**《数据结构综合设计》实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | 人工智能与大数据学院 | 专业 | 虚拟现实技术 | 班级 | 21级2班 | 学生姓名 | 赵扬帆 |
| 实验  周次 | 13-14 | 实验  日期 | 2023.5.26 | 学时 | 6 | 教师姓名 | 李昊康 |
| 项目名称 | | 排序的应用 | | | | | |
| 实验  类别 | 🗹验证型实验 🞎设计型实验 🞎综合型实验 🞎其它 | | | | | 成绩：90 | |
| 1. 实验目的及具体要求   实验目的：  1.实现多种类型的排序算法（插入排序、交换排序、选择排序、归并排序等）；  2.理解排序过程；  3.计算比较次数和移动次数，对比分析算法性能的优劣与适用场景；  具体要求：  编写程序实现插入排序、希尔排序、冒泡排序、快速排序、简单选择排序、堆排序。中任意3种。   1. 实验仪器、设备和材料   硬设备：PC机  软件环境：Windows VS2019   1. 实验内容、步骤及实验数据记录   冒泡排序  #include <iostream>  #include <string>  using namespace std;  int main() {  int n[10];  int i, j;  int temp; //用于缓存需要交换的数字  cout << "请输入十个数字！" << endl;  for (i = 0; i < 10; i++) {  cin >> n[i];  }  for (i = 0; i < 9; i++) { //共进行9步  for (j = 0; j < 9 - i; j++) { //在每一步进行10-i次两两比较  if (n[j] > n[j + 1]) {  temp = n[j];  n[j] = n[j + 1];  n[j + 1] = temp;  }  }  }  cout << "排序后的数据是：" << endl;  for (i = 0; i < 10; i++)  {  cout << n[i] << ' ';  }  cout << endl;  system("pause");  return 0;  }    快速排序  #include <stdio.h>  #include <iostream>  #include <math.h>  #include <algorithm>  using namespace std;  int part(int\* r, int low, int hight) //划分函数  {  int i = low, j = hight, pivot = r[low]; //基准元素  while (i < j)  {  while (i<j && r[j]>pivot) //从右向左开始找一个 小于等于 pivot的数值  {  j--;  }  if (i < j)  {  swap(r[i++], r[j]); //r[i]和r[j]交换后 i 向右移动一位  }  while (i < j && r[i] <= pivot) //从左向右开始找一个 大于 pivot的数值  {  i++;  }  if (i < j)  {  swap(r[i], r[j--]); //r[i]和r[j]交换后 i 向左移动一位  }  }  return i; //返回最终划分完成后基准元素所在的位置  }  void Quicksort(int\* r, int low, int hight)  {  int mid;  if (low < hight)  {  mid = part(r, low, hight); // 返回基准元素位置  Quicksort(r, low, mid - 1); // 左区间递归快速排序  Quicksort(r, mid + 1, hight); // 右区间递归快速排序  }  }  int main()  {  int a[10001];  int N;  cout << "请输入要排序的数据的个数： " << endl;  cin >> N;  cout << "请输入要排序的数据： " << endl;  for (int i = 0; i < N; i++)  {  cin >> a[i];  }  cout << endl;  Quicksort(a, 0, N - 1);  cout << "排序后的序列为： " << endl;  for (int i = 0; i < N; i++)  {  cout << a[i] << " ";  }  cout << endl;  return 0;  }    堆排序  #include<cstdio>  #include<iostream>  #include<cstring>  #include<algorithm>  using namespace std;  void adjust(int arr[], int len, int index)  {  int left = 2 \* index + 1;  int right = 2 \* index + 2;  int maxIdx = index;  if (left<len && arr[left] > arr[maxIdx]) maxIdx = left;  if (right<len && arr[right] > arr[maxIdx]) maxIdx = right; // maxIdx是3个数中最大数的下标  if (maxIdx != index) // 如果maxIdx的值有更新  {  swap(arr[maxIdx], arr[index]);  adjust(arr, len, maxIdx); // 递归调整其他不满足堆性质的部分  }  }  void heapSort(int arr[], int size)  {  for (int i = size / 2 - 1; i >= 0; i--) // 对每一个非叶结点进行堆调整(从最后一个非叶结点开始)  {  adjust(arr, size, i);  }  for (int i = size - 1; i >= 1; i--)  {  swap(arr[0], arr[i]); // 将当前最大的放置到数组末尾  adjust(arr, i, 0); // 将未完成排序的部分继续进行堆排序  }  }  int main()  {  int array[8] = { 8, 1, 14, 3, 21, 5, 7, 10 };  heapSort(array, 8);  for (auto it : array)  {  cout << it << endl;  }  return 0;  }    分析：  冒泡排序：  稳定性：我们从代码中可以看出只有前一个元素大于后一个元素才可能交换位置，所以相同元素的相对顺序不可能改变，所以它是稳定排序  比较性：因为排序时元素之间需要比较，所以是比较排序  时间复杂度：因为它需要双层循环n\*(n-1))，所以平均时间复杂度为O(n^2)  空间复杂度：只需要常数个辅助单元，所以空间复杂度为O(1)，我们把空间复杂度为O(1)的排序成为原地排序（in-place）  记忆方法：想象成气泡，一层一层的往上变大  快速排序：  优点：平均性能好，O(nlog2n)，2为下标  缺点：不稳定，初始序列有序或基本有序时，时间复杂度降为O(n^2)。  适用场景：快速排序时间复杂度为O(nlogn)，是目前基于比较的内部排序中被认为最好的方法，当数据过大且数据杂乱无章时，则适合采用快速排序  堆排序：  堆排序的时间复杂度是O(nlgn)，堆排序的时间复杂度不是O(n2)，因为堆采用了额外的空间，来降低了时间复杂度。堆排序有一个明显的问题，不管数组是否有序或者无序，都要从头到尾进行一遍排序，就好像是没有优化过的冒泡排序一样，从头冒到尾。因此，使得在数据量较少时，堆排序的时间复杂度的常数比较高，进而导致排序时间较长。 适用场景：优先级队列，利用堆求topK | | | | | | | |

说明：1. 实验周次：填写实际上课周，如第5-8周上课填“5-8”或第10周上课填“10”。

1. 实验报告各部分内容需详实填写，按实验指导书上的评分标准给出分数。
2. 实验类型参考实验类型说明文件。