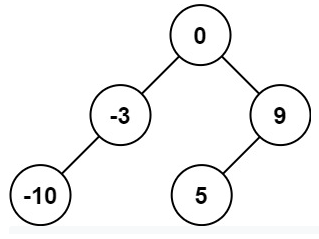
<https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/>

Given an integer array nums where the elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree*.

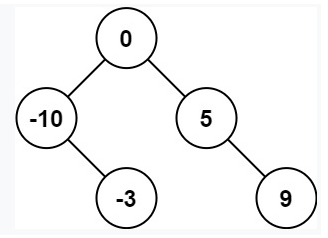
**Example 1:**



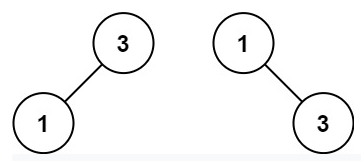
Input: nums = [-10,-3,0,5,9]

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 <= nums.length <= 104
* -104 <= nums[i] <= 104
* nums is sorted in a **strictly increasing** order.

**Attempt 1: 2022-11-10**

**Solution 1:  Recursive traversal and build BST based on sorted array (10min)**

/\*\*

\* 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 TreeNode sortedArrayToBST(int[] nums) {

return helper(nums, 0, nums.length - 1);

}

private TreeNode helper(int[] nums, int start, int end) {

// Base case

if(start > end) {

return null;

}

int mid = start + (end - start) / 2;

TreeNode root = new TreeNode(nums[mid]);

root.left = helper(nums, start, mid - 1);

root.right = helper(nums, mid + 1, end);

return root;

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(logn)

**Refer to**

<https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/discuss/35224/Python-optimal-solution>

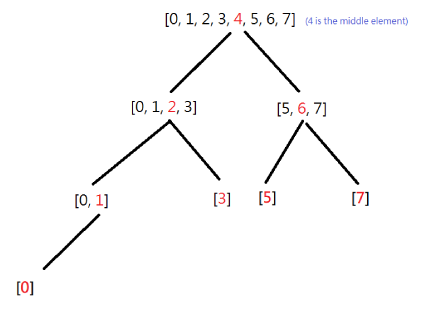
Since nums is a sorted list, the middle element nums[len(nums)//2] must be the root node of nums.Thus, after setting the middle element be the root, finding the middle element in the left subarray nums[:len(nums)//2] and right subarray nums[len(nums)//2 + 1 : ]

For example, nums = [0, 1, 2, 3, 4, 5, 6, 7] as a sorted array, the left half will be in the left subtree, middle value as the root, right half in the right subtree. This holds true for every node:

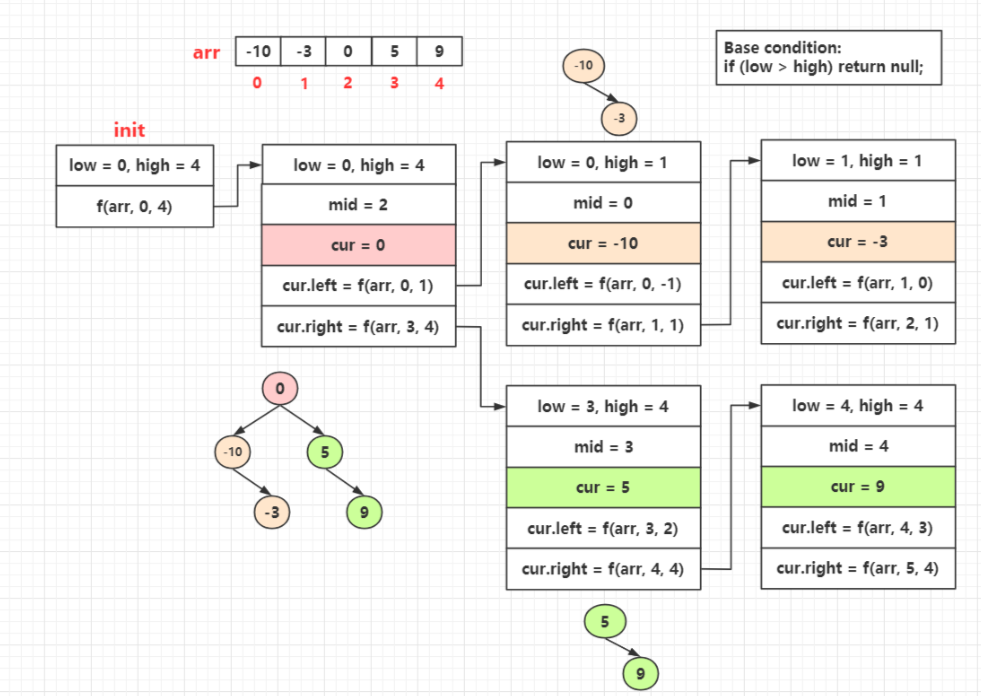
[1, 2, 3, 4, 5, 6, 7] -> left: [1, 2, 3], root: 4, right: [5, 6, 7]

[1, 2, 3] -> left: [1], root: 2, right: [3]

[5, 6, 7] -> left: [5], root: 6, right: [7]



<https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/discuss/35220/My-Accepted-Java-Solution/505514>



**Solution 2:  Iterative traversal and build BST based on sorted array (10min)**

/\*\*

\* 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 TreeNode sortedArrayToBST(int[] nums) {

TreeNode root = new TreeNode(nums[(nums.length - 1) / 2]);

Node node = new Node(root, 0, nums.length - 1);

Queue<Node> q = new LinkedList<Node>();

q.offer(node);

while(!q.isEmpty()) {

int size = q.size();

for(int i = 0; i < size; i++) {

Node cur = q.poll();

int mid = cur.start + (cur.end - cur.start) / 2;

if(cur.start != mid) {

TreeNode left = new TreeNode(nums[cur.start + (mid - 1 - cur.start) / 2]);

cur.node.left = left;

q.offer(new Node(left, cur.start, mid - 1));

}

if(cur.end != mid) {

TreeNode right = new TreeNode(nums[mid + 1 + (cur.end - 1 - mid) / 2]);

cur.node.right = right;

q.offer(new Node(right, mid + 1, cur.end));

}

}

}

return root;

}

}

class Node {

TreeNode node;

int start;

int end;

public Node(TreeNode node, int start, int end) {

this.node = node;

this.start = start;

this.end = end;

}

}

**Refer to**

<https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/discuss/35218/Java-Iterative-Solution/33435>

we assemble the tree from upper level to lower level, from left sibling to rightmost sibling

public class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

if (nums == null || nums.length == 0) {

return null;

}

Queue<MyNode> queue = new LinkedList<>();

int left = 0;

int right = nums.length - 1;

int val = nums[left + (right - left) / 2];

TreeNode root = new TreeNode(val);

queue.offer(new MyNode(root, left, right));

while (!queue.isEmpty()) {

int size = queue.size();

for (int i = 0; i < size; i++) {

MyNode cur = queue.poll();

int mid = cur.lb + (cur.rb - cur.lb) / 2;

if (mid != cur.lb) {

TreeNode leftChild = new TreeNode(nums[cur.lb + (mid - 1 - cur.lb) / 2]);

cur.node.left = leftChild;

queue.offer(new MyNode(leftChild, cur.lb, mid - 1));

}

if (mid != cur.rb) {

TreeNode rightChild = new TreeNode(nums[mid + 1 + (cur.rb - mid - 1) / 2]);

cur.node.right = rightChild;

queue.offer(new MyNode(rightChild, mid + 1, cur.rb));

}

}

}

return root;

}

private static class MyNode {

TreeNode node;

int lb;

int index;

int rb;

public MyNode(TreeNode n, int theLeft, int theRight) {

this.node = n;

this.lb = theLeft;

this.rb = theRight;

}

}

}

Time Complexity: O(n), where n is number of nodes in the Binary Tree

Space Complexity: O(n)