```
#include <iostream>
#include <fstream>
using namespace std;

class Node{
    public:

    Node* parent;
    int key1;
    int key2;
    Node* child1;
    Node* child2;
    Node* child3;

    // Constructor for internal node
    Node(Node* newParent, int newKey1, int newKey2, Node* newChild1, Node* newChild2, Node* newChild3){
        parent = newParent;
        key1 = newKey1;
        key2 = newKey2;
        child1 = newChild1;
        child2 = newChild2;
        child3 = newChild3;
    } //Node()

    // Constructor for leaf node
    Node(Node* newParent, int data){
        parent = newParent;
        key1 = -1;
        key2 = data;
    }//Node()

    void printNode(fstream& outstream){
        outstream << "(parent.key1 = ";

        if(parent!=NULL)
            outstream << parent->key1;
        else
            outstream << "0";

        outstream << ", key1 = " << key1;
        outstream << ", key2 = " << key2 << ", child1 = ";

        if(child1 != NULL)
            outstream << child1->key1;
        else
            outstream << "-1";

        outstream << ", child2 = ";

        if(child2 != NULL)
            outstream << child2->key1;
        else
            outstream << "-1";

        outstream << ", child3 = ";

        if(child3 != NULL)
            outstream << child2->key1;
        else
            outstream << "-1";

        outstream << ")" << endl;
    }//printNode()
};//Node class

class Tree{
    public:

    Node* root;

    Tree(){
        root = new Node(NULL, -1, -1, NULL, NULL, NULL);
    }//Tree()

    void updateKeys(Node* spot){
        // If we're past the root of the tree (this function traverses up the tree),
        // we are done
        if(spot==NULL)
            return;

        if(spot->child2 != NULL && spot->child2->key1 == -1)
            spot->key1 = spot->child2->key2;

        else if(spot->child2 == NULL)
            spot->key1 = -1;
```

```
        else if(spot->child2 != NULL && spot->child2->key1 != -1){
            //Find the smallest node in the second subtree
            Node* p = spot->child2;
            while(p->key1 != -1)
                p = p->child1;
            spot->key1 = p->key2;
        }//else if

        if(spot->child3 != NULL && spot->child3->key1 == -1)
            spot->key2 = spot->child3->key2;

        else if(spot->child3 == NULL)
            spot->key2 = -1;

        else if(spot->child3 != NULL && spot->child3->key1 != -1){
            Node* p = spot->child3;
            while(p->key1 != -1)
                p = p->child1; //while
            spot->key2 = p->key2;
        }//else if
        updateKeys(spot->parent);
    }//updateKeys()

    void insert(Node* spot, Node* newData){
        cout << "In insert();" << endl <<
            "newData->key1 = " << newData->key1 << ", newData->key2 = " << newData->key2 << endl;

        // Case 0: Spot has 0 children
        if(spot->child1 == NULL && spot->child2 == NULL && spot->child3 == NULL){
        cout << "Case 0" << endl;
            newData->parent=spot;
            spot->child1 = newData;
        }

        // Case 1: Spot has 1 child
        else if(spot->child1 != NULL && spot->child2 == NULL && spot->child3 == NULL){
            cout << "Case 1" << endl;
            newData->parent=spot;
            if(spot->child1->key2 <= newData->key2){
                spot->child2 = newData;
                cout << "spot->child2 = newData" << endl;
            }
            else {
                spot->child2 = spot->child1;
                spot->child1 = newData;
            }//else

        }//else if

        // Case 2: Spot has 2 children
        else if(spot->child1 != NULL && spot->child2 != NULL && spot->child3 == NULL){
            cout << "Case 2" << endl;
            newData->parent=spot;
            // Modified bubble sort from beginning of semester
            // to put children in ascending order
            // It's O(n^2), but we only have 4 items max
            Node* children[3] = {spot->child1, spot->child2, newData};
            int begin=0, end=2, swapflag=1, walker;

            while(end>begin && swapflag>0){
                walker=begin;
                if(swapflag>=1)
                    swapflag=0;
                while(walker < end){
                    if(children[walker]->key2 > children[walker+1]->key2){
                        //swap
                        Node* temp=children[walker];
                        children[walker]=children[walker+1];
                        children[walker+1]=temp;
                        swapflag++;
                        walker++;
                    }//if
                    end--;
                }//while
            }//while
            spot->child1 = children[0];
            spot->child2 = children[1];
            spot->child3 = children[2];
        }//if

        // Case 3: Spot has 3 children
        else if(spot->child1 != NULL && spot->child2 != NULL && spot->child3 != NULL){
            cout << "Case 3" << endl;
            Node* children[4] = {spot->child1, spot->child2,
```

```
            spot->child3, newData};
        int begin=0, end=3, swapflag=1, walker;
        cout << "Done building array; starting sort" << endl;

        while(end>begin && swapflag>0){
            swapflag=0;
            walker=begin;
            while(walker < end){
                cout << "In while(); walker = " << walker << endl;
                if(children[walker]->key1 > children[walker+1]->key1){
                    cout << "In if();"<< endl;
                    cout << "children[" << walker << "]==" << children[walker] << endl;
                    //swap
                    Node* temp=children[walker];
                    cout << "Node* temp=children[walker];" << endl;
                    children[walker]=children[walker+1];
                    cout << "children[walker]=children[walker+1];" << endl;
                    children[walker+1]=temp;
                    cout << "children[walker+1]=temp;" << endl;
                    swapflag++;
                    walker++;
                }//if
                end--;
            }//while
        }//while
        cout << "Done sorting" << endl;

        cout << "Printing contents of 4 nodes" << endl;
        for(int i=0; i<4; i++){
            cout << "children[" << i << "].key2 = " << children[i]->key2 << endl;
        }

        spot->child1 = children[0];
        spot->child2 = children[1];

        Node* newNode = new Node(NULL, -1, -1, NULL, NULL, NULL);
        Node* newChild1 = children[2];
        Node* newChild2 = children[3];
        newNode->child1 = newChild1;
        newNode->child2 = newChild2;
        newNode->child1->parent = newNode;
        newNode->child2->parent = newNode;

        updateKeys(spot);
        updateKeys(newNode);


        //cout << "About to make recursive call of insert()" << endl;

        if(spot->parent==NULL){
            root = spot->parent = new Node(NULL, -1, -1, spot, newNode, NULL);
            newNode->parent = spot->parent;
        }//if
        insert(spot->parent, newNode);
    }//else if


    // Update the keys
    updateKeys(spot);
}//insert()

void printTree(Node* nodeToPrint, fstream& outstream){
    if(nodeToPrint==NULL)
        return;
    else{
        nodeToPrint->printNode(outstream);
        printTree(nodeToPrint->child1, outstream);
        printTree(nodeToPrint->child2, outstream);
        printTree(nodeToPrint->child3, outstream);
    }//else
}//printTree()

Node* findSpot(Node* currentNode, int data){
    if(currentNode->child1 == NULL || currentNode->child1->key1 == -1)
        return currentNode;

    else if(data < currentNode->key1)
        return findSpot(currentNode->child1, data);

    else if((data >= currentNode->key1 && data > currentNode->key2) || currentNode->key2 == -1)
        return findSpot(currentNode->child2, data);

    else if(data >= currentNode->key2 && currentNode->key2 == -1)
        return findSpot(currentNode->child3, data);
}//findSpot()
```

```
};//Tree class

int main(){
    fstream infile;
    infile.open("infile.txt", fstream::in);

    if(!infile.is_open()){
        cout << "Error opening infile.txt, quitting." << endl;
        return 0;
    }//if

    fstream outfile;
    outfile.open("outfile.txt", fstream::out | fstream::trunc);

    if (!outfile.is_open()) {
        cout << "Error opening outfile.txt, quitting." << endl;
        return 0;
    }//if

    Tree* tree = new Tree();

    while(!infile.eof()){
        int input;
        infile >> input;
        Node* spot = tree->findSpot(tree->root, input);
        Node* newNode = new Node(NULL, input);
        outfile << "Tree with " << input << " inserted:" << endl;
        tree->insert(spot, newNode);
        tree->printTree(tree->root, outfile);
        outfile << "--------" << endl;
    }//while
    cout << "Completed insert operation" << endl;
    return 0;
}//main()
```