山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202300130183	姓名： 宋浩宇	班级：23 级人工智能班
实验题目：2024 数据结构-数据/智能 实验 2 排序算法		
实验学时：2	实验日期：2024/9/18	

实验目的：

练习名次排序、选择排序、冒泡排序、插入排序的代码的编写和使用。

软件开发工具：

1. visual studio code 2022（使用 C/C++、C/C++ Extension Pack、C/C++ Themes 插件）
2. mingw64 工具包

1. 实验内容

完成 2024 数据结构-数据/智能 实验 2 排序算法 中 A 题排序算法实验。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

我们分别来描述名次排序、选择排序、冒泡排序、插入排序。

一、名次排序

名次排序是十分简明的排序，算法的核心思路就是将数组里的元素依次和其他元素进行比较，以此可以知道该数字大于/小于多少个其他的元素，因为本实验要求我们使用升序排序，因此我们只需知道每个元素大于多少个元素就可以得到该元素在排序上的位置（后续分析算法时默认升序），该算法在元素均互异的时候是非常容易实现的，但如果存在相同的元素，则需要另外一个数组用于存储每个元素的个数，在获得结果的时候按照元素个数进行重复。算法的时间复杂度为 O(n^2)

二、选择排序

选择排序也是十分简单的排序，简单来说就是对于第 i 个位置上的元素，就是遍历第 i 到第 n 个元素，将其中最小的元素转移到第 i 个位置上（反过来排列最大的也一样）。即每次都挑选出剩余元素里最小的元素纳入已排序序列中，显然，使用这种贪心策略的算法可以完成所有元素的排序，且与名次排序不同的是，该算法不需要考虑元素重复个数的问题，但该算法有着需要考虑排序终点的问题，而因为该算法本身并不存在一个仅靠是否操作就可判定该序列是否有序的检查点，所以我们需要在每一轮选择时候检查一次数组是否有序，如果有序则终止排序，作为排序的可控终点。算法的时间复杂度为 O(n^2/2)

三、冒泡排序

冒泡排序与选择排序类似，也是在每一轮循环中将剩余元素中的最小元素排入已排序序列中（反过来排列最大的也一样）。可以证明，对于剩余未排序序列中的最小元素，因为它是最小元素，所以它会一直被交换，直到交换到“水面”上，因此显然该算法可以在 n 次循环后得到有序序列。该算法可以优化的地方比选择排序要多，首先在每轮冒泡之后都可以挑选出一个最小值，因此我们可以每一轮冒泡之后少考虑一个元素，即开始是对元素个数为 n 的序列进行排序，第二轮就是对 n-1 个元素进行排序，以此类推，可以减少相当大的一部分计算量，再有是排序终点，与选择排序不同，冒泡排序有一个很明显的终点，即在一轮“冒泡”中未触发交换这一操作，此时序列是有序的，因此未触发交换操作可以作为排序的可控终点。算法的时间复杂度为 O(n^2/2)

四、插入排序

插入排序的主要思想是将一个元素插入到本身已经有序的序列中，如果序列中的元素大于要被插入的元素，则将该元素后移，继续向前遍历直到找到序列中小于等于要插入元素的元素，将要被插入的元素放入该元素的下一位（即上一个元素后移所“空”出来的位置）。与上述三种排序算法相比，该算法的平均时间复杂度是最低的，算法在最极端情况下（即序列为降序序列）时间复杂度为 O(n^2/2)

3. 测试结果（测试输入，测试输出）
   对于名次排序，测试输入为：
   5
   5 5 4 3 2
   输出为：
   2 3 4 5 5

   对于选择排序，测试输入为：
   5
   5 4 3 2 1
   输出为：
   1 2 3 4 5

   对于冒泡排序，测试输入为：
   5
   5 1 2 3 4
   输出为：
   1 2 3 4 5

   对于插入排序，测试输入为：
   5
   5 4 3 2 1
   输出为：
   1 2 3 4 5

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）
从结果来看，我们的排序算法成功解决了这个实验的问题，另外，经过测试，我们在代码中加入的优化部分也成功降低了算法的耗时，特别的，对于名次排序优化的部分，从结果过来看我们成功解决了元素相同的问题。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```cpp
/*2024 数据结构-数据智能 实验 2 排序算法 A 排序算法.cpp*/
#include<iostream>
using namespace std;

template<class T>
class Solution
```

```
{
private:
    void swap(T& a, T& b)
    {
        T c;
        c = a;
        a = b;
        b = c;
    }
public:
    //名次排序
    void rankSort();
    //及时终止的选择排序
    void chooseSort();
    //及时终止的冒泡排序
    void bubbleSort();
    //插入排序
    void insertSort();
};


template<class T>
void Solution<T>::rankSort()
{
    int n;
    cin >> n;
    T data[n];
    T result[n];
    for (int i = 0;i < n;i++)
    {
        cin >> data[i];
    }
    int rank[n];
    int cnt[n];
    for (int i = 0;i < n;i++)
    {
        rank[i] = 0;
        cnt[i] = 0;
    }

    for (int i = 0;i < n;i++)
    {
        for (int j = 0;j < n;j++)
        {
            if (data[i]>data[j])
            {
```

```
                rank[i]++;
            }
            if (data[i]==data[j])
            {
                cnt[i]++;
            }

        }
    }
    for (int i = 0;i < n;i++)
    {
        for (int j = 0; j < cnt[i]; j++)
        {
            result[rank[i] + j] = data[i];
        }

    }

    for (int i = 0;i < n;i++)
    {
        cout << result[i] << " ";
    }
}

template<class T>
void Solution<T>::chooseSort()
{
    int n;
    cin >> n;
    T data[n];
    for (int i = 0;i < n;i++)
    {
        cin >> data[i];
    }
    for (int i = 0; i < n;i++)
    {
        int min_index = i;
        int min_num = data[i];
        for (int j = i; j < n; j++)
        {
            if (data[j] < min_num)
            {
                min_index = j;
                min_num = data[j];
            }
        }
```

```
99.             swap(data[i], data[min_index]);
100.            bool mark = 1;
101.            for (int j = 0;j < n-1;j++)
102.            {
103.                if (data[j]>data[j+1])
104.                {
105.                    mark = 0;
106.                }
107.            }
108.            if (mark)
109.            {
110.                break;
111.            }
112.
113.        }
114.        for (int i = 0;i < n;i++)
115.        {
116.            cout << data[i] << " ";
117.        }
118.    }
119.
120.    template<class T>
121.    void Solution<T>::bubbleSort()
122.    {
123.        int n;
124.        cin >> n;
125.        T data[n];
126.        for (int i = 0;i < n;i++)
127.        {
128.            cin >> data[i];
129.        }
130.
131.        for (int i = 0;i < n;i++)
132.        {
133.            bool mark = 1;
134.            for (int j = 0;j < n - i - 1;j++)
135.            {
136.                if (data[j] > data[j + 1])
137.                {
138.                    swap(data[j], data[j + 1]);
139.                    mark = 0;
140.                }
141.            }
142.            if (mark)
143.            {
144.                break;
```

```
145.         }
146.     }
147.
148.     for (int i = 0;i < n;i++)
149.     {
150.         cout << data[i] << " ";
151.     }
152. }
153.
154. template<class T>
155. void Solution<T>::insertSort()
156. {
157.         int n;
158.     cin >> n;
159.     T data[n];
160.     for (int i = 0;i < n;i++)
161.     {
162.         cin >> data[i];
163.     }
164.
165.     for (int i = 0; i < n; i++)
166.     {
167.         T temp = data[i];
168.         for (int j = i - 1;j >= 0;j--)
169.         {
170.             if (data[j]>temp)
171.             {
172.                 data[j + 1] = data[j];
173.             }
174.             else
175.             {
176.                 data[j + 1] = temp;
177.                 break;
178.             }
179.             if (j == 0)
180.             {
181.                 data[j] = temp;
182.                 break;
183.             }
184.
185.         }
186.
187.     }
188.
189.
190.
```

```
191.     for (int i = 0;i < n;i++)
192.     {
193.         cout << data[i] << " ";
194.     }
195. }
196. 
197. int main()
198. {
199.     Solution<int> test;
200.     // test.rankSort();
201.     // test.chooseSort();
202.     // test.bubbleSort();
203.     test.insertSort();
204.     return 0;
205. }
```