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：201805130155 | 姓名： 赵雨晗 | | 班级： 18计科3班 |
| 实验题目：排序算法 | | | |
| 实验学时：2 | | 实验日期： 2019.9.18 | |
| 实验目的：  1、 掌握各种简单排序算法。 | | | |
| 软件开发工具：  Windows : Vs Code + MingW | | | |
| 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、 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）。     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）   1、选择排序第一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后再从剩余的未排序元素中寻找到最小（大）元素，然后放到已排序的序列的末尾。以此类推，直到全部待排序的数据元素的个数为零。  2、冒泡排序重复地走访过要排序的元素列，依次比较两个相邻的元素，如果他们的顺序（如从大到小、首字母从A到Z）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。  3、插入排序如果有一个已经有序的数据序列，要求在这个已经排好的数据序列中插入一个数，但要求插入后此数据序列仍然有序，这个时候就要用到一种新的排序方法——插入排序法,插入排序的基本操作就是将一个数据插入到已经排好序的有序数据中，从而得到一个新的、个数加一的有序数据，算法适用于少量数据的排序，   1. 测试结果（测试输入，测试输出）      1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   一开始一直WA，后来发现在使用了ios::sync\_with\_stdio的同时使用scanf造成了错误  并无问题   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<cstdio>  #include<iostream>  #include<cstring>  #include<string>  #include <algorithm>  using namespace std;  template <class T>  class ArrayList{  protected:  T\* \_Ele;  int Arr\_len;  int list\_size;  public:  ArrayList(int init\_L = 10);  ArrayList(const ArrayList<T>& );  void push\_back(const T &x);  void ch\_sort();  void rank\_sort();  void Bubble\_sort();  void insert\_sort();  void output();  };  template <class T>  ArrayList<T> :: ArrayList(int init\_L){  list\_size = init\_L;  \_Ele = new T[init\_L];  Arr\_len = 0;  };  template <class T>  ArrayList<T> ::ArrayList(const ArrayList<T> &t) {  list\_size = t.list\_size;  \_Ele = new T[list\_size];  Arr\_len = t.Arr\_len;  copy(t.\_Ele, t.\_Ele + t.Arr\_len, \_Ele);  }  template <class T>  void ArrayList<T> ::push\_back(const T &x) {  if (list\_size == Arr\_len) {  ArrayList A(\*this);  list\_size = list\_size \* 2;  delete[] \_Ele;  \_Ele = new T[list\_size];  }  \_Ele[Arr\_len++] = x;  }  template <class T>  void ArrayList<T> ::ch\_sort() {  bool sorted = false;  for (int size = Arr\_len; !sorted && (size > 1); size--){  int ind = 0;  sorted = true;  for (int i = 1; i < size; ++i)  if (\_Ele[ind] <= \_Ele[i]) ind = i;  else sorted = false;  swap(\_Ele[ind], \_Ele[size - 1]);  }  }  template <class T>  void ArrayList<T> ::Bubble\_sort() {  bool swapd = true;  for (int i = Arr\_len; i > 1 && swapd; i--){  swapd = false;  for (int j = 0; j < i - 1; j++)  if (\_Ele[j] > \_Ele[j + 1]){  swap(\_Ele[j], \_Ele[j+1]);  swapd = 1;  }  }  }  template <class T>  void ArrayList<T> ::insert\_sort() {  for (int i = 1; i < Arr\_len; ++i){  T t = \_Ele[i];  int j;  for (j = i - 1; j >= 0 && t < \_Ele[j]; --j)  \_Ele[j + 1] = \_Ele[j];  \_Ele[j + 1] = t;  }  }  template <class T>  void ArrayList<T> ::rank\_sort() {  int \*r;  r = new int[list\_size];  memset(r, 0, list\_size \* 4);  for (int i = 0; i < Arr\_len; ++i){  for (int j = i + 1; j < Arr\_len; ++j){  if (\_Ele[i] > \_Ele[j])r[i]++;  else r[j]++;  }  }  // for (int i = 0; i < Arr\_len; ++i)cout<<r[i]<<' '<<endl;  for (int i = 0; i < Arr\_len; ++i){  while(r[i] != i){  swap(\_Ele[i], \_Ele[r[i]]);  swap(r[i], r[r[i]]);  }  }  }  template <class T>  void ArrayList<T> ::output() {  for (int i = 0; i < Arr\_len - 1; ++i)cout<<\_Ele[i]<<' ';  cout<<\_Ele[Arr\_len - 1];  }  int main()  {  ArrayList<int> \_L(1010);  int n, a;  // int t[1010];  // ios::sync\_with\_stdio(false);  cin>>n;  for (int i = 1; i <= n; ++i){  scanf("%d", &a);  // t[i] = a;  \_L.push\_back(a);  }  // sort(t + 1, t + n + 1);  // for (int i = 1; i < n; ++i) printf("%d ", t[i]);  // cout<<t[n];  \_L.Bubble\_sort();  \_L.output();  system("pause");  return 0;    } | | | |