《数据结构与算法》实验报告——排序算法

吉林大学数学学院2019级计算X班 姓名 学号

日期

一、实验题目

分别编写选择排序（SelectSort）、插入排序（InsertSort）、冒泡排序（BubbleSort）、快速排序（QuickSort）、合并排序（MergeSort）的程序，并验证结果。

二、算法简述（）

记待排序顺序表长度为n。

选择排序：顺序遍历前n-1个元素。将每个元素与其后比它小的最小元素交换（如果有的话）。

插入排序：顺序遍历第2~n个元素。此时每个元素前应是已经排好序的子序列，将该元素插入前面序列合适的位置。

冒泡排序：逆序遍历前n-1个元素。将每个元素“赶”到后面，直到遇到比它还大的元素为止。

快速排序：找到一个基准点（如中间点）。用“将把比基准点小的元素赶到左边，比基准点大的元素赶到右边”的方式，找到基准点排好序以后的位置。然后以此为边界，对基准点左右两个子序列递归应用该方法。

合并排序：递归分割子序列，将两个相邻的子序列按排好序的要求合并得到一个稍大的序列，递归应用该方法。

三、程序代码

|  |
| --- |
| InternalSorting.cpp |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  #include <windows.h>  #define MAXSIZE 20  typedef int keyType;  typedef char infoType; // 假设其他数据类型为char  typedef struct{  keyType key;  infoType otherInfo;  }redType;  typedef struct{  redType r[MAXSIZE + 1]; // r[0]作为临时存储  int length;  }SqList;  void printKey\_sq(SqList L); // 打印顺序表关键字项  void init\_sq(SqList &L); // 初始化顺序表  void push\_sq(SqList &L, keyType k); // 在顺序表末尾加一项，假设otherInfo为' '  void swap\_sq(SqList &L, int i, int j); // 交换两项  void SelectSort(SqList &L); // 选择排序  void InsertSort(SqList &L); // 插入排序  void BubbleSort(SqList &L); // 冒泡排序  void QuickSort(SqList &L); // 快速排序  void QuickSortHelper(SqList &L, int left, int right); // 快速排序递归  int partion(SqList &L, int left, int right); // 快速排序子序列分割  void MergeSort(SqList &L); // 合并排序  void MergeSortHelper(SqList &L, SqList &copyBuffer, int left, int right); // 合并排序递归  void Merge(SqList &L, SqList &copyBuffer, int left, int mid, int right); // 合并两部分  void printResult(void (\*sortFunction)(SqList &L), char\* funcName); // 打印实验结果  int main(){  int i;  void(\*funcName[6])(SqList &L) = {0, SelectSort, InsertSort, BubbleSort, QuickSort, MergeSort};  char funcMean[6][5] = {"", "选择", "插入", "冒泡", "快速", "合并"};  for (i = 1; i <= 5; i ++) printResult(funcName[i], funcMean[i]);  return 0;  }  void printKey\_sq(SqList L){  int i;  for (i = 1; i <= L.length; i ++) {  printf("%d", L.r[i]);  if (i != L.length) printf(" ");  else printf("\n");  }  }  void init\_sq(SqList &L){  L.length = 0;  }  void push\_sq(SqList &L, keyType k){  L.length += 1;  L.r[L.length].key = k;  L.r[L.length].otherInfo = ' ';  }  void swap\_sq(SqList &L, int i, int j){  L.r[0] = L.r[i];  L.r[i] = L.r[j];  L.r[j] = L.r[0];  }  void SelectSort(SqList &L){  int i, j, minIndex;  for (i = 1; i <= L.length - 1; i ++){  minIndex = i;  for (j = i + 1; j <= L.length; j ++){  if (L.r[j].key < L.r[minIndex].key) minIndex = j;  }  if (minIndex != i){  swap\_sq(L, i, minIndex);  }  }  }  void InsertSort(SqList &L){  int i, j, insertIndex;  for (i = 2; i <= L.length; i++){  if (L.r[i].key < L.r[i - 1].key){ // 需要插入已排好序的列  insertIndex = i;  for (j = i - 1; j >= 1; j --){  if(L.r[i].key < L.r[j].key) insertIndex = j;  else break;  }  L.r[0].key = L.r[i].key;  for (j = i - 1; j >= insertIndex; j --) L.r[j + 1].key = L.r[j].key;  L.r[insertIndex].key = L.r[0].key;  }  }  }  void BubbleSort(SqList &L){  int i, j;  for (i = L.length - 1; i >= 1; i --){  for(j = i + 1; j <= L.length; j ++){  if(L.r[i].key > L.r[j].key){  swap\_sq(L, i, j);  i = j;  }  else break;  }  }  }  void QuickSort(SqList &L){  QuickSortHelper(L, 1, L.length);  }  void QuickSortHelper(SqList &L, int left, int right){  if (left < right){  int pivotLocation = partion(L ,left, right);  QuickSortHelper(L, left, pivotLocation - 1);  QuickSortHelper(L, pivotLocation + 1, right);  }  }  int partion(SqList &L, int left, int right){  keyType pivot;  int mid = (left + right) / 2, boundary = left, i = left;  pivot = L.r[mid].key; // 基准点的值  swap\_sq(L, mid, right);  while (i < right){  for(i = boundary; i <= right; i ++){  if (L.r[i].key < pivot){  if (i != boundary) swap\_sq(L, boundary, i);  boundary ++;  break;  }  }  }  if(boundary != right) swap\_sq(L, boundary, right);  return boundary;  }  void MergeSort(SqList &L){  SqList copyBuffer = L; // 用于复制用的工具顺序表  MergeSortHelper(L, copyBuffer, 1, L.length);  }  void MergeSortHelper(SqList &L, SqList &copyBuffer, int left, int right){  if (left < right){  int mid = (left + right) / 2;  MergeSortHelper(L, copyBuffer, left, mid);  MergeSortHelper(L, copyBuffer, mid + 1, right);  Merge(L, copyBuffer, left, mid, right);  }  }  void Merge(SqList &L, SqList &copyBuffer, int left, int mid, int right){  int i1 = left, i2 = mid + 1, i;  for (i = left; i <= right; i ++){  if (i1 > mid){  copyBuffer.r[i] = L.r[i2];  i2 ++;  }  else if (i2 > right){  copyBuffer.r[i] = L.r[i1];  i1 ++;  }  else if (L.r[i1].key > L.r[i2].key){  copyBuffer.r[i] = L.r[i2];  i2 ++;  }  else {  copyBuffer.r[i] = L.r[i1];  i1 ++;  }  }  for (i = left; i <= right; i ++) L.r[i] = copyBuffer.r[i];  }  void printResult(void (\*sortFunction)(SqList &L), char\* funcMean){  SqList L;  int i;  init\_sq(L);  Sleep(1000); // 为了保证每次循环随机出来的都不一样，停顿1秒  srand(time(NULL) \* (time(NULL) + 1)); // 以time的函数作为seed进行随机  for (i = 1; i <= MAXSIZE; i ++) push\_sq(L, (rand()%90 + 10)); // 向顺序表中随机输入20个两位数  printf("%s排序：\n排序前：", funcMean);  printKey\_sq(L);  sortFunction(L);  printf("排序后：");  printKey\_sq(L);  if (sortFunction != MergeSort) printf("\n");  } |

四、实验结果

|  |
| --- |
| 第一次 |
| 选择排序：  排序前：35 35 37 49 63 30 81 70 54 46 15 70 71 12 65 20 27 86 44 22  排序后：12 15 20 22 27 30 35 35 37 44 46 49 54 63 65 70 70 71 81 86  插入排序：  排序前：17 76 94 34 57 25 77 87 83 14 53 75 22 47 97 18 83 65 15 55  排序后：14 15 17 18 22 25 34 47 53 55 57 65 75 76 77 83 83 87 94 97  冒泡排序：  排序前：13 96 42 79 63 51 88 99 97 17 66 79 61 47 36 34 30 76 40 60  排序后：13 17 30 34 36 40 42 47 51 60 61 63 66 76 79 79 88 96 97 99  快速排序：  排序前：98 95 79 91 79 93 24 25 96 53 53 89 82 19 80 69 46 29 44 27  排序后：19 24 25 27 29 44 46 53 53 69 79 79 80 82 89 91 93 95 96 98  合并排序：  排序前：18 81 98 65 25 60 64 28 80 40 22 88 85 38 33 31 50 15 11 56  排序后：11 15 18 22 25 28 31 33 38 40 50 56 60 64 65 80 81 85 88 98  --------------------------------  Process exited after 5.418 seconds with return value 0  请按任意键继续. . . |
| 第二次 |
| 选择排序：  排序前：55 53 26 64 69 16 18 80 16 84 60 66 12 88 93 60 15 22 76 44  排序后：12 15 16 16 18 22 26 44 53 55 60 60 64 66 69 76 80 84 88 93  插入排序：  排序前：45 87 11 70 90 69 33 52 10 61 79 43 47 43 81 52 81 22 53 49  排序后：10 11 22 33 43 43 45 47 49 52 52 53 61 69 70 79 81 81 87 90  冒泡排序：  排序前：50 99 85 38 41 55 72 12 72 79 65 26 64 52 84 63 45 53 91 18  排序后：12 18 26 38 41 45 50 52 53 55 63 64 65 72 72 79 84 85 91 99  快速排序：  排序前：52 98 51 56 93 57 27 66 28 41 25 89 63 26 84 92 79 27 10 48  排序后：10 25 26 27 27 28 41 48 51 52 56 57 63 66 79 84 89 92 93 98  合并排序：  排序前：70 76 97 43 66 82 79 33 60 45 49 61 60 55 84 50 93 34 73 49  排序后：33 34 43 45 49 49 50 55 60 60 61 66 70 73 76 79 82 84 93 97  --------------------------------  Process exited after 5.501 seconds with return value 0  请按任意键继续. . . |
| 第三次 |
| 选择排序：  排序前：25 32 98 50 35 86 61 27 92 28 67 75 96 11 19 85 94 85 92 76  排序后：11 19 25 27 28 32 35 50 61 67 75 76 85 85 86 92 92 94 96 98  插入排序：  排序前：83 84 25 57 17 91 28 17 97 45 24 98 12 45 97 37 42 94 26 62  排序后：12 17 17 24 25 26 28 37 42 45 45 57 62 83 84 91 94 97 97 98  冒泡排序：  排序前：50 41 33 25 18 36 99 93 88 14 54 29 81 51 83 91 60 38 13 11  排序后：11 13 14 18 25 29 33 36 38 41 50 51 54 60 81 83 88 91 93 99  快速排序：  排序前：31 59 31 52 38 80 87 74 64 99 51 48 58 22 75 80 57 15 62 22  排序后：15 22 22 31 31 38 48 51 52 57 58 59 62 64 74 75 80 80 87 99  合并排序：  排序前：10 64 19 39 69 65 98 50 25 46 29 66 17 48 66 87 35 23 82 85  排序后：10 17 19 23 25 29 35 39 46 48 50 64 65 66 66 69 82 85 87 98  --------------------------------  Process exited after 5.441 seconds with return value 0  请按任意键继续. . . |