

Problem 145: Binary Tree Postorder Traversal

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the

root

of a binary tree, return

the postorder traversal of its nodes' values

Example 1:

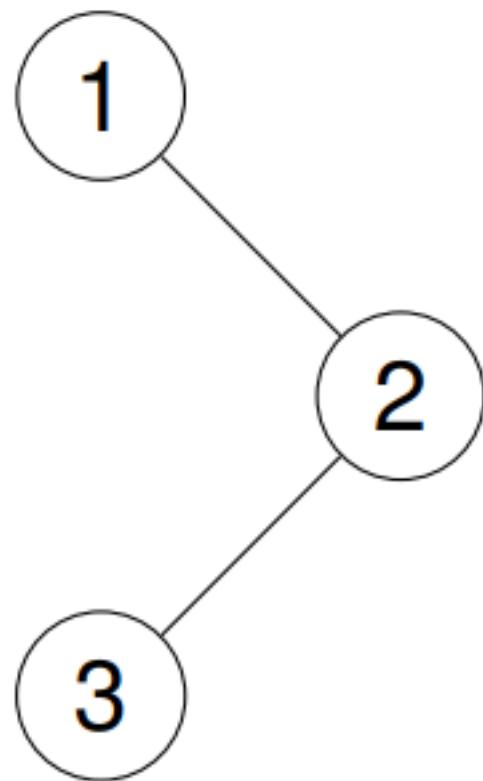
Input:

root = [1,null,2,3]

Output:

[3,2,1]

Explanation:



Example 2:

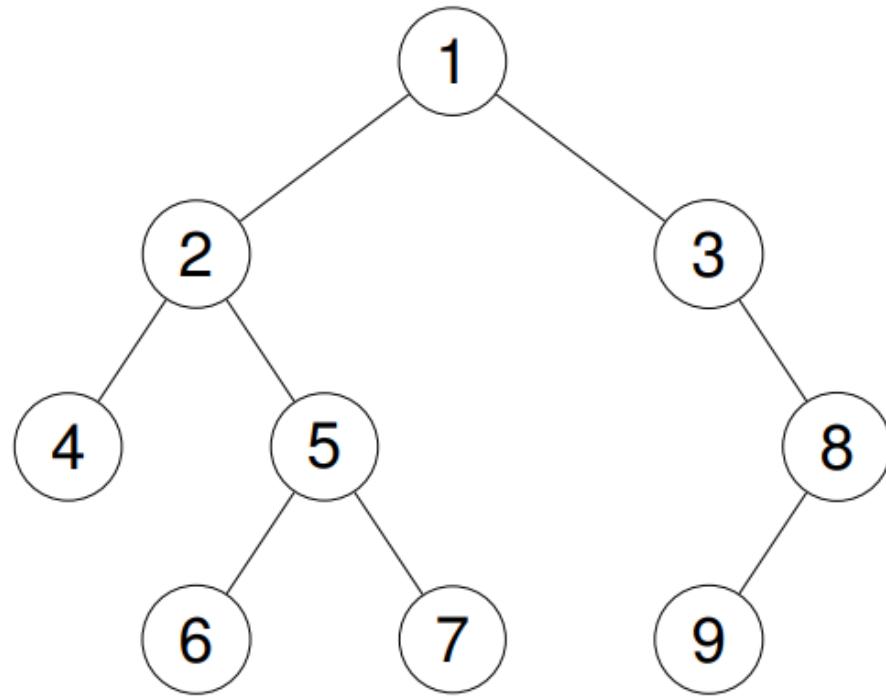
Input:

```
root = [1,2,3,4,5,null,8,null,null,6,7,9]
```

Output:

```
[4,6,7,5,2,9,8,3,1]
```

Explanation:



Example 3:

Input:

```
root = []
```

Output:

```
[]
```

Example 4:

Input:

```
root = [1]
```

Output:

```
[1]
```

Constraints:

The number of the nodes in the tree is in the range

[0, 100]

.

-100 <= Node.val <= 100

Follow up:

Recursive solution is trivial, could you do it iteratively?

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> postorderTraversal(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 List<Integer> postorderTraversal(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 postorderTraversal(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 postorderTraversal(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 postorderTraversal = 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 postorderTraversal(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 IList<int> PostorderTraversal(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* postorderTraversal(struct TreeNode* root, int* returnSize) {
    }
}

```

Go:

```

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

```

```

    * Right *TreeNode
    *
    */
func postorderTraversal(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 postorderTraversal(root: TreeNode?): List<Int> {
        }
    }
}

```

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 postorderTraversal(_ 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 postorder_traversal(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 postorder_traversal(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 $root
 * @return Integer[]
 */
function postorderTraversal($root) {

}
}

```

Dart:

```

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

```

```

* TreeNode? left;
* TreeNode? right;
* TreeNode([this.val = 0, this.left, this.right]);
*
*/
class Solution {
List<int> postorderTraversal(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 postorderTraversal(root: TreeNode): List[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 postorder_traversal(TreeNode.t | nil) :: [integer]
def postorder_traversal(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 postorder_traversal(Root :: #tree_node{} | null) -> [integer()].
postorder_traversal(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 (postorder-traversal root)
  (-> (or/c tree-node? #f) (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Binary Tree Postorder Traversal
 * 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> postorderTraversal(TreeNode* root) {
        }

    };
}
```

Java Solution:

```
/**
 * Problem: Binary Tree Postorder Traversal
 * 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 List<Integer> postorderTraversal(TreeNode root) {
}

}
}

```

Python3 Solution:

```

"""
Problem: Binary Tree Postorder Traversal
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 postorderTraversal(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 postorderTraversal(self, root):
#         """
#         :type root: Optional[TreeNode]
#         :rtype: List[int]
#         """
#         ...
```

JavaScript Solution:

```
/**
 * Problem: Binary Tree Postorder Traversal
 * 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 postorderTraversal = function(root) {  
};
```

TypeScript Solution:

```
/**  
 * Problem: Binary Tree Postorder Traversal  
 * 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 postorderTraversal(root: TreeNode | null): number[] {  
};
```

C# Solution:

```
/*  
 * Problem: Binary Tree Postorder Traversal  
 * 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 IList<int> PostorderTraversal(TreeNode root) {
}
}

```

C Solution:

```

/*
* Problem: Binary Tree Postorder Traversal
* 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* postorderTraversal(struct TreeNode* root, int* returnSize) {

}

```

Go Solution:

```

// Problem: Binary Tree Postorder Traversal
// 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 postorderTraversal(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 postorderTraversal(root: TreeNode?): List<Int> {
        ...
    }
}

```

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 postorderTraversal(_ root: TreeNode?) -> [Int] {
        ...
    }
}

```

Rust Solution:

```

// Problem: Binary Tree Postorder Traversal
// 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 postorder_traversal(root: Option<Rc<RefCell<TreeNode>>>) -> Vec<i32> {
        let mut result = Vec::new();
        if let Some(node) = root {
            let node_ref = node.borrow();
            result.extend(postorder_traversal(node_ref.left.clone()));
            result.extend(postorder_traversal(node_ref.right.clone()));
            result.push(node_ref.val);
        }
        result
    }
}

```

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 postorder_traversal(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 $root
 * @return Integer[]
 */
function postorderTraversal($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> postorderTraversal(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 postorderTraversal(root: TreeNode): List[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 postorder_traversal(root :: TreeNode.t() | nil) :: [integer]
def postorder_traversal(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 postorder_traversal(Root :: #tree_node{} | null) -> [integer()].  
postorder_traversal(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 (postorder-traversal root)  
(-> (or/c tree-node? #f) (listof exact-integer?))  
)
```