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：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 实验题目：排序算法 | | | |
| 实验学时：2 | | 实验日期： 2018.9.29 | |
| 实验目的：  掌握各种简单排序算法 | | | |
| 软件环境：  Mingw32-w64 8.1.0  Sublime Text3 | | | |
| 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）。   1. 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）   数据结构：动态数组  算法：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  按名次排序：依次计算出每个元素在数组中的排名，最后按照排名归位  及时终止的选择排序：在选择排序的基础上，如果发现前一个元素都比下一个元素小，则认为数组已经有序，此时退出  及时终止的冒泡排序：在冒泡排序的基础上，如果发现当前轮冒牌没有元素发生交换，则认为有序，此时退出  插入排序：将当前元素插入已经有序的数组片段中   1. 测试结果（测试输入，测试输出，结果分析）   Input:  10  Output:  原始:  164 527 73 691 259 776 128 395 431 551  按名次排序:  73 527 73 691 259 776 128 395 431 551  比较: 45, 移动: 1  73 128 73 691 259 776 128 395 431 551  比较: 45, 移动: 2  73 128 164 691 259 776 128 395 431 551  比较: 45, 移动: 3  73 128 164 259 259 776 128 395 431 551  比较: 45, 移动: 4  73 128 164 259 395 776 128 395 431 551  比较: 45, 移动: 5  73 128 164 259 395 431 128 395 431 551  比较: 45, 移动: 6  73 128 164 259 395 431 527 395 431 551  比较: 45, 移动: 7  73 128 164 259 395 431 527 551 431 551  比较: 45, 移动: 8  73 128 164 259 395 431 527 551 691 551  比较: 45, 移动: 9  73 128 164 259 395 431 527 551 691 776  比较: 45, 移动: 10  73 128 164 259 395 431 527 551 691 776  比较: 45, 移动: 10  及时终止的选择排序:  164 527 73 691 259 551 128 395 431 776  比较: 3, 移动: 1  164 527 73 431 259 551 128 395 691 776  比较: 5, 移动: 2  164 527 73 431 259 395 128 551 691 776  比较: 7, 移动: 3  164 128 73 431 259 395 527 551 691 776  比较: 8, 移动: 4  164 128 73 395 259 431 527 551 691 776  比较: 9, 移动: 5  164 128 73 259 395 431 527 551 691 776  比较: 10, 移动: 6  164 128 73 259 395 431 527 551 691 776  比较: 11, 移动: 7  73 128 164 259 395 431 527 551 691 776  比较: 11, 移动: 8  73 128 164 259 395 431 527 551 691 776  比较: 12, 移动: 9  73 128 164 259 395 431 527 551 691 776  比较: 12, 移动: 9  及时终止的冒泡排序:  164 527 73 691 259 776 128 395 431 551  比较: 0, 移动: 0  164 73 527 691 259 776 128 395 431 551  比较: 1, 移动: 1  164 73 527 691 259 776 128 395 431 551  比较: 1, 移动: 1  164 73 527 259 691 776 128 395 431 551  比较: 2, 移动: 2  164 73 527 259 691 776 128 395 431 551  比较: 2, 移动: 2  164 73 527 259 691 128 776 395 431 551  比较: 3, 移动: 3  164 73 527 259 691 128 395 776 431 551  比较: 4, 移动: 4  164 73 527 259 691 128 395 431 776 551  比较: 5, 移动: 5  164 73 527 259 691 128 395 431 551 776  比较: 6, 移动: 6  73 164 527 259 691 128 395 431 551 776  比较: 7, 移动: 7  73 164 527 259 691 128 395 431 551 776  比较: 7, 移动: 7  73 164 259 527 691 128 395 431 551 776  比较: 8, 移动: 8  73 164 259 527 691 128 395 431 551 776  比较: 8, 移动: 8  73 164 259 527 128 691 395 431 551 776  比较: 9, 移动: 9  73 164 259 527 128 395 691 431 551 776  比较: 10, 移动: 10  73 164 259 527 128 395 431 691 551 776  比较: 11, 移动: 11  73 164 259 527 128 395 431 551 691 776  比较: 12, 移动: 12  73 164 259 527 128 395 431 551 691 776  比较: 12, 移动: 12  73 164 259 527 128 395 431 551 691 776  比较: 12, 移动: 12  73 164 259 527 128 395 431 551 691 776  比较: 12, 移动: 12  73 164 259 128 527 395 431 551 691 776  比较: 13, 移动: 13  73 164 259 128 395 527 431 551 691 776  比较: 14, 移动: 14  73 164 259 128 395 431 527 551 691 776  比较: 15, 移动: 15  73 164 259 128 395 431 527 551 691 776  比较: 15, 移动: 15  73 164 259 128 395 431 527 551 691 776  比较: 15, 移动: 15  73 164 259 128 395 431 527 551 691 776  比较: 15, 移动: 15  73 164 128 259 395 431 527 551 691 776  比较: 16, 移动: 16  73 164 128 259 395 431 527 551 691 776  比较: 16, 移动: 16  73 164 128 259 395 431 527 551 691 776  比较: 16, 移动: 16  73 164 128 259 395 431 527 551 691 776  比较: 16, 移动: 16  73 164 128 259 395 431 527 551 691 776  比较: 16, 移动: 16  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 17  插入排序:  164 527 73 691 259 776 128 395 431 551  比较: 0, 移动: 1  73 164 527 691 259 776 128 395 431 551  比较: 2, 移动: 2  73 164 527 691 259 776 128 395 431 551  比较: 2, 移动: 3  73 164 259 527 691 776 128 395 431 551  比较: 4, 移动: 4  73 164 259 527 691 776 128 395 431 551  比较: 4, 移动: 5  73 128 164 259 527 691 776 395 431 551  比较: 9, 移动: 6  73 128 164 259 395 527 691 776 431 551  比较: 12, 移动: 7  73 128 164 259 395 431 527 691 776 551  比较: 15, 移动: 8  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 9  73 128 164 259 395 431 527 551 691 776  比较: 17, 移动: 9   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   (1).经查看结果得知4种排序方法的结果均为升序，可以确定算法正确  (2).随机数的生成。  要取得[a,b)的随机整数，使用(rand() % (b-a))+ a;  要取得[a,b]的随机整数，使用(rand() % (b-a+1))+ a;  要取得(a,b]的随机整数，使用(rand() % (b-a))+ a + 1;   1. 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 <cstdlib>  #include <ctime>  using namespace std;  template<typename T>  class Sort  {  public:  Sort(const T\*, int);  Sort(const Sort<T>&);  ~Sort();  void rankSort();  void selectionSort();  void bubbleSort();  void insertSort();  void output();  void update(const T\*, int);  protected:  int n, cmp, mv;  T\* a;  };  template<typename T>  Sort<T>::Sort(const T\* arr, int len)  {  n = len;  a = new T[n];  copy(arr, arr + n, a);  cmp = mv = 0;  }  template<typename T>  Sort<T>::Sort(const Sort<T>& s)  {  n = s.n;  a = new T[n];  copy(s.a, s.a + n, a);  cmp = mv = 0;  }  template<typename T>  Sort<T>::~Sort() { delete[]a; }  template<typename T>  void Sort<T>::rankSort()  {  T\* rk = new T[n];  T\* tmp = new T[n];  for (int i = 0; i < n; ++i) rk[i] = 0;  for (int i = 0; i < n; ++i)  {  for (int j = 0; j < i; ++j)  {  if (a[i] >= a[j]) rk[i]++;  else rk[j]++;  cmp++;  }  }  for (int i = 0; i < n; ++i) tmp[rk[i]] = a[i];  for (int i = 0; i < n; ++i)  {  mv++;  a[i] = tmp[i];  output();  }  delete[]tmp;  delete[]rk;  }  template<typename T>  void Sort<T>::selectionSort()  {  bool sorted = false;  for (int i = n; i > 1 && !sorted; --i)  {  int maxIndex = 0;  sorted = true;  for (int j = 0; j < i; ++j)  {  if (a[maxIndex] < a[j])  {  cmp++;  maxIndex = j;  }  else sorted = false;  }  swap(a[i - 1], a[maxIndex]);  mv++;  output();  }  }  template<typename T>  void Sort<T>::bubbleSort()  {  bool sorted = false;  for (int i = n; i > 1 && !sorted; --i)  {  sorted = true;  for (int j = 0; j < i - 1; ++j)  {  if (a[j] > a[j + 1])  {  cmp++;  mv++;  swap(a[j], a[j + 1]);  sorted = false;  }  output();  }  }  }  template <typename T>  void Sort<T>::insertSort()  {  for (int i = 1; i < n; ++i)  {  T tmp = a[i];  int j;  for (j = i - 1; j >= 0 && tmp < a[j]; --j)  {  cmp++;  a[j + 1] = a[j];  }  mv++;  a[j + 1] = tmp;  output();  }  }  template<typename T>  void Sort<T>::output()  {  for (int i = 0; i < n; ++i) cout << a[i] << ' ';  cout << endl << "比较: " << cmp << ", 移动: " << mv << endl;  }  template<typename T>  void Sort<T>::update(const T\* arr, int len)  {  delete[]a;  n = len;  a = new T[n];  copy(arr, arr + n, a);  cmp = mv = 0;  }  int main()  {  srand(time(0));  int n;  cin >> n;  int \*a = new int[n];  for (int i = 0; i < n; ++i) a[i] = rand() % 1001;  cout << "原始:" << endl;  for (int i = 0; i < n; ++i) cout << a[i] << ' ';  cout << endl;  cout << endl << "按名次排序:" << endl;  Sort<int> s(a, n);  s.rankSort();  s.output();  cout << endl << "及时终止的选择排序:" << endl;  s.update(a, n);  s.selectionSort();  s.output();  cout << endl << "及时终止的冒泡排序:" << endl;  s.update(a, n);  s.bubbleSort();  s.output();  cout << endl << "插入排序: " << endl;  s.update(a, n);  s.insertSort();  s.output();  return 0;  } | | | |