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：201700140056 | 姓名：李港 | | 班级：18.2 |
| 实验题目：实验二 排序算法 | | | |
| 实验学时：2h | | 实验日期：2019.09.19 | |
| 实验目的：  掌握各种简单排序算法。 | | | |
| 软件开发工具：  Virtual Studio 2019 | | | |
| 1. **实验内容**    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. 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. **总体思路：**       1. 本次实验使用LgSorts类将所有功能函数进行打包。       2. 排序函数为公有，交换与取随机数等函数私有。       3. LgSorts类负责输出排序过程与结果，不负责数据的初始化，不负责内存的申请与释放。       4. LgSorts类提供了重排函数，可以将数据打乱，以便进行下一次排序测试。       5. 输入数据后，每次排序之后都打乱顺序，测试另一个排序函数。    2. **名次排序**       1. 关键点在于：元素的名次 **=** 比它小的元素的数量 **+** 该元素左侧与它相等的元素的数量，这就要求每个元素都与其他元素对比一次；因此有两个循环：       2. 外层循环负责指定当前对比的元素并限定比较范围       3. 内层元素执行循环比较操作       4. 只要元素两两对比过就好，但为了操作方便，将数组分为已比较区和未比较区，    3. **及时终止的选择排序**       1. 数组分为排序区与未排序区，每次在未排序区选择一个最大（小）的数，与两区交接的未排序元素交换位置，如此反复。最终完成排序。       2. **及时终止：**在从未排序区选择最值的过程中，可以设计策略：          1. 若要选择最大值，则要从将来排好序时最小值所在一侧开始比较。          2. 若要选择最小值，则与上述规则相反。          3. 若每次比较均需将新的被比较者设为新的最值，则未排序区已经排好序，可以终止选择排序过程。    4. **插入排序**       1. 数组分为排序区与未排序区，每次选择两区边界的未排序元素，经过逐步比较插入到排序区适合它的位置那里。与选择排序存在相似性。    5. **及时终止的冒泡排序**       1. 不分排序区与未排序区，每次循环都对下标相邻的两值进行比较，将每次比较的两者按照固定大小顺序进行交换，像冒泡一样将最值交换到相应一侧，最终实现所有元素交换到合适的位置上。       2. **及时终止：**若一次循环中不再有元素需要交换，则说明已经排序完成，应停止排序。   **数据结构：**本实验对数据结构要求不高，采用C原生数组  **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   1. **测试结果（测试输入，测试输出）**   **输入：**  11  **输出：**  39 720 239 438 856 798 366 286 451 613 854  排序源数组初始化完成，  名次排序测试----------------  排名结果：0 7 1 4 10 8 3 2 5 6 9  名次排序结束----------------  插入排序测试----------------  第 1 轮循环：438 613 720 366 856 286 451 39 239 854 798  第 2 轮循环：438 613 720 366 856 286 451 39 239 854 798  第 3 轮循环：366 438 613 720 856 286 451 39 239 854 798  第 4 轮循环：366 438 613 720 856 286 451 39 239 854 798  第 5 轮循环：286 366 438 613 720 856 451 39 239 854 798  第 6 轮循环：286 366 438 451 613 720 856 39 239 854 798  第 7 轮循环：39 286 366 438 451 613 720 856 239 854 798  第 8 轮循环：39 239 286 366 438 451 613 720 856 854 798  第 9 轮循环：39 239 286 366 438 451 613 720 854 856 798  第 10 轮循环：39 239 286 366 438 451 613 720 798 854 856  扫描次数：10 比较次数：35 移动次数：28  插入排序结束----------------  及时终止选择排序测试---------  第 1 轮循环：720 438 856 366 286 39 239 854 451 613 798  第 2 轮循环：856 438 720 366 286 39 239 854 451 613 798  第 3 轮循环：856 854 720 366 286 39 239 438 451 613 798  第 4 轮循环：856 854 798 366 286 39 239 438 451 613 720  第 5 轮循环：856 854 798 720 286 39 239 438 451 613 366  第 6 轮循环：856 854 798 720 613 39 239 438 451 286 366  第 7 轮循环：856 854 798 720 613 451 239 438 39 286 366  第 8 轮循环：856 854 798 720 613 451 438 239 39 286 366  第 9 轮循环：856 854 798 720 613 451 438 366 39 286 239  第 10 轮循环：856 854 798 720 613 451 438 366 286 39 239  第 11 轮循环：856 854 798 720 613 451 438 366 286 239 39  排序完成，终止！  扫描次数：11 比较次数：66 移动次数：10  选择排序结束----------------  及时终止冒泡排序测试---------  第 1 轮循环：39 366 613 720 854 239 438 798 451 286 856  第 2 轮循环：39 366 613 720 239 438 798 451 286 854 856  第 3 轮循环：39 366 613 239 438 720 451 286 798 854 856  第 4 轮循环：39 366 239 438 613 451 286 720 798 854 856  第 5 轮循环：39 239 366 438 451 286 613 720 798 854 856  第 6 轮循环：39 239 366 438 286 451 613 720 798 854 856  第 7 轮循环：39 239 366 286 438 451 613 720 798 854 856  第 8 轮循环：39 239 286 366 438 451 613 720 798 854 856  第 9 轮循环：39 239 286 366 438 451 613 720 798 854 856  无可交换数据，终止！  扫描次数：9 比较次数：54 移动次数：28  冒泡排序结束----------------  **随机生成内容符合范围，排序结果正确。**   1. **分析与探讨（结果分析，若存在问题，探讨解决问题的途径）**   经过测试，结果正常，本程序在执行逻辑上暂不存在问题，在代码方面存在的问题如下：   1. 记录循环、比较、移动次数等数据均需要变量来记录，这降低了代码的可读性。 2. 本程序严格遵循if-else配对原则，可能造成代码冗余。   结合实际情况，对上述问题不进行更改。   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）**   程序2-1 排序类及其测试 sort.cpp  #include<cstdlib>  #include<ctime>  #include<iostream>  using std::cout;  using std::endl;  #define LGRAND(min,max) ((rand()%(max-min+(int)1))+(int)min )  /\*\*  \* @brief 本类包含了四种排序函数  \*/  template<typename T>  class LgSorts {  private:  T\* \_arr;  int \_len;  void \_LgSwap(T\* A, T\* B) {  T temp = \*A;  \*A = \*B;  \*B = temp;  }  /\*\*  \* @brief 取随机数 返回min-max中间的数值  \*/  int \_LgRand(int min, int max) {  return ((rand() % (max - min + 1)) + min);  }  /\*\*  \* @brief 将数组中所有值展示一次  \* int times 循环轮次  \*/  void \_Show(int times) {  cout << "第 " << times << " 轮循环：";  for (int times = 0; times < \_len; times++) {  cout << \_arr[times] << " ";  }  cout << endl;  }  public:  LgSorts(int in, T\* iarr) :\_len(in), \_arr(iarr) {}  /\*\*  \* @brief 每次排序后进行打乱，以便再次排序  \*/  void ReArrange() {  for (int i = 0; i < \_len; i++) {  \_LgSwap(\_arr + i, \_arr + LGRAND(0, \_len - 1));  }  return;  }  /\*\*  \* @brief 冒泡排序  \*/  void BubbleSort() {  int cmp\_times = 0; //记录比较次数  int move\_times = 0; //记录交换次数  int loop = 0;  /\*冒泡排序核心代码，每次扫描都将所有元素扫描一遍，根据其大小进行互换\*/  for (int j = 0; j < \_len - 1; j++) {  loop++;  /\*单次扫描\*/  bool no\_swap = 1; //记录该次扫描是否有交换行为  for (int i = 0; i < \_len - j - 1; i++) {  cmp\_times++;  if (\_arr[i] > \_arr[i + 1]) {  no\_swap = 0;  move\_times++;  \_LgSwap(\_arr + i, \_arr + i + 1);  } else { //严格if-else配对  continue;  }  }  \_Show(j + 1);  /\*判断该次扫描是否有过交换行为，若无，则说明以排好序，提前终止\*/  if (no\_swap) {  cout << "无可交换数据，终止！" << endl;  break;  }  }  cout << "扫描次数：" << loop << " 比较次数：" << cmp\_times << " 移动次数：" << move\_times << endl;  }  /\*\*  \* @brief 选择排序  \*/  void SelectSort() {  int cmp\_times = 0;  int move\_times = 0;  int loop = 0;  /\*选择排序核心代码，从大到小，左测为排序区，右测未排序区\*/  for (int i = 0; i < \_len; i++) {  loop++;  int maxindex = \_len - 1;//该下标有讲究，从目标最低处开始，如果他一直变化，说明未排序区也排好序了  bool sorted = true;  for (int j = \_len - 1; j > i - 1; j--) {  cmp\_times++;  if (\_arr[j] >= \_arr[maxindex]) {  maxindex = j;  } else {//如果maxindex有一次未变化，说明未排序区还没有完全排序  sorted = false;  }  }  \_Show(i + 1);  if (sorted) {  cout << "排序完成，终止！" << endl;  break;  }  \_LgSwap(\_arr + maxindex, \_arr + i);  move\_times++;  }  cout << "扫描次数：" << loop << " 比较次数：" << cmp\_times << " 移动次数：" << move\_times << endl;  }  /\*\*  \* @brief 插入排序  \*/  void InsertSort() {  int cmp\_times = 0;  int move\_times = 0;  int loop = 0;  for (int i = 1; i < \_len; i++) { // i到\_len-1是未排序区域，0到i-1是已排序区域  T temp = \_arr[i];  for (int j = i; 0 < j; j--) {  cmp\_times++;  //把\_arr【i】插入到a【0：1-1】  /\*以排序区域右侧为最大值，若它小于temp，则temp就是最新的最大值\*/  if (\_arr[j - 1] <= temp) {  break;  } else {  move\_times++;  \_arr[j] = \_arr[j - 1];  \_arr[j - 1] = temp;  }  }  \_Show(i);  loop++;  }  cout << "扫描次数：" << loop << " 比较次数：" << cmp\_times << " 移动次数：" << move\_times << endl;  }  /\*\*  \* @brief 排名次  \*/  void Rank() {  int\* r = new int[\_len];  memset(r, 0, sizeof(int) \* \_len);  /\*Rank核心代码\*/  //目的是让所有元素都与其他元素比较一次  //左侧为已主动比较的，右侧为未主动比较的  for (int i = 1; i < \_len; i++) {  for (int j = 0; j < i; j++) {  if (\_arr[j] <= \_arr[i]) { //名次等于比他小的元素的数量加上左侧同大小元素的数量  r[i]++;  } else {  r[j]++;  }  }  }  cout << "排名结果：";  for (int i = 0; i < \_len; i++) {  cout << r[i] << " ";  }  cout << endl;  }  };  int main() {  srand(0);  /\*初始化待排序数组\*/  int num = 10;  int\* iarr = new int[num];  for (int i = 0; i < num; i++) {  iarr[i] = (int)LGRAND(1, 1000);  cout << iarr[i] << " ";  }  cout << endl;  cout << "排序源数组初始化完成，";  cout << endl;  /\*进行排序\*/  LgSorts<int> s(num, iarr);  cout << "名次排序测试----------------" << endl;  s.Rank();  cout << "名次排序结束----------------" << endl << endl;  cout << "插入排序测试----------------" << endl;  s.ReArrange();  s.InsertSort();  cout << "插入排序结束----------------" << endl << endl;  cout << "及时终止选择排序测试---------" << endl;  s.ReArrange();  s.SelectSort();  cout << "选择排序结束----------------" << endl << endl;  cout << "及时终止冒泡排序测试---------" << endl;  s.ReArrange();  s.BubbleSort();  cout << "冒泡排序结束----------------" << endl << endl;  /\*释放资源\*/  delete[] iarr;  return 0;  } | | | |