

南京信息工程大学 数据结构 I 实验(实习)报告

实验(实习)名称 排序和二分查找 日期 2024.12. 得分 _____ 指导教师 _____
学院 计院 专业 计科 _____

一、实验目的

- 1、掌握二分查找算法并能在有序表中进行查找操作；
- 2、掌握查找的不同方法，并能用高级语言应用线性结构实现查找；
- 3、掌握直接插入排序、冒泡排序、简单选择排序算法及应用。

二、实验内容与步骤

1、从键盘输入无序关键字序列：49, 38, 65, 97, 76, 13, 27, 49

(1) 编写程序分别采用直接插入排序（见教材 P265 算法 10.1）、冒泡排序、简单选择排序算法（见教材 P277 算法 10.9），对该序列由小到大进行排序，并输出排序结果。

(2) 利用二分查找法在上题排序结果的基础上对从键盘接收的任一个关键字 key 进行查找，若找到则提示查找成功并输出 key 所在的位置，否则提示没有找到信息（见教材算法 9.2）。
报告要求：

(1) 分别写出直接插入排序、冒泡排序、简单选择排序算法思想及时间复杂度；

直接插入排序：逐步构建有序序列来进行排序。初始时，将第一个元素视为有序部分，从第二个元素开始依次将未排序的元素插入到有序部分的正确位置。插入时需要通过比较找到插入位置，并将后续元素依次后移，为待插入元素空出位置。

时间复杂度： $O(n^2)$

冒泡排序：通过多次比较相邻元素，将较大的元素逐步冒泡到序列末尾。每一次排序完成后，末排序部分的最大元素会被移动到正确位置。通过多次遍历，直到整个有序序列有序。

时间复杂度： $O(n^2)$

简单选择排序：在未排序部分中每次找到最小值，将其与当前排序位置的元素交换。通过多次选择和交换操作，最终将整个序列排成有序。每一趟排序固定一个元素到正确位置，且不依赖序列的初始状态。

时间复杂度： $O(n^2)$

(2) 写出二分查找的应用条件及其算法思想；

应用条件：数据必须有序，从大到小或从小到大排序；数据以顺序存储；数据集合在查找过程中不发生变化；数据量较大的数据适用，时间复杂度为 $O(\log n)$ 。

算法思想：在一个有序的数组中，通过比较待查值（key）与数组中间元素的大小关系，逐步缩小查找范围；每次将查找范围缩小为当前范围的一半，知道找到目标值或者范围为空。

(3) 完整的实现代码（文本）和测试结果（图片）。

实验代码：

```
#include <iostream>
```

```
using namespace std;
```

```
#define MAX_SIZE 20
typedef int KeyType; // 定义关键字类型为整型数据
typedef struct{
```

```

    KeyType key;           // 关键字项
}RedType;                 // 记录类型
typedef struct{
    RedType r[MAX_SIZE+1]; // r[0]闲置或用作哨兵单元
    int length;            // 顺序表长度
}SqList;                  // 顺序表类型

// 直接插入排序
void InsertSort(SqList &L){
    int i,j;
    for(i=2;i<=L.length;i++){
        if(L.r[i].key<L.r[i-1].key){
            L.r[0]=L.r[i];
            L.r[i]=L.r[i-1];
            for(j=i-2;L.r[0].key<L.r[j].key;j--){
                L.r[j+1]=L.r[j]; // 记录后移
            }
            L.r[j+1]=L.r[0];
        }
    }
}

// 找到最小关键字的索引
int SelectMinKey(SqList &L,int start){
    int min=start;
    for(int i=start+1;i<=L.length;i++){
        if(L.r[i].key<L.r[min].key){
            min=i;
        }
    }
    return min;
}

// 简单选择排序
void SelectSort(SqList &L){
    for(int i=1;i<L.length;i++){
        int minIndex=SelectMinKey(L,i);
        if(i!=minIndex){
            L.r[0]=L.r[i];
            L.r[i]=L.r[minIndex];
            L.r[minIndex]=L.r[0];
        }
    }
}

// 冒泡排序实现
void BubbleSort(SqList &L){

```

```

        for(int i=0;i<L.length;i++){
            for(int j=1;j<L.length;j++){
                if(L.r[j].key>L.r[j+1].key){
                    RedType temp=L.r[j];
                    L.r[j]=L.r[j+1];
                    L.r[j+1]=temp;
                }
            }
        }
    }

void PrintList(SqList &L){
    for(int i=1;i<=L.length;i++){
        cout<<L.r[i].key<<" ";
    }
    cout<<endl;
}

// 二分查找
int BinarySearch(SqList &L,KeyType key){
    int low=1,high=L.length; // 初始化查找范围
    while(low<=high){
        int mid=(low+high)/2;
        if(L.r[mid].key==key){
            return mid;
        }else if(L.r[mid].key<key){
            low=mid+1;
        }else{
            high=mid-1;
        }
    }
    return 0; // 未找到返回 0
}

// 二分查找第一个索引
int BinarySearchFirst(SqList &L,KeyType key){
    int low=1,high=L.length; // 初始化查找范围
    int result=0;
    while(low<=high){
        int mid=(low+high)/2;
        if(L.r[mid].key==key){
            result=mid;
            high=mid-1;
        }else if(L.r[mid].key<key){
            low=mid+1;
        }else{
            high=mid-1;
        }
    }
}

```

```

        }
    }

    return result; // 未找到返回 0
}

// 二分查找最后一个索引
int BinarySearchLast(SqList &L,KeyType key){
    int low=1,high=L.length; // 初始化查找范围
    int result=0;
    while(low<=high){
        int mid=(low+high)/2;
        if(L.r[mid].key==key){
            result=mid;
            low=mid+1;
        }else if(L.r[mid].key<key){
            low=mid+1;
        }else{
            high=mid-1;
        }
    }
    return result; // 未找到返回 0
}

int main() {
    SqList L;
    L.length = 0;
    cout<<"请输入无序关键字序列: ";
    int input;
    while(cin>>input){
        if(L.length>=MAX_SIZE){
            cout<<"越界"<<endl;
            return 1;
        }
        L.r[+L.length].key=input;
        if(cin.peek()=='\n') break; // 判断是否按下了 Enter 键
    }
    // 排序前输出
    cout<<"排序前的序列: ";
    PrintList(L);
    cout<<"请选择排序算法: 1.直接插入排序 2.选择排序 3.冒泡排序"<<endl;
    int chose;
    cin>>chose;
    switch(chose){
        case 1:
            InsertSort(L);
            cout<<"直接插入排序: ";

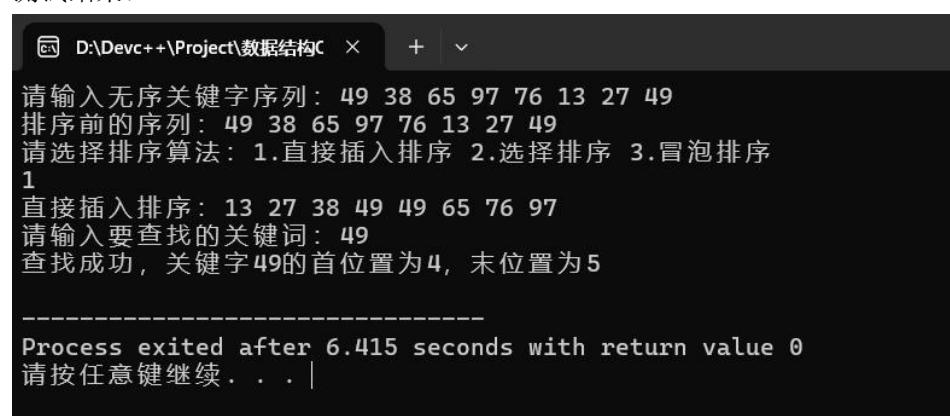
```

```

    PrintList(L);
    break;
case 2:
    SelectSort(L);
    cout<<"选择排序: ";
    PrintList(L);
    break;
case 3:
    BubbleSort(L);
    cout<<"冒泡排序: ";
    PrintList(L);
    break;
default:
    cout<<"无效选择, 请重新选择" << endl;
    break;
}
// 进行二分查找
cout<<"请输入要查找的关键词: ";
int key;
cin>>key;
int positionfirst=BinarySearchFirst(L,key);
int positionlast=BinarySearchLast(L,key);
if(positionfirst&&positionlast&&positionfirst!=positionlast){
    cout<<"查找成功, 关键字"<<key<<"的首位置为"<<positionfirst<<", 末位置为
"<<positionlast<< endl;
}else if(positionfirst==positionlast){
    cout<<"查找成功, 关键字"<<key<<"的位置为"<<positionfirst<< endl;
}else{
    cout<<"查找失败" << endl;
}
return 0;
}
// 49 38 65 97 76 13 27 49

```

测试结果：



```

D:\DevC++\Project\数据结构C

请输入无序关键字序列: 49 38 65 97 76 13 27 49
排序前的序列: 49 38 65 97 76 13 27 49
请选择排序算法: 1.直接插入排序 2.选择排序 3.冒泡排序
1
直接插入排序: 13 27 38 49 49 65 76 97
请输入要查找的关键词: 49
查找成功, 关键字49的首位置为4, 末位置为5

-----
Process exited after 6.415 seconds with return value 0
请按任意键继续. . .

```

```
D:\DevC++\Project\数据结构C X + ▾  
请输入无序关键字序列：49 38 65 97 76 13 27 49  
排序前的序列：49 38 65 97 76 13 27 49  
请选择排序算法：1.直接插入排序 2.选择排序 3.冒泡排序  
2  
选择排序：13 27 38 49 49 65 76 97  
请输入要查找的关键词：23  
查找失败  
-----  
Process exited after 5.195 seconds with return value 0  
请按任意键继续... .
```

```
D:\DevC++\Project\数据结构C X + ▾  
请输入无序关键字序列：49 38 65 97 76 13 27 49  
排序前的序列：49 38 65 97 76 13 27 49  
请选择排序算法：1.直接插入排序 2.选择排序 3.冒泡排序  
3  
冒泡排序：13 27 38 49 49 65 76 97  
请输入要查找的关键词：76  
查找成功，关键字76的位置为7  
-----  
Process exited after 24.62 seconds with return value 0  
请按任意键继续... .
```

三、实验心得（必写）

在本次实验中，我深入学习和实践了三种经典排序算法——直接插入排序、冒泡排序、简单选择排序，并结合二分查找实现了对有序表的高效查找。这次实验让我更加理解了算法的基本思想和实现细节，也对算法的时间复杂度有了更深刻的认识。

首先，通过直接插入排序的实现，我体会到其逐步构建有序序列的过程，并理解了插入时需要通过比较找到正确位置的重要性。冒泡排序让我感受到算法的直观性，其多次比较和交换的方式清晰易懂，但在效率上较为一般。而简单选择排序在未排序部分中找到最小值并交换位置的思路简单明确，适合数据规模较小的场景。

其次，在实现二分查找时，我认识到它对于有序表的重要性和高效性。通过进一步扩展，实现了查找第一个和最后一个出现位置的功能，从而能够应对重复元素的情况，增强了算法的实用性。

通过这次实验，我加深了对数据结构和算法的理解，尤其是排序与查找之间的关系。同时，我也进一步掌握了C++语言在实现算法中的灵活性和细节处理能力，如数组的操作、输入输出控制等。此外，我发现了程序调试的重要性，例如处理输入越界和无效数据时，代码的健壮性显得尤为关键。总而言之，收获颇丰。