

# 1932. Merge BSTs to Create Single BST

## Description

You are given `n` **BST (binary search tree) root nodes** for `n` separate BSTs stored in an array `trees` (**0-indexed**). Each BST in `trees` has **at most 3 nodes**, and no two roots have the same value. In one operation, you can:

- Select two **distinct** indices `i` and `j` such that the value stored at one of the **leaves** of `trees[i]` is equal to the **root value** of `trees[j]`.
- Replace the leaf node in `trees[i]` with `trees[j]`.
- Remove `trees[j]` from `trees`.

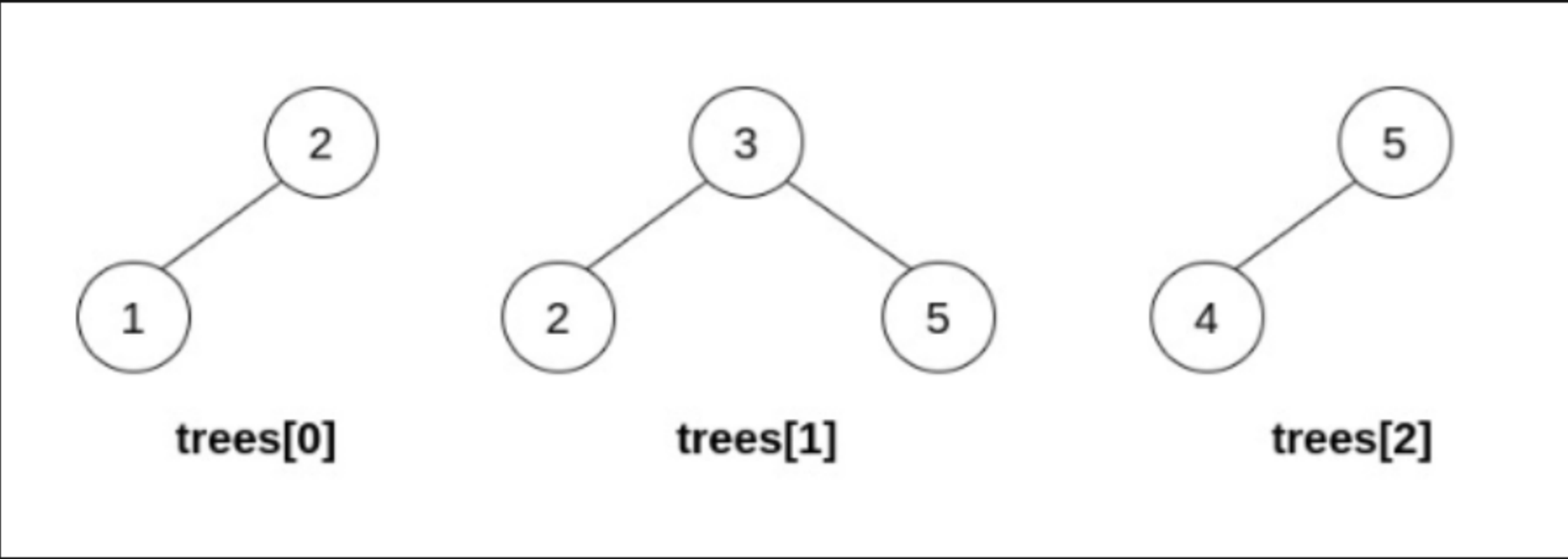
Return *the root of the resulting BST if it is possible to form a valid BST after performing `n - 1` operations, or `null` if it is impossible to create a valid BST*.

A BST (binary search tree) is a binary tree where each node satisfies the following property:

- Every node in the node's left subtree has a value **strictly less** than the node's value.
- Every node in the node's right subtree has a value **strictly greater** than the node's value.

A leaf is a node that has no children.

### Example 1:



**Input:** `trees = [[2,1],[3,2,5],[5,4]]`  
**Output:** `[3,2,5,1,null,4]`  
**Explanation:**  
In the first operation, pick `i=1` and `j=0`, and merge `trees[0]` into `trees[1]`. Delete `trees[0]`, so `trees = [[3,2,5,1],[5,4]]`.

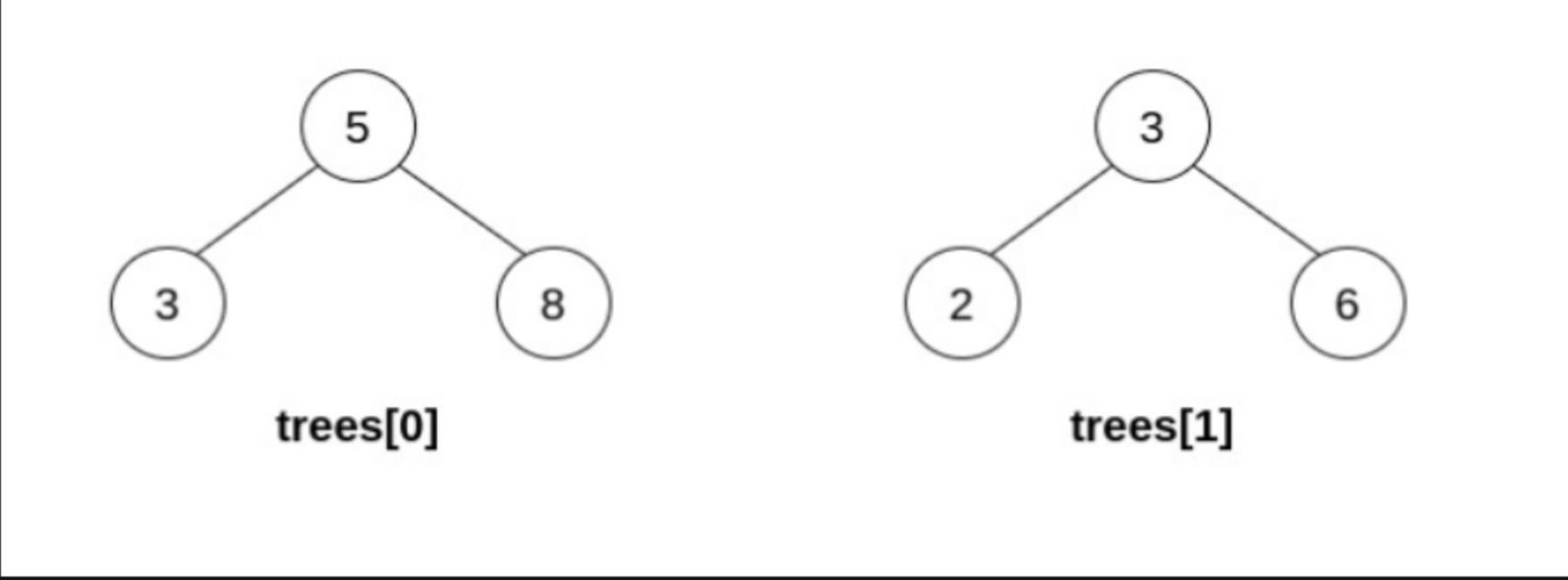
```
graph TD
    3((3)) --- 2((2))
    2 --- 1((1))
    3 --- 5_1((5))
    5_2((5)) --- 4_2((4))
```

**Diagram description:** The diagram shows the state after the first merge. The tree rooted at 3 now has a left child 2, which in turn has a left child 1. The original right child 5 of root 3 remains. Separately, there is a tree rooted at 5 with a left child 4. The nodes 2, 1, and 5 in the first tree are highlighted in blue in the original image.

**Diagram description:** The diagram shows the state after the second merge. The tree rooted at 3 now has a left child 2 (with left child 1) and a right child 5 (with left child 4). The node 5 in the second tree is highlighted in blue in the original image.

The resulting tree, shown above, is a valid BST, so return its root.

### Example 2:

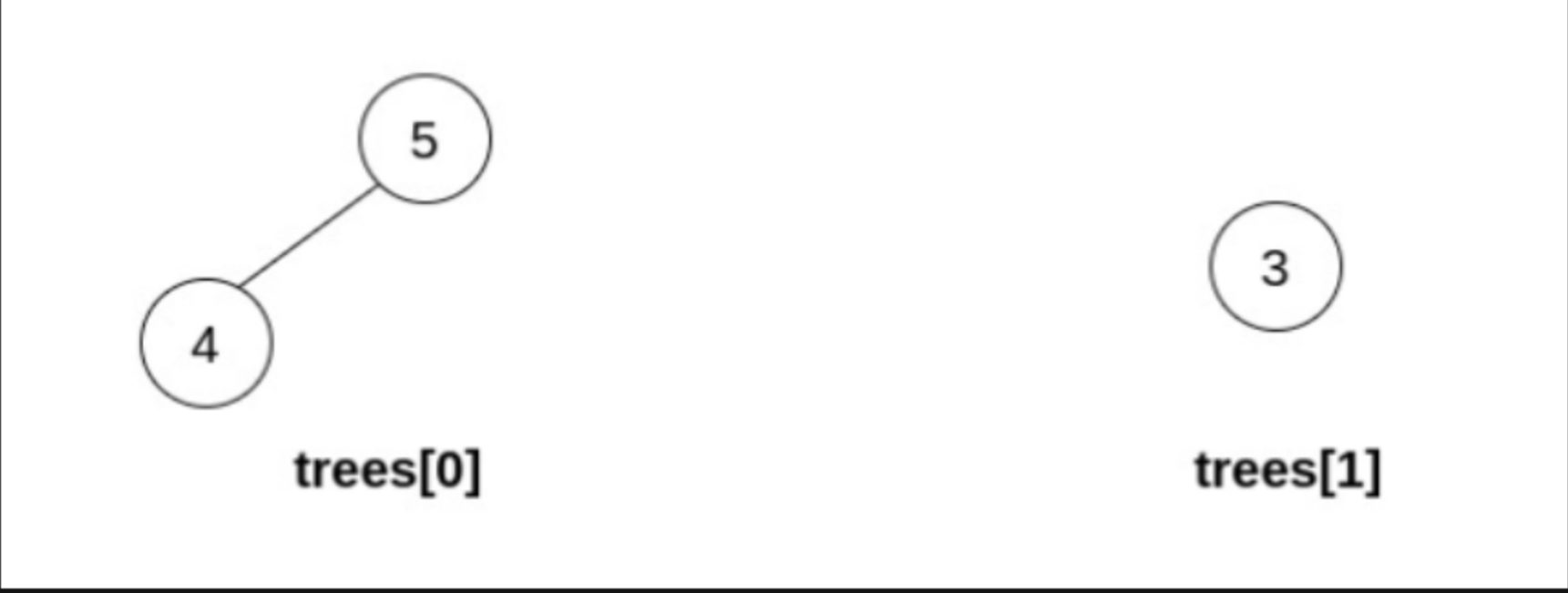


**Input:** `trees = [[5,3,8],[3,2,6]]`  
**Output:** `[]`  
**Explanation:**  
Pick `i=0` and `j=1` and merge `trees[1]` into `trees[0]`. Delete `trees[1]`, so `trees = [[5,3,8,2,6]]`.

```
graph TD
    5((5)) --- 3((3))
    3 --- 2((2))
    3 --- 6((6))
    5 --- 8((8))
```

The resulting tree is shown above. This is the only valid operation that can be performed, but the resulting tree is not a valid BST, so return `null`.

### Example 3:



**Input:** `trees = [[5,4],[3]]`  
**Output:** `[]`  
**Explanation:** It is impossible to perform any operations.

### Constraints:

- `n == trees.length`
- `1 <= n <= 5 * 104`
- The number of nodes in each tree is in the range `[1, 3]`.
- Each node in the input may have children but no grandchildren.
- No two roots of `trees` have the same value.
- All the trees in the input are **valid BSTs**.
- `1 <= TreeNode.val <= 5 * 104`.

