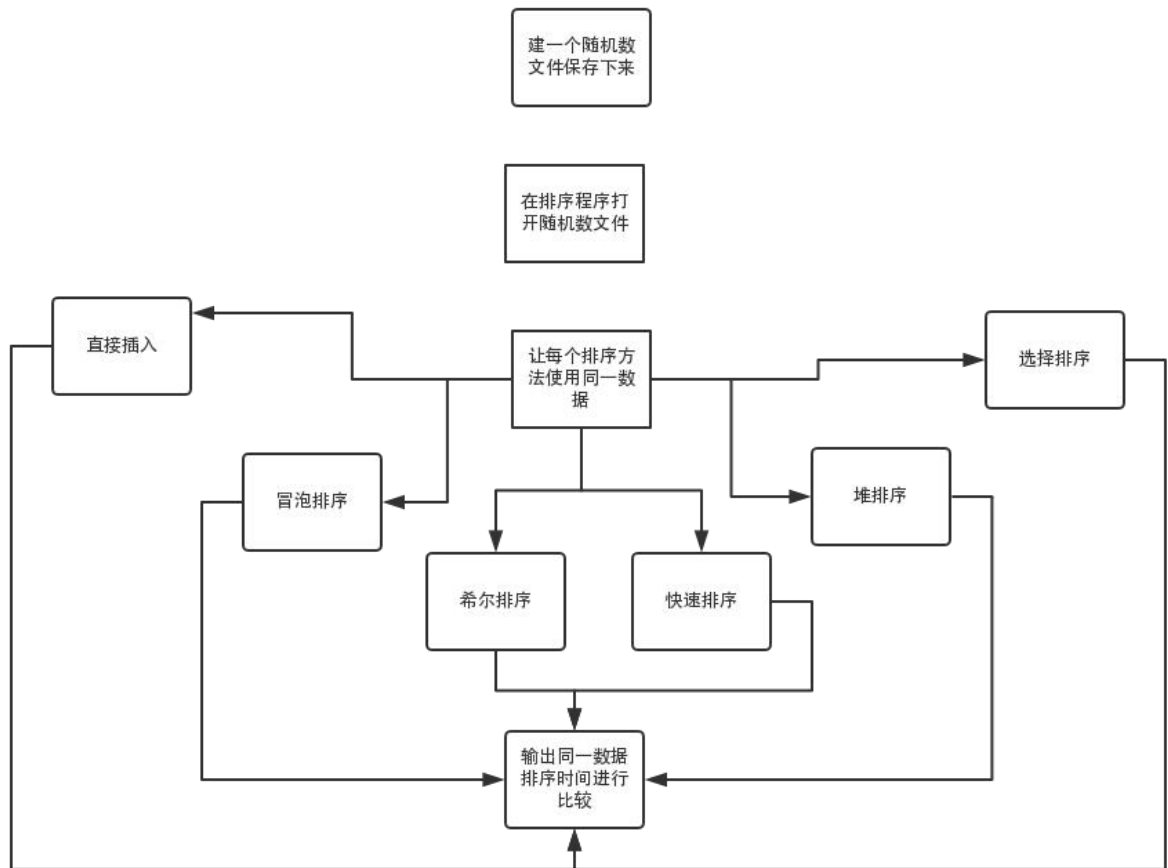
3. 冒泡排序

4. 选择排序

5. 堆排序

6. 快速排序

4.3 详细设计

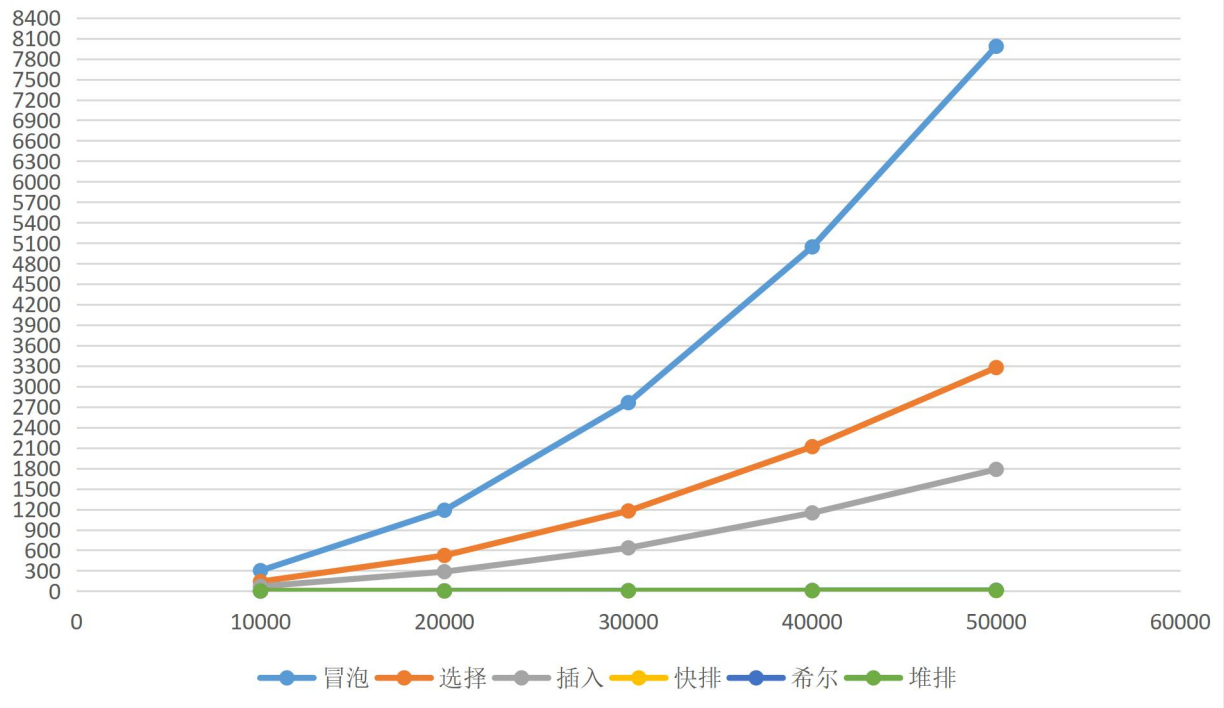


五、测试及结果分析

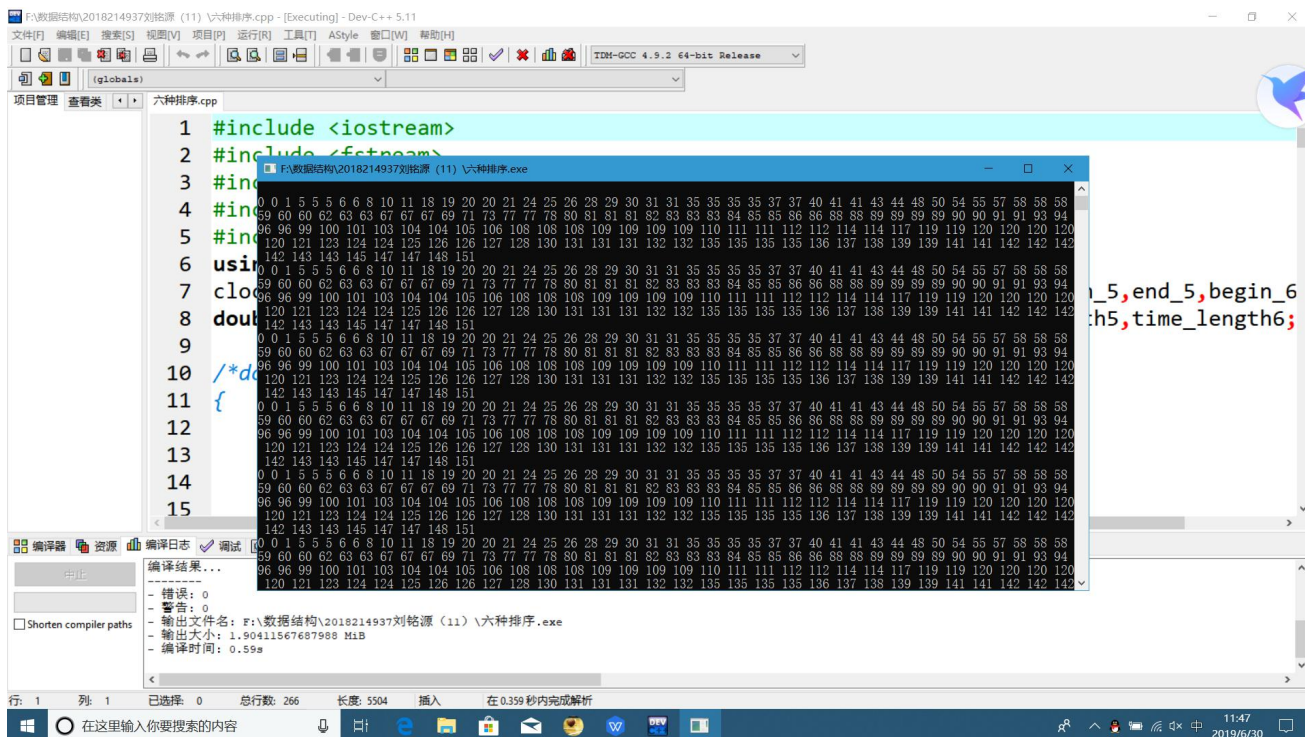
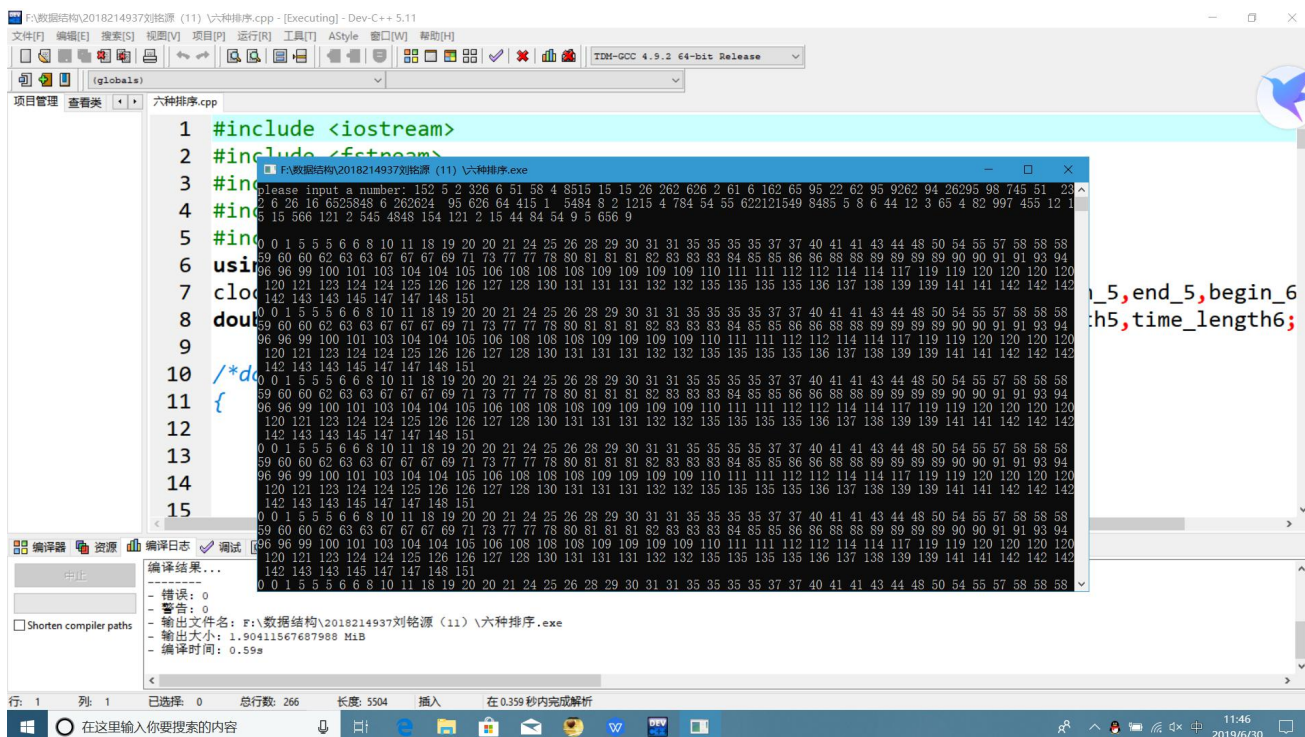
5.1 实验

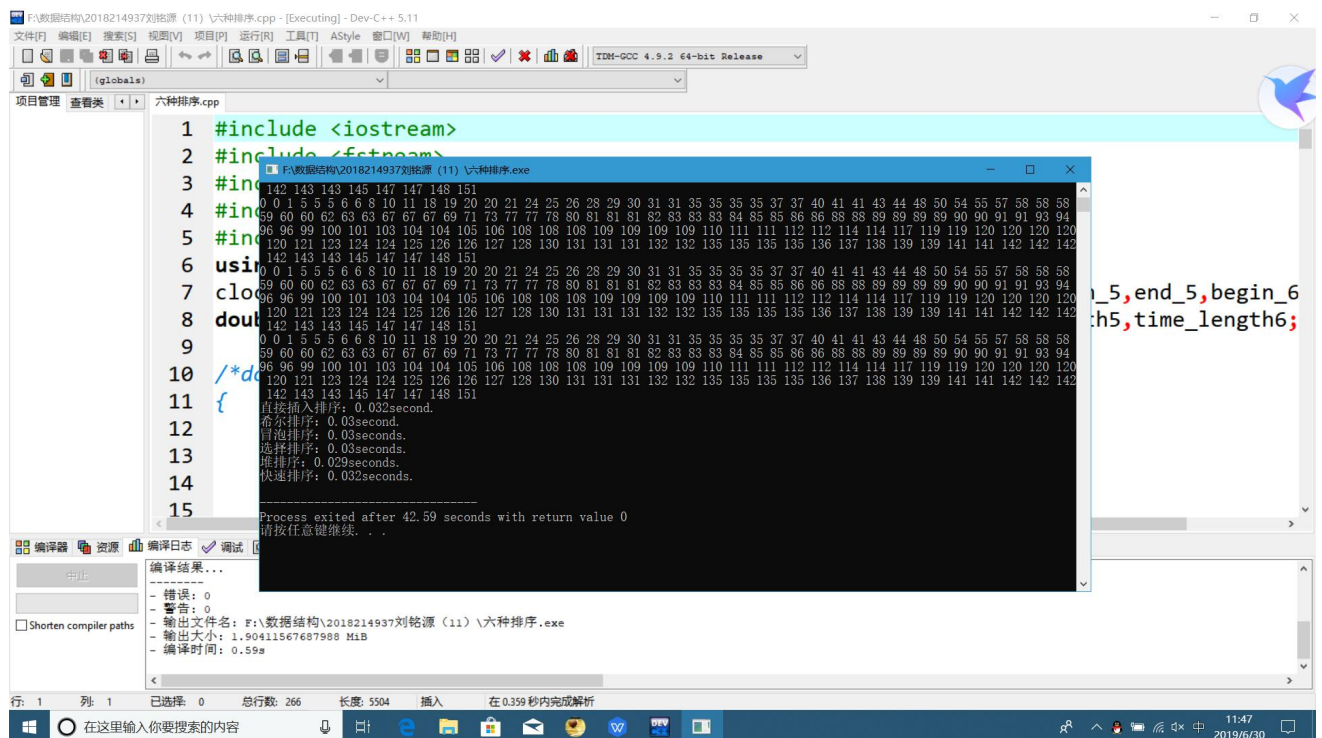
	平均情况	最好情况	最坏情况
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
基数排序	$O(n)$	$O(n)$	$O(n)$
快速排序	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
希尔排序	$O(n^{1.5})$	$O(n)$	$O(n^{1.5})$
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$

图表标题



5.2 结果及分析





```
1 #include <iostream>
2 #include <fstream>
3 #include <vector>
4 #include <algorithm>
5 #include <chrono>
6 using namespace std;
7 const int N = 100;
8 double time_length6;
9 vector<int> v(N);
10 /*定义100个元素的数组*/
11 {
12     直接插入排序: 0.032seconds.
13     希尔排序: 0.03seconds.
14     冒泡排序: 0.03seconds.
15     选择排序: 0.03seconds.
16     堆排序: 0.029seconds.
17     快速排序: 0.03seconds.
18 }
19 int main()
20 {
21     //生成100个随机数
22     for (int i = 0; i < N; i++)
23     {
24         v[i] = rand() % 100000;
25     }
26     //输出初始数组
27     for (int i = 0; i < N; i++)
28     {
29         cout << v[i] << " ";
30         if (i % 10 == 9)
31             cout << endl;
32     }
33     //排序
34     chrono::time_point<chrono::system_clock> begin, end;
35     begin = chrono::system_clock::now();
36     //直接插入排序
37     sort(v.begin(), v.end());
38     end = chrono::system_clock::now();
39     time_length6 = chrono::duration<double>(end - begin).count();
40     cout << "直接插入排序耗时: " << time_length6 << endl;
41     //希尔排序
42     begin = chrono::system_clock::now();
43     //冒泡排序
44     sort(v.begin(), v.end());
45     end = chrono::system_clock::now();
46     time_length6 = chrono::duration<double>(end - begin).count();
47     cout << "冒泡排序耗时: " << time_length6 << endl;
48     //选择排序
49     begin = chrono::system_clock::now();
50     //堆排序
51     sort(v.begin(), v.end());
52     end = chrono::system_clock::now();
53     time_length6 = chrono::duration<double>(end - begin).count();
54     cout << "堆排序耗时: " << time_length6 << endl;
55     //快速排序
56     begin = chrono::system_clock::now();
57     //归并排序
58     sort(v.begin(), v.end());
59     end = chrono::system_clock::now();
60     time_length6 = chrono::duration<double>(end - begin).count();
61     cout << "快速排序耗时: " << time_length6 << endl;
62     //输出排序后的数组
63     for (int i = 0; i < N; i++)
64     {
65         cout << v[i] << " ";
66         if (i % 10 == 9)
67             cout << endl;
68     }
69     return 0;
70 }
```

直接插入排序：插入排序类似于整理扑克牌，基本操作是将一个记录插入到已经排好序的有序数列中，从而得到一个有序但记录数加一的有序数列。

插入排序的时间复杂度为 $O(n^2)$ ，是稳定的排序方法，适用于数量较少的排序。

希尔排序：希尔排序基本思想是将相距某个增量 d 的记录组成一个子序列，通过插入排序使得这个子序列基本有序，然后减少增量继续排序。操作上先取一个小于 n 的整数 d_1 作为第一个增量，把全部记录分成 d_1 个组，所有距离为 d_1 的倍数的记录放在同一个组中。先在各组内进行直接插入排序，然后取第二个增量 $d_2 < d_1$ 重复上述的分组和排序，直至所取的增量 $d_t = 1$ ($d_t < d_{t-1} < \dots < d_2 < d_1$)，即所有记录放在同一组中进行直接插入排序为止。

希尔排序的时间复杂度可以达到 $O(n^{3/2})$

冒泡排序：在最好的情况下，也就是数列本身是排好序的，需要进行 $n - 1$ 次比较；在最坏的情况下，也就是数列本身是逆序的，需要进行 $n(n-1)/2$ 次比较。

因此冒泡排序总的时间复杂度是 $O(n^2)$ 。

选择排序：选择排序的基本思想是每一趟在 $n - i + 1$ ($i = 1, 2, \dots, n - 1$) 个记录中选取关键字最小(或最大)的记录作为有序序列的第 i 个记录，直到所有元素排序完成。选择排序是不稳定的排序算法。

选择排序的时间复杂度为 $O(n^2)$ ，但性能上略优于冒泡排序。

堆排序：堆是具有下列性质的完全二叉树：

1. 每个节点的值都大于或等于其左右孩子节点的值，称为大顶堆；
2. 每个节点的值都小于或等于其左右孩子节点的值，称为小顶堆。

堆排序是指利用堆这种数据结构所设计的一种排序算法。基本思想是把待排序的序列构造成一个大顶堆，此时序列的最大值就是队顶元素，把该元素放在最后，然后对剩下的 $n - 1$ 个元素继续构造大顶堆，直到排序完成。

堆排序的时间复杂度为 $O(n \log n)$ ，由于要构造堆，因此不适用于序列个数较少的情况。

快速排序：快速排序是对冒泡排序的一种改进。基本思想是通过一趟排序将待排记录分割成独立的两部分，其中一部分的记录都比另一部分小，然后再分别对这两个部分进行快速排序，最终实现整个序列的排序。

快速排序的时间复杂度为 $O(n \log n)$ ，是一种不稳定的排序算法

六、实验收获

通过本次实验我学会了几种排序方法，熟练掌握了这几种排序，从一开始只会用冒泡

排序到现在的很多排序方法，比如对比快速的大数据和快速，冒泡排序的速度慢收获了很多知识，这次实验排序让我明白解决方法的多样性，我理解了六种排序的时间和空间复杂度，可以熟练掌握六种排序存储和逻辑结构。

七、参考文献

Csdn

八. 附录（源代码）

详情请打开代码部分