《数据结构与算法实验》第 14 次实验

学院： 专业： 年级：
姓名： 学号 日期： 2022 年 6 月 20 日

第一部分 验证实验

一、 实验目的

1. 实现散列查找算法

二、 实验内容

1. 实现散列查找算法（参考实验书 p234）
   - 构造一个散列表。
   - 对散列表进行查找。

三、代码实现（注：实验编码格式为 UTF-8）

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

const int Max = 11;
const char Empty = '#';
int HashSearch(int ht[], int m, int k, int &j, int &count);
void HashDelete(int ht[], int m, int x);

int main()
{
    int s[9] = {47, 7, 29, 11, 16, 92, 22, 8, 3};
    int ht[Max] = {0};
    int temp, i = 0, index = 0, count = 0;
    for ( i = 0; i < 9; i++)
    {
        HashSearch(ht, Max, s[i], index, count);
        cout<<"the index of the element "<<s[i]<<" is "<<index<<endl;
    }
```

```
    cout<<"the elements in the Hash: "<<endl;
    for ( i = 0; i < Max; i++)
        cout<<ht[i]<<" ";
    cout<<endl;
    srand(time(NULL));
    temp = 3;
    HashSearch(ht, Max, temp, index, count);
    cout<<"the index of the element "<<temp<<" is "<<index<<endl;
    cout<<"the amount of comparasion: "<<count<<endl;
    return 0;
}

int HashSearch(int ht[], int m, int k, int &j, int &count)
{
    int i;
    j = k % m;
    count = 1;
    if (ht[j] == k)  return 1;
    else if (ht[j] == 0)
        {
            ht[j] = k;
            return 0;
        }
    i = (j + 1) % m;
    while (ht[i] != 0 && i != j)
    {
        count++;
        if (ht[i] == k)
        {
            j = i;
            return 1;
        }
        else i = (i + 1) % m;
    }
    if (i == j)
    {
        cout<<"overflow!"<<endl;
        return 0;
    }
    else
    {
        ht[i] = k;
        j = i;
        return 0;
```

```
    }
}
```

四、运行与测试

```
the elements in the Hash:
11 22 0 47 92 16 3 7 29 8 0
the index of the element 38 is 10
the amount of comparasion: 5
PS F:\DataStructure>
```

另外，由于散列查找在闭散列表删除中有直接应用，故代码文件中将这两道题放在同一个 cpp 文件里。

第二部分 设计实验

一、实验目的

通过 2 个实验，补充对散列查找的技巧掌握，实现散列的删除和查找性能比较。

通过 9 个实验，掌握排序相关操作。

二、实验内容

1. 闭散列表删除(T02)： 参考课本 p217

在用线性探测解决冲突的散列表中，设计算法实现闭散列表的删除操作删除某个关键码时需要保证探测序列不断开，使得后序查找能够进行。为此，可以在删除关键码时，用一个特殊符号（例如 #）代替，在查找时遇到这个特殊符号则继续执行探测操作，然后再统一将所有特殊符号删除。

2. 闭散列表和开散列表性能比较(T03)： 参考实验书 p236

- 用线性探测法处理冲突建立闭散列表。
- 用拉链法处理冲突建立开散列表。
- 设计合理的测试数据，比较二者的查找性能。

3. 插入排序算法(T05)： 参考实验书 p240

简单起见，假定待排序记录为整数，并按升序排列。直接插入排序算法和希尔排序算法参见主教材 8.3 节。为了避免每次运行程序都从键盘上输入数据，可以设计一个函数自动生成待排序记录。

4. 交换排序算法(T06)： 参考实验书 p241

简单起见，假定待排序记录为整数，并按升序排列。起泡排序算法和快速排序算法参见主教材 8.3 节。为了避免每次运行程序都从键盘上输入数据，可以设计一个函数自动生成待排序记录。

5. 选择排序算法(T07)： 参考实验书 p244

简单起见，假定待排序记录为整数，并按升序排列。简单选择排序算法和堆排序算法参见主教材 8.4 节。为了避免每次运行程序都从键盘上输入数据，可以设计一个函数自动生成待排序记录。

6. 改进插入排序(T08)： 参考课本 p252

直接插入排序中寻找插入位置的操作可以通过折半查找来实现。据此写一个改进的插入排序的算法。

插入排序的基本思想是：每趟从无序区中取出一个元素，再按键值大小插入到有序区中。对于有序区，当然可以采用折半查找来确定插入位置。

7-8. 单链表直接插入排序/简单选择排序(T09-10)： 参考课本 p252

设待排序的记录序列用单链表作存储结构，写出直接插入排序算法和简单选择算法。

直接插入：本算法采用的存储结构是带头结点的单链表。首先找到元素的插入位置，然后把元素从链表中原位置删除，再插入到相应的位置处。函数作为成员函数定义在单链表的类中。

简单选择：在找到最小元素后，将最小元素与无序序列的第一个元素相交换，而不交换结点，这样可以避免指针的修改。函数作为成员函数定义在单链表的类中。

9. 关键码查找(T11)： 参考课本 p252

对给定的序号，要求在无序记录 A[1]~A[n]中找到按关键码从小到大排在第 j 位上的记录，利用快速排序的划分思想设计算法实现上述查找。

10. 非递归快速排序(T12)： 参考课本 p252

先调用划分函数 Quickpass，以确定中间位置，然后再借助栈分别对中间元素的左、右两边的区域进行快速排序。

11. 正负分离(T13)： 参考课本 p253

一个线性表中的元素为正整数或负整数，设计算法将正整数和负整数分开，使线性表的前一半为负整数，后一半为正整数。不要求对这些元素排序，但要求尽量减少比较次数。

本题的基本思想是：先设置好上、下界和轴值，然后分别从线性表两端查找正数和负数，找到后进行交换，直到上下界相遇。

12. 堆调整(T14)： 参考课本 p253

增加一个元素应从叶子向根方向调整，假设调整为大根堆。

13. 归并序列(T15)： 参考课本 p253

采用二路归并排序中一次归并的思想，设三个参数 i,j,k 分别指向两个待归并的有序序列和最终有序序列的当前记录，初始时 i,j 分别指向两个有序序列的第一个记录，即 i=0,j=0,k 指向存放归并结果的位置，即 k=0。然后，比较 i 和 j 所指记录的关键码，取出较小者作为归并结果存人 k 所指位置，直至两个有序序列之一的所有记录都取完，再将另一个有序序列的剩余记录顺序送到归并后的有序序列中。

14. 双向起泡排序(T16)： 参考实验书 p248

对一组数据进行双向起泡排序（假定按升序排列）。

- 设计双向起泡排序算法;
- 将双向起泡排序算法的时间性能与起泡排序算法的时间性能进行比较。

双向起泡排序的基本思想是：从两端（奇数趟排序从后向前，偶数趟排序从前向后）两两比较相邻记录，如果反序则交换，直到没有反序的记录为止。

三、设计编码（注：实验编码格式为 UTF-8/GB2312）

1. 闭散列表删除(TO2)：参考课本 p217

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

const int Max = 11;
const char Empty = '#';
int HashSearch(int ht[], int m, int k, int &j, int &count);
void HashDelete(int ht[], int m, int x);

int main()
{
    int s[9] = {47, 7, 29, 11, 16, 92, 22, 8, 3};
    int ht[Max] = {0};
    int temp, i = 0, index = 0, count = 0;
    for ( i = 0; i < 9; i++)
    {
        HashSearch(ht, Max, s[i], index, count);
        cout<<"the index of the element "<<s[i]<<" is "<<index<<endl;
    }
    cout<<"the elements in the Hash: "<<endl;
    for ( i = 0; i < Max; i++)
        cout<<ht[i]<<" ";
    cout<<endl;
    srand(time(NULL));
    temp = 3;
    HashSearch(ht, Max, temp, index, count);
    cout<<"the index of the element "<<temp<<" is "<<index<<endl;
    cout<<"the amount of comparasion: "<<count<<endl;
    cout<<"Delete the element "<<temp<<endl;
    HashDelete(ht, Max, temp);
    cout<<"the elements in the Hash: "<<endl;
    for ( i = 0; i < Max; i++)
        cout<<ht[i]<<" ";
    cout<<endl;
    return 0;
}

int HashSearch(int ht[], int m, int k, int &j, int &count)
```

```
{
    int i;
    j = k % m;
    count = 1;
    if (ht[j] == k)  return 1;
    else if (ht[j] == 0)
        {
            ht[j] = k;
            return 0;
        }
    i = (j + 1) % m;
    while (ht[i] != 0 && i != j)
    {
        count++;
        if (ht[i] == k)
        {
            j = i;
            return 1;
        }
        else i = (i + 1) % m;
    }
    if (i == j)
    {
        cout<<"overflow!"<<endl;
        return 0;
    }
    else
    {
        ht[i] = k;
        j = i;
        return 0;
    }
}

void HashDelete(int ht[], int m, int x)
{
    int i, j = x % m, count = 0;
    if (ht[j] != x) i = (j + 1) % m;
    while (ht[i] != Empty && i != j)
    {
        if (ht[i] == x) break;
        else i = (i + 1) % m;
    }
    if (i == j)  cout<<"error! x no record"<<endl;
```

```
    else ht[j] = '#';
}
```

2. 闭散列表和开散列表性能比较(T03)：参考实验书 p236

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

const int Max = 11;

struct Node
{
    int data;
    Node * next;
};

const char Empty = '#';
int HashSearch(int ht[], int m, int k, int &j, int &count);
Node * HashSearch2(Node * ht2[], int m, int k,  int &count);

int main()
{
    int s[9] = {47, 7, 29, 11, 16, 92, 22, 8, 3};
    int ht[Max] = {0};
    Node * ht2[Max];
    int temp, i = 0, index = 0, count = 0;
    for ( i = 0; i < 9; i++)
    {
        HashSearch(ht, Max, s[i], index, count);
        cout<<"the index of the element "<<s[i]<<" is "<<index<<endl;
    }
    cout<<"the elements in the Hash: "<<endl;
    for ( i = 0; i < Max; i++)
        cout<<ht[i]<<" ";
    cout<<endl;

    cout<<"enter the element to search: ";
    cin>>temp;
    int p = HashSearch(ht, Max, temp, index, count);
    if (p == 1)
    {
        cout<<"success! the index of the element "<<temp<<" is "<<index<<endl;
```

```
        cout<<"the amount of comparasion (in closed hashing) is "<<count<<endl;
    }
    else  cout<<"fail! "<<endl;

    cout<<"----------------"<<endl;
    int count2 = 0;
    for ( i = 0; i < Max; i++)
        ht2[i] = NULL;
    for ( i = 0; i < 9; i++)
        Node * p = HashSearch2(ht2, Max, s[i], count2);
    Node *p2 = HashSearch2(ht2, Max, temp, count);
    if (p != NULL)
        cout<<"success! the amount of comparasion (in open hashing) is
"<<count<<endl;
    else  cout<<"fail! "<<endl;

    return 0;
}

int HashSearch(int ht[], int m, int k, int &j, int &count)
{
    int i;
    j = k % m;
    count = 1;
    if (ht[j] == k)  return 1;
    else if (ht[j] == 0)
        {
            ht[j] = k;
            return 0;
        }
    i = (j + 1) % m;
    while (ht[i] != 0 && i != j)
    {
        count++;
        if (ht[i] == k)
        {
            j = i;
            return 1;
        }
        else i = (i + 1) % m;
    }
    if (i == j)
    {
        cout<<"overflow!"<<endl;
```

```
        return 0;
    }
    else
    {
        ht[i] = k;
        j = i;
        return 0;
    }
}

Node * HashSearch2(Node * ht2[], int m, int k,  int &count)
{
    int j = k % m;
    Node * p = ht2[j];
    count = 1;
    while (p != NULL && p -> data != k)
    {
        p = p -> next;
        count++;
    }
    if (p != NULL)
    {
        if (p -> data == k)  return p;
        else
        {
            Node * q = new Node;
            q -> data = k;
            q -> next = ht2[j];
            ht2[j] = q;
        }
    }
    if (p == NULL)
    {
        Node * q = new Node;
        q -> data = k;
        q -> next = ht2[j];
        ht2[j] = q;
    }
}
```

3. **插入排序算法(T05)：** 参考实验书 p240

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

```cpp
#include <time.h>
using namespace std;

const int Max = 10;
void Creat(int r[], int n);
void InsertSort(int r[], int n);
void ShellSort(int r[], int n);

int main()
{
    int a[Max + 1] = {0}, b[Max + 1] = {0};
    int i = 0;
    Creat(a, Max);
    for ( i = 1; i <= Max; i++)
        b[i] = a[i];
    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    InsertSort(a, Max);
    cout<<"after perform straight insertion: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;

    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    ShellSort(b, Max);
    cout<<"after perform Shell insertion: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}
```

```
void InsertSort(int r[], int n)
{
    for (int i = 2; i <= n; i++)
    {
        r[0] = r[i];
        int j;
        for ( j = i - 1; r[0] < r[j]; j--)
            r[j + 1] = r[j];
        r[j + 1] = r[0];
    }
}

void ShellSort(int r[], int n)
{
    for (int d = n/2; d >= 1; d = d/2)
    {
        for (int i = d + 1; i <= n; i++)
        {
            r[0] = r[i];
            int j;
            for ( j = i - d; j > 0 && r[0] < r[j]; j = j - d)
                r[j + d] = r[j];
            r [j + d] = r[0];
        }
    }
}
```

4. **交换排序算法(T06)：** 参考实验书 p241

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

const int Max = 10;
void Creat(int r[], int n);
void BubbleSort(int r[], int n);
int Partition(int r[], int first, int end);
void QuickSort(int r[], int first, int end);

int main()
{
    int a[Max + 1] = {0}, b[Max + 1] = {0};
```

```
    int i = 0;
    Creat(a, Max);
    for ( i = 1; i <= Max; i++)
        b[i] = a[i];
    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    BubbleSort(a, Max);
    cout<<"after perform BubbleSort: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;

    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    QuickSort(b, 1, Max);
    cout<<"after perform QuickSort: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

void BubbleSort(int r[], int n)
{
    int exchange = n, bound = n;
    while (exchange != 0)
    {
        bound = exchange;
        exchange = 0;
        for (int j = 0; j < bound; j++)
            if (r[j] > r[j + 1])
            {
```

```
                r[0] = r[j];
                r[j] = r[j + 1];
                r[j + 1] = r[0];
                exchange = j;
            }
    }
}

int Partition(int r[], int first, int end)
{
    int i = first, j = end;
    while (i < j)
    {
        while (i < j && r[i] <= r[j])  j--;
        if (i < j)
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            i++;
        }
        while (i < j && r[i] <= r[j])  i++;
        if (i < j)
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            j--;
        }
    }
    return i;
}

void QuickSort(int r[], int first, int end)
{
    if (first < end)
    {
        int pivot = Partition(r, first, end);
        QuickSort(r, first, pivot - 1);
        QuickSort(r, pivot + 1, end);
    }
}
```

5. 选择排序算法(T07)：参考实验书 p244

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

const int Max = 10;
void Creat(int r[], int n);
void SelectSort(int r[], int n);
void Sift(int r[], int k, int m);
void HeapSort(int r[], int n);

int main()
{
    int a[Max + 1] = {0}, b[Max + 1] = {0};
    int i = 0;
    Creat(a, Max);
    for ( i = 1; i <= Max; i++)
        b[i] = a[i];
    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    SelectSort(a, Max);
    cout<<"after perform straight selection: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;

    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    HeapSort(b, Max);
    cout<<"after perform Heapsort: ";
    for ( i = 1; i <= Max ; i++)
        cout<<b[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
```

```
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

void SelectSort(int r[], int n)
{
    for (int i = 1; i < n; i++)
    {
        int index = i;
        for (int j = i + 1; j <= n; j++)
            if (r[j] < r[index])  index = j;
        if (index != i)
        {
            r[0] = r[i];
            r[i] = r[index];
            r[index] = r[0];
        }
    }
}

void Sift(int r[], int k, int m)
{
    int i = k, j = 2 * i;
    while (j <= m)
    {
        if (j < m && r[j] < r[j + 1])  j++;
        if (r[i] > r[j])  break;
        else
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            i = j;
            j = 2 * i;
        }
    }
}

void HeapSort(int r[], int n)
{
    int i = 0;
    for ( i = n/2; i >= 1; i--)
        Sift(r, i, n);
```

```
    for ( i = 1; i < n; i++)
    {
        r[0] = r[1];
        r[1] = r[n - i + 1];
        r[n - i + 1] = r[0];
        Sift(r, 1, n-i);
    }
}
```

6. 改进插入排序(T08)：参考课本 p252

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

const int Max = 10;
void Creat(int r[], int n);
void StraightSort(int r[], int n);

int main()
{
    int a[Max + 1] = {0}, b[Max + 1] = {0};
    int i = 0;
    Creat(a, Max);
    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    StraightSort(a, Max);
    cout<<"after perform binary-straight insertion: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}
```

```
void StraightSort(int r[], int n)
{
    int j;
    for (int i = 2; i <= n; i++)
    {
        r[0] = r[i];
        int low = 1, high = i - 1, flag = 1, mid;
        while (low <= high && flag)
        {
            mid = (low + high)/2;
            if (r[0] < r[mid]) high = mid - 1;
            else if (r[0] > r[mid]) low = mid + 1;
                else flag = 0;
        }
        for ( j = i - 1; j >= low; j--)
            r[j + 1] = r[j];
        r[low] = r[0];
    }
}
```

7-8. 单链表直接插入排序/简单选择排序(T09-10)：参考课本 p252

(1)LinkList.h

```
#ifndef LinkList_H
#define LinkList_H

struct Node
{
    int data;
    Node * next;
};

class LinkList
{
    public:
        LinkList();
        LinkList(int a[], int n);
        ~LinkList();
        void PrintList();
        Node * getfirst();
        friend void StraightSort(LinkList &A);
        friend void SelectSort(LinkList &A);
    private:
        Node * first;
```

```
};

#endif
```

(2)LinkList.cpp

```
#include <iostream>
#include "LinkList.h"
using namespace std;

LinkList::LinkList()
{
    first = new Node;
    first -> next = NULL;
}

LinkList::LinkList(int a[], int n)
{
    Node * r, * s;
    first = new Node;
    r = first;
    for (int i = 0; i < n; i++)
    {
        s = new Node;
        s -> data = a[i];
        r -> next = s;
        r = s;
    }
    r -> next = NULL;
}

LinkList::~LinkList()
{
    Node * q = NULL;
    while (first != NULL)
    {
        q = first;
        first = first -> next;
        delete q;
    }
}

void LinkList::PrintList()
{
```

```
    Node * p = first -> next;
    while (p != NULL)
    {
        cout<<p -> data<<" ";
        p = p -> next;
    }
    cout<<endl;
}

Node * LinkList::getfirst()
{
    return first;
}

void StraightSort(LinkList &A)
{
    Node* pre = A.first, * p = A.first->next, *q = p->next,*u;
    while (q != NULL)
    {
        while (q != p)
        {
            while (p->data < q->data)
            {
                pre = p;
                p = p->next;
            }
            if (p != q)
            {
                u = q->next;
                pre->next = q;
                q->next = p;
                q = u;
            }
            else q = p->next;
        }
        pre = A.first;
        p = A.first->next;
    }
}

void SelectSort(LinkList& A)
{
    Node*p = A.first->next,*s,*q;
    int exchange;
```

```
    while (p != NULL)
    {
        s = p;
        q = p->next;
        while (q != NULL)
        {
            if (q->data < s->data)s = q;
            q = q->next;
        }
        if (p != s)
        {
            exchange = p->data;
            p->data = q->data;
            q->data = exchange;
        }
        p = p->next;
    }
}
```

(3)LinkList_main.cpp

```
#include<iostream>
#include"LinkList.cpp"
using namespace std;

int main( )
{
    int a[6] = { 7, 4, 23, 9, 8, 5}, b[6], i;
    for (i = 0;i < 6;i++)
        b[i] = a[i];
    cout<<"for unordered list: ";
    for (i = 0; i < 6; i++)
        cout<<a[i]<< " ";
    cout<<endl;
    LinkList L1(a, 6);
    LinkList L2(b, 6);
    L1.PrintList();
    cout<<"after perform StraightSort: "<<endl;
    StraightSort(L1);
    L1.PrintList();
    cout<<"after perform SelectSort: "<<endl;
    SelectSort(L2);
    L2.PrintList();
    return 0;
```

```
}
```

9. 关键码查找(T11)：参考课本 p252

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

const int Max = 10;
void Creat(int r[], int n);
int Search(int r[], int n, int j);
int Devo(int r[], int low, int high);

int main()
{
    int a[Max + 1] = {0};
    int i = 0;
    Creat(a, Max);
    cout<<"for unordered list: "<<endl;
    for (i = 1; i <= Max; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    cout<<"enter j: ";
    int j;cin>>j;
    cout<<"sort from small to large, the data with the index "<<j<<" is ";
    cout<<Search(a, Max, j);
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for (i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

int Search(int r[], int n, int j)
{
    int s = 1, t = n;
    int k = Devo(r, s, t);
    while (k != j)
    {
```

```
        if (k < j)  k = Devo(r, k + 1, t);
        else  k = Devo(r, s, k - 1);
    }
    return r[j];
}

int Devo(int r[], int low, int high)
{
    int i = low, j = high, x = r[low];
    while (i < j)
    {
        while (r[j] >= x && i < j)  j--;
        if (i < j)
        {
            r[i] = r[j];
            i++;
        }
        while (r[i] < x && i < j)  i++;
        if (i < j)
        {
            r[j] = r[i];
            j--;
        }
    }
    r[i] = x;
    return i;
}
```

10. 非递归快速排序(T12)：参考课本 p252

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

const int Max = 10;
void Creat(int r[], int n);
int QuickPass(int r[], int first, int end);
void QuickSort(int r[], int n);
void show(int r[], int n);

int main()
{
    int a[Max + 1] = { 0, 2, 8, 1, 3, 7, 10, 6, 4, 5, 10};
```

```
    int i = 0;
    Creat(a, Max);
    cout<<"for unordered list: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    QuickSort(a, Max);
    cout<<"after perform QuickSort: ";
    for ( i = 1; i <= Max ; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

int QuickPass(int r[], int first, int end)
{
    int i = first, j = end;
    while (i < j)
    {
        while (i < j && r[i] <= r[j])
            j--;
        if (i < j)
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            i++;
        }
        while (i < j && r[i] <= r[j])
            i++;
        if (i < j)
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            j--;
```

```
        }
    }
    return i;
}

void QuickSort(int r[], int n)
{
    int top = -1, S[Max + 1];
    int low = 1, high = n, i, thigh = n;
    while (low < high || top != -1)
    {
        while (low < high)
        {
            i = QuickPass(r, low, high);
            //cout<<"low: "<<low<<", high: "<<high<<", i: "<<i<<endl;
            ++top;
            S[top] = i;
            high = i - 1;
        }
        if (top != -1)
        {
            i = S[top];
            top--;
            low = i + 1;
            high = n;
            //cout<<"low: "<<low<<", high: "<<high<<", i: "<<i<<endl;
        }
    }
}

void show(int r[], int n)
{
    for (int i = 1; i <= Max; i++)
        cout<<r[i]<<" ";
    cout<<endl;
}
```

11. 正负分离(T13)：参考课本 p253

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

```
const int Max = 10;
void Creat(int r[], int n);
int Devot(int r[], int n);

int main()
{
    int a[Max + 1] = {0};
    int i = 0;
    Creat(a, Max);
    cout<<"for unordered list: ";
    for (i = 0; i < Max; i++)
        cout<<a[i]<<"  ";
    cout<<endl;
    Devot(a, Max);
    cout<<"after perform Seperation: ";
    for (i = 0; i < Max; i++)
        cout<<a[i]<<"  ";
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for (i = 0; i < n; i++)
    {
        r[i] = 1 + rand() % 100;
        int t = rand() % 2;
        if (t == 1)
        {
            r[i] = r[i] * (-1);
        }
    }
}
int Devot(int r[], int n)
{
    int i = 0, j = n - 1, t;
    while (i < j)
    {
        while (r[j] > 0 && i < j)
        {
            j--;
        }
        while (r[i] < 0 && i < j)
```

```
        {
            i++;
        }

        if (i < j)
        {
            t = r[i];
            r[i] = r[j];
            r[j] = t;
            i++;
            j--;
        }
    }
}
```

12. 堆调整(T14)：参考课本 p253

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <stdio.h>
#include <time.h>
using namespace std;

const int Max = 8;
const int Max2 = Max + 1;

void Creat(int r[], int n);
void InsertHeap(int r[],int k);
void Sift(int r[], int k, int m);

int main()
{
    int a[Max2 + 1] = {0,26,47,26,35,19,13,7,18,0};

    int i = 0;
    Creat(a, Max2);
    for(i = Max/2; i >= 1; i--)
        Sift(a,i,Max);

    cout<<"初始的大根堆为：";
    for (i = 1; i <= Max; i++)
        cout<<a[i]<<"  ";
    cout<<endl;
```

```
    a[Max2]=a[Max2]%30;
    cout<<"插入"<<a[Max2]<<", 大根堆为: ";

    InsertHeap(a,Max);

    for (i = 1; i <= Max2; i++)
        cout<<a[i]<<"  ";
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

void InsertHeap(int r[],int k)
{
    int x = r[k+1];
    int i = k+1;
    while(i/2 > 0 && r[i/2] > x)
    {
        r[i] = r[i/2];
        i = i/2;
    }
    r[i] = x;
}

void Sift(int r[], int k, int m)
{
    int i = k, j = 2 * i;
    while (j <= m)
    {
        if (j < m && r[j] < r[j + 1])  j++;
        if (r[i] > r[j])  break;
        else
        {
            r[0] = r[i];
            r[i] = r[j];
            r[j] = r[0];
            i = j;
```

```
            j = 2 * i;
        }
    }
}
```

13. 归并序列(T15)：参考课本 p253

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

const int N = 10;
const int M = 5;
void Creat(int r[], int n);
void Union(int A[], int n, int B[], int m, int C[]);

int main()
{
    int A[N], B[M], C[N+M];
    int i = 0;
    Creat(A, N);
    cout<<"A: ";
    for (i = 0; i < N; i++)
        cout<<A[i]<<" ";
    cout<<endl;
    Creat(B, M);
    cout<<"B: ";
    for (i = 0; i <M; i++)
        cout<<B[i]<<"  ";
    cout<<endl;
    Union(A, N, B, M, C);
    cout<<"after Union, C: "<<endl;
    for (i = 0; i < M + N; i++)
        cout<<C[i]<<" ";
    cout<<endl;
    return 0;
}

void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL)+n);
    for (i = 0; i < n; i++)
```

```
    {
        if(i==0)  r[i] = 1 + rand() % 100;
        else  r[i] = r[i-1] + rand() % 20;
    }
}

void Union(int A[], int n, int B[], int m, int C[])
{
    int i = 0, j = 0, k = 0;
    while(i < n && j < m)
    {
        if(A[i]<=B[j])  C[k++] = A[i++];
        else  C[k++] = B[j++];
    }
    while(i < n)  C[k++] = A[i++];
    while(j < m)  C[k++] = B[j++];
}
```

14. 双向起泡排序(T16)：参考实验书 p248

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

const int Max = 10;
void Creat(int r[], int n);
void BiBubble(int r[], int n);

int main()
{
    int a[Max + 1], i = 0;
    Creat(a, Max);
    cout<<"for unordered list: ";
    for (i = 0; i < Max; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    BiBubble(a, Max);
    cout<<"after perform Bibubble: ";
    for (i = 0; i < Max; i++)
        cout<<a[i]<<" ";
    cout<<endl;
    return 0;
}
```

```
void Creat(int r[], int n)
{
    int i = 0;
    srand(time(NULL));
    for ( i = 1; i <= n; i++)
        r[i] = 1 + rand() % 100;
}

void BiBubble(int r[],int n)
{
    int flag = 1,i = 0, j;
    while(flag == 1)
    {
        flag = 0;
        for(j = n-i-1; j > i; j--)
        {
            if(r[j-1] > r[j])
            {
                flag = 1;
                int t = r[j];
                r[j] = r[j-1];
                r[j-1] = t;
            }
        }
        for(j = i+1; j < n-i-1; j++)
        {
            if(r[j] > r[j+1])
            {
                flag = 1;
                int t = r[j];
                r[j] = r[j+1];
                r[j+1] = t;
            }
        }
        i++;
    }
}
```

四、运行结果

1. 闭散列表删除(T02)

```
the elements in the Hash:
11 22 0 47 92 16 3 7 29 8 0
the index of the element 3 is 6
the amount of comparasion: 4
Delete the element 3
the elements in the Hash:
11 22 0 35 92 16 3 7 29 8 0
PS F:\DataStructure>
```

2. 闭散列表和开散列表性能比较(T03)

```
the index of the element 47 is 3
the index of the element 7 is 7
the index of the element 29 is 8
the index of the element 11 is 0
the index of the element 16 is 5
the index of the element 92 is 4
the index of the element 22 is 1
the index of the element 8 is 9
the index of the element 3 is 6
the elements in the Hash:
11 22 0 47 92 16 3 7 29 8 0
enter the element to search: 47
success! the index of the element 47 is 3
the amount of comparasion (in closed hashing) is 1
---------------
success! the amount of comparasion (in open hashing) is 2
PS F:\DataStructure>
```

3. 插入排序算法(T05)

```
for unordered list: 92 27 81 22 70 10 13 20 41 58
after perform straight insertion: 10 13 20 22 27 41 58 70 81 92
for unordered list: 92 27 81 22 70 10 13 20 41 58
after perform Shell insertion: 10 13 20 22 27 41 58 70 81 92
PS F:\DataStructure>
```

4. 交换排序算法(T06)

```
for unordered list: 16 82 10 100 78 39 88 38 51 69
after perform BubbleSort: 10 16 38 39 51 69 78 82 88 100
for unordered list: 16 82 10 100 78 39 88 38 51 69
after perform QuickSort: 10 16 38 39 51 69 78 82 88 100
PS F:\DataStructure>
```

5. 选择排序算法(T07)

```
for unordered list: 90 2 97 15 63 33 12 36 34 45
after perform straight selection: 2 12 15 33 34 36 45 63 90 97
for unordered list: 90 2 97 15 63 33 12 36 34 45
after perform Heapsort: 2 12 15 33 34 36 45 63 90 97
PS F:\DataStructure>
```

6. 改进插入排序(T08)

```
for unordered list: 68 27 89 23 34 94 55 15 8 2
after perform binary-straight insertion: 2 8 15 23 27 34 55 68 89 94
PS F:\DataStructure>
```

7-8. 单链表直接插入排序/简单选择排序(T09-10)

```
Microsoft Visual Studio 调试控制台
对于无序序列: 7 4 23 9 8 5
执行插入操作前数据为:
7 4 23 9 8 5
执行直接插入排序后数据为:
4 5 7 8 9 23
执行选择排序后数据为:
4 5 7 8 9 23
```

9. 关键码查找(T11)

```
for unordered list:
94 74 36 20 90 6 62 12 26 10
enter j: 5
sort from small to large, the data with the index 5 is 26
PS F:\DataStructure>
```

10. 非递归快速排序(T12)

```
for unordered list: 25 83 18 6 82 85 9 69 59 80
after perform QuickSort: 6 9 18 25 59 69 80 82 83 85
PS F:\DataStructure>
```

11. 正负分离(T13)

```
for unordered list: 63  -33  50  -42  14  25  17  -81  -21  82
after perform Seperation: -21  -33  -81  -42  14  25  17  50  63  82
PS F:\DataStructure>
```

12. 堆调整(T14)

```
初始的大根堆为: 93  53  70  51  41  43  28  50
插入21，大根堆为: 21  93  70  53  41  43  28  50  51
PS F:\DataStructure>
```

13. 归并序列(T15)

```
A: 6 8 10 29 45 47 58 63 67 80
B: 89  105  122  136  147
after Union, C:
6 8 10 29 45 47 58 63 67 80 89 105 122 136 147
PS F:\DataStructure>
```

14. 双向起泡排序(T16)

```
for unordered list: 8 50 11 53 65 63 73 79 78 90
after perform Bibubble: 8 11 50 53 63 65 73 78 79 90
PS F:\DataStructure>
```

第三部分 综合实验：个人电话号码查询

一、问题描述

人们在日常生活中经常需要查找某个人或某个单位的电话号码，本实验将实现一个简单的个人电话号码查询系统，根据用户输入的信息（例如姓名等）进行快速查询。

注：本实验在原实验要求中为第 4 题。

二、基本要求

- 在外存上，用文件保存电话号码信息。
- 在内存中，设计数据结构存储电话号码信息。
- 提供查询功能，根据姓名实现快速查询。
- 提供其他维护功能，例如插入、删除、修改等。

三、设计编码（注：实验编码格式为 UTF-8）

1. 算法设计

设计思想：

由于需要管理的电话号码信息较多，而且要在程序运行结束后仍然保存电话号码信息，所以电话号码信息采用文件的形式存放到外存中。在系统运行时，需要将电话号码信息从文件调入内存来进行查找等操作，为了接收文件中的内容，要有一个数据结构与之对应，可以设计数组来接收数据。

2. 代码实现

```
#include <iostream>
using namespace std;
#include <stdlib.h>
#include <time.h>
const int Max = 10;

struct TeleNumber
{
    string name;
    char landLine[10];
    char mobileNumber[12];
    char email[20];
} Tele[Max + 1];

int BinSearch(TeleNumber r[], int n, string find, int num[]) //从数组下标 1 开始存放待查集合
{
    int low = 1, high = n;
    int mid;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (find < r[num[mid]].name)
        {
            high = mid - 1;
        }
        else if (find > r[num[mid]].name)
        {
            low = mid + 1;
        }
        else
        {
```

```
            return mid;
        } //查找成功，返回元素序号
    }
    return 0; //查找失败，返回 0
}

void Sort(TeleNumber r[], int n, int num[])
{
    for (int i = 1; i <= n; i++)
    {
        for (int j = 2; j <= n; j++)
        {
            int tj1 = num[j - 1];
            int tj2 = num[j];

            if (r[tj1].name > r[tj2].name)
            {
                swap(num[j], num[j - 1]);
            }
        }
    }
}

int main()
{
    freopen("a.txt", "r", stdin);
    int n, l, num[Max + 1];
    cout<<"Input the number of people"<<endl;
    cin>>n;
    cout<<"Inpyt name, landline, mobile, email: "<<endl;
    for (int i = 1; i <= n; i++)
    {
        cin>>Tele[i].name>>Tele[i].landLine>>Tele[i].mobileNumber>>Tele[i].email;
        num[i] = i;
    }
    Sort(Tele, n, num);

    cout<<"Find name: "<<endl;
    string find;
    cin>>find;
    l = BinSearch(Tele, n, find, num);
    cout<<endl;
    cout<<find<<endl<<"landline: "<<Tele[num[l]].landLine<<endl;
    cout<<"mobile: "<<Tele[num[l]].mobileNumber<<endl;
```

```
    cout<<"email: "<<Tele[num[l]].email<<endl;
    return 0;
}
```

四、运行结果

```
Input the number of people
Inpyt name, landline, mobile, email:
Find name:

GGG
landline: 77777777
mobile: 1397777777
email: GGG@mail.sysu.edu.cn
PS F:\DataStructure>
```

其他维护功能，例如插入、删除、修改等可直接在 txt 文档中实现。

五、总结与心得

本次实验的数量较多，内容较为繁杂，但是实现起来多为单文件即可，而且很多代码由书上给出。其中书上“插入排序算法”的代码实现案例中出现了小错误，将最后几行的 mid 改为 low 即可正常运行。

另外，对于综合实验的“电话号码查找”，由于对引用文件的相关语句不够熟悉，无法在运行终端进行电话号码本维护功能的实现，而是要在 txt 文件中进行手动修改。在熟悉引用文件的相关语句熟悉之后应该可以解决问题。