```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>

struct node
{
    struct node *left;
    int data;
    struct node *right;
};

struct node *root = NULL, *newnode;

void insertAt(struct node *new, struct node *t)
{
    if (root == NULL)
    {
        root = newnode;
    }
    else if (t->data > new->data)
    {
        if (t->left == NULL)
        {
            t->left = new;
        }
        else
        {
            insertAt(new, t->left);
        }
    }
    else if (t->data < new->data)
    {
        if (t->right == NULL)
        {
            t->right = new;
        }
        else
        {
            insertAt(new, t->right);
        }
    }
}

void insert(int n)
{
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = n;
```

```
    newnode->right = NULL;
    newnode->left = NULL;
    insertAt(newnode, root);
}

void display(struct node *temp)
{
    if (temp != NULL)
    {
        display(temp->left);
        printf("%d\n", temp->data);
        display(temp->right);
    }
}

int ancestor(struct node *root, int n1, int n2)
{
    if (root->data > n1 && root->data > n2)
    {
        return ancestor(root->left, n1, n2);
    }
    else if (root->data < n1 && root->data < n2)
    {
        return ancestor(root->right, n1, n2);
    }
    else
    {
        return root->data;
    }
}

void displayrange(struct node *temp, int n1, int n2)
{
    if (temp != NULL)
    {
        displayrange(temp->left, n1, n2);
        if (temp->data >= n1 && temp->data <= n2)
        {
            printf("%d\n", temp->data);
        }
        displayrange(temp->right, n1, n2);
    }
}

int heightTree(struct node *root)
{
    int ans;
    if (root == NULL)
```

```
    {
        return 0;
    }
    else
    {
        int leftHeight = heightTree(root->left);
        int rightHeight = heightTree(root->right);
        ans = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
        return ans;
    }
}

void smallest(struct node *root)
{
    if (root->left != NULL)
    {
        smallest(root->left);
    }
    else
    {
        printf("%d", root->data);
    }
}

void largest(struct node *root)
{
    if (root->right != NULL)
    {
        largest(root->right);
    }
    else
    {
        printf("%d", root->data);
    }
}

bool balancedbst(struct node *root, int *height)
{
    int leftHeight = 0, rightHeight = 0;
    int l = 0, r = 0;

    if (root == NULL)
    {
        *height = 0;
        return 1;
    }

    l = balancedbst(root->left, &leftHeight);
```

```
    r = balancedbst(root->right, &rightHeight);

    *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

    if ((leftHeight - rightHeight >= 2) || (rightHeight - leftHeight >= 2))
    {
        return 0;
    }
    else
    {
        return l && r;
    }
}

void main()
{
    int ch;
    int n;
    printf("number of elements in the tree: ");
    scanf("%d", &n);
    printf("enter the elements: ");
    int a[n];
    for (int i = 0; i < n; i++)
    {
        scanf("%d", &a[i]);
        insert(a[i]);
    }
    do
    {
        printf("\nEnter the operation: \n1.ancestor\n2.height of the
tree\n3.display range\n4.smallest\n5.largest\n6.balanced\n7.Exit: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
        {
            int n1, n2;
            printf("enter the two numbers of which ancestor has to be found:
");
            scanf("%d %d", &n1, &n2);
            printf("Ancestor of %d and %d is %d\n", n1, n2, ancestor(root, n1,
n2));
            break;
        }
        case 2:
        {
            printf("the height of the tree is %d\n", heightTree(root));
            break;
```

```
        }
        case 3:
        {
            int n1, n2;
            printf("enter the two numbers: ");
            scanf("%d %d", &n1, &n2);
            printf("the numbers between the two numbers in a tree are:\n");
            displayrange(root, n1, n2);
            break;
        }
        case 4:
        {
            printf("The smallest number of the tree is: ");
            smallest(root);
            break;
        }
        case 5:
        {
            printf("the largest number of the tree is: ");
            largest(root);
            break;
        }
        case 6:
        {
            int height = 0;
            if (balancedbst(root, &height))
            {
                printf("The tree is balanced\n");
            }
            else
            {
                printf("The tree is not balanced\n");
            }
            break;
        }
        }
    } while (ch != 7);
}
```

```
Enter the operation:
1)ancestor
2)hieght of the tree
3)display range
 4)smallest
5)largest 6)balanced
7)Exit
1
enter the two numbers of which ancestor has to be found 76 105
Ancestor of 76 and 105 :80
Enter the operation:
1)ancestor
2)hieght of the tree
3)display range
 4)smallest
5)largest 6)balanced
7)Exit
2
the hieght of the tree is :4
Enter the operation:
1)ancestor
2)hieght of the tree
3)display range
 4)smallest
5)largest 6)balanced
7)Exit
3
enter the two numbers 25 82
the numbers between the two numbers in a tree are:32
40
76
80
```