

《数据结构》

实验报告

姓名： 庄李晨

学号： 10522238

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2023年12月

【实验名称】排序算法的实现

【实验目的】

一、掌握插入类排序算法的实现。

二、掌握交换类排序算法的实现。

三、掌握选择类排序算法的实现。

【实验内容】

1. 实验要求：

1． 仔细阅读排序实验代码，补充下面6个函数的实现过程。补充完成main（）函数的定义，给出测试结果。

（1）创建排序表

void Create(SqList &L) { }

（2）输出排序表元素

void Traverse(SqList L){ }

（3）直接插入排序

void InsertSort(SqList &ST) { }

（4）冒泡排序

void BubbleSort(SqList &L){ }

（5）快速排序

void QuickSort(SqList &L){ }

（6）简单选择排序

void SelectSort(SqList &L) { }

2.选作题目

内部排序算法比较

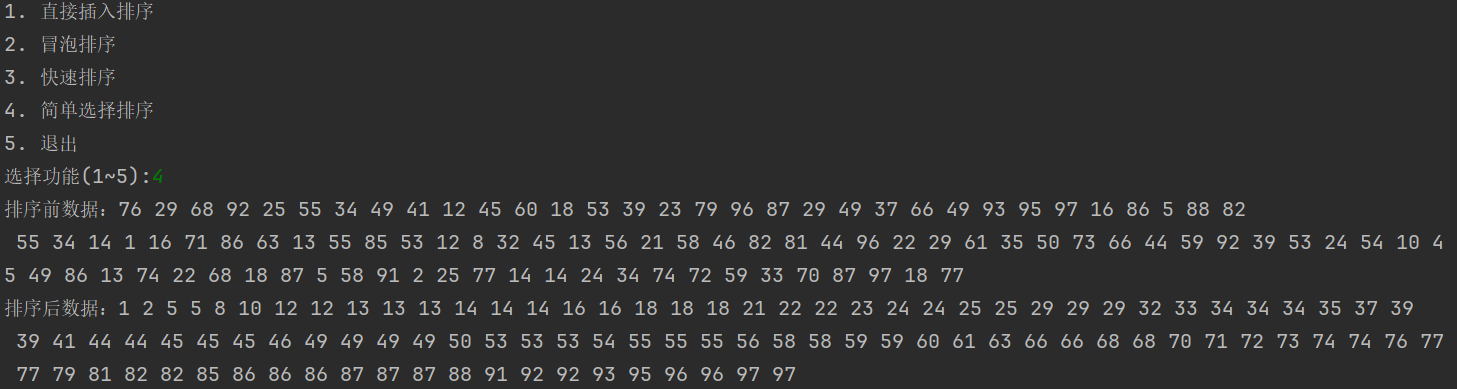
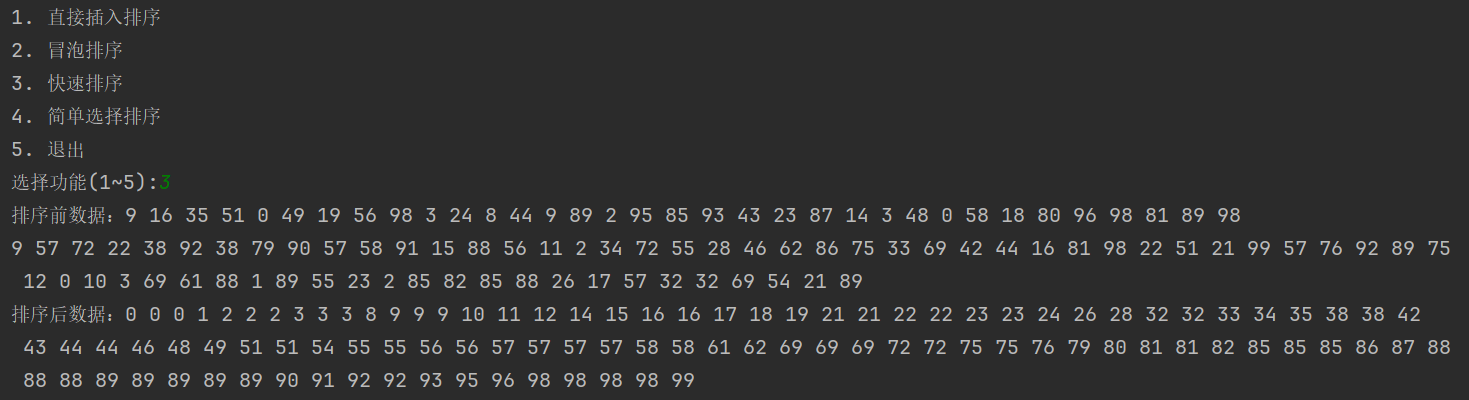
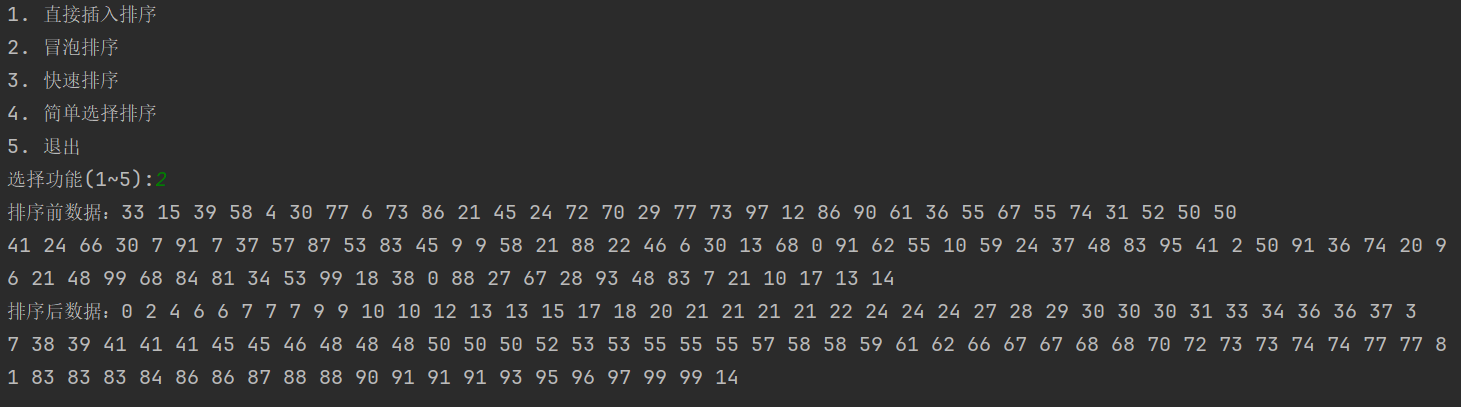
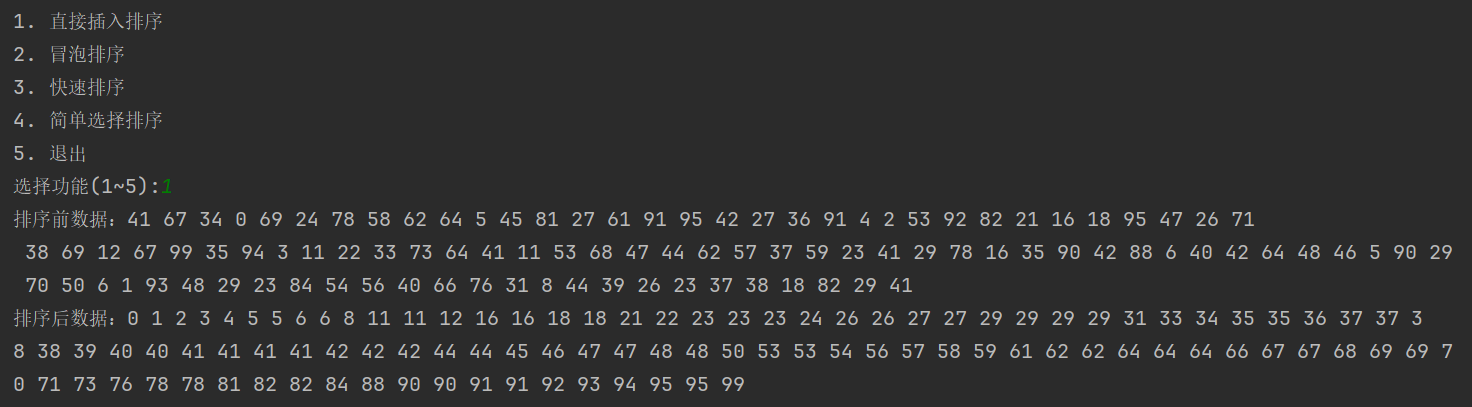
在教科书中，各种内部排序算法的时间复杂度分析结果只给出了算法执行时间的阶，或大概执行时间。试通过随机数据比较各种算法的关键字比较次数和关键字移动次数，以取得直观感受。对7种常用的内部算法进行比较：冒泡排序、直接插入排序、简单选择排序、快速查找排序、希尔排序、堆排序、归并排序。

（1）要求待排序的表长不小于10000，其中的数据要用程序随机产生，至少要用5组不同的输入数据做比较，比较的指标为有关键字参加的比较次数和关键字的移动次数（关键字交换记为3次移动）；

（2）输入数据：整数，由程序随机生成；

（3）输出：每组测试数据针对不同排序算法的关键字参加的比较次数和关键字的移动次数、算法执行时间。

1. 程序清单
2. #include <iostream>
3. #include <stdlib.h>
4. using namespace std;
5. #define MAXSIZE 100
6. typedef int ElemType;
7. typedef struct
8. {
9. ElemType elem[MAXSIZE + 1]; *//数据元素存储空间基址，0号单元留空*
10. int length;                 *//表长度*
11. } SqList;
12. *//创建排序表*
13. void Create(SqList &L)
14. {
15. for (int i = 1; i <= MAXSIZE; i++)
16. {
17. L.elem[i] = rand() % 100; *//生成0~99的随机数*
18. }
19. L.length = MAXSIZE;
20. }
21. *//输出排序表元素*
22. void Traverse(SqList L)
23. {
24. for (int i = 1; i <= L.length; i++)
25. {
26. cout << L.elem[i] << " ";
27. }
28. cout << endl;
29. }
30. *//直接插入排序*
31. void InsertSort(SqList &ST)
32. {
33. int i, j;
34. for (i = 2; i <= ST.length; i++)
35. {
36. if (ST.elem[i] < ST.elem[i - 1])
37. {
38. ST.elem[0] = ST.elem[i];
39. for (j = i - 1; ST.elem[j] > ST.elem[0]; j--)
40. {
41. ST.elem[j + 1] = ST.elem[j];
42. }
43. ST.elem[j + 1] = ST.elem[0];
44. }
45. }
46. }
47. *//冒泡排序*
48. void BubbleSort(SqList &L)
49. {
50. int i, j;
51. bool flag;
52. for (i = 1; i < L.length; i++)
53. {
54. flag = false;
55. for (j = L.length - 1; j >= i; j--)
56. {
57. if (L.elem[j] < L.elem[j - 1])
58. {
59. swap(L.elem[j], L.elem[j - 1]);
60. flag = true;
61. }
62. }
63. if (!flag)
64. {
65. return;
66. }
67. }
68. }
69. *//快速排序*
70. void QuickSort(SqList &L, int low, int high)
71. {
72. if (low < high)
73. {
74. int i = low, j = high, pivot = L.elem[low];
75. while (i < j)
76. {
77. while (i < j && L.elem[j] >= pivot)
78. {
79. j--;
80. }
81. L.elem[i] = L.elem[j];
82. while (i < j && L.elem[i] <= pivot)
83. {
84. i++;
85. }
86. L.elem[j] = L.elem[i];
87. }
88. L.elem[i] = pivot;
89. QuickSort(L, low, i - 1);
90. QuickSort(L, i + 1, high);
91. }
92. }
93. *//简单选择排序*
94. void SelectSort(SqList &L)
95. {
96. int i, j, min\_index;
97. for (i = 1; i < L.length; i++)
98. {
99. min\_index = i;
100. for (j = i + 1; j <= L.length; j++)
101. {
102. if (L.elem[j] < L.elem[min\_index])
103. {
104. min\_index = j;
105. }
106. }
107. if (min\_index != i)
108. {
109. swap(L.elem[i], L.elem[min\_index]);
110. }
111. }
112. }
113. int main(void)
114. {
115. SqList L;
116. int c = 0;
117. while (c != 5)
118. {
119. cout << endl
120. << "1. 直接插入排序";
121. cout << endl
122. << "2. 冒泡排序";
123. cout << endl
124. << "3. 快速排序";
125. cout << endl
126. << "4. 简单选择排序";
127. cout << endl
128. << "5. 退出";
129. cout << endl
130. << "选择功能(1~5):";
131. cin >> c;
132. switch (c)
133. {
134. case 1:
135. Create(L);
136. cout << "排序前数据：";
137. Traverse(L);
138. InsertSort(L);
139. cout << "排序后数据：";
140. Traverse(L);
141. break;
142. case 2:
143. Create(L);
144. cout << "排序前数据：";
145. Traverse(L);
146. BubbleSort(L);
147. cout << "排序后数据：";
148. Traverse(L);
149. break;
150. case 3:
151. Create(L);
152. cout << "排序前数据：";
153. Traverse(L);
154. QuickSort(L, 1, L.length);
155. cout << "排序后数据：";
156. Traverse(L);
157. break;
158. case 4:
159. Create(L);
160. cout << "排序前数据：";
161. Traverse(L);
162. SelectSort(L);
163. cout << "排序后数据：";
164. Traverse(L);
165. break;
166. case 5:
167. cout << "结束操作" << endl;
168. break;
169. }
170. }
171. return 0;
172. }
173. 结果截图



【实验体会】

实验心得：

在这个实验中，我实现了四种常见的排序算法：直接插入排序、冒泡排序、快速排序和简单选择排序。通过实验，我对这些排序算法有了更深入的了解。

在直接插入排序算法中，我使用了一个哨兵元素来简化插入过程。具体实现中，我从第二个元素开始，将当前元素与前面已排序好的元素进行比较，如果当前元素小于前面的元素，则将前面的元素后移，直到找到插入位置，然后将当前元素插入到该位置。

在冒泡排序算法中，我使用了一个布尔类型的标志来判断是否发生了交换，如果没有发生交换，则说明排序已经完成，可以提前退出循环。具体实现中，我依次比较相邻的两个元素，如果前面的元素大于后面的元素，则交换它们的位置，重复这个过程直到排序完成。

在快速排序算法中，我使用了递归的方式来实现。具体实现中，我选择一个基准元素，并将待排序序列分成两个子序列，小于基准元素的放在左边，大于基准元素的放在右边，然后对左右两个子序列分别递归地进行快速排序。

在简单选择排序算法中，我每次都选择一个最小的元素，将其与当前位置的元素交换。具体实现中，我依次遍历未排序的元素，找到最小的元素并记录其位置，然后将最小的元素与当前位置的元素交换，重复这个过程直到排序完成。

通过这次实验，我进一步巩固了排序算法的原理和实现方式。我发现不同的排序算法在时间复杂度和稳定性上都有所不同，选择合适的排序算法可以提高排序的效率。同时，我也学会了使用适当的数据结构来存储和操作排序表，以及如何进行元素的交换和比较操作。