Binary Tree

No1:

int identical(BTNode \*tree1, BTNode \*tree2)

{

    /\* add your code here \*/

    if(tree1 == NULL && tree2 == NULL) return 1;

    else if(tree1 == NULL || tree2 == NULL) return 0;

    else{

        if(tree1->item != tree2->item) return 0;

        else return identical(tree1->left, tree2->left)\*identical(tree1->right, tree2->right);

    }

}

No2:

int maxHeight(BTNode \*node)

{

    /\* add your code here \*/

    int l;

    int r;

    if(node == NULL) return -1;

    else {

        l = maxHeight(node->left);

        r = maxHeight(node->right);

        if(l>r) return l + 1;

        else return r + 1;

    }

}

No3:

int countOneChildNodes(BTNode \*node)

{

    /\* add your code here \*/

    if(node == NULL) return 0;

    else{

        if((node->left == NULL && node->right != NULL)||(node->left != NULL && node->right == NULL)){

            return 1 + countOneChildNodes(node->left) + countOneChildNodes(node->right);

        }

        else{

            return countOneChildNodes(node->left) + countOneChildNodes(node->right);

        }

    }

}

No4:

int sumOfOddNodes(BTNode \*node)

{

    /\* add your code here \*/

    if(node == NULL){return 0;}

    else{

        if(node->item % 2 == 1 || node->item % 2 == -1){

            return node->item + sumOfOddNodes(node->left) + sumOfOddNodes(node->right);

        }

        else return sumOfOddNodes(node->left) + sumOfOddNodes(node->right);

    }

}

No5:

void mirrorTree(BTNode \*node)

{

    /\* add your code here \*/

    if(node == NULL) return;

    else{

        BTNode \*temp;

        temp = node->left;

        node->left = node->right;

        node->right = temp;

        mirrorTree(node->left);

        mirrorTree(node->right);

    }

}

No6:

void printSmallerValues(BTNode \*node, int m)

{

    /\* add your code here \*/

    if(node == NULL) {}

    else {

        if(node->item <m){

            printf("%d ",node->item);

        }

        printSmallerValues(node->left, m);

        printSmallerValues(node->right, m);

    }

}

No7:

int smallestValue(BTNode \*node)

{

    /\* add your code here \*/

    int l;

    int r;

    if(node == NULL) return 99999999;

    else if(node->left == NULL && node->right == NULL){

        return node->item;

    }

    else{

        l = smallestValue(node->left);

        r = smallestValue(node->right);

        if(l>r) return r;

        else return l;

    }

}

No8:

int hasGreatGrandchild(BTNode \*node)

{

    /\* add your code here \*/

    int l,r,f;

    if(node == NULL) return -1;

    else{

        l = hasGreatGrandchild(node->left);

        r = hasGreatGrandchild(node->right);

        if(l > r) f = l;

        else f = r;

        f = f + 1;

        if(f >= 3) printf("%d ",node->item);

        return f;

    }

}