**要求代码和实验报告规范，在算法思想中：对实验涉及的数据结构进行有效设计和分析；对算法进行分析并给出时间、空间复杂度的结论；清晰表达实验思路、出现的问题及解决方法。**

**一、调试成功程序及说明**

**1、**

**题目：**实现二叉排序树的插入和删除。

**算法思想：**借助之前的二叉树知识，递归实现二叉排序树的建立、插入与删除

**运行结果：**输入需要建树的元素，创建二叉树后在进行插入删除

**结果分析：**正确

**附源程序。**

#include <stdlib.h>

#include <stdio.h>

#include<iostream>

using namespace std;

typedef int KeyType;

typedef struct node

{

KeyType key;

struct node \*left,\*right;

}BSTNode;

BSTNode \*BSTInsert( BSTNode \*T, KeyType key ) //递归插入算法

{

if(T==NULL)

{

T = (BSTNode \*)malloc(sizeof(BSTNode));

T->key = key;

T->left = T->right = NULL;

return T;

}

else if( T->key == key)

return T;

else if( T->key > key)

T->left = BSTInsert( T->left, key);

else

T->right = BSTInsert( T->right, key);

return T;

}

BSTNode \*CreateBST(int n) //二叉排序树的创建算法

{

BSTNode \*T = NULL;

KeyType key;

printf("请输入数据元素的值：");

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

{

scanf("%d",&key);

T = BSTInsert( T, key);

}

return T;

}

BSTNode \*BSTDelete( BSTNode \*T, KeyType key) //二叉排序树的删除算法

{

BSTNode \*p = T;

BSTNode \*pa = NULL;

BSTNode \*f,\*q;

while(p)

{

if(p->key==key)

break;

else if( p->key<key )

{

pa = p;

p = p->right;

}

else

{

pa = p;

p = p->left;

}

}

if(!p)

return T;

if( !p->left && !p->right )

{

if(pa)

{

if( pa->key < key)

pa->right = NULL;

else

pa->left = NULL;

free(p);

return T;

}

else

{

free(p);

return NULL;

}

}

else if( !p->left && p->right )

{

if( pa )

{

if( pa->key < key)

pa->right = p->right;

else

pa->left = p->right;

free(p);

return T;

}

else

{

T = p->right;

free(p);

return T;

}

}

else if( p->left && !p->right)

{

if( pa )

{

if( pa->key < key)

pa->right = p->left;

else

pa->left = p->left;

free(p);

return T;

}

else

{

T = p->left;

free(p);

return T;

}

}

else

{

f = p;

q = p->left;

while( q->right )

{

f = q;

q = q->right;

}

p->key = q->key;

if( !q->left && !q->right )

{

if( f->key < q->key )

f->right = NULL;

else

f->left = NULL;

free(q);

}

else

{

if( f->key < q->key )

f->right = q->left;

else

f->left = q->left;

free(p);

}

return T;

}

}

void InOrderTraverse( BSTNode \*T)

{

if( T!=NULL )

{

if( T->left )

InOrderTraverse(T->left);

printf("%d ",T->key);

if( T->right )

InOrderTraverse(T->right);

}

}

int main()

{

int n,m,choice;

printf("构建二叉树：\n");

printf("请输入数据元素的个数：");

cin>>n;

BSTNode \*T;

T = CreateBST(n);

printf("中序遍历二叉排序树为：");

InOrderTraverse(T);

while(1)

{

cout<<"\n1.插入元素 2.删除元素 3.退出"<<endl;

cout<<"请选择：";

cin>>choice;

if(choice==1)

{

printf("\n请输入想要插入的元素：");

cin>>m;

T = BSTInsert(T,m);

printf("插入元素后二叉排序树为：");

InOrderTraverse(T);

}

else if(choice==2)

{

printf("\n请输入想要删去的元素：");

cin>>m;

T = BSTDelete(T,m);

printf("删除元素后二叉排序树为：");

InOrderTraverse(T);

}

else

break;

}

return 0;

}

**2、**

**题目：**实现交换、选择、归并等简单排序算法；

**算法思想：**交换、选择与归并的基本思想。交换：时间复杂度O(n^2),空间复杂度O(1),稳定；选择：时间复杂度O(n^2),空间复杂度O(1),不稳定；归并：时间复杂度O(nlog2底n),空间复杂度O(n),稳定。

**运行结果：**输入无序数列，选择排序算法并输出

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

#define MAXSIZE 256

#define STACKINITSIZE 256

typedef int KeyType;

typedef struct

{

KeyType key;

char info;

}RecordType;

typedef struct

{

RecordType r[MAXSIZE+1];

int length;

}SqList;

void BubbleSort( SqList &L )

{

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

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

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

{

RecordType temp = L.r[j];

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

L.r[j+1] = temp;

}

}

void SelectSort( SqList &L )

{

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

{

int min = L.r[i].key;

int k = i;

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

{

if( L.r[j].key<min )

{

min = L.r[j].key;

k = j;

}

}

RecordType temp = L.r[i];

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

L.r[k] = temp;

}

}

void Merge( SqList &L, int low, int mid, int high)

{

SqList T;

int i = low;

int j = mid+1;

int k = 0;

while( i<=mid && j<=high )

{

if( L.r[i].key<=L.r[j].key )

T.r[k++].key = L.r[i++].key;

else

T.r[k++].key = L.r[j++].key;

}

while( i<=mid )

T.r[k++].key = L.r[i++].key;

while( j<=high )

T.r[k++].key = L.r[j++].key;

for( k=0,i=low; i<=high; k++,i++)

L.r[i].key = T.r[k].key;

}

void MSort( SqList &L, int len)

{

int i = 1;

while( i+2\*len <= L.length)

{

Merge( L, i, i+len-1, i+2\*len-1);

i = i+2\*len;

}

if( i+len <= L.length)

Merge( L,i,i+len-1,L.length);

}

void MergeSort(SqList &L)

{

for(int len=1; len<=L.length; len=2\*len)

MSort(L,len);

}

int main()

{

SqList L1,L2,L3;

int n,choice;

cout<<"输入需要排序的整数个数：";

cin>>n;

L1.length = n;

L2.length = n;

L3.length = n;

cout<<"请输入整数：";

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

{

cin>>L1.r[i].key ;

L2.r[i].key =L1.r[i].key ;

L3.r[i].key =L1.r[i].key ;

}

int flag=1;

cout<<"创建三组排序L1,L2,L3均为："<<endl;

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

{

cout<<L1.r[i].key<<" " ;

}

while(flag==1)

{

cout<<"\n请选择："<<endl;

cout<<"1.交换排序 2.选择排序"<<endl;

cout<<"3.归并排序 4.退出"<<endl;

cin>>choice;

switch(choice)

{

case(1):

BubbleSort(L1);

cout<<"L1(冒泡)是：";

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

{

cout<<L1.r[i].key<<" " ;

}

cout<<endl;

break;

case(2):

SelectSort(L2);

cout<<"L2(选择)是：";

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

{

cout<<L2.r[i].key<<" " ;

}

cout<<endl;

break;

case(3):

MergeSort(L3);

cout<<"L3(归并)是：";

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

{

cout<<L3.r[i].key<<" " ;

}

cout<<endl;

break;

case(4):

flag=0;

break;

}

}

return 0;

}

**3、**

**题目：**实现快速排序算法；

**算法思想：**快速排序算法思想，时间复杂度O(nlogn),空间复杂度O(logn),不稳定。

**运行结果：**输入无序数列，输出快速排序后的数列

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

#define MAXSIZE 256

typedef int KeyType;

typedef struct

{

KeyType key;

char info;

}RecordType;

typedef struct

{

RecordType r[MAXSIZE+1];

int length;

}SqList;

int Partition( SqList &L, int low, int high) //一趟快速排序

{

RecordType temp = L.r[low];

int pivotkey = L.r[low].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] = temp;

return low;

}

void QSort( SqList &L, int low, int high ) //low和high区间的快速排序算法

{

if( low<high )

{

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

QSort(L, low, pivotloc-1);

QSort(L, pivotloc+1, high);

}

}

void QuickSort( SqList &L) //快速排序算法

{

QSort(L, 1, L.length);

}

int main()

{

SqList L;

int n;

cout<<"输入需要排序的整数个数：";

cin>>n;

L.length = n;

cout<<"请输入整数：";

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

{

cin>>L.r[i].key ;

}

QuickSort(L);

cout<<"快速排序后序列是："<<endl;

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

{

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

}

return 0;

}

**4、**

**题目：**实现堆排序算法；

**算法思想：**堆排序算法思想，时间复杂度O(nlog2底n),空间复杂度O(1),不稳定。

**运行结果：**输入无序数列，输出堆排序后的数列

**结果分析：**正确

**附源程序。**

#include <stdio.h>

#include <stdlib.h>

#include<iostream>

using namespace std;

#define MAXSIZE 256

typedef int KeyType;

typedef struct

{

KeyType key;

char info;

}RecordType;

typedef struct

{

RecordType r[MAXSIZE+1];

int length;

}SqList;

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

{

RecordType temp = L.r[low];

int i = low;

int j = 2\*i;

while( j<=high )

{

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

j++;

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

break;

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

i = j;

j = 2\*i;

}

L.r[i] = temp;

}

void HeapSort( SqList &L )

{

for( int i=L.length/2; i>0; i--)

HeapAdjust(L, i, L.length);

for( int i=L.length; i>1; i--)

{

RecordType temp = L.r[1];

L.r[1] = L.r[i];

L.r[i] = temp;

HeapAdjust(L,1,i-1);

}

}

int main()

{

SqList L;

int n;

cout<<"输入需要排序的整数个数：";

cin>>n;

L.length = n;

cout<<"请输入整数：";

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

{

cin>>L.r[i].key ;

}

HeapSort(L);

cout<<"\n堆排序后序列是：";

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

{

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

}

return 0;

}

**5、**

**题目：**CSP题目 垃圾回收站

**算法思想：**简单数组运算

**运行结果：**按题输入垃圾坐标，输出不同得分垃圾站选址个数

**结果分析：**正确

**附源程序。**

#include<stdlib.h>

#include<stdio.h>

#include<iostream>

using namespace std;

typedef struct Point

{

int x;

int y;

}Point;

Point pos[256];

int main()

{

int n;

cin>>n;

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

cin>>pos[i].x>>pos[i].y;

int grade[5]={0};

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

{

int flag = 0;

int row = pos[i].x;

int col = pos[i].y;

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

{

if(j!=i)

{

if(pos[j].x==row && pos[j].y==col+1)

flag++;

if(pos[j].x==row && pos[j].y==col-1)

flag++;

if(pos[j].x==row+1 && pos[j].y==col)

flag++;

if(pos[j].x==row-1 && pos[j].y==col)

flag++;

}

}

if(flag == 4)

{

int temp = 0;

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

{

if(j!=i)

{

if(pos[j].x==row+1 && pos[j].y==col+1)

temp++;

if(pos[j].x==row-1 && pos[j].y==col-1)

temp++;

if(pos[j].x==row+1 && pos[j].y==col-1)

temp++;

if(pos[j].x==row-1 && pos[j].y==col+1)

temp++;

}

}

if(temp == 0)

grade[0]++;

else if(temp==1)

grade[1]++;

else if(temp==2)

grade[2]++;

else if(temp==3)

grade[3]++;

else if(temp==4)

grade[4]++;

}

}

for(int i=0;i<5;i++)

{

cout<<grade[i]<<endl;

}

return 0;

}

**三、代码行数及小结**

行数：合计550左右

小结：本次最后一次上机，了解了二叉排序树的建立与使用，复习冒泡与选择排序法，新学习了快速排序、归并排序、堆排序等排序方法，具有很强的实用性。