

学生学号	0121510880209	实验课成绩	
------	---------------	-------	--

武汉理工大学

学 生 实 验 报 告 书

实验课程名称 数据结构

开 课 学 院 计算机科学与技术学院

指导老师姓名 钟欣

学 生 姓 名 彭玉全

学生专业班级 软件 1503

2016 — 2017 学年 第 1 学期

实验课程名称： 数据结构

实验项目名称	查找算法的设计及实现			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-12-12

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

实验题 7 二叉排序树的查找

【问题描述】读入一串整数，构造一棵二叉排序树。然后在此树上查找值为 x 的结点。

【实现提示】二叉排序树的构成，可从空的二叉树开始，每输入一个结点，就将其插入到当前已形成的二叉排序树中，所以，它实际是利用了二叉排序树的插入算法。

【测试数据】

读入一串整数：10,8,23,5,67,98,34,20，构造 BST。

二、实验基本原理与设计(算法与程序设计)

问题分析：

二叉排序树问题，当二叉排序树不为空是，首先将给定值和根节点的关键字比较，若相等，则查找成功，否则依据给定值和根节点的关键字之间的大小关系，分别在左子树或右子树上继续查找

算法分析：

```

BiTree SrarchBST_1(BiTree T, KeyType key) {
//在指针T所指的二叉排序树中递归地查找某关键字等于key的数据元素
//若查找成功则返回指向该数据元素的指针，否返回null
    if ((!T) || EQ(key, T->data))
        return T;
    else if (LT(key, T->data))
        return (SrarchBST_1(T->lchild, key)); //左子树
    else return (SrarchBST_1(T->rchild, key)); //右子树
}

int SearchBST_2(BiTree T, KeyType key, BiTree f, BiTree &p) {
//在指针T所指的二叉排序树中递归地查找某关键字等于key的数据元素
//若查找成功则返回指向该数据元素的指针，并返回true 否则指针p指向查找
//路径上访问的最后一个节点并返回false 指针f指向T的双亲，其初始值null
    if (!T) {
        p = f; return 0; //返回失败
    }
}

```

```

    }
    else if (EQ(key, T->data)) {
        p = T; return 1;
    }
    else if (LT(key, T->data))
        return SearchBST_2(T->lchild, key, T, p);
    else
        return SearchBST_2(T->rchild, key, T, p);
}

int InsertBST(BiTree &T, ElemType e) {
//当二叉排序树中不存在关键字等于key的元素时 插入e并返回true
    BiTree p, s;
    if (!SearchBST_2(T, e, NULL, p)) { //查找不成功
        s = (BiTree)malloc(sizeof(BiTNode));
        s->data = e; s->lchild = s->rchild = NULL;
        if (!p) {
            T = s; //被插入节点为新的节点
        }
        else if (LT(e, p->data))
            p->lchild = s; //被插入节点 *s为
        else
            p->rchild = s;
        return 1;
    }
    else
        return 0;
}

```

源代码:

```

#include<iostream>
#include<stdlib.h>

//对数值比较的约定定义
#define EQ(a,b) ((a) == (b))
#define LT(a,b) ((a)<(b))
#define LQ(a,b) ((a)<=(b))

using namespace std;
typedef int ElemType; //元素的数据类型为Elemtype
typedef int KeyType;

typedef struct BiTNode { //定义二叉链表结构
    ElemType data;

```

```

    struct BiTNode * lchild, *rchild; //左右孩子指针
}BiTNode,*BiTree;

/*****二叉排序树的查找算法*****/
BiTree SrarchBST_1(BiTree T, KeyType key) {
    if ((!T) || EQ(key, T->data))
        return T;
    else if (LT(key, T->data))
        return (SrarchBST_1(T->lchild, key));
    else return (SrarchBST_1(T->rchild, key));
}

int SearchBST_2(BiTree T, KeyType key, BiTree f, BiTree &p) {
    if (!T) {
        p = f; return 0; //返回失败
    }
    else if (EQ(key, T->data)) {
        p = T; return 1;
    }
    else if (LT(key, T->data))
        return SearchBST_2(T->lchild, key, T, p);
    else
        return SearchBST_2(T->rchild, key, T, p);
}

int InsertBST(BiTree &T, ElemType e) {
    BiTree p, s;
    if (!SearchBST_2(T, e, NULL, p)) { //查找不成功
        s = (BiTree)malloc(sizeof(BiTNode));
        s->data = e; s->lchild = s->rchild = NULL;
        if (!p) {
            T = s; //被插入节点为新的节点
        }
        else if (LT(e, p->data))
            p->lchild = s; //被插入节点 *s为
        else
            p->rchild = s;
        return 1;
    }
    else
        return 0;
}

void Print(BiTree root)

```

```

{
    BiTree p = root;
    if (p != NULL) {
        Print(p->lchild);
        cout << p->data << ", ";
        Print(p->rchild);
    }
}

int main()
{
    BiTree T = NULL;
    BiTree p = NULL;
    int num = 0;
    int key;
    cout << "请输入关键字个数: ";
    cin >> num;
    cout << "请依次输入关键字: " << endl;
    for (int i = 0; i < num; i++) {
        int temp = 0;
        cin >> temp;
        InsertBST(T, temp);
    }
    cout << "请输入你要查找的值: ";
    cin >> key;
    p = SrarchBST_1(T, key);
    if (p) {
        cout << "查找成功: " << p->data << endl;
    }
    else
        cout << "未找到你要的关键字" << endl;
    Print(T);
    return 0;
}

```

三、主要仪器设备及耗材

1. 实验设备

PC 机

2. 开发环境

VS 2015 Pro

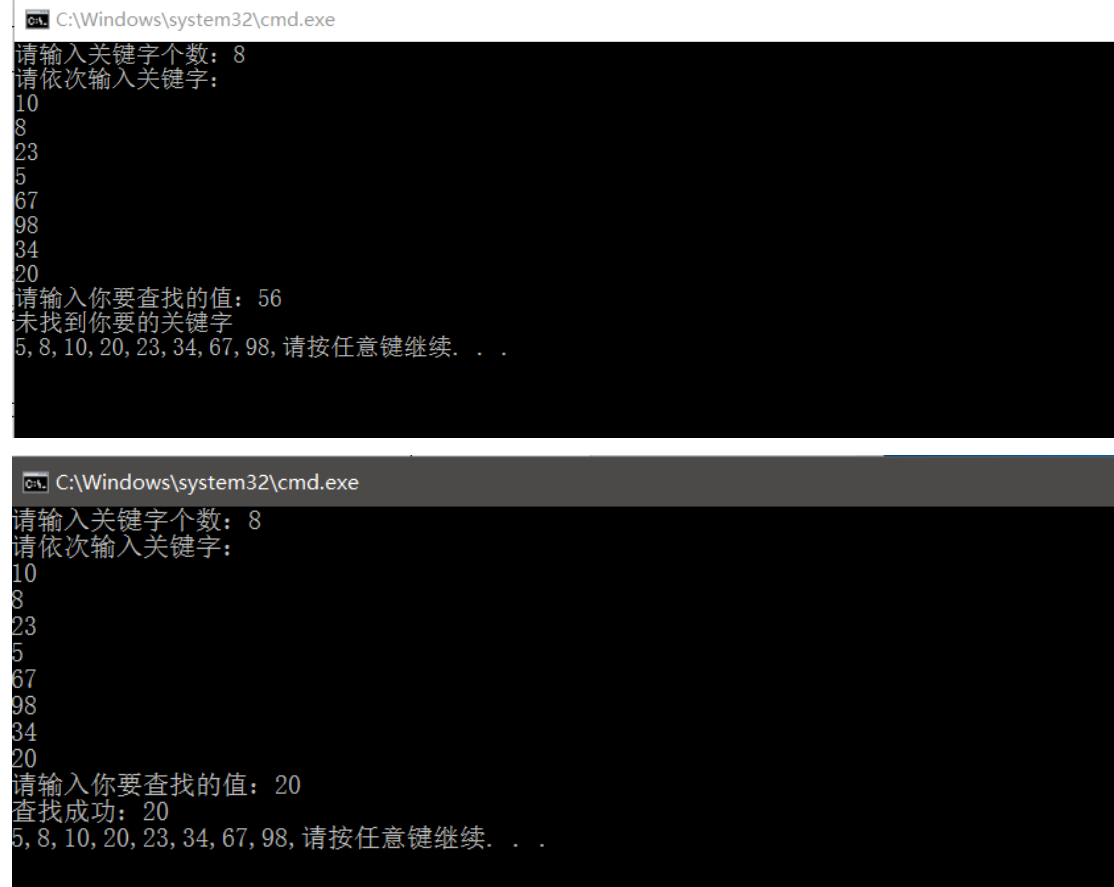
第二部分：实验调试与结果分析（可加页）

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2. 实验输入/输出数据记录



```
C:\Windows\system32\cmd.exe
请输入关键字个数: 8
请依次输入关键字:
10
8
23
5
67
98
34
20
请输入你要查找的值: 56
未找到你要的关键词
5, 8, 10, 20, 23, 34, 67, 98, 请按任意键继续. . .

C:\Windows\system32\cmd.exe
请输入关键字个数: 8
请依次输入关键字:
10
8
23
5
67
98
34
20
请输入你要查找的值: 20
查找成功: 20
5, 8, 10, 20, 23, 34, 67, 98, 请按任意键继续. . .
```

二、实验结果

经过问题分析，算法分析，程序编写，调试分析，程序最终通过运行。

三、实验小结、建议及体会

经过本次实验，我对查找一章中的动态查找有了更深的了解，强化了课堂上学到的知识，加深的了对二叉排序树算法的熟悉程度。树的结构不是一次生成的而是在查找过程中，当树中不存在关键字等于给定值得结点是在金乡插入。新插入的节点一定是叶子结点，并且是查找不成功是查找路径上访问的最后一个结点的左孩子或右孩子。二叉排序树即拥有类似折半查找的特性，又采用了链表作为存储结构。体会：编程本身就是让人快乐的，特别是解决了一个问题之后，以后要多动手不能只是空看代码。

实验课程名称： 数据结构

实验项目名称	选择排序的实现与分析			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-12-12

第一部分：实验分析与设计（可加页）

一 实验内容描述（问题域描述）

实验题 8 选择排序的实现与分析

【问题描述】给定 n 个整数，写一算法用简单选择排序算法和堆排序算法按从大到小的顺序输出前 m

名最大值。并分析算法的效率。

【测试数据】

(1) 从键盘输入 10 个整数, $m=10$

(2) 随机产生 10000 个数据, $m=10$

二 实验基本原理与设计(算法与程序设计)

算法分析:

简单选择排序算法:

一趟简单选择排序的操作为: 通过 $n-1$ 次关键字间的比较, 从 $n-i+1$ 个记录中选出关键字最小的记录, 并和第 i 个记录交换之。

int SelectMinKey(SqList &L, int i)

```
{
    int k = 0;
    compareNum[0] += L.length - i;
    for (k = i; k < L.length; k++)
    {
        compareNum[0]++; //记录的是 i 与 length 的比较
        compareNum[0]++; //下面的选择语句中的比较
        if (L.elem[i] > L.elem[k])
        {
            k = i; changeNum[0]++;
        }
    }
    return k;
}
```

void SelectSort(SqList &L)

```
{
```



```

int j = 0;
int temp = 0;
//对顺序表进行简单排序
for (int i = 0; i < L.length; ++i)
{
    compareNum[0]++; //记录的是 i 与 length 的比较
    j = SelectMinKey(L, i);
    compareNum[0]++;
    if (i != j)
    {
        temp = L.elem[i];
        L.elem[i] = L.elem[j];
        L.elem[j] = temp;
        changeNum[0] += 3; //交换+3
    }
}
}

```

堆排序算法：

若在输出堆顶的最小值后，使得剩余 $n-1$ 个元素的序列重新又建成一个堆，则得到 n 个元素的次小值。如此反复执行，便能得到一个有序序列。

```

void HeadAdjust(SqList &H, int s, int m)
{
    int rc = H.elem[s];
    for (int j = 2 * s; j <= m; j *= 2)
    {
        compareNum[1]++; //for 循环的判断
        if (j < m && (compareNum[1]++) && (H.elem[j] < H.elem[j + 1]))
            ++j;
        if (rc < H.elem[j])
        {
            compareNum[1]++; break;
        }
        H.elem[s] = H.elem[j]; s = j;
        changeNum[1]++;
    }
    H.elem[s] = rc;
    changeNum[1]++;
}

```

```

void HeadSort(SqList &H)
{
    int temp; //中间变量用于保存数值，

```

```

        for (int i = H.length / 2; i>0; --i) {
            compareNum[1]++;
            HeadAdjust(H, i, H.length); //后续的调整
        }
        for (int i = H.length; i>1; --i) {
            compareNum[1]++;
            temp = H.elem[1]; H.elem[1] = H.elem[i]; H.elem[i] = temp; //最后的一次
            记录相互交换
            changeNum[1] += 3;
            HeadAdjust(H, 1, i - 1); //第一次的调整
        }
    }
}

```

源代码：

```

#include<iostream>
#include<time.h>
using namespace std;

#define LIST_INIT_SIZE 100 //线性表存储空间的初始分配量
#define LISTINCREMENT 10 //分配增量
#define random(x) (rand()%x) //随机数

int compareNum[2]; //用来记录比较的次数
int changeNum[2]; //用来比较交换的次数

typedef struct
{
    int *elem; //存储空间基址 存储数据的类型
    int length; //当前长度
    int listsize; //当前分配的存储容量
}SqList;

void InitList_Sq(SqList &L)
{
    //构造一个空的线性表
    L.elem = (int *)malloc(LIST_INIT_SIZE * sizeof(int));
    if (!L.elem)
    {

```

```

        cout << "链表创建失败/n";
        exit(1);
    }
    L.length = 0;
    L.listsize = LIST_INIT_SIZE;
}

void InsertList_Sq(SqList &L, int i, int e)
{
    int *newbase = NULL;
    int *p, *q;
    if (i<1 || i>L.length+1)
    {
        cout << "插入点 i 值超标 错误" << endl;
        exit(1);
    }
    if (L.length >= L.listsize)
    {
        newbase = (int*)realloc(L.elem, (L.listsize +
LISTINCREMENT)*sizeof(int));

        if (!newbase)
        {
            cout << "新增空间分配错误!!/n";
            exit(1);
        }
        L.elem = newbase;//新基址
        L.listsize = L.listsize + LISTINCREMENT;//增加存储容量
    }
    q = &(L.elem[i - 1]);//插入位置
    for (p = &(L.elem[L.length - 1]); p >= q; p--)//元素右移动
    {
        *(p + 1) = *p;
    }
    *q = e;//插入 e
    ++L.length;//表增长 1
}

```

```

void PrintSqList(SqList &L,int num)
{
    int i;
    if (L.length == 0)
    {
        cout << "错误 空表\n";
        return;
    }
    for (i = 0; i < num; i++)
        cout << L.elem[i]<<" ";
    cout << endl;
}

/*****简单排序算法*****/
int SelectMinKey(SqList &L, int i)
{
    int k = 0;
    compareNum[0] += L.length - i;
    for (k = i; k < L.length; k++)
    {
        compareNum[0]++; //记录的是 i 与 length 的比较
        compareNum[0]++; //下面的选择语句中的比较
        if (L.elem[i] > L.elem[k])
        {
            k = i; compareNum[0]++;
        }
    }
    return k;
}

void SelectSort(SqList &L)
{
    int j = 0;
    int temp = 0;
    //对顺序表进行简单排序
    for (int i = 0; i < L.length; ++i)
    {
        compareNum[0]++; //记录的是 i 与 length 的比较
        j = SelectMinKey(L, i);
        compareNum[0]++;
    }
}

```

```

        if (i != j)
        {
            temp = L.elem[i];
            L.elem[i] = L.elem[j];
            L.elem[j] = temp;
            changeNum[0] += 3; //交换+3
        }
    }
}

/*****堆排序算法*****/
void HeadAdjust(SqList &H, int s, int m)
{
    int rc = H.elem[s];
    for (int j = 2 * s; j <= m; j *= 2)
    {
        compareNum[1]++; //for 循环的判断
        if (j < m && (compareNum[1]++) && (H.elem[j] < H.elem[j + 1]))
            ++j;
        if (rc < H.elem[j])
        { compareNum[1]++; break; }
        H.elem[s] = H.elem[j]; s = j;
        changeNum[1]++;
    }
    H.elem[s] = rc;
    changeNum[1]++;
}

void HeadSort(SqList &H)
{
    int temp; //中间变量用于保存数值,
    for (int i = H.length / 2; i > 0; --i) {
        compareNum[1]++;
        HeadAdjust(H, i, H.length); //后续的调整
    }
    for (int i = H.length; i > 1; --i) {
        compareNum[1]++;
        temp = H.elem[1]; H.elem[1] = H.elem[i]; H.elem[i] = temp; //最后的一次
        记录相互交换
        changeNum[1] += 3;
        HeadAdjust(H, 1, i - 1); //第一次的调整
    }
}

```

```

    }
}
int main()
{
    int na = 10;
    int nb = 10000;
    int m = 10;
    SqList La, Lb, La1, Lb1;
    InitList_Sq(La);
    InitList_Sq(Lb);
    InitList_Sq(La1);
    InitList_Sq(Lb1);

    for (int i = 1; i <= na; i++)
    {
        int temp;
        cout << "请输入第" << i << "个值=";
        cin >> temp;
        InsertList_Sq(La, i, temp);
        InsertList_Sq(La1, i, temp);
    }

    srand((int)time(0));
    //随机产生一万个 20000 以内的数
    for (int i = 1; i < nb; i++)
    {
        int temp = random(20000);
        InsertList_Sq(Lb, i, temp);
        InsertList_Sq(Lb1, i, temp);
    }

    //简单选择排序
    //PrintSqList(La, m);
    SelectSort(La);
    cout << "简单选择排序 对第一组比较了" << compareNum[0] << "次" << endl;
    cout << "简单选择排序 对第一组交换了" << changeNum[0] << "次" << endl;
    PrintSqList(La, m);
    cout << endl;
}

```

```

//PrintSqList(Lb, m);
SelectSort(Lb);
cout << "简单选择排序 对第二组比较了" << compareNum[0] << "次" << endl;
cout << "简单选择排序 对第二组交换了" << changeNum[0] << "次" << endl;
PrintSqList(Lb, m);
cout << endl;

//堆排序
//PrintSqList(La1, m);
HeadSort(La1);
cout << "堆排序 对第一组比较了" << compareNum[1] << "次" << endl;
cout << "堆排序 对第一组交换了" << changeNum[1] << "次" << endl;
PrintSqList(La, m);
cout << endl;

//PrintSqList(Lb1, m);
HeadSort(Lb1);
cout << "堆排序 对第二组比较了" << compareNum[1] << "次" << endl;
cout << "堆排序 对第二组交换了" << changeNum[1] << "次" << endl;
PrintSqList(Lb, m);
cout << endl;

return 0;
}

```

三、主要仪器设备及耗材

1. 实验设备

PC 机

2. 开发环境

VS 2015 Pro

第二部分：实验调试与结果分析（可加页）

四、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2. 实验输入/输出数据记录

```
C:\Windows\system32\cmd.exe
请输入第1个值=48
请输入第2个值=15
请输入第3个值=26
请输入第4个值=59
请输入第5个值=79
请输入第6个值=46
请输入第7个值=23
请输入第8个值=1
请输入第9个值=2
请输入第10个值=3
简单选择排序 对第一组比较了185次
简单选择排序 对第一组交换了28次
79, 59, 48, 46, 26, 23, 15, 3, 2, 1,

简单选择排序 对第二组比较了150005183次
简单选择排序 对第二组交换了106469次
19992, 19990, 19984, 19983, 19978, 19977, 19974, 19974, 19971, 19970,

堆排序 对第一组比较了51次
堆排序 对第一组交换了50次
79, 59, 48, 46, 26, 23, 15, 3, 2, 1,

堆排序 对第二组比较了63796次
堆排序 对第二组交换了48784次
19992, 19990, 19984, 19983, 19978, 19977, 19974, 19974, 19971, 19970,

请按任意键继续. . .
```

```
C:\Windows\system32\cmd.exe
请输入第1个值=1
请输入第2个值=2
请输入第3个值=3
请输入第4个值=4
请输入第5个值=5
请输入第6个值=6
请输入第7个值=7
请输入第8个值=8
请输入第9个值=9
请输入第10个值=11
简单选择排序 对第一组比较了185次
简单选择排序 对第一组交换了40次
11, 9, 8, 7, 6, 5, 4, 3, 2, 1,

简单选择排序 对第二组比较了150005183次
简单选择排序 对第二组交换了107650次
19998, 19996, 19988, 19986, 19980, 19980, 19979, 19978, 19975, 19975,

堆排序 对第一组比较了53次
堆排序 对第一组交换了50次
11, 9, 8, 7, 6, 5, 4, 3, 2, 1,

堆排序 对第二组比较了63816次
堆排序 对第二组交换了48840次
19998, 19996, 19988, 19986, 19980, 19980, 19979, 19978, 19975, 19975,

请按任意键继续. . .
```

五、实验结果小结体会

通过本次实验，对两种常见的排序算法有了更为深刻的认识，简单排序所需进行记录移动的操作次数较少，然而无论记录的初始排列如何，所需及拟向的关键字间的比较次数相同，均为 $n(n-1)/2$ ，时间复杂度为 $O(n^2)$ 。堆排序算法不适合记录较少的文件，对 n 很大的文件比较适用，其运行时间主要耗费在建初始

堆和调整建新堆时进行的反复筛选上。代码中实现了对两种算法比较和交换次数的打印，据此判断两种排序算法的优劣。

