重庆交通大学信息科学与工程学院

《数据结构 A》实验报告

实验名称	搜索与排序实验
课程名称	数据结构 A
专业班级	曙光 2101 班
学　　号	632107060510
姓　　名	王靖
指导教师	鲁云平

2022 年 12 月

《数据结构 A》综合性实验评分标准

评分等级	综合性实验评分标准
优	程序演示完全正确，界面美观，能正确回答 90%及以上的问题；报告规范，分析清楚，严格按照要求条目书写，阐述清楚。能针对综合性题目进行分析，并设计合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
良	按要求完成 80%及以上功能，界面尚可，能正确回答 80%及以上的问题；报告规范，分析清楚，个别条目书写不完全符合要求，阐述基本清楚。能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
中	按要求完成 70%及以上功能，能回答 70%及以上的问题；报告基本规范，分析基本清楚，存在 30%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
及格	按要求完成 60%及以上功能，能回答老师多数问题；报告基本规范，存在 40%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析和设计较合适的解决方案，并能使用合适的编程平台进行编程、调试、测试和实现，成功调试功能达 60%以上。
不及格	存在 40%以上功能未完成或抄袭；报告不规范或存在 40%以上条目书写不完全符合要求。无法采用正确的方法完成项目分析和解决方案设计或成功调试测试功能超过 40%未完成。

教师签名	
综合性实验成绩	

说明:

1、综合性实验报告总体要求为格式规范，内容丰富、全面、深入；严禁大量拷贝。

2、请提交 word 文档和 PDF 文档，文件命名格式：学号-姓名。

3、报告主要内容参考下页示例

目录

一、实验目的

1. 搜索

(1) 有序顺序表的折半搜索

I. 生成 10000 个无重复的随机数据的顺序表

II. 对顺序表进行排序

III. 利用折半搜索并计算平均搜索长度

(2) 二叉搜索树的查找

I. 生成 10000 个无重复的随机数据的二叉搜索树

II. 实现二叉搜索树的插入、删除

III. 实现二叉搜索树的查找并计算平均搜索长度

2. 排序

(1) 生成 100000 个随机数据组成的数组

(2) 设计插入排序、冒泡排序、快速排序、归并排序算法对数组进行排序

(3) 分析比较不同排序算法的性能

三、系统分析

（1）数据方面：标准类型数据

（2）功能方面：

I. 生成随机数据

II. 实现顺序表和二叉搜索树

III. 实现顺序表的二分查找

IV. 实现四种内排序算法

四、系统设计

（1）数据结构的设计

- 顺序表类

```
class SQList
{
private:
    int* data;
    int num;
public:
    SQList();
    SQList(int p[],int n);
    ~SQList();
    // 快速排序
    void quick_sort(int l, int r);
    // 二分查找
```

```
    int find(int x, int& cnt);
    int getNum();
    void print();
};
```

- 二叉树节点类

```
struct TreeNode
{
    int data; // 数据
    TreeNode* left; // 左指针
    TreeNode* right;// 右指针
    TreeNode();
    //带参构造函数
    TreeNode(int data);
    //析构函数
    ~TreeNode();
};
```

- 二叉树类

```
class Tree
{
private:
    TreeNode* root;
    int numOfNode;
public:
    Tree();
    Tree(int p[], int n);
    //析构函数
    ~Tree();
    // 插入节点
    TreeNode* insert(TreeNode* node, int x);
    // 删除节点
    TreeNode* remove(TreeNode* node, int x);
    // 找最小元素
    TreeNode* findMin(TreeNode* node);
    // 找最大元素
    TreeNode* findMax(TreeNode* node);
    //查找
    TreeNode* find(TreeNode* node, int x, int& cnt);
    //获取根节点
    TreeNode* getRoot();
    //获取节点个数
    int getNumOfNode();
    //前序遍历
    void preOrder(TreeNode* node);
    //中序遍历
```

```
    void inOrder(TreeNode* node);
    //后序遍历
    void postOrder(TreeNode* node);
    //层序遍历
    void levelOrder();
};
```

（2）基本操作的设计

- 顺序表的快速排序

快速排序采用分治的思想不断划分左右部分进行排序，选定一基准数，通过交换使得基准数左边的值都小于基准数，右边的值都大于基准数。通过不断分块排序使得最终序列有序。

- 顺序表的二分查找并计算平均长度

在顺序表已经有序的情况下，从中间位置开始查找需要的值，通过比较每次可缩小一半查找范围。直到最后仅剩一个数值

- 二叉搜索树的建立

二叉搜索树的建立可以看作是向一棵空树中插入节点

- 二叉搜索树的插入

插入过程可看作是一种递归，从根节点开始依次比较数值的大小，寻找插入的位置。直到寻找到合适的插入位置。

- 二叉搜索树删除节点

删除节点时要考虑该节点是否为叶节点，若是叶节点可以直接删除，否则需要进行一些调整。

- 二叉搜索树的查找并计算查找长度

查找过程也可看作是一种递归，从根节点开始逐次向下查找，直到找到目标值。

五、系统实现

- 顺序表的快速排序

```
void SQList::quick_sort(int l, int r)
{
    if (l >= r) return;
    int x = data[l + r >> 1];
    int i = l - 1, j = r + 1;
    while (i < j)
    {
        do i++; while (data[i] < x);
        do j--; while (data[j] > x);
        if (i < j)
        {
```

```
            int temp = data[i];
            data[i] = data[j];
            data[j] = temp;
        }
    }
    quick_sort(l, j);
    quick_sort(j + 1, r);
}
```

- 顺序表的二分查找并计算平均长度

```
int SQList::find(int x, int& cnt)
{
    int len = 0;
    int l = 0, r = num - 1;
    int mid = 0;
    while (l < r)
    {
        mid = l + r >> 1;
        len++;
        if (data[mid] > x) r = mid - 1;
        else if (data[mid] < x) l = mid + 1;
        else break;
    }
    cnt = len;
    if (data[mid] == x) return mid;
    else return -1;
}
```

- 二叉搜索树的建立

```
Tree::Tree(int p[], int n)
{
    root = NULL;
    numOfNode = 0;
    for (int i = 0; i < n; i++)
    {
        root = insert(this->root, p[i]);
    }
}
```

- 二叉搜索树的插入

```
TreeNode* Tree::insert(TreeNode* node, int x)
{
    if (node == NULL) node = new TreeNode(x), numOfNode++;
    else
    {
        if (x < node->data) node->left = insert(node->left, x);
        else if (x > node->data) node->right = insert(node->right, x);
```

```
    }
    return node;
}
```

- 二叉搜索树删除节点

```
TreeNode* Tree::remove(TreeNode* node, int x)
{
    TreeNode* p;
    if (node == NULL) return node;
    if (x < node->data) node->left = remove(node->left, x);
    else if (x > node->data) node->right = remove(node->right, x);
    else
    {
        if (node->left != NULL && node->right != NULL)
        {
            p = findMin(node->right);
            node->data = p->data;
            node->right = remove(node->right, node->data);
        }
        else
        {
            p = node;
            if (node->left != NULL) node = node->right;
            else node = node->left;
            delete p;
        }
    }
    return node;
}
```

- 二叉搜索树的查找并计算查找长度

```
TreeNode* Tree::find(TreeNode* node, int x, int& cnt)
{
    int len = 0;
    while (node != NULL)
    {
        len++;
        if (x < node->data) node = node->left;
        else if (x > node->data) node = node->right;
        else break;
    }
    cnt = len;
    return node;
}
```

- 数组的插入排序

```
void insert_sort(int p[], int n)
```

```
{
    int temp, pos;
    for (int i = 1; i < n; i++)
    {
        temp = p[i];
        pos = i - 1;
        while (pos >= 0 && temp < p[pos])
        {
            p[pos + 1] = p[pos];
            pos--;
        }
        p[pos + 1] = temp;
    }
}
```

- 数组的冒泡排序

```
void bubble_sort(int p[], int n)
{
    for (int i = 1; i <= n - 1; i++)
    {
        for (int j = 0; j < n - i; j++)
        {
            if (p[j] > p[j + 1])
            {
                int temp = p[j];
                p[j] = p[j + 1];
                p[j + 1] = temp;
            }
        }
    }
}
```

- 数组的快速排序

```
void quick_sort(int p[], int l, int r)
{
    if (l >= r) return;
    int x = p[l + r >> 1];
    int i = l - 1, j = r + 1;
    while (i < j)
    {
        do i++; while (p[i] < x);
        do j--; while (p[j] > x);
        if (i < j)
        {
            int temp = p[i];
            p[i] = p[j];
```

```
            p[j] = temp;
        }
    }
    quick_sort(p, l, j);
    quick_sort(p, j + 1, r);
}
```

- 数组的归并排序

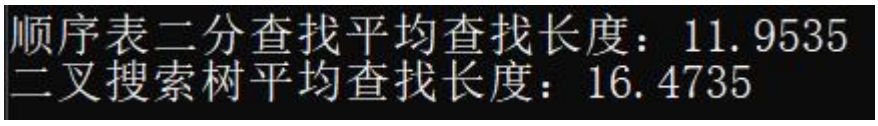
```
void merge_sort(int p[], int l, int r)
{
    if (l >= r) return;
    int mid= l + r >> 1;
    merge_sort(p, l, mid);
    merge_sort(p, mid + 1, r);

    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
    {
        if (p[i] <= p[j]) temp[k++] = p[i++];
        else temp[k++] = p[j++];
    }
    while (i <= mid) temp[k++] = p[i++];
    while (j <= r) temp[k++] = p[j++];
    for (int i = l, j = 0; i <= r; i++, j++)
    {
        p[i] = temp[j];
    }
}
```

六、系统测试与结果

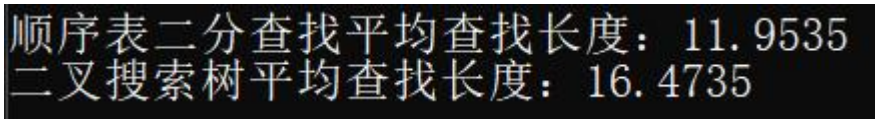
- 顺序表的二分查找并计算平均查找长度

  计算 10000 个数据的平均查找长度



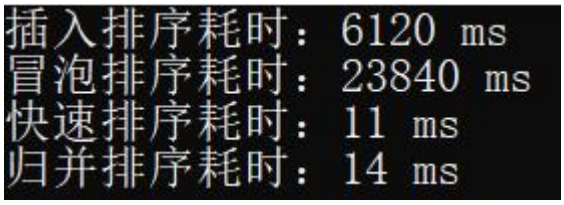
- 二叉搜索树的查找并计算平均查找长度

  计算 10000 个数据的平均查找长度



- 排序算法的比较

  比较四种排序算法对于 100000 数据的排序时间



七、实验分析

（1）算法的性能分析

- 顺序表的快速排序

快速排序的利用分治思想，每次调整的时间复杂度为$O(n)$，因此总的平均时间复杂度为$O(n*log_2n)$。

- 顺序表的二分查找并计算平均长度

二分查找每次将查找范围缩小一半，逐次缩小，因此平均时间复杂度为$O(n*log_2n)$。

- 二叉搜索树的插入

二叉搜索树插入过程最多经过的比较次数是树的高度，因此平均时间复杂度为$O(log_2n)$。

- 二叉搜索树删除节点

二叉搜索树删除节点过程最多经过的比较次数是树的高度，因此平均时间复杂度为$O(log_2n)$。

- 二叉搜索树的查找并计算查找长度

二叉搜索树查找过程最多经过的比较次数是树的高度，因此平均时间复杂度为$O(log_2n)$。

- 数组的插入排序

数组的插入排序过程中每个数将和序列中已有的数进行比较，因此平均时间复杂度为$O(n^2)$。

- 数组的冒泡排序

数组的冒泡排序过程中每相邻两个数会进行比较，直到最终序列有序，因此平均时间复杂度为$O(n^2)$。

- 数组的快速排序

快速排序的利用分治思想，每次调整的时间复杂度为$O(n)$，因此总的平均时间复杂度为$O(n*log_2n)$。

- 数组的归并排序

归并排序的利用分治思想，分割的过程时间复杂度为$O(log_2n)$，最终和为有序序列，因此总的平均时间复杂度为$O(n*log_2n)$。

（2）数据结构的分析

二叉搜索树极大提高了搜索的效率，能够将大规模数据的搜索的平均时间复杂度达到$O(log_2 n)$。但二叉搜索树的搜索性能受树的形状的影响，有时可能会使搜索过程的时间复杂度达到$O(n)$。因此二叉搜索树的改进平衡二叉树可以做到将搜索的时间复杂度严格控制在$O(log_2 n)$。

八、实验总结

这次实验成功的完成了二叉搜索树的创建以及一些操作。首先，必须要说的是，这次实验与往常的有非常大的不同，采用了大量的递归的应用，经过这次实验之后，对这种晦涩的递归，产生了更深的理解。其次是对于二叉搜索树产生了更多的可能，能不能通过一定的限制条件，来使二叉搜索树具备插入、删除等操作，又或者能让搜索更加的快等等。通过对四种排序算法的比较分析，也能更加深入理解排序算法。

总结一下，经过这次实验，第一是收获了对于递归的理解和应用，第二是对于“二叉搜索树是不是具有扩展？”的想象。为后续学习打下了良好的基础。

参考文献：

[1]李春葆.《数据结构教程（第6版）》.北京：清华大学出版社.
[2]陈　越,何钦铭.《数据结构（第2版）》.北京：高等教育出版社.
[3]殷人昆.《数据结构（用面向对象与C++语言描述）（第3版）》.北京：清华大学出版社
[4]严蔚敏.《数据结构（C语言版）》.北京.清华大学出版社.