```cpp
#include <bits/stdc++.h>

using namespace std;

class treeNode
{

public:
    int data;
    treeNode *leftchild;
    treeNode *rightchild;

    treeNode(int value)
    {
        data = value;
        leftchild = NULL;
        rightchild = NULL;
    }
};

void spacePrint(int level)
{
    for (int i = 0; i < level; i++)
    {
        cout << "   ";
    }
}

void printTree(treeNode *root, int level)
{
    if (root == NULL)
    {
        return;
    }
    if (root->leftchild == NULL && root->rightchild == NULL)
    {
        cout << root->data << endl;
    }
    else
    {
        cout << endl;
        spacePrint(level);
        cout << " Root: " << root->data << endl;
    }

    if (root->leftchild != NULL)
    {
        spacePrint(level);
        cout << " Left: ";
        printTree(root->leftchild, level + 1);
    }

    if (root->rightchild != NULL)
    {
        spacePrint(level);
        cout << " Right: ";
        printTree(root->rightchild, level + 1);
    }
}

void inOrder(treeNode *root, string &chk)
{
    if (root == NULL)
        return;

    inOrder(root->leftchild, chk);
    chk += to_string(root->data) + " ";
    inOrder(root->rightchild, chk);
}

treeNode *insertionBST(treeNode *root, int value)
{
    treeNode *newNode = new treeNode(value);
    if (root == NULL)
    {
        root = newNode;
        return root;
    }

    if (value < root->data)
    {
        root->leftchild = insertionBST(root->leftchild, value);
    }
    else if (value > root->data)
    {
        root->rightchild = insertionBST(root->rightchild, value);
    }
    return root;
}

treeNode *searchBST(treeNode *root, int value)
{
    if (root == NULL)
    {
        return NULL;
    }
    if (root->data == value)
    {
        cout << root->data;
        return root;
    }
    if (value < root->data)
    {
        cout << root->data << "->";
        searchBST(root->leftchild, value);
    }

    else
    {
        cout << root->data << "->";
        searchBST(root->rightchild, value);
    }
}

treeNode *inordersucc(treeNode *root)
{
    treeNode *curr = root;
    while (curr->leftchild != NULL)
    {
        curr = curr->leftchild;
```

```c
    }
    return curr;
}

treeNode *deliationBST(treeNode *root, int value)
{

    if (value < root->data)
    {

        root->leftchild = deliationBST(root->leftchild, value);
    }

    else if (value > root->data)
    {

        root->rightchild = deliationBST(root->rightchild, value);
    }
    else
    {
        if (root->rightchild == NULL)
        {
            treeNode *tmp = root->leftchild;
            free(root);
            return tmp;
        }
        else if (root->leftchild == NULL)
        {
            treeNode *tmp = root->rightchild;
            free(root);
            return tmp;
        }
        else
        {
            treeNode *tmp = inordersucc(root->rightchild);
            root->data = tmp->data;
            root->rightchild = deliationBST(root->rightchild, tmp->data);
        }
        return root;
    }
```

```
}

void zigzagTravarsal(treeNode *root)
{

    stack<treeNode *> currentLevel;
    stack<treeNode *> nextLevel;

    bool lefttoRight = true;

    currentLevel.push(root);

    while (!currentLevel.empty())
    {

        treeNode * x = currentLevel.top();
        currentLevel.pop();

        cout<<x->data << " ";

        if (lefttoRight)
        {
            if (x->leftchild)
            {
                nextLevel.push(x->leftchild);
            }
            if (x->rightchild)
            {
                nextLevel.push(x->rightchild);
            }
        }
        else
        {
            if (x->rightchild)
            {
                nextLevel.push(x->rightchild);
            }
            if (x->leftchild)
            {
                nextLevel.push(x->leftchild);
            }
        }
        if (currentLevel.empty())
        {
            lefttoRight != lefttoRight;
            swap(currentLevel, nextLevel);
        }
    }
}

int main()
{

    int n;
    cin >> n;
    treeNode *root = NULL;
    for (int i = 0; i < n; i++)
    {
        int value;
        cin >> value;
        root = insertionBST(root, value);
    }
    string travarsal = "";
    inOrder(root, travarsal);
    cout << travarsal << endl;

    zigzagTravarsal(root);
    return 0;
}

/*

10

11 5 9 43 34 1 2 7 8 21

*/
```