《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 马昆 | 班级 | 22级软件工程专升本1班 | 学号 | 2206831544 |
| 实验名称 | **实验八 排序技术** | | | | |
| 实验时间 | 2023年05月25号 | | 成绩 | |  |
| 1. **实验目的：**   1.掌握插入排序、交换排序算法的基本思想；  2.掌握插入排序、交换排序算法的实现方法；  3.掌握插入排序、交换排序算法的时间性能。  **二、实验工具：**  软件平台：Windows7或以上版本，Visual Studio 2019  **三、实验原理：**  1、算法描述：  （1）插入排序函数：我这里使用了两个函数来描述此算法，分别为升序排序与降序排序，这里只描述降序排序。函数接收一个数组与一个整型长度，无返回值。定义4个整型，两个整型用于两层for循环的循环变量，一个整型表示某次循环的最大值，一个整型表示某次循环找到的最大值的下标。定义一个外层for循环，从0开始，循环数组的长度-1次。循环体内假设本次循环的最大数为第一个数，并且将最大值下标设为本次循环的第一个数的下标。在外层for循环内定义内层for循环，循环变量从外层循环的变量值开始，到数组的长度-1结束。内层for循环内判断外层循环的当前数后的数是否大于它，如果大于就将最大值变量赋值为此数，并将最大值下标更新为此数的下标。内层for循环结束，在外层for循环的最后，将未排序数的最大值放到排序数的最末尾。  （2）希尔排序函数：函数接收一个数组形参与一个数组长度形参，无返回值。定义一个整型用于装分组间隔，初始为数组长度的一半。定义while循环，当分组间隔大于0时才能进入循环。循环体内定义for循环，对每个分组进行插入排序。最后在for循环外即while循环的最末尾缩小分组间隔，缩小为原来的一半。  （3）起泡排序函数：我这里使用了两个函数来描述此算法，分别为升序排序与降序排序，这里只描述升序排序。函数接收一个数组形参，一个数组长度形参，无返回值。定义三个整型，两个用于for循环的循环变量，一个用于交换两变量值的中间变量。定义外层for循环，从0开始到数组长度-1结束，外层循环变量每次循环自增1。外层循环内定义内层循环，循环变量从0开始，到数组长度-1并减外层循环的循环变量的差次，循环变量每次循环自增1。内层循环内判断数组中索引为内层循环当前循环变量值的数是否比下一个数的值大，如果大就使用临时变量替换数组中这两个数的值。  （4）函数接收一个整数数组和数组长度作为参数，没有返回值。在函数内部，定义三个整型变量，其中两个用于控制循环，一个用于交换两个变量的值。首先，选择数组的最后一个元素作为基准元素。然后，使用一个循环将数组中的元素进行划分。该循环从数组的第一个元素遍历到倒数第二个元素。在循环内部，检查当前元素是否小于或等于基准元素。如果是，将较小元素的索引递增，并交换当前元素与较小元素的位置。这样，每次交换操作都将较小的元素放在了基准元素的左边。完成一次循环后，将基准元素放置在正确的位置。具体操作是将基准元素与较小元素的下一个位置交换。这样，基准元素就被放置到了正确的位置上，并且基准元素左侧的元素都小于它，右侧的元素都大于它。接下来，通过递归调用快速排序函数，对基准元素左右两侧的子数组进行排序。递归调用的参数分别为数组的起始位置到基准元素的前一个位置，和基准元素的后一个位置到数组的结束位置。这样，通过递归调用，可以对整个数组进行排序。通过多轮的划分和排序操作，最终实现了整个数组的排序。快速排序的核心在于不断地选取基准元素，并将小于基准的元素放在左边，大于基准的元素放在右边。通过递归调用，可以对每个子数组进行相同的划分和排序操作，最终实现整个数组的有序化。  **四、实验步骤和内容：**  1.随机产生500个1～2000之间的整数并输出，完成以下操作：  （1）对给定整数进行直接插入排序，显示排序所用时间；  （2）对给定整数进行希尔排序，显示排序所用时间；  （3）对给定整数进行起泡排序，显示排序所用时间；  （4）对给定整数进行快速排序，显示排序所用时间。  部分参考代码如下：  #include<ctime> //使用函数库ctime  ……  int main()  {  clock\_t start,finish; //定义查找开始时间和结束时间变量  ……  start=clock(); //记录查找算法的开始时间  查找算法  finish=clock(); //记录查找算法的结束时间  ……  }   |  | | --- | | **Sort.h**  #pragma once  // 快速排序-升序  void quickSort(int arr[], int length);  // 希尔排序-升序  void shellAsc(int nums[], int len);  // 希尔排序-降序  void shellDesc(int nums[], int len);  // 插入排序-降序  void insertDesc(int nums[], int len);  // 插入排序-升序  void insertAsc(int nums[], int len);  // 冒泡排序-升序  void bubbleAsc(int nums[], int len);  // 冒泡排序-降序  void bubbleDesc(int nums[], int len); | | **Sort.cpp**  #include "Sort.h"  // 快速排序递归函数  void quickSort(int arr[], int low, int high) {  if (low < high) {  // 以数组最后一个元素作为基准，将小于基准的元素放在左边，大于基准的元素放在右边  int pivot = arr[high]; // 基准元素  int i = low - 1; // 较小元素的索引  int temp;  for (int j = low; j < high; j++) {  if (arr[j] <= pivot) {  i++;  temp = arr[i];  arr[i] = arr[j];  arr[j] = temp;  }  }  temp = arr[i + 1];  arr[i + 1] = arr[high];  arr[high] = temp;  pivot = i + 1;  // 递归对基准左右两侧的子数组进行排序  quickSort(arr, low, pivot - 1);  quickSort(arr, pivot + 1, high);  }  }  // 快速排序函数，接受一个整数数组和数组长度作为参数  void quickSort(int arr[], int length) {  quickSort(arr, 0, length - 1);  }  // 希尔排序-升序  void shellAsc(int nums[], int len) {  // 计算增量序列的初始值  int gap = len / 2;  // 不断缩小增量，直到增量为1  while (gap > 0) {  // 对每个增量进行插入排序  for (int i = gap; i < len; i++) {  int temp = nums[i];  int j = i;  // 在当前增量下，将元素插入到已排序序列的正确位置  while (j >= gap && nums[j - gap] > temp) {  nums[j] = nums[j - gap];  j -= gap;  }  nums[j] = temp;  }  // 缩小增量  gap /= 2;  }  }  // 希尔排序-降序  void shellDesc(int nums[], int len) {  // 计算增量序列的初始值  int gap = len / 2;  // 不断缩小增量，直到增量为1  while (gap > 0) {  // 对每个增量进行插入排序  for (int i = gap; i < len; i++) {  int temp = nums[i];  int j = i;  // 在当前增量下，将元素插入到已排序序列的正确位置  while (j >= gap && nums[j - gap] < temp) {  nums[j] = nums[j - gap];  j -= gap;  }  nums[j] = temp;  }  // 缩小增量  gap /= 2;  }  }  // 插入排序-降序  void insertDesc(int nums[], int len) {  int i, j, max, maxIndex;  // 从未排序的数字中找到最大数，放到已排序数字的末尾  for (i = 0; i < len - 1; i++) {  // 将最大值假设为未排序数中的第一个数  max = nums[maxIndex = i];  // 遍历未排序的数字  for (j = i; j < len - 1; j++) {  if (max < nums[j + 1]) {  max = nums[maxIndex = j + 1];  }  }  // 将未排序数的最大值放到排序数的最末尾  nums[maxIndex] = nums[i];  nums[i] = max;  }  }  // 插入排序-升序  void insertAsc(int nums[], int len) {  int i, j, min, minIndex;  // 从未排序的数字中找到最小数，放到已排序数字的末尾  for (i = 0; i < len - 1; i++) {  // 将最小值假设为未排序数中的第一个数  min = nums[minIndex = i];  // 遍历未排序的数字  for (j = i; j < len - 1; j++) {  if (min > nums[j + 1]) {  min = nums[minIndex = j + 1];  }  }  // 将未排序数的最小值放到排序数的最末尾  nums[minIndex] = nums[i];  nums[i] = min;  }  }  // 冒泡排序-降序  void bubbleDesc(int nums[], int len) {  int i, j, temp;  for (i = 0; i < len - 1; i++) {  for (j = 0; j < len - i - 1; j++) {  // 如果前面的数小于后面的数，就交换  if (nums[j] < nums[j + 1]) {  temp = nums[j + 1];  nums[j + 1] = nums[j];  nums[j] = temp;  }  }  }  }  // 冒泡排序-升序  void bubbleAsc(int nums[], int len) {  int i, j, temp;  for (i = 0; i < len - 1; i++) {  for (j = 0; j < len - i - 1; j++) {  // 如果前面的数大于后面的数，就交换  if (nums[j] > nums[j + 1]) {  temp = nums[j + 1];  nums[j + 1] = nums[j];  nums[j] = temp;  }  }  }  } | | **Main.cpp**  #include <iostream>  #include <chrono>  #include <cstdlib>  #include <ctime>  #include "Sort.h"  using namespace std;  // 打印字符串  void printArr(int nums[], int len) {  int i = 0;  for (; i < len; i++) {  cout << nums[i] << '\t';  if (i % 11 == 10) cout << endl;  }  cout << endl;  }  int main()  {  // 随机数种子初始化  srand(time(NULL));  // 生成500个1~2000之间的随机整数  int nums[500] = { 0 };  for (int i = 0; i < 500; i++) {  nums[i] = rand() % 2000 + 1;  }    int len = sizeof(nums) / sizeof(int);    auto start\_time = chrono::high\_resolution\_clock::now();  quickSort(nums, len);  auto end\_time = chrono::high\_resolution\_clock::now();  auto elapsed\_time = chrono::duration\_cast<chrono::nanoseconds>(end\_time - start\_time).count();  cout << "排序所用时间：" << elapsed\_time << " 纳秒" << endl;  printArr(nums, len);  return 0;  } | | **运行结果1-插入排序**    **运行结果2-希尔排序**    **运行结果3-起泡排序**    **运行结果4-快速排序** |   **五、实验总结：**  通过本次实验，学习了四种常用的排序算法，其中插入排序与冒泡排序吸收得比较好，而希尔排序与快速排序能够明白其算法思想， 但是代码实现稍微有点儿困难，需要花费很长得时间去完成代码，代码运行后实际记录的时间与预期有些差别，例如冒泡排序应该效率比较低，但是与其他算法的时间记录比起来还挺快的，可能是编译的原因。  **六、教师评语：** | | | | | |