10.比较多种排序方法

void insertArray (T\* arru, int len) {

\_root = new node;

//数组中每个元素均执行一次插入操作，i为其关键词

for (int i = 0; i < len; i++) {

node\* p = \_root;

node\* pp = nullptr;

while (p != nullptr) {

pp = p;

if (i < p->element.key)

p = p->left;

else

p = p->right;

}

node\* newNode= new node(i,arri[i]);

if (root != nullptr) {

if (i < pp->element.key)pp->left = newNode;

else pp->right = newNode;

} else {

\_root = newNode;

}

\_size++;

}

}

//position指向需要放置元素的位置，每次放置元素会导致其加一；

//遍历完成后，position指向有序数组的最后一个元素

void inOrderToArr (node\* t,E\* position) {

if (t != nullptr) {

inOrderToArr (t->leftChild,position);

position[0]=t->key;

position++;

inOrderToArr (t->leftChild,position);

return;

} else {

return;

}

}

建立二叉树时间复杂度为O(n)

中序遍历该二叉树时间复杂度为O(n)

故使用二叉搜索树进行遍历时间复杂度为O(n)

而插入排序时间复杂度为O(n^2)，堆排序时间复杂度为O(nlgn)

很明显二叉搜索树排序更优

15.删除最大节点

void deleteMax () {

node\* troot = \_root;//指向要删除的节点

node\* troot\_f = \_root;//指向要删除的元素的父节点

//判断两种特殊情况

if (\_root == nullptr) {

return

}

if (\_root->right == nullptr) {

troot = \_root;

\_root = \_root->left;

delete troot;

return;

}

//找到最右的右子树

while (troot->right != nullptr) {

troot\_f = troot;

troot = troot->right;

}

//如果最右子节点还有左子树，则该左子树变成最右子节点的父节点的右子树

if (troot->left != nullptr) {

troot\_f->right = troot->left;

}

//删除右子树

delete troot;

return;

}