

# Problem 501: Find Mode in Binary Search Tree

## Problem Information

**Difficulty:** [Easy](#)

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given the

root

of a binary search tree (BST) with duplicates, return

all the

mode(s)

(i.e., the most frequently occurred element) in it

If the tree has more than one mode, return them in

any order

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys

less than or equal to

the node's key.

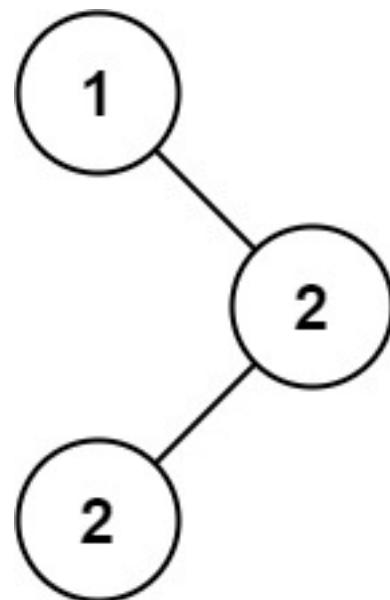
The right subtree of a node contains only nodes with keys

greater than or equal to

the node's key.

Both the left and right subtrees must also be binary search trees.

Example 1:



Input:

```
root = [1,null,2,2]
```

Output:

```
[2]
```

Example 2:

Input:

```
root = [0]
```

Output:

[0]

Constraints:

The number of nodes in the tree is in the range

[1, 10]

4

]

.

-10

5

$\leq \text{Node.val} \leq 10$

5

Follow up:

Could you do that without using any extra space? (Assume that the implicit stack space incurred due to recursion does not count).

## Code Snippets

C++:

```
/**  
 * Definition for a binary tree node.  
 * struct TreeNode {  
 *     int val;  
 *     TreeNode *left;
```

```

* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
vector<int> findMode(TreeNode* root) {

}
};


```

### Java:

```

/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public int[] findMode(TreeNode root) {

}
}


```

### Python3:

```

# Definition for a binary tree node.
# class TreeNode:
# def __init__(self, val=0, left=None, right=None):

```

```
# self.val = val
# self.left = left
# self.right = right
class Solution:

    def findMode(self, root: Optional[TreeNode]) -> List[int]:
```

### Python:

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):

    def findMode(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[int]
        """


```

### JavaScript:

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number[]}
 */
var findMode = function(root) {

};
```

### TypeScript:

```

    /**
 * Definition for a binary tree node.
 * class TreeNode {
 * val: number
 * left: TreeNode | null
 * right: TreeNode | null
 * constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
 *
 * this.val = (val===undefined ? 0 : val)
 * this.left = (left===undefined ? null : left)
 * this.right = (right===undefined ? null : right)
 *
 * }
 *
 */
function findMode(root: TreeNode | null): number[] {
}

```

### C#:

```

    /**
 * Definition for a binary tree node.
 * public class TreeNode {
 * public int val;
 * public TreeNode left;
 * public TreeNode right;
 * public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 * this.val = val;
 * this.left = left;
 * this.right = right;
 * }
 *
 * }
 *
 */
public class Solution {
public int[] FindMode(TreeNode root) {

}
}

```

### C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* findMode(struct TreeNode* root, int* returnSize) {

}

```

## Go:

```

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func findMode(root *TreeNode) []int {
}

```

## Kotlin:

```

/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 *
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */
class Solution {
    fun findMode(root: TreeNode?): IntArray {
}

```

```
}
```

```
}
```

## Swift:

```
/**  
 * Definition for a binary tree node.  
 * public class TreeNode {  
 *     public var val: Int  
 *     public var left: TreeNode?  
 *     public var right: TreeNode?  
 *     public init() { self.val = 0; self.left = nil; self.right = nil; }  
 *     public init(_ val: Int) { self.val = val; self.left = nil; self.right = nil; }  
 *     public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {  
 *         self.val = val  
 *         self.left = left  
 *         self.right = right  
 *     }  
 * }  
 * }  
 */  
class Solution {  
    func findMode(_ root: TreeNode?) -> [Int] {  
        ...  
    }  
}
```

## Rust:

```
// Definition for a binary tree node.  
// #[derive(Debug, PartialEq, Eq)]  
// pub struct TreeNode {  
//     pub val: i32,  
//     pub left: Option<Rc<RefCell<TreeNode>>>,  
//     pub right: Option<Rc<RefCell<TreeNode>>>,  
// }  
//  
// impl TreeNode {  
//     #[inline]  
//     pub fn new(val: i32) -> Self {  
//         TreeNode {
```

```

// val,
// left: None,
// right: None
// }
// }
// }

use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
pub fn find_mode(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<i32> {
    }

}
}
}

```

## Ruby:

```

# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
#   @val = val
#   @left = left
#   @right = right
# end
# end
# @param {TreeNode} root
# @return {Integer[]}
def find_mode(root)

end

```

## PHP:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *     public $val = null;
 *     public $left = null;
 *     public $right = null;
 *     function __construct($val = 0, $left = null, $right = null) {
 *         $this->val = $val;
 *         $this->left = $left;
 *     }
 * }

```

```

* $this->right = $right;
* }
* }
*/
class Solution {

/**
* @param TreeNode $root
* @return Integer[]
*/
function findMode($root) {

}

}

```

### Dart:

```

/***
* Definition for a binary tree node.
* class TreeNode {
* int val;
* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
* }
*/
class Solution {
List<int> findMode(TreeNode? root) {

}
}

```

### Scala:

```

/**
* Definition for a binary tree node.
* class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
* var value: Int = _value
* var left: TreeNode = _left
* var right: TreeNode = _right
* }

```

```

*/
object Solution {
def findMode(root: TreeNode): Array[Int] = {

}
}

```

### Elixir:

```

# Definition for a binary tree node.
#
# defmodule TreeNode do
# @type t :: %__MODULE__
# val: integer,
# left: TreeNode.t() | nil,
# right: TreeNode.t() | nil
# }
# defstruct val: 0, left: nil, right: nil
# end

defmodule Solution do
@spec find_mode(TreeNode.t() | nil) :: [integer]
def find_mode(root) do

end
end

```

### Erlang:

```

%% Definition for a binary tree node.
%%
%% -record(tree_node, {val = 0 :: integer(),
%% left = null :: 'null' | #tree_node{},
%% right = null :: 'null' | #tree_node{}}).

-spec find_mode(tree_node() | null) -> [integer()].
find_mode(Root) ->
.
```

### Racket:

```

; Definition for a binary tree node.
#|
;

; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
  (val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
  (tree-node val #f #f))

|#
(define/contract (find-mode root)
  (-> (or/c tree-node? #f) (listof exact-integer?)))
)
```

## Solutions

### C++ Solution:

```

/*
 * Problem: Find Mode in Binary Search Tree
 * Difficulty: Easy
 * Tags: tree, search, stack
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 */
```

```

* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
vector<int> findMode(TreeNode* root) {

}
};

```

### Java Solution:

```

/**
 * Problem: Find Mode in Binary Search Tree
 * Difficulty: Easy
 * Tags: tree, search, stack
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
class Solution {
public int[] findMode(TreeNode root) {

```

```
}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Find Mode in Binary Search Tree
Difficulty: Easy
Tags: tree, search, stack

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:

    def findMode(self, root: Optional[TreeNode]) -> List[int]:
        # TODO: Implement optimized solution
        pass
```

### Python Solution:

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):

    def findMode(self, root):
        """

:type root: Optional[TreeNode]
:rtype: List[int]
"""
```

## JavaScript Solution:

```
/**  
 * Problem: Find Mode in Binary Search Tree  
 * Difficulty: Easy  
 * Tags: tree, search, stack  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
/**  
 * Definition for a binary tree node.  
 * function TreeNode(val, left, right) {  
 *   this.val = (val===undefined ? 0 : val)  
 *   this.left = (left===undefined ? null : left)  
 *   this.right = (right===undefined ? null : right)  
 * }  
 */  
/**  
 * @param {TreeNode} root  
 * @return {number[]}   
 */  
var findMode = function(root) {  
  
};
```

## TypeScript Solution:

```
/**  
 * Problem: Find Mode in Binary Search Tree  
 * Difficulty: Easy  
 * Tags: tree, search, stack  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
/**  
 * Definition for a binary tree node.  
 * class TreeNode {
```

```

* val: number
* left: TreeNode | null
* right: TreeNode | null
* constructor(val?: number, left?: TreeNode | null, right?: TreeNode | null)
{
  this.val = (val==undefined ? 0 : val)
  this.left = (left==undefined ? null : left)
  this.right = (right==undefined ? null : right)
}
*/
function findMode(root: TreeNode | null): number[] {
}

```

## C# Solution:

```

/*
 * Problem: Find Mode in Binary Search Tree
 * Difficulty: Easy
 * Tags: tree, search, stack
 *
 * Approach: DFS or BFS traversal
 * Time Complexity: O(n) where n is number of nodes
 * Space Complexity: O(h) for recursion stack where h is height
 */

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left;
 *     public TreeNode right;
 *     public TreeNode(int val=0, TreeNode left=null, TreeNode right=null) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */

```

```
public class Solution {  
    public int[] FindMode(TreeNode root) {  
  
    }  
}
```

### C Solution:

```
/*  
 * Problem: Find Mode in Binary Search Tree  
 * Difficulty: Easy  
 * Tags: tree, search, stack  
 *  
 * Approach: DFS or BFS traversal  
 * Time Complexity: O(n) where n is number of nodes  
 * Space Complexity: O(h) for recursion stack where h is height  
 */  
  
/**  
 * Definition for a binary tree node.  
 * struct TreeNode {  
 *     int val;  
 *     struct TreeNode *left;  
 *     struct TreeNode *right;  
 * };  
 */  
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* findMode(struct TreeNode* root, int* returnSize) {  
  
}
```

### Go Solution:

```
// Problem: Find Mode in Binary Search Tree  
// Difficulty: Easy  
// Tags: tree, search, stack  
//  
// Approach: DFS or BFS traversal  
// Time Complexity: O(n) where n is number of nodes
```

```

// Space Complexity: O(h) for recursion stack where h is height

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func findMode(root *TreeNode) []int {
}

```

### Kotlin Solution:

```

/**
 * Example:
 * var ti = TreeNode(5)
 * var v = ti.`val`
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */
class Solution {
    fun findMode(root: TreeNode?): IntArray {
        }
    }
}

```

### Swift Solution:

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     public var val: Int
 *     public var left: TreeNode?
 *     public var right: TreeNode?
 *     public init() { self.val = 0; self.left = nil; self.right = nil; }
 */

```

```

* public init(_ val: Int) { self.val = val; self.left = nil; self.right =
nil; }
* public init(_ val: Int, _ left: TreeNode?, _ right: TreeNode?) {
*   self.val = val
*   self.left = left
*   self.right = right
* }
* }
*/
class Solution {
func findMode(_ root: TreeNode?) -> [Int] {

}
}

```

### Rust Solution:

```

// Problem: Find Mode in Binary Search Tree
// Difficulty: Easy
// Tags: tree, search, stack
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// Definition for a binary tree node.
// #[derive(Debug, PartialEq, Eq)]
// pub struct TreeNode {
//   pub val: i32,
//   pub left: Option<Rc<RefCell<TreeNode>>,
//   pub right: Option<Rc<RefCell<TreeNode>>,
// }
//
// impl TreeNode {
//   #[inline]
//   pub fn new(val: i32) -> Self {
//     TreeNode {
//       val,
//       left: None,
//       right: None
//     }
//   }
// }

```

```

    // }
    // }

use std::rc::Rc;
use std::cell::RefCell;
impl Solution {
    pub fn find_mode(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<i32> {
        }

    }
}

```

### Ruby Solution:

```

# Definition for a binary tree node.
# class TreeNode
# attr_accessor :val, :left, :right
# def initialize(val = 0, left = nil, right = nil)
#   @val = val
#   @left = left
#   @right = right
# end
# end

# @param {TreeNode} root
# @return {Integer[]}
def find_mode(root)

end

```

### PHP Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 *     public $val = null;
 *     public $left = null;
 *     public $right = null;
 *     function __construct($val = 0, $left = null, $right = null) {
 *         $this->val = $val;
 *         $this->left = $left;
 *         $this->right = $right;
 *     }
 * }

```

```

*/
class Solution {

/**
 * @param TreeNode $root
 * @return Integer[]
 */
function findMode($root) {

}
}

```

### Dart Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 * int val;
 * TreeNode? left;
 * TreeNode? right;
 * TreeNode([this.val = 0, this.left, this.right]);
 * }
 */
class Solution {
List<int> findMode(TreeNode? root) {

}
}

```

### Scala Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(_value: Int = 0, _left: TreeNode = null, _right: TreeNode =
null) {
 * var value: Int = _value
 * var left: TreeNode = _left
 * var right: TreeNode = _right
 * }
 */
object Solution {

```

```
def findMode(root: TreeNode): Array[Int] = {  
    }  
    }
```

### Elixir Solution:

```
# Definition for a binary tree node.  
#  
# defmodule TreeNode do  
# @type t :: %__MODULE__{  
# val: integer,  
# left: TreeNode.t() | nil,  
# right: TreeNode.t() | nil  
# }  
# defstruct val: 0, left: nil, right: nil  
# end  
  
defmodule Solution do  
@spec find_mode(TreeNode.t() | nil) :: [integer]  
def find_mode(root) do  
  
end  
end
```

### Erlang Solution:

```
%% Definition for a binary tree node.  
%%  
%% -record(tree_node, {val = 0 :: integer(),  
%% left = null :: 'null' | #tree_node{},  
%% right = null :: 'null' | #tree_node{}}).  
  
-spec find_mode(tree_node() | null) -> [integer()].  
find_mode(Root) ->  
.
```

### Racket Solution:

```
; Definition for a binary tree node.  
#|
```

```
; val : integer?
; left : (or/c tree-node? #f)
; right : (or/c tree-node? #f)
(struct tree-node
  (val left right) #:mutable #:transparent)

; constructor
(define (make-tree-node [val 0])
  (tree-node val #f #f))

|#

(define/contract (find-mode root)
  (-> (or/c tree-node? #f) (listof exact-integer?)))
)
```