**实验6 查找与排序**

学号：2022217587姓名： 党存远 专业：物联网工程

|  |  |  |  |
| --- | --- | --- | --- |
| **“数据结构课程实验”验收细则** | | | |
| **成绩等级** | **具体表现** | **教师评分**  **（百分制）** | |
| A（100-90] | 1）能较快完成实验；2）编码正确；3）能完整正确地回答教师提问； | □ |  |
| B（90-80] | 1）能够在规定时间内完成实验；2）编码正确；3）能正确地回答教师提问； | □ |  |
| C（80-70] | 1）能够在规定时间内完成实验； | □ |  |
| D（70-60] | 1）虽完成实验但未能在规定时间内完成验收的； | □ |  |
| E（<60） | 1）未能进行验收的； | □ |  |

**6.1 实验目的**

掌握查找、排序中的基本算法。

**6.2 实验要求**

（1）对下列数据表，分别采用二分查找算法实现查找，给出查找过程依次所比较的元素（的下标）。

第一组测试数据：

数据表为 (1,2,3,4,6,7,8,9,10,11,12,13,17,18,19,20,24,25,26,30,35,40,45,50,,100)

查找的元素分别为： 2， 30，5，33，110

第二组数据：

数据表为 (2,3,5,7,8,10,12,15,18,20,22,25,30,35,40,45,50,55,60, 80,100)

查找的元素分别为： 8，80，3，100，13，

（2）设计出在二叉排序树中插入结点的算法，在此基础上实现构建二叉排序树的算法。

测试数据：构建二叉排序树的输入序列如下：

第一组数据：

100，150，120，50，70，60，80，170，180，160，110，30，40，35，175

第二组数据：

100，70，60，80，150，120，50，160，30，40，170，180，175，35

并在二叉排序树上分别实现3个数值的查找

（3）分别实现直接插入排序、希尔排序、冒泡排序、快速排序算法；

测试数据：随机生成5组整数数据，长度1000；

要求：各个算法运行5次以上，求取平均排序时长，进行排序时长对比；

（4）按链式基数排序的方法实现对整数表的排序。

测试数据如下：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）

（5）完成堆排序。

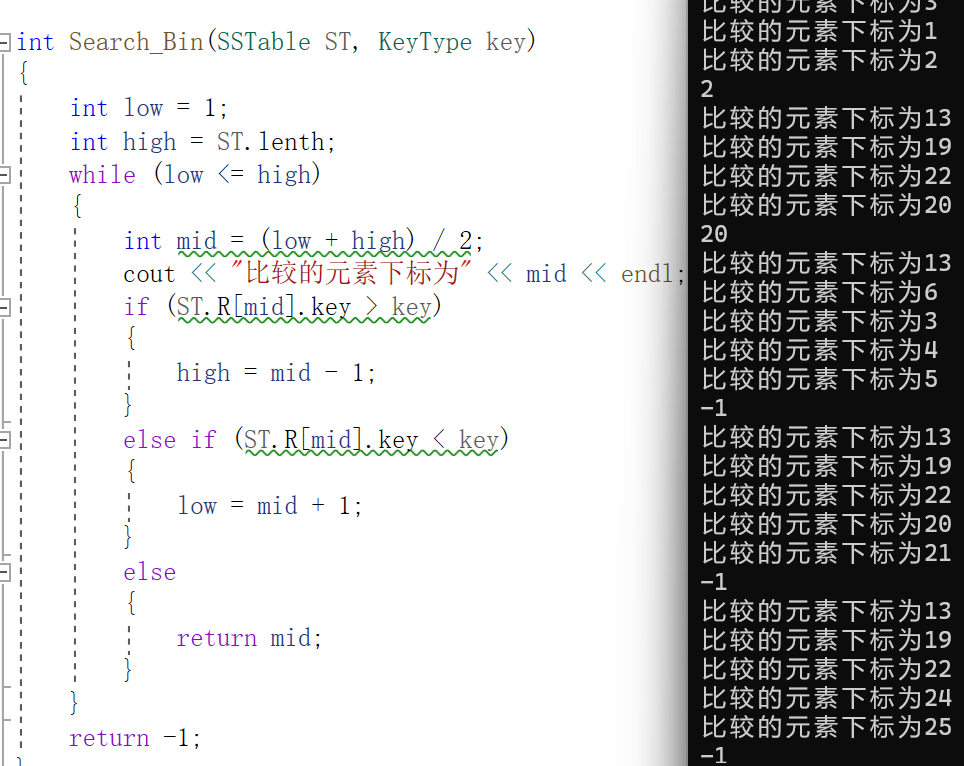
测试数据：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）

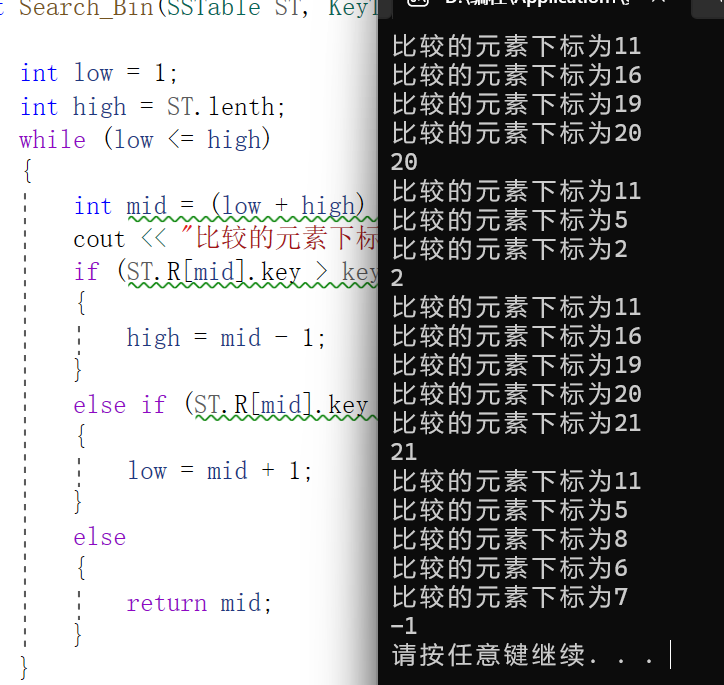
**6.3 实验数据要求**

无。

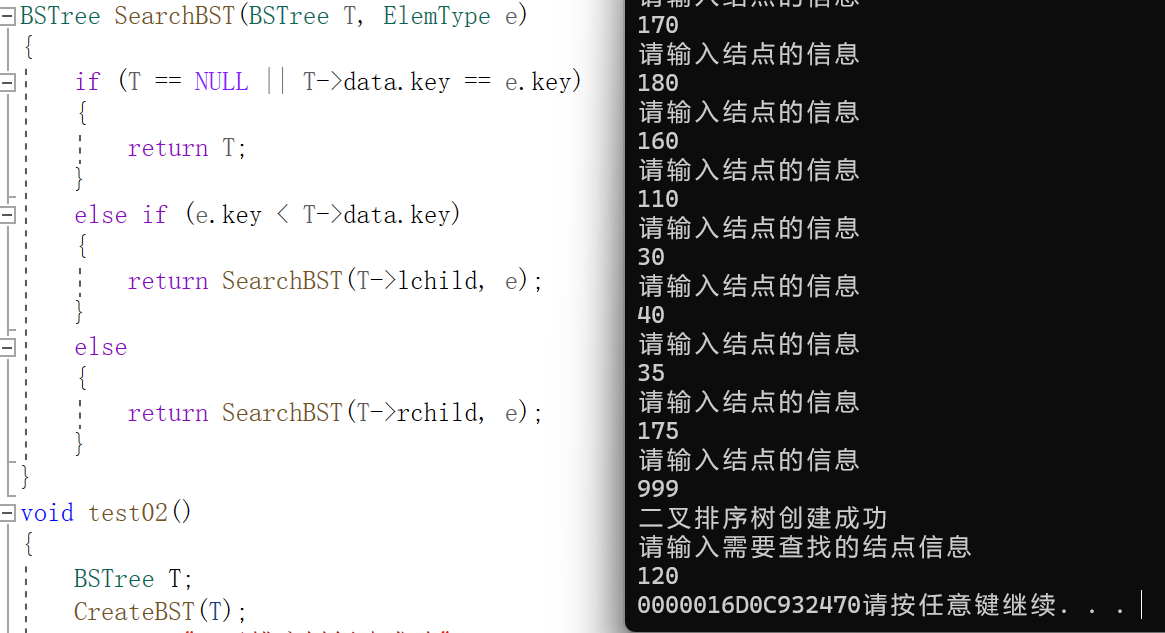
**6.4 运行结果截图及说明**

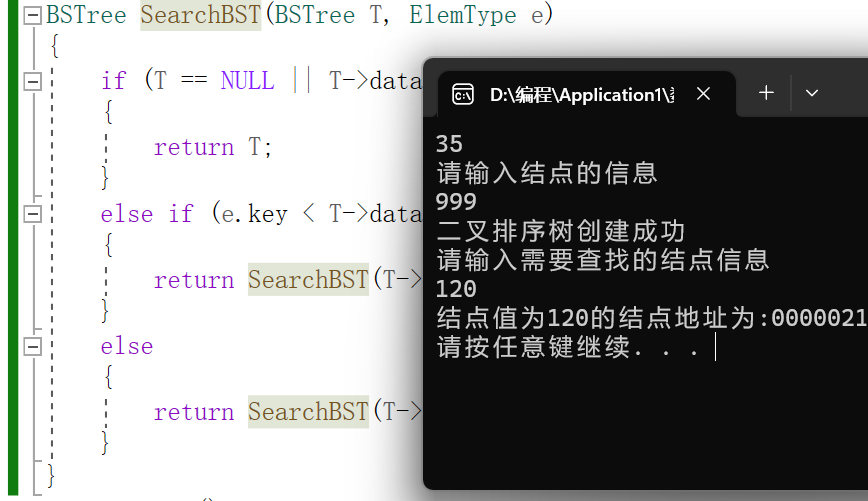
1. 对下列数据表，分别采用二分查找算法实现查找，给出查找过程依次所比较的元素（的下标）：





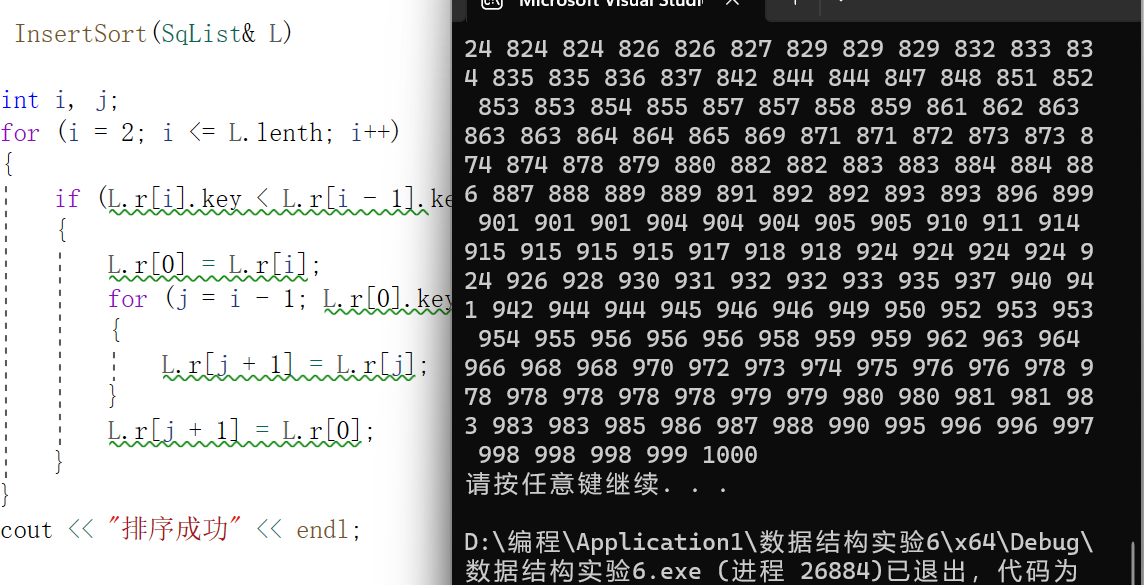
2、设计出在二叉排序树中插入结点的算法，在此基础上实现构建二叉排序树的算法。



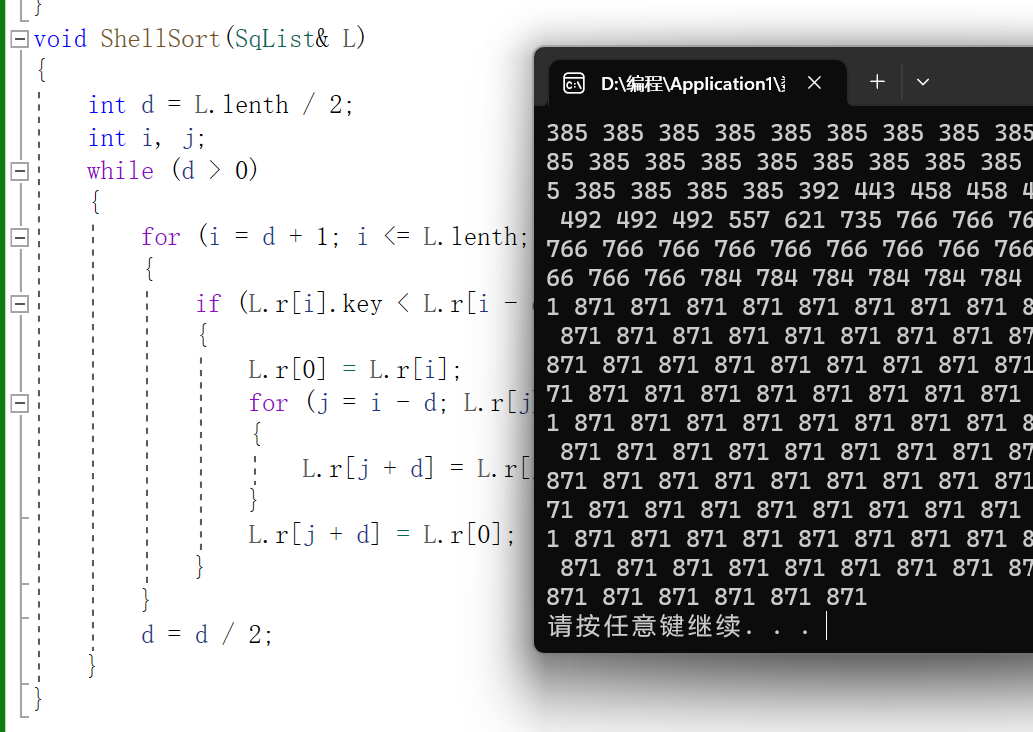


3、分别实现直接插入排序、希尔排序、冒泡排序、快速排 序算法；

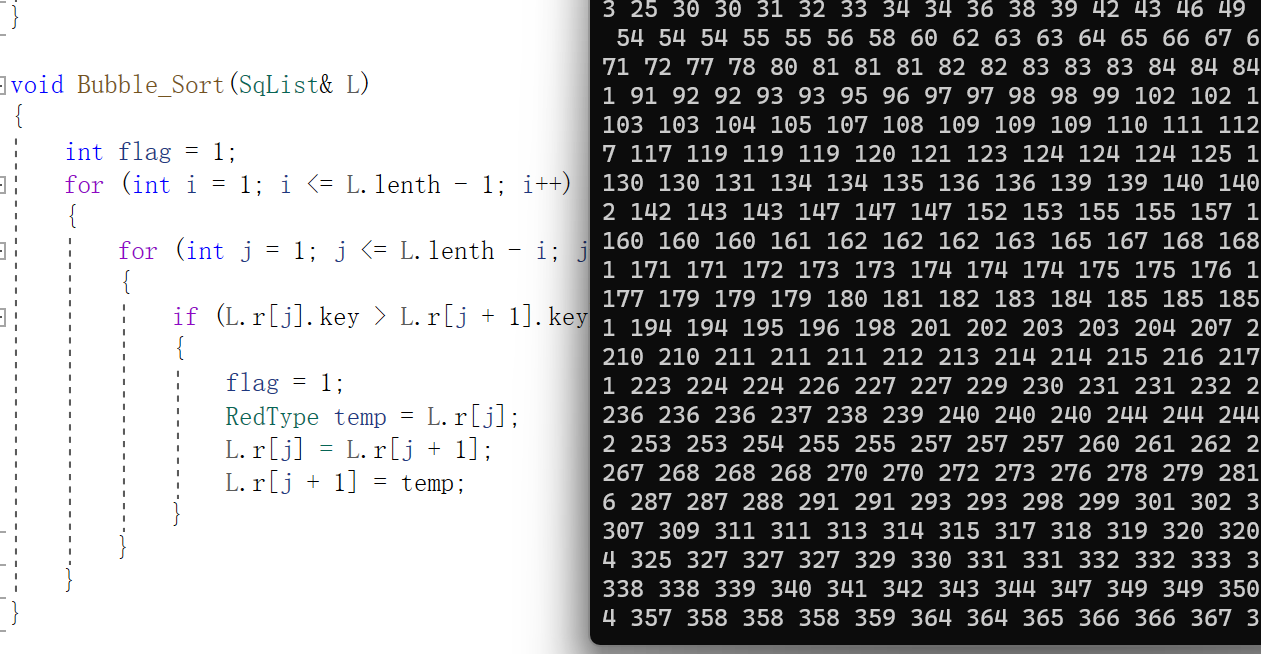
测试数据：随机生成5组整数数据，长度1000；

****

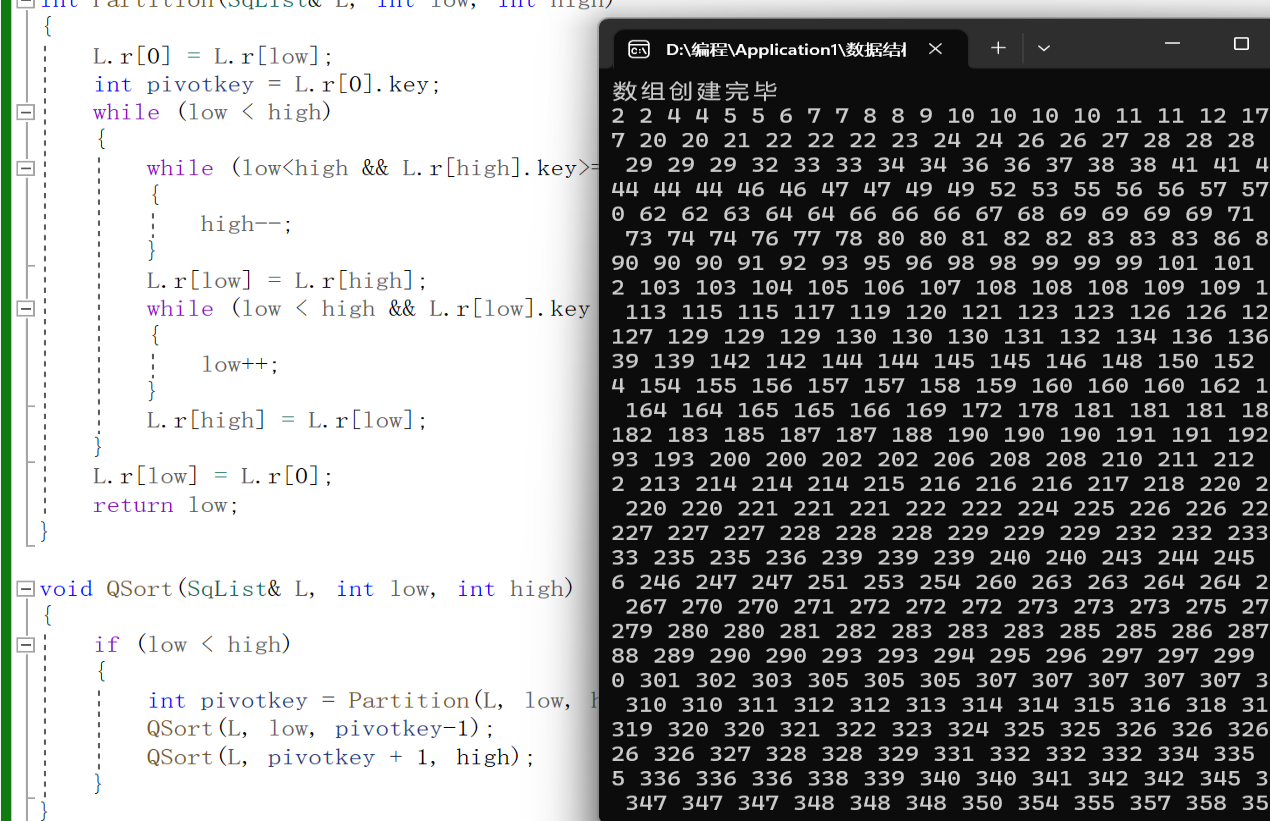
**插入排序**

****

**希尔排序**

****

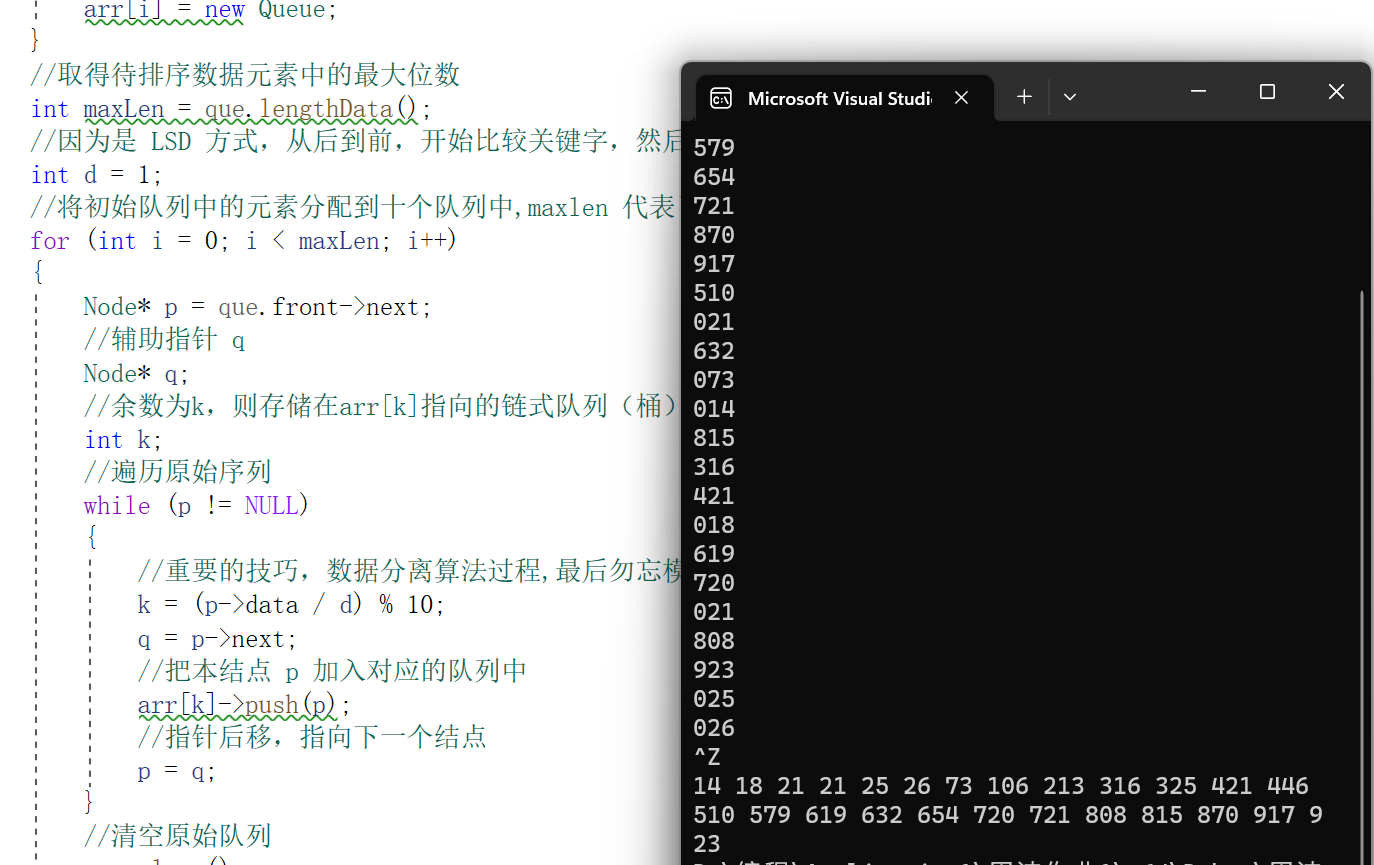
**冒泡排序**

****

**快速排序**

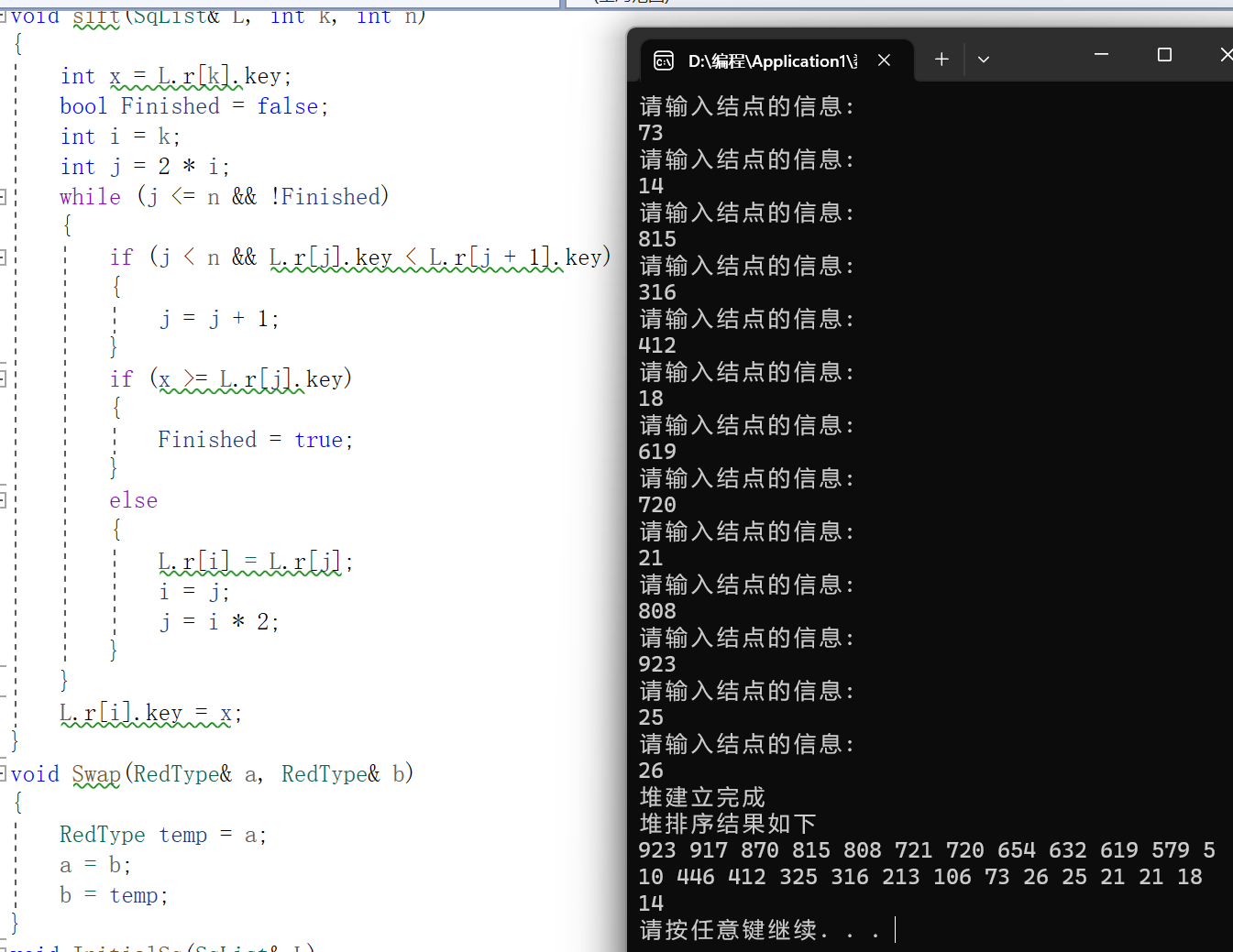
4、按链式基数排序的方法实现对整数表的排序。

测试数据如下：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）

****

5、完成堆排序。

测试数据：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）

****

**6.5 附源代码**

1.对下列数据表，分别采用二分查找算法实现查找，给出查找过程依次所比较的元素（的下标）：int Search\_Bin(SSTable ST, KeyType key)

{

int low = 1;

int high = ST.lenth;

while (low <= high)

{

int mid = (low + high) / 2;

cout << "比较的元素下标为" << mid << endl;

if (ST.R[mid].key > key)

{

high = mid - 1;

}

else if (ST.R[mid].key < key)

{

low = mid + 1;

}

else

{

return mid;

}

}

return -1;

}

2、设计出在二叉排序树中插入结点的算法，在此基础上实现构建二叉排序树的算法。

oid Insert(BSTree &T,ElemType e)

{

if (!T)

{

BSTNode\* S = new BSTNode;

S->data = e;

S->lchild = S->rchild = NULL;

T = S;//将新结点链接到已找到位置

}

else if (e.key < T->data.key)

{

Insert(T->lchild, e);

}

else if (e.key>T->data.key)

{

Insert(T->rchild, e);

}

}

void CreateBST(BSTree& T)

{

T = NULL;

ElemType e;

cout << "请输入结点的信息" << endl;

cin >> e.key;

while (e.key != ENDFLAG)

{

Insert(T, e);

cout << "请输入结点的信息" << endl;

cin >> e.key;

}

Insert(T, e); //将结束位置的结点也进行插入

}

BSTree SearchBST(BSTree T, ElemType e)

{

if (T == NULL || T->data.key == e.key)

{

return T;

}

else if (e.key < T->data.key)

{

return SearchBST(T->lchild, e);

}

else

{

return SearchBST(T->rchild, e);

}

}

3、分别实现直接插入排序、希尔排序、冒泡排序、快速排序算法；

测试数据：随机生成5组整数数据，长度1000:

void PrintList(SqList L)

{

for (int i = 1; i <= L.lenth; i++)

{

cout << L.r[i].key << " ";

}

}

void CreateList(SqList &L)

{

L.lenth = 1000;

srand(time(NULL));

for (int i = 1; i <= L.lenth; i++)

{

L.r[i].key= (rand()%(1000-1+1))+1;

}

cout << "数组创建完毕" << endl;

}

void InsertSort(SqList& L)

{

int i, j;

for (i = 2; i <= L.lenth; i++)

{

if (L.r[i].key < L.r[i - 1].key)

{

L.r[0] = L.r[i];

for (j = i - 1; L.r[0].key < L.r[j].key; j--)

{

L.r[j + 1] = L.r[j];

}

L.r[j + 1] = L.r[0];

}

}

cout << "排序成功" << endl;

}

void ShellSort(SqList& L)

{

int d = L.lenth / 2;

int i, j;

while (d > 0)

{

for (i = d + 1; i <= L.lenth; i++)

{

if (L.r[i].key < L.r[i - d].key)

{

L.r[0] = L.r[i];

for (j = i - d; L.r[j].key > L.r[0].key; j--)

{

L.r[j + d] = L.r[j];

}

L.r[j + d] = L.r[0];

}

}

d = d / 2;

}

}

void Bubble\_Sort(SqList& L)

{

int flag = 1;

for (int i = 1; i <= L.lenth - 1; i++)

{

for (int j = 1; j <= L.lenth - i; j++)

{

if (L.r[j].key > L.r[j + 1].key)

{

flag = 1;

RedType temp = L.r[j];

L.r[j] = L.r[j + 1];

L.r[j + 1] = temp;

}

}

}

}

int Partition(SqList& L, int low, int high)

{

L.r[0] = L.r[low];

int pivotkey = L.r[0].key;

while (low < high)

{

while (low<high && L.r[high].key>=pivotkey)

{

high--;

}

L.r[low] = L.r[high];

while (low < high && L.r[low].key <= pivotkey)

{

low++;

}

L.r[high] = L.r[low];

}

L.r[low] = L.r[0];

return low;

}

void QSort(SqList& L, int low, int high)

{

if (low < high)

{

int pivotkey = Partition(L, low, high);

QSort(L, low, pivotkey-1);

QSort(L, pivotkey + 1, high);

}

}

4、按链式基数排序的方法实现对整数表的排序。

测试数据如下：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）

void RadixSort(Queue& que)

{

//声明一个指针数组，该指针数组中存放十个指针，这十个指针需要分别指向十个队列，这是模拟10个桶，因为是0-9的数字，取值范围为10

Queue\* arr[10];

//初始化这十个队列

for (int i = 0; i < 10; i++)

{

//初始化建立头结点

arr[i] = new Queue;

}

//取得待排序数据元素中的最大位数

int maxLen = que.lengthData();

//因为是 LSD 方式，从后到前，开始比较关键字，然后分配再收集，故开始设置数据分离算法中的除数为 1

int d = 1;

//将初始队列中的元素分配到十个队列中,maxlen 代表了需要分配和收集的次数

for (int i = 0; i < maxLen; i++)

{

Node\* p = que.front->next;

//辅助指针 q

Node\* q;

//余数为k，则存储在arr[k]指向的链式队列（桶）中

int k;

//遍历原始序列

while (p != NULL)

{

//重要的技巧，数据分离算法过程,最后勿忘模10，取余数，分离出需要的关键字位

k = (p->data / d) % 10;

q = p->next;

//把本结点 p 加入对应的队列中

arr[k]->push(p);

//指针后移，指向下一个结点

p = q;

}

//清空原始队列

que.clear();

//分配完毕，马上将十个队列中的数据收集到原始队列中

for (int i = 0; i < 10; i++)

{

if (!arr[i]->empty())

{

//从首节点开始遍历，不是头结点开始

Node\* p = arr[i]->front->next;

//辅助指针 q

Node\* q;

while (p != NULL)

{

q = p->next;

//收集到原始队列中，这就是为什么每次分配完毕，需要清除原始队列

que.push(p);

p = q;

}

}

}

//一趟的分配收集完毕，最后要清空十个队列

for (int i = 0; i < 10; i++)

{

arr[i]->clear();

}

//进行下一趟的分配和收集

d \*= 10;

}

//输出队列中排好序的元素

print(que);

}

};5、完成堆排序。

测试数据：(106,213,325,446,579,654,721,870,917,510,21,632,73,14,815,316,412,18,619,720,21,808,923,25,26）:

void sift(SqList& L, int k, int n)

{

int x = L.r[k].key;

bool Finished = false;

int i = k;

int j = 2 \* i;

while (j <= n && !Finished)

{

if (j < n && L.r[j].key < L.r[j + 1].key)

{

j = j + 1;

}

if (x >= L.r[j].key)

{

Finished = true;

}

else

{

L.r[i] = L.r[j];

i = j;

j = i \* 2;

}

}

L.r[i].key = x;

}

void Swap(RedType& a, RedType& b)

{

RedType temp = a;

a = b;

b = temp;

}

void InitialSq(SqList& L)

{

int n = 0;

cout << "请输入结点的个数" << endl;

cin >> n;

L.lenth = n;

int m;

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

{

cout << "请输入结点的信息:" << endl;

cin >> m;

L.r[i].key = m;

}

}

void HeadSort(SqList& L, int n)

{

for (int i = n / 2; i >= 1; i--)

{

sift(L, i, n);

}

cout << "堆建立完成" << endl;

cout << "堆排序结果如下" << endl;

for (int i = n; i >= 2; i--)

{

cout << L.r[1].key << " ";

Swap(L.r[i], L.r[1]);

sift(L, 1, i - 1);

}

cout << L.r[1].key << " ";

}

**6.6 调试过程中出现的bug总结**

1、在编写二分查找算法时需要设置low和high以及mid，每一次通过low和high求出mid，之后通过mid进行元素的查找，在使用二分查找法时，为了节省空间，利用数组中下标为0号的元素，将该位置设定为哨兵，通过哨兵来判断是否找到所需数据。

2、在编写二叉排序树的创建算法时，首先编写插入算法，申请内存空间，当传入的结点为NULL时，将插入的结点信息赋值给该结点，之后将该结点链接在树上，若树不空，则判断传入结点的数值，若比当前结点数值小，则将该结点插入在左子树，若比该结点的数值大则插入在右子树上，如果和该结点的数值大小相同就不进行插入操作，之后编写构建树的函数，当每次传入信息后进行插入操作，此时二叉树创建完毕。之后在编写在二叉树上查找结点的算法时，返回当前指针的地址，如果T==NULL或者T->data.key==key,此时返回T即可，如果不是，则进行进一步判断，若该结点的值小于结点数值，则在左子树上进行查找，反之在右子树上进行查找。

3、在进行插入排序的算法编写过程中，从数组下标为2号的位置开始，一直到L.lenth长度进行插入，之后进行一次判断操作，若比前一位置数据小则进行移动元素并且插入的操作，需要注意的是由于之前的序列已经有序，因此只要L.r[0].key < L.r[j].key，循环就继续直到相等时结束。

4、进行希尔排序算法编写的过程中，类似于直接插入排序，将插入排序中元素之间的距离改为d即可，同时每一次排序结束更新d的数值。开始排序时也是从d+1号位置开始排序直到L.lenth。

5、进行冒泡排序时，首先设置一个是否交换的标志来减少算法的时间效率，之后判断若进行了交换则循环得以继续，否则退出循环。总共需要进行L.lenth-1次循环，每一次循环比较的次数为:L.lenth-i，之后进行判断操作，如果该位置的元素大于下一位置元素，则进行交换操作，每一次找到最小值。

6、在编写快速排序算法时，首先编写查找中间位置的算法，将数组中第一个位置的元素设置为中心点，之后让low从前往后寻找比中心点大的元素，high从后往前寻找比中心点小的元素，将寻找到的位置进行交换，最后当low==high时，该位置就是中心点的位置。然后返回中心点的位置，之后进行快速排序的操作，当low小于high查找中心点的位置并进行排序，使用递归进行算法编写，将中心点的左边部分继续调用快速排序算法，中心点的右边部分同样也继续调用快速排序算法。

7、进行堆排序时【利用大根堆进行顺序排列，小根堆进行倒序排列】，首先编写对堆进行调整的算法，参数设置为需要调整的位置的下标以及堆中元素的个数。之后对2\*i以及2\*i+1位置与i位置进行比较，找到最大的元素，如果原来的传入的结点的位置就是最大的元素，则设置Finished为true，无需再进行调整，否则进行交换之后更新下标位置直到最后一个结点，继续重复上述步骤，当最后调整结束以后，将i位置的数据赋值为最初传入的数据，则堆调整结束。之后通过堆调整的方法进行堆的建立操作，从最后一个非叶子节点开始(即：数组下标为n/2的位置，i>=1)进行堆调整，当堆调整完毕以后，从最后一个结点开始，与第一个结点进行交换，之后对剩余元素进行重新的堆调整找到次大值。

8、进行简单选择排序时，首先假定第一个位置的元素就是最小值，然后从i+1的位置开始寻找最小值，之后判断min是否=最开始的i，若不等于则进行交换操作。

9、在使用链式数据结构进行基数排序时，利用了链队列来保存数据。首先以静态链表存储待排记录，并令表头指针指向第一个记录，之后进行分配操作。  
在分配时，按当前关键字位所取值，将记录分配到不同的链队列中，每个队列中记录的关键字位相同。最后在收集时，按当前关键字位取值从小到大将各队列首尾相链成一个链表。最后对每个关键字位均重复 2 和 3 两步。在算法编写时，首先使用链式存储结构将数据存储起来，并令表头指针指向第一个数据，之后声明一个指针数组，该指针数组中存放10个指针指向10个队列。

之后计算出待排序数据中元素的最大位数也就是需要分配收集的次数，之后通过循环计算出原始数据中当前数位的元素之后进行入队操作，当所有数据都进入队列之后清空原始队列，之后在进行收集操作，将10个队列中的数据依次收集进入原队列，当收集结束后将10个队列清空，最后更新d的数值然后进入下一次的循环直到所有数均被放入队列中。