#include<iostream>

#include<cmath>

using namespace std; //===============================

struct node

{

int info;

node \*left;

node \*right;

}; //===================================

class cbt

{

node \*temp,\*temp1,\*temp2,\*temp0;

bool flag;

int count,num;

double totalheight,totalmaxnode,totalminnode;

public:

node \*root;

int key; //================================================

cbt()

{

root=temp=temp1=temp2=temp0=0;

key=count=0;

totalheight=0,totalmaxnode=0,totalminnode=0;

num=1;

flag=true;

} //=====================================================

void insert(node \*temp)

{

num=1;

if(root==NULL)

{

root=new node;

root->info=key;

root->left=root->right=0;

temp=root;

count++;

return;

}

totalheight=height(root);

totalminnode=pow(2.0,totalheight);

totalmaxnode=pow(2.0,(totalheight+1))-1;

if(count==totalmaxnode)

{

while(temp->left!=NULL)

{

temp=temp->left;

}

temp->left=new node;

temp->left->info=key;

temp->left->left=temp->left->right=0;

count++;

percolate(root);

return;

}

update(root,totalheight);

if(flag==true)

{

temp1->left=new node;

temp1->left->info=key;

temp1->left->left=temp1->left->right=0;

count++;

percolate(root);

return;

}

if(flag==false)

{

temp1->right=new node;

temp1->right->info=key;

temp1->right->left=temp1->right->right=0;

count++;

percolate(root);

return;}

percolate(root);

} //=================================================

void percolate(node \*temp)

{int change;

if((temp->left!=NULL)&&(temp->right==NULL))

{

if(temp->info < temp->left->info)

{ change =temp->info;

temp->info=temp->left->info;

temp->left->info=change;

}

}

if((temp->left!=NULL)&&(temp->right!=NULL))

{ if((temp->info<temp->right->info || temp->info<temp->left->info )||(temp->info<temp->right->info && temp->info<temp->left->info ))

{

if(temp->right->info<temp->left->info)

{ change =temp->info;

temp->info=temp->left->info;

temp->left->info=change;

}

else

{ change =temp->info;

temp->info=temp->right->info;

temp->right->info=change;

}

}

}

if(temp->left!=NULL)

percolate(temp->left);

if(temp->right!=NULL)

percolate(temp->right);

} //================================================

void heapdelete(node \*temp)

{

if(root==NULL)

{

cout<<"empty root not deleted "<<endl;

return;

}

last(temp);

if(temp0->right==NULL)

{root->info=temp0->left->info;

temp0->left=NULL;

count--;

percolate(root);

return;}

else

{ root->info=temp0->right->info;

temp0->right=NULL;

count--;

percolate(root);

return;}

} //==============================================

void update(node \*temp,double ht)

{

if(temp->left==NULL)

{

temp1=temp;

flag=true;

num=2;

return;

}

if(temp->right==NULL)

{

temp1=temp;

flag=false;

num=2;

return;

}

if(ht>1)

update(temp->left,ht-1);

if(ht>1 && num==1)

update(temp->right,ht-1);

} //==================================================

int height(node \*temp)

{

int step=0;

while(temp->left!=NULL)

{

temp=temp->left;

step++;

}

return step;

} //===========================================

void inorder(node \*temp)

{

if(temp==NULL)

{ cout<<"THe tree is empty \n";

return;

}

if(temp->left!=NULL)

inorder(temp->left);

cout<<"\t"<<temp->info<<" ";

if(temp->right!=0)

inorder(temp->right);

return;

}

//=====================================================

int htr(node \*temp)

{

int step=1;

if(temp->right!=NULL)

{temp=temp->right;

while(temp->left!=NULL)

{

temp=temp->left;

step++;

}

}

else

step=0;

return step;

}

void last(node \*temp)

{int a,b;

if(temp->left->left!=NULL)

{

a=height(temp);

b=htr(temp);

if(a==b)

last(temp->right);

else

last(temp->left);

}

else

{

temp0=temp;

return;

}

} //=====================================================

};

void main()

{

cbt obj;

int a;

cout<<"insert";

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

{

cin>>obj.key;

obj.insert(obj.root);

}

obj.inorder(obj.root);

cout<<endl;

cout<<"delete";

obj.heapdelete(obj.root);

obj.inorder(obj.root);

cout<<endl;

cout<<"delete";

obj.heapdelete(obj.root);

obj.inorder(obj.root);

cout<<endl;

cout<<"delete";

obj.heapdelete(obj.root);

obj.inorder(obj.root);

cout<<endl;

system("pause");

}