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：201700130033	姓名： 武学伟	班级： 17 计科 2 班
实验题目：排序算法		
实验学时：4	实验日期： 2018-10-6	

实验目的：

掌握各种简单排序

软件环境：

Win10_home X64

CodeBlocks

1. 实验内容（题目内容，输入要求，输出要求）

1） 创建排序类：数据含有 n 个整数，使用动态数组存储；提供操作：按名次排序、及时终止的选择排序、及时终止的冒泡排序、插入排序；

2） 键盘输入 n，随机生成 n 个 0~1000 之间的整数建立排序实例；输出各种排序算法的排序过程。

3） 统计每一种排序所耗费的时间（即比较次数和移动次数）。

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）

1） 名次排序

首先计算名次，一个元素的名次等于所有比它小的元素个数加上在它左边出现的相同大小元素的个数，从第二个元素开始遍历，当前面的数比后面的小，则后面的名次加一。

之后进行排序，动态创建一个新的数组，将原数组中的元素按照名次数组的顺序，对应的输入到新数组中，再将新数组输入到需要输出的数组中并删除动态建立的数组

2） 插入排序

假设只有一个元素的数组是有序的，将第 2 个元素插入这个数组，得到一个含有两个元素的有序数组，再插入第 3 个元素，直至插入 n 个元素。

3） 及时终止的选择排序

在 n 个元素中选择出最大的数，并与 a[n-1]交换，在剩余的 n-1 个元素中，再选择出最大的元素，与 a[n-2]交换，直至交换到 a[1]。

在交换的过程中数组可能已经有序或者一开始就是有序的，所以为了避免重复的交换，定义 bool 变量 sorted，一开始初始化为 false，当 sorted 为假时，进行选择排序，在寻找最大值的 for 循环中，将 sorted 更改为 true，检查是否有序，即数组的前一个元素总小于后一个元素即为有序，否则将 sorted 重新更新为 false。

这样就可以在数组已经有序时及时终止。

4） 及时终止的冒泡排序

一次冒泡，将最大元素移到数组的右边，遍历数组，比较相邻元素，前大于后，就交换这两个值。

冒泡排序，每次对数组进行冒泡，最大值在右边，第一次对 a[0:n-1]，第

二次 a[0:n-2]，第 n-1 次 a[0:1].

在冒泡中，也可能在过程或者一开始就有序，所以定义 bool 变量 swapped，初始化为 true，冒泡排序的 for 循环在 swapped 为 true 时进行，在循环中更新为 false，若是发生了交换，则将 swapped 更新为 true。

这样就可以在数组已经有序时及时终止。

5) 对于类的数据成员的问题

采用动态建立数组的方式，并且建立两个动态数组，一个用于生成随机数组，另一个用于存储不同排序方法产生的数组。

3. 测试结果（测试输入，测试输出，结果分析）：

第一组

输入：

```
Input the length of your Array
10
The random number:
472 33 309 727 517 516 770 24 688 9
```

输出：

名次排序

```
Rank sort:
9 33 309 727 517 516 770 24 688 9
9 24 309 727 517 516 770 24 688 9
9 24 33 727 517 516 770 24 688 9
9 24 33 309 517 516 770 24 688 9
9 24 33 309 472 516 770 24 688 9
9 24 33 309 472 516 770 24 688 9
9 24 33 309 472 516 517 24 688 9
9 24 33 309 472 516 517 688 688 9
9 24 33 309 472 516 517 688 727 9
9 24 33 309 472 516 517 688 727 770

result:
9 24 33 309 472 516 517 688 727 770
Compare_times: 45
Move_times: 20
```

插入排序

```
Insert sort:
33 472 309 727 517 516 770 24 688 9
33 309 472 727 517 516 770 24 688 9
33 309 472 727 517 516 770 24 688 9
33 309 472 517 727 516 770 24 688 9
33 309 472 516 517 727 770 24 688 9
33 309 472 516 517 727 770 24 688 9
24 33 309 472 516 517 727 770 688 9
24 33 309 472 516 517 688 727 770 9
9 24 33 309 472 516 517 688 727 770

result:
9 24 33 309 472 516 517 688 727 770
Compare_times: 23
Move_times: 41
```

及时终止的选择排序

```
Selection sort:
472 33 309 727 517 516 9 24 688 770
472 33 309 688 517 516 9 24 727 770
472 33 309 24 517 516 9 688 727 770
472 33 309 24 9 516 517 688 727 770
472 33 309 24 9 516 517 688 727 770
9 33 309 24 472 516 517 688 727 770
9 33 24 309 472 516 517 688 727 770
9 24 33 309 472 516 517 688 727 770
9 24 33 309 472 516 517 688 727 770

result:
9 24 33 309 472 516 517 688 727 770
Compare_times: 45
Move_times: 27
```

及时终止的冒泡排序

```
Bubble sort:
33 309 472 517 516 727 24 688 9 770
33 309 472 516 517 24 688 9 727 770
33 309 472 516 24 517 9 688 727 770
33 309 472 24 516 9 517 688 727 770
33 309 24 472 9 516 517 688 727 770
33 24 309 9 472 516 517 688 727 770
24 33 9 309 472 516 517 688 727 770
24 9 33 309 472 516 517 688 727 770
9 24 33 309 472 516 517 688 727 770

result:
9 24 33 309 472 516 517 688 727 770
Compare_times: 23
Move_times: 69
```

第二组

输入：

```
Input the length of your Array
10
The random number:
980 800 797 640 583 703 460 98 698 413
```

输出：

名次排序

```
Rank sort:
98 800 797 640 583 703 460 98 698 413
98 413 797 640 583 703 460 98 698 413
98 413 460 640 583 703 460 98 698 413
98 413 460 583 583 703 460 98 698 413
98 413 460 583 640 703 460 98 698 413
98 413 460 583 640 698 460 98 698 413
98 413 460 583 640 698 703 98 698 413
98 413 460 583 640 698 703 797 698 413
98 413 460 583 640 698 703 797 800 413
98 413 460 583 640 698 703 797 800 980

result:
98 413 460 583 640 698 703 797 800 980
Compare_times: 45
Move_times: 20
```

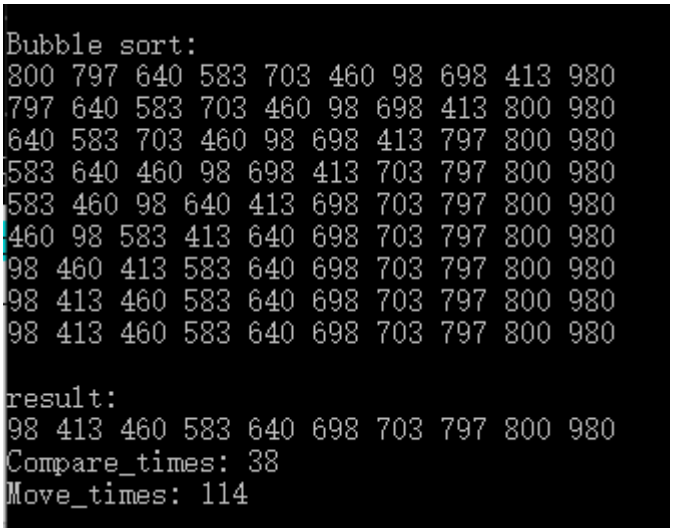
插入排序

```
Insert sort:
800 980 797 640 583 703 460 98 698 413
797 800 980 640 583 703 460 98 698 413
640 797 800 980 583 703 460 98 698 413
583 640 797 800 980 703 460 98 698 413
583 640 703 797 800 980 460 98 698 413
460 583 640 703 797 800 980 98 698 413
98 460 583 640 703 797 800 980 698 413
98 460 583 640 698 703 797 800 980 413
98 413 460 583 640 698 703 797 800 980

result:
98 413 460 583 640 698 703 797 800 980
Compare_times: 38
Move_times: 56
```

及时终止的选择排序

```
Selection sort:
413 800 797 640 583 703 460 98 698 980
413 698 797 640 583 703 460 98 800 980
413 698 98 640 583 703 460 797 800 980
413 698 98 640 583 460 703 797 800 980
413 460 98 640 583 698 703 797 800 980
413 460 98 583 640 698 703 797 800 980
413 460 98 583 640 698 703 797 800 980
413 98 460 583 640 698 703 797 800 980
98 413 460 583 640 698 703 797 800 980

result:
98 413 460 583 640 698 703 797 800 980
Compare_times: 45
Move_times: 27
```

及时终止的冒泡排序



4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

从第一组和第二组的数据来看，
名次排序的移动和比较次数是一致的，
插入排序的移动与比较次数处于最好与最坏的情况之间，
及时终止的均是最坏的情况，
及时终止的冒泡排序也是处于正常的范围内。

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）

```
#include <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;

class ArraySort
{
    public :
        ArraySort();//构造函数
        ~ArraySort();//析构函数
        void RankSort();//名次排序
        void SelectionSort();//及时终止的选择排序
        void InsertSort();//插入排序
        void BubbleSort();//及时终止的冒泡排序
        void Output();//输出排序完成的数组

    private:
        int n;
        int *l;  //用于存储随机数
        int *L;  //用于输出排序的数组
};
```

```
/*构造函数*/
ArraySort::ArraySort()
{
    cout << "Input the length of your Array" << endl;
    cin >> n;
    l = new int[n];
    L = new int[n];
    cout << "The random number: " << endl;
    srand ((int) time (0));//用时间做随机种子
    for (int i=0; i<n; i++)
    {
        l[i] = (rand()%1000); //生成 n 个 0~1000 的随机数
        cout << l[i] << " ";
    }
    cout << endl << endl;
}

/*析构函数*/
ArraySort::~ArraySort()
{
    delete []l; //删除存储随机数的数组
    delete []L; //删除排序的数组
}

/*输出排序完成的数组*/
void ArraySort::Output()
{
    for (int i=0; i<n; i++)
        cout << L[i] << " ";
    cout << endl;
}

/*名次排序*/
void ArraySort::RankSort()
{
    int compare_times = 0;//统计比较次数
    int move_times = 0;//统计移动次数
    cout << "Rank sort: " << endl;
    int *r = new int[n]; //动态建立一个名次数组
    for (int i=0; i<n; i++) //名次数组初始化
        r[i] = 0;
    for (int i=1; i<n; i++)//计算名次
        for (int j=0; j<i; j++)
        {
```

```
            if (l[j]<=l[i])//如果前面的数比后面的小，则后面的名次＋1
            {
                r[i]++;
                compare_times++;
            }
            else            //否则前面的名次＋1
            {
                r[j]++;
                compare_times++;
            }
        }
    for (int i=0; i<n; i++)
        L[i] = l[i];   //将获取的随机数组赋值给输出的数组
    int *u = new int[n]; //创建一个附加数组
    for (int i=0; i<n; i++)  //根据名次将被排序的数组中的元素移动到正确的位置
    {
        u[r[i]] = l[i];
        move_times++;
    }
    for (int i=0; i<n; i++)  //将排序好的数组赋值给用于输出的数组
    {
        L[i] = u[i];
        move_times++;
        Output();
    }
    cout << endl << "result:" << endl;
    Output();
    delete []r;//删除动态数组
    delete []u;//删除动态数组
    cout << "Compare_times: " << compare_times << endl;
    cout << "Move_times: " << move_times << endl;
    cout << endl;
}

/*插入排序*/
void ArraySort::InsertSort()
{
    int compare_times = 0;//统计比较次数
    int move_times = 0;//统计移动次数
    cout << "Insert sort: " << endl;
    for (int i=0; i<n; i++)
        L[i] = l[i];   //将获取的随机数组赋值给输出的数组
```

```
    for (int i=1; i<n; i++) //对输出的数组进行插入排序
    {
        int t = L[i];
        move_times++;
        int j;
        for (j = i-1; (j>=0) && (t<L[j]); j--)
        {
            L[j+1] = L[j];
            move_times++;
            compare_times++;
        }
        L[j+1] = t;
        move_times++;
        Output();
    }
    cout << endl << "result:" << endl;
    Output();
    cout << "Compare_times: " << compare_times << endl;
    cout << "Move_times: " << move_times << endl;
    cout << endl;
}

/*及时终止的选择排序*/
void ArraySort::SelectionSort()
{
    int compare_times = 0;//统计比较次数
    int move_times = 0;//统计移动次数
    cout << "Selection sort: " << endl;
    for (int i=0; i<n; i++)
        L[i] = l[i];   //将获取的随机数组赋值给输出的数组
    bool sorted = false;
    for (int length = n; !sorted && (length>1); length--)
    {
        int indexofmax = 0;
        sorted = true;
        for (int i=1; i<length; i++)  //获取最大值，若大小关系有序，
则布尔值 sorted 为真，退出循环
        {
            if (L[indexofmax] <= L[i])
            {
                indexofmax = i;
                compare_times++;
            }
            else
```

```
            {
                sorted = false;
                compare_times++;
            }
        }
        int temp = L[indexofmax];
        L[indexofmax] = L[length-1];
        L[length-1] = temp;
        //swap(L[indexofmax],L[length-1]);//交换最大值与数组第length-1个元素的数值
        move_times = move_times+3;
        Output();
    }
    cout << endl << "result:" << endl;
    Output();
    cout << "Compare_times: " << compare_times << endl;
    cout << "Move_times: " << move_times << endl;
    cout << endl;
}

/*及时终止的冒泡排序*/
void ArraySort::BubbleSort()
{
    int compare_times = 0;//统计比较次数
    int move_times = 0;//统计移动次数
    cout << "Bubble sort: " << endl;
    for (int i=0; i<n; i++)
        L[i] = l[i];   //将获取的随机数组赋值给输出的数组
    bool swapped = true;
    for (int i=n; i>1&&swapped; i--)
    {
        swapped = false;
        for (int j=0; j<i-1; j++)
        {
            if (L[j]>L[j+1])
            {
                int temp = L[j];
                L[j] = L[j+1];
                L[j+1] = temp;
                //swap(L[j],L[j+1]);
                swapped = true;    //当无序时进行了交换，swapped为真，继续循环
                compare_times++;
                move_times = move_times+3;
```

```
            }

        }
        Output();
    }
    cout << endl << "result:" << endl;
    Output();
    cout << "Compare_times: " << compare_times << endl;
    cout << "Move_times: " << move_times << endl;
    cout << endl;
}

int main()
{
    ArraySort a;
    a.RankSort();
    a.InsertSort();
    a.SelectionSort();
    a.BubbleSort();
    return 0;
}
```