**《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1班 | 学生姓名 | 翟家杏 |
| 实验  周次 | 13-14 | 实验  日期 | 2023.5.29 | 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：95 | |
| 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   插入排序：  #include<bits/stdc++.h>  using namespace std;  void InsertSort(int a[],int l)  {  int temp;  int j;  for(int i=1;i<l;i++)  {  if(a[i]<a[i-1])  {  temp=a[i];  for(j=i-1;j>=0&&temp<a[j];j--)  {  a[j+1]=a[j];  }  a[j+1]=temp;  }  for(int k=0;k<l;k++)  cout<<a[k]<<" ";  cout<<endl;  }  }  int main()  {  int a[10]={2,5,8,3,6,9,1,4,7};  int b[10]={1,2,3,4,5,6,7,8,9};  int len=9;  InsertSort(a,len);  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)  {      int i = low, j = hight, pivot = r[low];      while (i < j)      {          while (i<j && r[j]>pivot)          {              j--;          }          if (i < j)          {              swap(r[i++], r[j]);          }          while (i < j && r[i] <= pivot)          {              i++;          }          if (i < j)          {              swap(r[i], r[j--]);          }      }      return i;  }  void Quicksort(int\* r, int low, int hight)  {      int mid;      if (low < hight)      {          mid = part(r, low, hight);          Quicksort(r, low, mid - 1);          Quicksort(r, mid+1, hight);      }  }  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;  }    希尔排序：  #include <iostream>  using namespace std;  void Print(int array[],int length){      for(int i=0;i<length;i++){          cout<<array[i]<<" ";      }      cout<<endl;  }  void ShellSort(int array[],int length){      for(int step=length/2;step>0;step=step/2){          for(int i=0;i<step;i++){              for(int j=i+step;j<length;j+=step){                  int temp=array[j];                  int m=j-step;                  for(m=j-step;m>=0&&array[m]>temp;m-=step){                      array[m+step]=array[m];                  }                  array[m+step]=temp;              }          }      Print(array,length);      }  }  int main(){      int array[]={49,38,65,97,76,13,27,49};      int length=sizeof(array)/sizeof(\*array);      Print(array,length);      ShellSort(array,length);      return 0;  }    分析：**插入排序**：元素集合越接近有序，直接插入排序算法的时间效率越高，它是一种稳定的排序算法。 适用场景：一个新元素需要插入到一组已经是有序的数组中，或者是一组基本有序的数组排序。 **快速排序**：整体的综合性能和使用场景都是比较好的，所以才敢叫快速排序。不稳定。此排序算法的效率在序列越乱的时候，效率越高。在数据有序时，会退化成冒泡排序； 适用场景：在给大量数据排序的时候，快排的效率尤为明显 **希尔排序**：希尔排序是对直接插入排序的优化。当gap > 1时都是预排序，目的是让数组更接近于有序。当gap == 1时，数组已经接近有序的了，这样就 会很快。这样整体而言，可以达到优化的效果。我们实现后可以进行性能测试的对比。 比较在希尔排序中是最主要的操作，而不是交换。用已知最好的步长序列的希尔排序比直接插入排序要快，甚至在小数组中比快速排序和堆排序还快，但在涉及大量数据时希尔排序还是不如快排；不稳定 适用场景：适合小数组小数据或使用于中规模数据量的排序。 | | | | | | | |

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。