

# Problem 2471: Minimum Number of Operations to Sort a Binary Tree by Level

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given the

root

of a binary tree with

unique values

In one operation, you can choose any two nodes

at the same level

and swap their values.

Return

the minimum number of operations needed to make the values at each level sorted in a

strictly increasing order

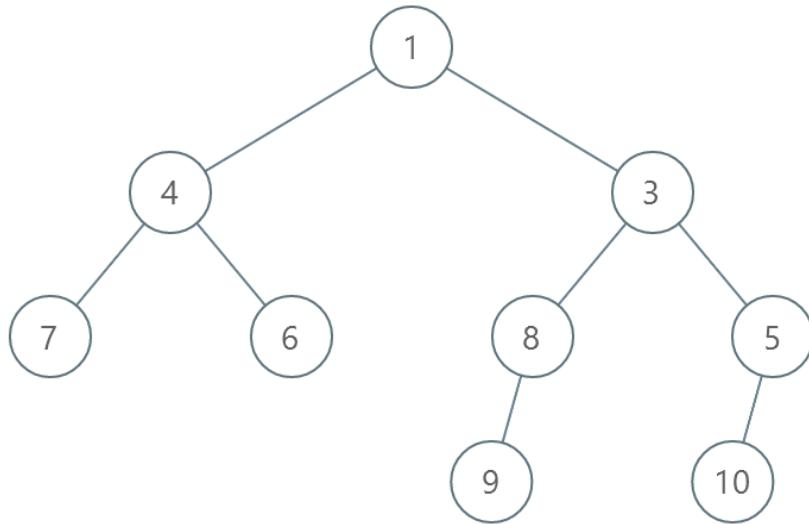
The

level

of a node is the number of edges along the path between it and the root node

.

Example 1:



Input:

root = [1,4,3,7,6,8,5,null,null,null,null,9,null,10]

Output:

3

Explanation:

- Swap 4 and 3. The 2

nd

level becomes [3,4]. - Swap 7 and 5. The 3

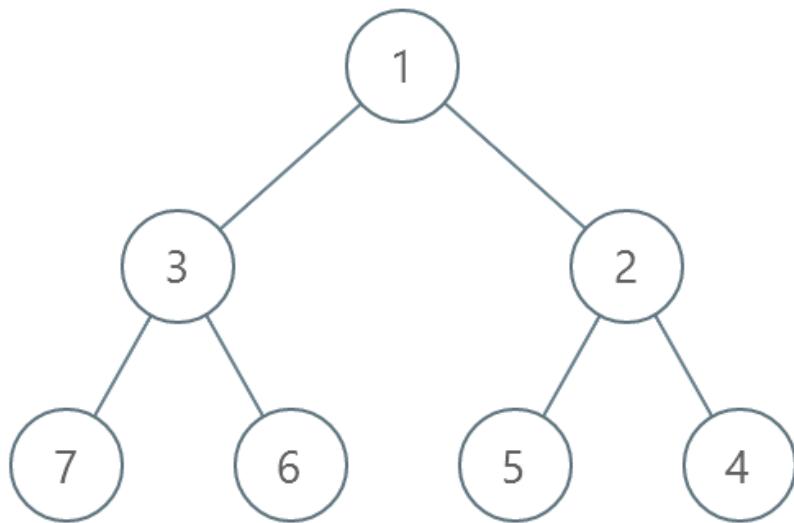
rd

level becomes [5,6,8,7]. - Swap 8 and 7. The 3

rd

level becomes [5,6,7,8]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed.

Example 2:



Input:

root = [1,3,2,7,6,5,4]

Output:

3

Explanation:

- Swap 3 and 2. The 2

nd

level becomes [2,3]. - Swap 7 and 4. The 3

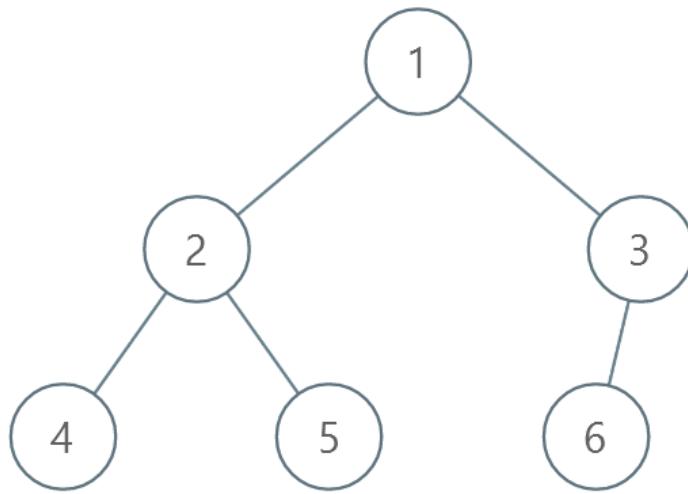
rd

level becomes [4,6,5,7]. - Swap 6 and 5. The 3

rd

level becomes [4,5,6,7]. We used 3 operations so return 3. It can be proven that 3 is the minimum number of operations needed.

Example 3:



Input:

root = [1,2,3,4,5,6]

Output:

0

Explanation:

Each level is already sorted in increasing order so return 0.

Constraints:

The number of nodes in the tree is in the range

[1, 10]

5

]

1 <= Node.val <= 10

5

All the values of the tree are

unique

## 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:
int minimumOperations(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 minimumOperations(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 minimumOperations(self, root: Optional[TreeNode]) -> 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 minimumOperations(self, root):
        """
:type root: Optional[TreeNode]
:rtype: 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 minimumOperations = 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 minimumOperations(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 MinimumOperations(TreeNode root) {
        }
    }
}

```

## C:

```

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

```

```
int minimumOperations(struct TreeNode* root) {  
}  
}
```

### Go:

```
/**  
 * Definition for a binary tree node.  
 * type TreeNode struct {  
 *     Val int  
 *     Left *TreeNode  
 *     Right *TreeNode  
 * }  
 */  
func minimumOperations(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 minimumOperations(root: TreeNode?): 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 minimumOperations(_ 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 minimum_operations(root: Option<Rc<RefCell<TreeNode>>>) -> 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 minimum_operations(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 minimumOperations($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 {  
int minimumOperations(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 minimumOperations(root: TreeNode): 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 minimum_operations(root :: TreeNode.t | nil) :: integer
def minimum_operations(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 minimum_operations(Root :: #tree_node{} | null) -> integer().
minimum_operations(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 (minimum-operations root)
  (-> (or/c tree-node? #f) exact-integer?))
)

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level
 * Difficulty: Medium
 * Tags: tree, sort, 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() : val(0), left(nullptr), right(nullptr) {
 *         // TODO: Implement optimized solution
 *         return 0;
 *     }
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {
 *         // TODO: Implement optimized solution
 *         return 0;
 *     }
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
 *     right(right) {
 *         // TODO: Implement optimized solution
 *         return 0;
 *     }
 */

```

```

    }
* } ;
*/
class Solution {
public:
int minimumOperations(TreeNode* root) {
    }
} ;

```

### Java Solution:

```

/**
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level
 * Difficulty: Medium
 * Tags: tree, sort, 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() {
// TODO: Implement optimized solution
return 0;
}
* 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 minimumOperations(TreeNode root) {  
    }  
}
```

### Python3 Solution:

```
"""  
Problem: Minimum Number of Operations to Sort a Binary Tree by Level  
Difficulty: Medium  
Tags: tree, sort, 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, val=0, left=None, right=None):  
#         self.val = val  
#         self.left = left  
#         self.right = right  
class Solution:  
    def minimumOperations(self, root: Optional[TreeNode]) -> 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 minimumOperations(self, root):  
        """  
        :type root: Optional[TreeNode]  
        :rtype: int
```

```
"""
```

### JavaScript Solution:

```
/**  
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level  
 * Difficulty: Medium  
 * Tags: tree, sort, 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, 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 minimumOperations = function(root) {  
  
};
```

### TypeScript Solution:

```
/**  
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level  
 * Difficulty: Medium  
 * Tags: tree, sort, 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 minimumOperations(root: TreeNode | null): number {
}

```

## C# Solution:

```

/*
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level
 * Difficulty: Medium
 * Tags: tree, sort, 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 val=0, TreeNode left=null, TreeNode right=null) {
 * this.val = val;
 * this.left = left;

```

```

        * this.right = right;
        *
        *
    }

    public class Solution {
        public int MinimumOperations(TreeNode root) {
    }
}

```

## C Solution:

```

/*
 * Problem: Minimum Number of Operations to Sort a Binary Tree by Level
 * Difficulty: Medium
 * Tags: tree, sort, 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;
 * };
 */
int minimumOperations(struct TreeNode* root) {

}

```

## Go Solution:

```

// Problem: Minimum Number of Operations to Sort a Binary Tree by Level
// Difficulty: Medium
// Tags: tree, sort, 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 minimumOperations(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 minimumOperations(root: TreeNode?): 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 minimumOperations(_ root: TreeNode?) -> Int {
}
}

```

## Rust Solution:

```

// Problem: Minimum Number of Operations to Sort a Binary Tree by Level
// Difficulty: Medium
// Tags: tree, sort, 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 minimum_operations(root: Option<Rc<RefCell<TreeNode>>>) -> 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 minimum_operations(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 $root
     * @return Integer
     */
    function minimumOperations($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 {
  int minimumOperations(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 minimumOperations(root: TreeNode): 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 minimum_operations(TreeNode.t() | nil) :: integer
def minimum_operations(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 minimum_operations(#tree_node{} | null) -> integer().
minimum_operations(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 (minimum-operations root)  
(-> (or/c tree-node? #f) exact-integer?)  
)
```