2h 26m left

ALL

1 2 3 4 5

3. Binary Tree Search

In a binary search tree, each node holds a *value* and a reference to as many as *2 child* nodes, or *children*. The *root* node has no ancestors. The children are called *left* and *right*, and *subtrees* rooted at *left* and *right* are the left and right subtrees. If each node is considered the root of a subtree, each node value in its left subtree must be less than its own value. Likewise, each node in its right subtree must have a greater or equal value to the root. This allows for efficient searching.

For each value in a list of integers, determine if it is present in a tree . If it is, return the integer *1*, otherwise, return *0*.

Function Description
Complete the function *isPresent* in the editor below.

isPresent has the following parameter(s):
BSTreeNode root: reference to the root node of a tree of integers
int val[q] : an array of integer items to search for
Returns:
int[q]: an integer array where each value at index *i* denotes whether *val[i]* is found in the BST or not

Constraints

- $1 \le n, q \le 10^5$
- $1 \le val[i] \le 5 \times 10^4$

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function:

The first line contains an integer, *n* , the number of elements in the tree.
Each of the next *n* lines contains an integer, the value of *node[i]* where $0 \le i \le n$ and *node[i]* is the root
The next line contains an integer, *q*, the number of queries
Each of the next *q* lines contains an integer to search for

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       -----
11      →   tree size n = 11
20      →   node values = [20, 10, 30, 8, 12, 25, 40,
6, 11, 13, 23]
10
30
8
12
25
40
6
11
13
23
4       →   val[] size q = 4
30      →   val = [30, 10, 12, 15]
10
12
15
```

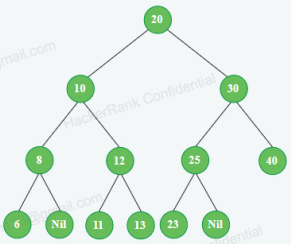
Test Values

```
30
10
12
15
```

Sample Output

```
1
1
1
0
```

Explanation



The tree is assembled as described in *Input Format for Custom Testing* by the provided code stub. Nodes marked "Nil" have no value and are placeholders to make *left* and *right* clear.

- Search for val[0] = 30. Start from the root of a tree. 30 > 20 : Search in the right subtree which has the root = 30. The item is found, return 1.
- Search for val[1] = 10. Start from the root of a tree. 10 < 20 : Search in the left subtree which has the root = 10. The item is found, return 1.
- Search for val[2] = 12. Start from the root of a tree. 12 < 20 : Search in the left subtree which has the root = 10. 12 > 10 : Search in the right subtree which has the root = 12. The item is found, return 1.

Language C | Environment | Autocomplete Ready

```c
#include <stdio.h>···

/* Any extra functions you would like to add, code here */

int isPresent(node* root, int val){
    /* For your reference
       struct node {
        struct node *left,*right;
        int val;
    };
    */
}

int main(){···
```

Line: 11 Col: 1

Test Results | Custom Input | Run Code | Run Tests | Submit

- Search for val[3] = 15. Start from the root of a tree. 15 < 20: Search in the left subtree which has the root = 10. 15 > 10 : Search in the right subtree which has the root = 12. 15 > 12 : Search in the right subtree which has the root = 13. End of the tree and the item is not found, return 0.
- The return values are [1, 1, 1, 0]

▼ Sample Case 1

Sample Input

```
STDIN       Function
-----       -----
11      →   tree size n = 11
20      →   node = [20, 10, 30, 8, 12, 25, 40, 6, 11,
13, 23]
10
30
8
12
25
40
6
11
13
23
4       →   val[] size q = 4
30      →   val = [79, 10, 20, 30, 40]
10
12
15
79
10
20
30
40
```

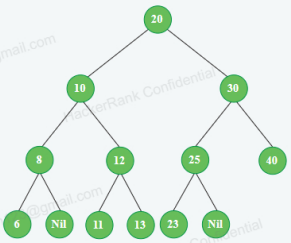
Test Values

```
79
10
20
30
40
```

Sample Output

```
0
1
1
1
1
```

Explanation



- Search for val[0] = 79. Start from the root of a tree. 79 > 20 : Search in the right subtree which has the root = 30. 79 > 30 : Search in the right subtree which has the root = 40. End of the tree and the item is not found, return 0.
- Search for val[1] = 10. Start from the root of a tree. 10 < 20 : Search in the left subtree which has the root = 10. The item is found, return 1.
- Search for val[2] = 20. Start from the root of a tree. 20 = 20 : The item is found, return 1.
- Search for val[3] = 30. Start from the root of a tree. 30 > 20 : Search in the right subtree which has the root = 30. The item is found, return 1.
- Search for val[4] = 40. Start from the root of a tree. 40 > 20 : Search in the right subtree which has the root = 30. 40 > 30: Search in the right subtree which has the root = 40. The item is found, return 1.
- The return values are [0, 1, 1, 1, 1]