**郑 州 轻 工 业 学 院**

**实 验 报 告**

**课程名称：** 数据结构

**实验名称：** 常用的内部排序算法

**院 （系）：** 计算机与通信工程学院

**姓 名：** 刘亚炜

**学 号：** 541707090128

**专业班级：** 物联网工程17-1

**指导教师：** 孟颍辉

2019 **年** 6 **月** 6 **日**

**实验报告成绩评定表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 评定项目 | 内 容 | 满 分 | 评 分 | 总 分 |
| 实验态度 | 实验认真，态度端正，遵守纪律，出勤情况。 | 20 |  |  |
| 实验过程 | 按实验要求完成各种功能或操作，代码书写规范，注释清晰，设计严谨，运行结果正确。 | 40 |  |
| 报告撰写 | 报告字迹整洁、内容丰富、条理清楚；图、表、文字表达准确规范，上交及时。 | 40 |  |
| 总成绩 | 采用五级分制：优、良、中、及格、不及格 |  | | |
| **评语：**  **指导老师签字： 年 月 日** | | | | |

实验报告正文

1. **实验目的**

掌握常见的内部排序算法的思想及其适用条件；掌握常见的内部排序算法的程序实现。

1. **实验内容及要求**
2. 任务描述

设计一个内部排序算法模拟系统，利用该系统实现常用的7种排序算法，并测试各种排序算法的性能。

1. 主要数据类型与变量

vector<int> arr;

#define MAXSIZE 100 /\*参加排序元素的最大个数\*/

typedef int KeyType;

typedef struct {

KeyType key;

InfoType otherinfo; // 其他字段（自己设计）

}RedType;

typedef struct

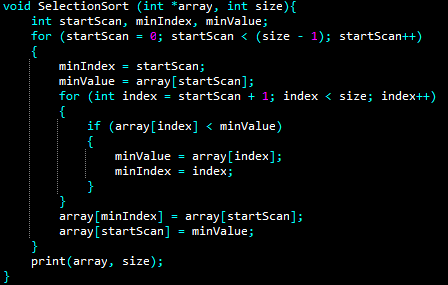
{

RedType r[MAXSIZE+1];

int length; /\*参加排序元素的实际个数\*/

}SqList;

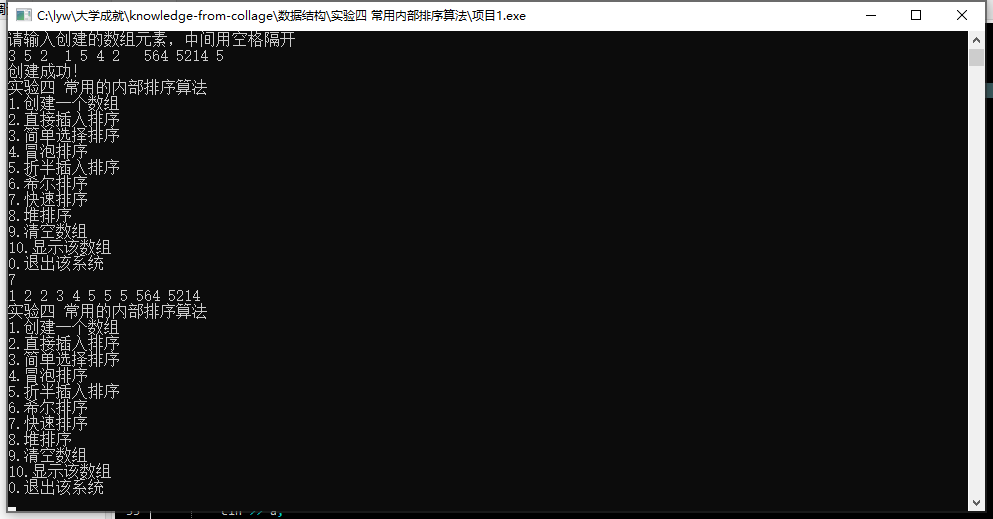
1. 算法或程序模块



1. **测试**
   1. 方案

使用随机生成的10万个数据进行排序，程序执行正常，但是快速排序的平均效率要高于其他排序算法很多。

* 1. 结果



1. **总结与讨论**

本次实验中每个排序算法都是以函数形式表现，对于数组的类型，如果是结构体类型的话则需要重载运算符，从大到小和从小到大排序时可以直接自行构建cmp函数进行大小比较（定义排序规则）。

**附：程序的源代码**

**#include <bits/stdc++.h>**

**#define left(x) 2\*x+1**

**#define right(x) 2\*(x+1)**

**using namespace std;**

**vector<int> arr;**

**int length;**

**void menu(){**

**cout<<"实验四 常用的内部排序算法\n";**

**cout<<"1.创建一个数组\n";**

**cout<<"2.直接插入排序\n";**

**cout<<"3.简单选择排序\n";**

**cout<<"4.冒泡排序\n";**

**cout<<"5.折半插入排序\n";**

**cout<<"6.希尔排序\n";**

**cout<<"7.快速排序\n";**

**cout<<"8.堆排序\n";**

**cout<<"9.清空数组\n";**

**cout<<"10.显示该数组\n";**

**cout<<"0.退出该系统\n";**

**}**

**void print(int \*a, int len){**

**for(int i = 0; i < len; i++) {**

**cout<<a[i]<<" ";**

**}**

**cout<<endl;**

**}**

**void creat\_array(){**

**cout<<"请输入创建数组的长度\n";**

**cin >> length;**

**cout<<"请输入创建的数组元素，中间用空格隔开\n";**

**for(int i = 0; i < length; i++) {**

**int a;**

**cin >> a;**

**arr.push\_back(a);**

**}**

**cout<<"创建成功！\n";**

**}**

**void InsertSort(int\* a,int len)//插入排序**

**{**

**int begin = 1;**

**int i = 0;**

**while(begin < len)**

**{**

**int key = a[begin];**

**for(i = begin-1;i>=0;i--)**

**{**

**if(a[i]<=key)**

**{**

**a[i+1] = key;**

**break;**

**}**

**a[i+1] = a[i];**

**}**

**if(i<0)**

**a[0] = key;//说明找完了整个有序子序列都没找到**

**begin++;**

**}**

**print(a, len);**

**}**

**void SelectionSort (int \*array, int size){**

**int startScan, minIndex, minValue;**

**for (startScan = 0; startScan < (size - 1); startScan++)**

**{**

**minIndex = startScan;**

**minValue = array[startScan];**

**for (int index = startScan + 1; index < size; index++)**

**{**

**if (array[index] < minValue)**

**{**

**minValue = array[index];**

**minIndex = index;**

**}**

**}**

**array[minIndex] = array[startScan];**

**array[startScan] = minValue;**

**}**

**print(array, size);**

**}**

**void BubbleSort(int \*array, int len){**

**int t;**

**for(int i = 0; i < len - 1; i++) {**

**for(int j = i + 1; j < len; j++) {**

**if(array[i]>array[j]){**

**t = array[j];**

**array[j] = array[i];**

**array[i] = t;**

**}**

**}**

**}**

**print(array, len);**

**}**

**void InsertBinarySort(int\* a, int n){**

**for(int i=1;i<n;++i)**

**{**

**if(a[i]<a[i-1])**

**{**

**int low = 0;**

**int high = i-1;**

**int j;**

**int temp = a[i];**

**while(low<=high)**

**{**

**int mid=(low+high)/2;**

**if(a[mid]==a[i])**

**{//将a[i]放在a[mid]右边**

**for( j=i-1;j>mid;--j)**

**{**

**a[j+1]=a[j];**

**}**

**a[mid+1] = temp;**

**}**

**else if(a[mid]<a[i])**

**{**

**low=mid+1;**

**}**

**else**

**{**

**high=mid-1;**

**}**

**}**

**if(low>high)/\*说明当前插入的元素不在已排好序的数列中,此时应该将元素放在index等于low处，或者index等于high+1处；\*/**

**{**

**for( j=i-1;j>=low;--j)**

**{**

**a[j+1]=a[j];**

**}**

**a[low] = temp;**

**}**

**}**

**}**

**print(a, n);**

**}**

**void ShellSort(int array[], int n){//希尔排序**

**int i, j;**

**int increment = n;**

**do{**

**increment = increment / 3 + 1;//增量序列**

**for (i = increment + 1; i <= n; i++)**

**{**

**if (array[i] < array[i - increment])**

**{**

**array[0] = array[i];**

**for (j = i - increment; j>0 && array[0] < array[j]; j -= increment)**

**array[j + increment] = array[j];**

**array[j + increment] = array[0];**

**}**

**}**

**}**

**while (increment>1);**

**print(array, n);**

**}**

**void quicksort(int data[], int first, int last){**

**int lower = first + 1;**

**int upper = last;**

**swap(data[first], data[(first + last) / 2]);**

**int bound = data[first];**

**while (lower <= upper)**

**{**

**while (data[lower] < bound)**

**lower++;**

**while (data[upper] > bound)**

**upper--;**

**if (lower < upper)**

**swap(data[lower++], data[upper--]);**

**else lower++;**

**}**

**swap(data[upper], data[first]);**

**if (first < upper - 1)**

**quicksort(data, first, upper - 1);**

**if (upper + 1 < last)**

**quicksort(data, upper + 1, last);**

**}**

**//假定对某一个节点i其左，右子树都是都是最大堆，但是对于节点i和它的左右子节点则可能破坏最大堆的性质，我们来写一个函数对这**

**//情况下的堆来进行维护使整体的堆满足最大堆性质**

**void MaxHeapify(int\* a,int i,int low,int high){/\*输入为要被排序的数组和根节点，数组a当中被维护的那一部分的下标low，high\*/**

**int l = left(i);//计算下标为i的节点的左子节点**

**int r = right(i);//计算下标为i的节点的右子节点**

**int largest;//保存i,l,r(即i和它的左右子节点)之间的最大数的下标**

**int temp;//交互数组中的数所使用的临时变量**

**//找到三个数当中最大的那个数，将最大的那个数和i进行互换**

**if (l<=high && a[l]>a[i]){**

**largest = l;**

**}else{**

**largest = i;**

**}**

**if (r<=high && a[r]>a[largest]){**

**largest = r;**

**}**

**if (largest != i){**

**temp = a[i];**

**a[i] = a[largest];**

**a[largest] = temp;**

**MaxHeapify(a, largest,low,high);//交换有可能破坏子树的最大堆性质，所以对所交换的那个子节点进行一次维护，而未交换的那个子节点，根据我们的假设，是保持着最大堆性质的。**

**}**

**}**

**//将数组建立为一个最大堆**

**//调整数组当中数的位置将其处理为一个最大堆**

**void BuildMaxHeap(int\* a,int length){**

**for (int i = length / 2-1; i >= 0; i--){**

**MaxHeapify(a, i, 0, length - 1);**

**}**

**}**

**//堆排序函数**

**void HeapSort(int\* a,int length){**

**int temp;**

**BuildMaxHeap(a,length);**

**for (int i = length - 1; i >= 1; i--){**

**//交换根节点和数组的最后一个节点**

**temp = a[i];**

**a[i] = a[0];**

**a[0] = temp;**

**MaxHeapify(a, 0, 0, i-1);//维护从下标为i-1到0的子数组**

**}**

**}**

**int main(){**

**while(1){**

**menu();**

**int ans;**

**cin >> ans;**

**switch(ans){**

**case 1:**

**creat\_array();**

**break;**

**case 2:**

**InsertSort(&arr[0], arr.size());**

**break;**

**case 3:**

**SelectionSort(&arr[0], arr.size());**

**break;**

**case 4:**

**BubbleSort(&arr[0], arr.size());**

**break;**

**case 5:**

**InsertBinarySort(&arr[0], arr.size());**

**break;**

**case 6:**

**ShellSort(&arr[0], arr.size());**

**break;**

**case 7:**

**sort(arr.begin(), arr.end());**

**print(&arr[0], arr.size());**

**break;**

**case 8:**

**HeapSort(&arr[0], arr.size());**

**print(&arr[0], arr.size());**

**break;**

**case 9:**

**arr.clear();**

**break;**

**case 10:**

**print(&arr[0], arr.size());**

**break;**

**case 0:**

**return 0;**

**}**

**}**

**}**