

# Problem 235: Lowest Common Ancestor of a Binary Search Tree

## Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

According to the

definition of LCA on Wikipedia

: "The lowest common ancestor is defined between two nodes

$p$

and

$q$

as the lowest node in

$T$

that has both

$p$

and

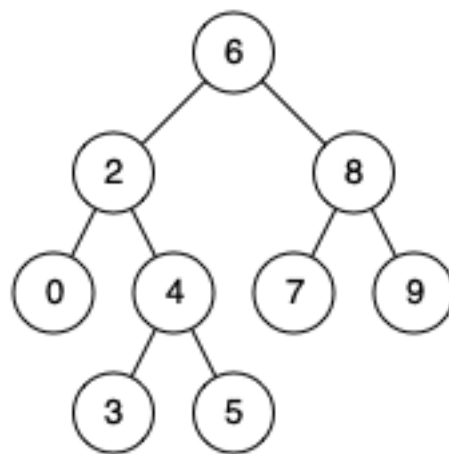
q

as descendants (where we allow

a node to be a descendant of itself

).”

Example 1:



Input:

root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

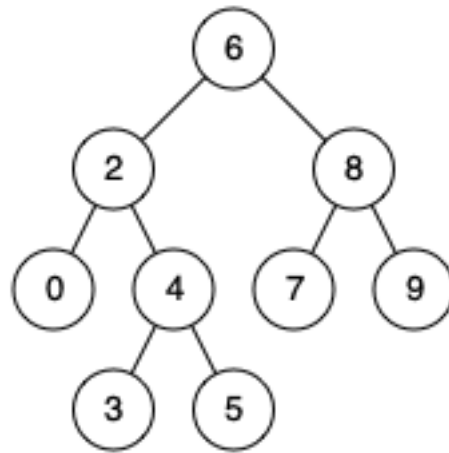
Output:

6

Explanation:

The LCA of nodes 2 and 8 is 6.

Example 2:



Input:

root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

Output:

2

Explanation:

The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

Example 3:

Input:

root = [2,1], p = 2, q = 1

Output:

2

Constraints:

The number of nodes in the tree is in the range

[2, 10

5

]

.

-10

9

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

9

All

Node.val

are

unique

.

$p \neq q$

p

and

q

will exist in the BST.

**Code Snippets**

## C++:

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

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {

    }
};
```

## Java:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *   int val;
 *   TreeNode left;
 *   TreeNode right;
 *   TreeNode(int x) { val = x; }
 * }
 */

class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

    }
}
```

## Python3:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
```

```

# self.left = None
# self.right = None

class Solution:
def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':

```

## Python:

```

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

class Solution(object):
def lowestCommonAncestor(self, root, p, q):
    """
    :type root: TreeNode
    :type p: TreeNode
    :type q: TreeNode
    :rtype: TreeNode
    """

```

## JavaScript:

```

/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *   this.val = val;
 *   this.left = this.right = null;
 * }
 */

/**
 * @param {TreeNode} root
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {TreeNode}
 */
var lowestCommonAncestor = function(root, p, q) {

```

```
};
```

## 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 lowestCommonAncestor(root: TreeNode | null, p: TreeNode | null, q:
TreeNode | null): TreeNode | null {

};
```

## C#:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *   public int val;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int x) { val = x; }
 * }
 */

public class Solution {
    public TreeNode LowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

    }
}
```

**C:**

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

struct TreeNode* lowestCommonAncestor(struct TreeNode* root, struct TreeNode*
p, struct TreeNode* q) {

}
```

**Go:**

```
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *   Val int
 *   Left *TreeNode
 *   Right *TreeNode
 * }
 */

func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {

}
```

**Kotlin:**

```
/**
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int = 0) {
 *   var left: TreeNode? = null
 *   var right: TreeNode? = null
 * }
 */

class Solution {
    fun lowestCommonAncestor(root: TreeNode?, p: TreeNode?, q: TreeNode?):
```

```
TreeNode? {

}

}
```

## Swift:

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

class Solution {
func lowestCommonAncestor(_ root: TreeNode?, _ p: TreeNode?, _ q: TreeNode?)
-> TreeNode? {

}

}
```

## 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 lowest_common_ancestor(root: Option<Rc<RefCell<TreeNode>>>, p:
    Option<Rc<RefCell<TreeNode>>>, q: Option<Rc<RefCell<TreeNode>>>) ->
    Option<Rc<RefCell<TreeNode>>> {

    }
    }

```

## Ruby:

```

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

# @param {TreeNode} root
# @param {TreeNode} p
# @param {TreeNode} q
# @return {TreeNode}
def lowest_common_ancestor(root, p, q)

end

```

## PHP:

```

/**
 * Definition for a binary tree node.
 * class TreeNode {
 * public $val = null;
 * public $left = null;
 * public $right = null;

```

```

* function __construct($value) { $this->val = $value; }
* }
*/

class Solution {
/**
 * @param TreeNode $root
 * @param TreeNode $p
 * @param TreeNode $q
 * @return TreeNode
 */
function lowestCommonAncestor($root, $p, $q) {

}
}

```

## Scala:

```

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

object Solution {
  def lowestCommonAncestor(root: TreeNode, p: TreeNode, q: TreeNode): TreeNode
  = {

  }
}

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Lowest Common Ancestor of a Binary Search Tree

```

```

* Difficulty: Medium
* Tags: tree, search
*
* 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(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {

    }
};

```

## Java Solution:

```

/**
 * Problem: Lowest Common Ancestor of a Binary Search Tree
 * Difficulty: Medium
 * Tags: tree, search
 *
 * 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(int x) { val = x; }
* }
*/

class Solution {
public:
    TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Lowest Common Ancestor of a Binary Search Tree
Difficulty: Medium
Tags: tree, search

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, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

# Definition for a binary tree node.
# class TreeNode(object):

```

```

# def __init__(self, x):
# self.val = x
# self.left = None
# self.right = None

class Solution(object):
def lowestCommonAncestor(self, root, p, q):
    """
    :type root: TreeNode
    :type p: TreeNode
    :type q: TreeNode
    :rtype: TreeNode
    """

```

## JavaScript Solution:

```

/**
 * Problem: Lowest Common Ancestor of a Binary Search Tree
 * Difficulty: Medium
 * Tags: tree, search
 *
 * 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) {
 *   this.val = val;
 *   this.left = this.right = null;
 * }
 */

/**
 * @param {TreeNode} root
 * @param {TreeNode} p
 * @param {TreeNode} q
 * @return {TreeNode}
 */
var lowestCommonAncestor = function(root, p, q) {

```

```
};
```

### TypeScript Solution:

```
/**
 * Problem: Lowest Common Ancestor of a Binary Search Tree
 * Difficulty: Medium
 * Tags: tree, search
 *
 * 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 lowestCommonAncestor(root: TreeNode | null, p: TreeNode | null, q:
TreeNode | null): TreeNode | null {

};
```

### C# Solution:

```
/*
 * Problem: Lowest Common Ancestor of a Binary Search Tree
 * Difficulty: Medium
 * Tags: tree, search
```

```

*
* 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 x) { val = x; }
* }
*/

public class Solution {
public TreeNode LowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

}
}

```

## C Solution:

```

/*
* Problem: Lowest Common Ancestor of a Binary Search Tree
* Difficulty: Medium
* Tags: tree, search
*
* 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;
* };

```

```

*/

struct TreeNode* lowestCommonAncestor(struct TreeNode* root, struct TreeNode*
p, struct TreeNode* q) {

}

```

## Go Solution:

```

// Problem: Lowest Common Ancestor of a Binary Search Tree
// Difficulty: Medium
// Tags: tree, search
//
// 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 lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {

}

```

## Kotlin Solution:

```

/**
 * Definition for a binary tree node.
 * class TreeNode(var `val`: Int = 0) {
 *     var left: TreeNode? = null
 *     var right: TreeNode? = null
 * }
 */

class Solution {

```

```

fun lowestCommonAncestor(root: TreeNode?, p: TreeNode?, q: TreeNode?):
TreeNode? {

}

}

```

### 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(_ val: Int) {
 *         self.val = val
 *         self.left = nil
 *         self.right = nil
 *     }
 * }
 */

class Solution {
    func lowestCommonAncestor(_ root: TreeNode?, _ p: TreeNode?, _ q: TreeNode?)
    -> TreeNode? {

    }

}

```

### Rust Solution:

```

// Problem: Lowest Common Ancestor of a Binary Search Tree
// Difficulty: Medium
// Tags: tree, search
//
// 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 lowest_common_ancestor(root: Option<Rc<RefCell<TreeNode>>>, p:
Option<Rc<RefCell<TreeNode>>>, q: Option<Rc<RefCell<TreeNode>>>) ->
Option<Rc<RefCell<TreeNode>>> {

  }
}

```

## Ruby Solution:

```

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

# @param {TreeNode} root
# @param {TreeNode} p
# @param {TreeNode} q
# @return {TreeNode}

```

```
def lowest_common_ancestor(root, p, q)

end
```

### PHP Solution:

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

class Solution {
/**
 * @param TreeNode $root
 * @param TreeNode $p
 * @param TreeNode $q
 * @return TreeNode
 */
function lowestCommonAncestor($root, $p, $q) {

}

}
```

### Scala Solution:

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

object Solution {
def lowestCommonAncestor(root: TreeNode, p: TreeNode, q: TreeNode): TreeNode
```

