

Problem 108: Convert Sorted Array to Binary Search Tree

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an integer array

nums

where the elements are sorted in

ascending order

, convert

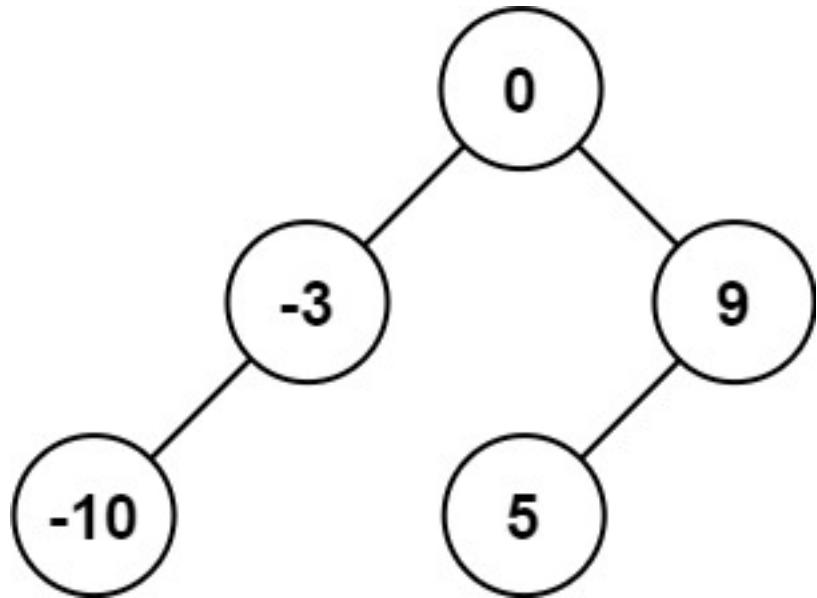
it to a

height-balanced

binary search tree

.

Example 1:



Input:

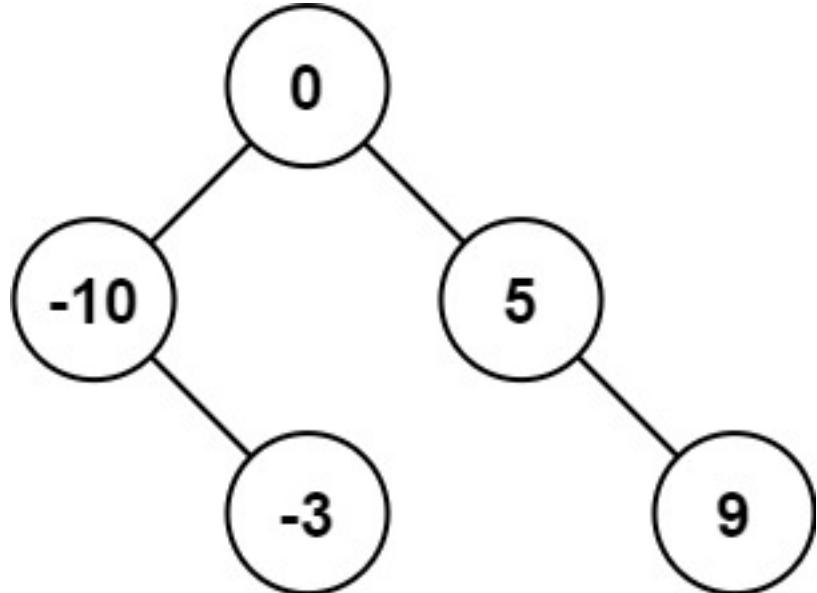
nums = [-10,-3,0,5,9]

Output:

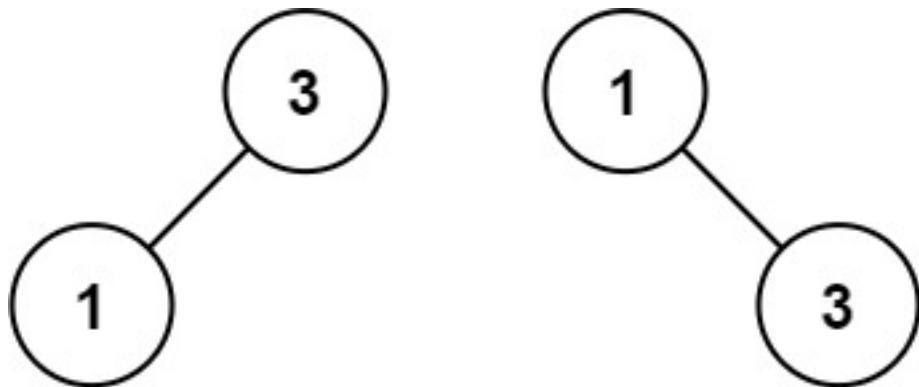
[0,-3,9,-10,null,5]

Explanation:

[0,-10,5,null,-3,null,9] is also accepted:



Example 2:



Input:

nums = [1,3]

Output:

[3,1]

Explanation:

[1,null,3] and [3,1] are both height-balanced BSTs.

Constraints:

$1 \leq \text{nums.length} \leq 10$

4

-10

4

$\leq \text{nums}[i] \leq 10$

4

nums

is sorted in a

strictly increasing

order.

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:
    TreeNode* sortedArrayToBST(vector<int>& nums) {

    }
};
```

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;
 *     }
 * }
```

```

* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public TreeNode sortedArrayToBST(int[] nums) {

}
}

```

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 sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

```

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 sortedArrayToBST(self, nums):
        """
:type nums: List[int]
:rtype: Optional[TreeNode]
"""

```

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 {number[]} nums
* @return {TreeNode}
*/
var sortedArrayToBST = function(nums) {

};

```

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 sortedArrayToBST(nums: number[]): TreeNode | null {

};

```

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 TreeNode SortedArrayToBST(int[] nums) {
        }
    }
}

```

C:

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* sortedArrayToBST(int* nums, int numsSize) {
    }
}

```

Go:

```

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

```

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 sortedArrayToBST(nums: IntArray): 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() { 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 sortedArrayToBST(_ nums: [Int]) -> 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 sorted_array_to_bst(nums: Vec<i32>) -> Option<Rc<RefCell<TreeNode>>> {
        }
    }
}

```

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 {Integer[]} nums
# @return {TreeNode}
def sorted_array_to_bst(nums)

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 Integer[] $nums
     * @return TreeNode
     */
    function sortedArrayToBST($nums) {

    }
}
```

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 {
    TreeNode? sortedArrayToBST(List<int> nums) {
    }
}
```

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 sortedArrayToBST(nums: Array[Int]): TreeNode = {  
  
  }  
}
```

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 sorted_array_to_bst(nums :: [integer]) :: TreeNode.t | nil  
  def sorted_array_to_bst(nums) 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 sorted_array_to_bst(Nums :: [integer()]) -> #tree_node{} | null.
sorted_array_to_bst(Nums) ->
.

```

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 (sorted-array-to-bst nums)
  (-> (listof exact-integer?) (or/c tree-node? #f)))
)
```

Solutions

C++ Solution:

```

/*
 * Problem: Convert Sorted Array to Binary Search Tree
 * Difficulty: Easy
 * Tags: array, tree, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
}
};


```

Java Solution:

```

/**
 * Problem: Convert Sorted Array to Binary Search Tree
 * Difficulty: Easy
 * Tags: array, tree, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 TreeNode sortedArrayToBST(int[] nums) {

}
}

```

Python3 Solution:

```

"""
Problem: Convert Sorted Array to Binary Search Tree
Difficulty: Easy
Tags: array, tree, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
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 sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:
        # 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 sortedArrayToBST(self, nums):
        """
        :type nums: List[int]
        :rtype: Optional[TreeNode]
        """

```

JavaScript Solution:

```

/**
 * Problem: Convert Sorted Array to Binary Search Tree
 * Difficulty: Easy
 * Tags: array, tree, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 {number[]} nums
 * @return {TreeNode}
 */
var sortedArrayToBST = function(nums) {

};


```

TypeScript Solution:

```

/**
 * Problem: Convert Sorted Array to Binary Search Tree
 * Difficulty: Easy
 * Tags: array, tree, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 sortedArrayToBST(nums: number[]): TreeNode | null {
}

```

C# Solution:

```

/*
 * Problem: Convert Sorted Array to Binary Search Tree
 * Difficulty: Easy
 * Tags: array, tree, sort, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * 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 TreeNode SortedArrayToBST(int[] nums) {
}
}

```

C Solution:

```

/*
* Problem: Convert Sorted Array to Binary Search Tree
* Difficulty: Easy
* Tags: array, tree, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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* sortedArrayToBST(int* nums, int numsSize) {
}

```

Go Solution:

```
// Problem: Convert Sorted Array to Binary Search Tree
// Difficulty: Easy
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 sortedArrayToBST(nums []int) *TreeNode {
}
```

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 sortedArrayToBST(nums: IntArray): 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() { 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 sortedArrayToBST(_ nums: [Int]) -> TreeNode? {
}
}

```

Rust Solution:

```

// Problem: Convert Sorted Array to Binary Search Tree
// Difficulty: Easy
// Tags: array, tree, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// 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 sorted_array_to_bst(nums: Vec<i32>) -> Option<Rc<RefCell<TreeNode>>> {

}
}

```

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 {Integer[]} nums
# @return {TreeNode}
def sorted_array_to_bst(nums)

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 Integer[] $nums
     * @return TreeNode
     */
    function sortedArrayToBST($nums) {

    }
}

```

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 {
TreeNode? sortedArrayToBST(List<int> nums) {
    }
}

```

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 sortedArrayToBST(nums: Array[Int]): TreeNode = {

}
}

```

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 sorted_array_to_bst(nums :: [integer]) :: TreeNode.t | nil
def sorted_array_to_bst(nums) 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 sorted_array_to_bst(Nums :: [integer()]) -> #tree_node{} | null.
sorted_array_to_bst(Nums) ->

```

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 (sorted-array-to-bst nums)  
(-> (listof exact-integer?) (or/c tree-node? #f)))  
)
```