《数据结构》实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 李爽 | 班级 | 22级软件工程专升本1班 | 学号 | 2206831522 |
| 实验名称 | **实验八 排序技术** | | | | |
| 实验时间 | 2023年5月25日 | | 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：将数组分为已排序和未排序两部分，初始时已排序部分只有一个元素，然后依次将未排序部分的元素插入到已排序部分的正确位置，直到所有元素都被插入完成。具体实现时，从第二个元素开始，将当前元素与已排序部分的元素逐个比较，找到插入位置后将其插入，并将已排序部分中的元素向后移动以腾出插入位置。  （2）希尔排序函数：将数组分割为多个子序列来进行排序，然后逐步缩小子序列的间隔，最终完成整个数组的排序。希尔排序的核心思想是通过插入排序的思想，在每个子序列中进行排序，从而减少逆序对的数量。希尔排序的关键是选择合适的间隔序列，不同的间隔序列会影响算法的效率。  （3）起泡排序函数：重复地遍历数组，比较相邻的元素并将较大的元素交换到右侧。通过多次遍历，每次遍历都将当前未排序部分的最大元素交换到正确的位置，直到整个数组排序完成。冒泡排序的特点是每一轮遍历都会将当前未排序部分的最大元素冒泡到最右侧。  （4）快速排序函数：选择一个基准元素，通过分割将数组划分为两个子数组，其中一个子数组的所有元素小于基准元素，另一个子数组的所有元素大于基准元素。然后对这两个子数组分别进行快速排序，最终将整个数组排序完成。快速排序使用了递归的思想，它的关键是选取合适的基准元素和实现划分操作。通过不断地划分和递归排序，快速排序能够快速地将大规模的问题分解为小规模的子问题，并且具有较高的排序效率。  **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  ……  }  #include <iostream>  #include <vector>  #include <random>  #include <chrono>  using namespace std;  using namespace chrono;  // 直接插入排序  void insertionSort(vector<int>& arr) {  int n = arr.size();  for (int i = 1; i < n; ++i) {  // 选取当前元素作为插入元素  int key = arr[i];  int j = i - 1;  // 向前查找插入位置并移动元素  while (j >= 0 && arr[j] > key) {  arr[j + 1] = arr[j];  j--;  }  // 插入元素到正确位置  arr[j + 1] = key;  }  }  // 希尔排序  void shellSort(vector<int>& arr) {  int n = arr.size();  // 根据增量进行分组  for (int gap = n / 2; gap > 0; gap /= 2) {  for (int i = gap; i < n; ++i) {  // 选取当前元素作为插入元素  int temp = arr[i];  int j;  // 向前查找插入位置并移动元素  for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {  arr[j] = arr[j - gap];  }  // 插入元素到正确位置  arr[j] = temp;  }  }  }  // 冒泡排序  void bubbleSort(vector<int>& arr) {  int n = arr.size();  for (int i = 0; i < n - 1; ++i) {  // 依次比较相邻的元素并交换  for (int j = 0; j < n - i - 1; ++j) {  if (arr[j] > arr[j + 1]) {  swap(arr[j], arr[j + 1]);  }  }  }  }  // 快速排序  int partition(vector<int>& arr, int low, int high) {  // 选取最后一个元素作为基准  int pivot = arr[high];  int i = low - 1;  for (int j = low; j <= high - 1; ++j) {  // 将小于基准的元素放到左边  if (arr[j] < pivot) {  i++;  swap(arr[i], arr[j]);  }  }  // 将基准放到正确位置  swap(arr[i + 1], arr[high]);  return (i + 1);  }  void quickSort(vector<int>& arr, int low, int high) {  if (low < high) {  // 划分数组，并获取基准的位置  int pi = partition(arr, low, high);  // 递归地对左半部分进行快速排序  quickSort(arr, low, pi - 1);  // 递归地对右半部分进行快速排序  quickSort(arr, pi + 1, high);  }  }  // 生成随机数  vector<int> generateRandomNumbers(int count, int minValue, int maxValue) {  vector<int> numbers(count);  random\_device rd;  mt19937 gen(rd());  uniform\_int\_distribution<int> dis(minValue, maxValue);  for (int i = 0; i < count; ++i) {  numbers[i] = dis(gen);  }  return numbers;  }  int main() {  // 生成随机数  vector<int> numbers = generateRandomNumbers(500, 1, 2000);  // 直接插入排序  vector<int> numbersInsertion = numbers;  auto startInsertion = high\_resolution\_clock::now();  insertionSort(numbersInsertion);  auto stopInsertion = high\_resolution\_clock::now();  auto durationInsertion = duration\_cast<microseconds>(stopInsertion - startInsertion);  cout << "直接插入排序所用时间（微秒）: " << durationInsertion.count() << endl;  // 希尔排序  vector<int> numbersShell = numbers;  auto startShell = high\_resolution\_clock::now();  shellSort(numbersShell);  auto stopShell = high\_resolution\_clock::now();  auto durationShell = duration\_cast<microseconds>(stopShell - startShell);  cout << "希尔排序所用时间（微秒）: " << durationShell.count() << endl;  // 冒泡排序  vector<int> numbersBubble = numbers;  auto startBubble = high\_resolution\_clock::now();  bubbleSort(numbersBubble);  auto stopBubble = high\_resolution\_clock::now();  auto durationBubble = duration\_cast<microseconds>(stopBubble - startBubble);  cout << "冒泡排序所用时间（微秒）: " << durationBubble.count() << endl;  // 快速排序  vector<int> numbersQuick = numbers;  auto startQuick = high\_resolution\_clock::now();  quickSort(numbersQuick, 0, numbersQuick.size() - 1);  auto stopQuick = high\_resolution\_clock::now();  auto durationQuick = duration\_cast<microseconds>(stopQuick - startQuick);  cout << "快速排序所用时间（微秒）: " << durationQuick.count() << endl;  return 0;  }    **五、实验总结：**  本次实验使用C++编程语言完成了以下任务：随机生成500个1～2000之间的整数，并对这些整数进行四种不同的排序算法：直接插入排序、希尔排序、冒泡排序和快速排序。通过记录每种算法的排序时间，我们可以评估它们的效率差异。实验结果表明，快速排序算法展现了最佳性能，其次是希尔排序和直接插入排序，而冒泡排序的效率相对较低。这次实验使我们更深入了解了不同排序算法的特点和应用场景，为我们在实际问题中选择合适的排序算法提供了参考。  **六、教师评语：** | | | | | |