Which of the following statements is true? Select all that apply:

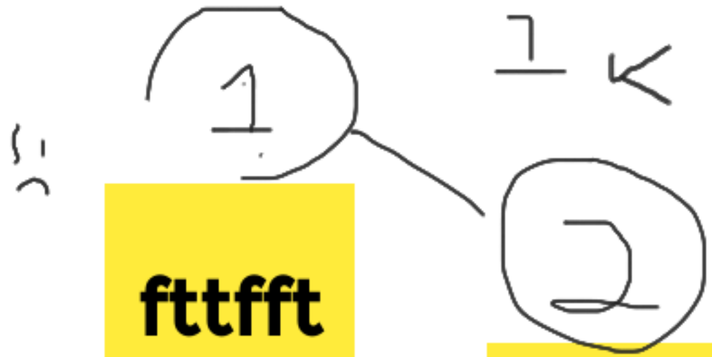
1. The in-order traversal of a binary tree is always sorted
2. The post-order traversal of a binary search tree *might* be sorted
3. The pre-order traversal of a binary tree *might* be sorted
4. The pre-order traversal of a binary search tree is always sorted
5. **All** ancestors of a node in a binary search tree are less than the node
6. **All** ancesors of a node in a binary tree *might* be less than the node

in-order is true ONLY FOR BST (actually false for regular binary trees)

post-order might be sorted, but isn't necessarily

1.False 2.f 3.True 4.

real



fttfft

2. 3, I forgor

4, 6 True

Did NXHX write this

1. False | 2. True | 3. True | 4. False | 5. . True

True: 1, 3, 6

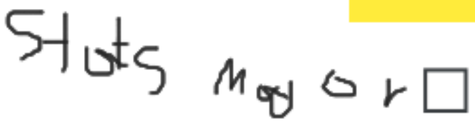
its so over

pre-order might be sorted, but not necessarily

REPENT

THE END IS NEAR

dude really



3, 6 TRUE rest False

plu

Given the in-order traversal of an arbitrary binary *search* tree, is it possible to reconstruct the original tree? If so, how? If not, why not?
Hint: Recall our discussion on AVL Trees from last week
Hint 2: Recall your lab on BST rotations

n >= 2

it is not possible because we do not know what the root value is

Therefore, the function is not injective.
QED.

Not possible. If we consider the function mapping BSTs to their in-order traversals, we see the function is not injective because multiple BSTs can map to a single in-order traversal.

the in order traversal of a BST will always return a sorted list, so it is impossible to reconstruct the original tree without more information

No because we don't know what number the tree is balanced around

No, because you cannot tell which nodes are children and which are parents.

From looking at a BST, it isn't possible to create the exact same tree given arbitrary nodes. You'd know the structure, but you wouldn't know the original context of the nodes

Yes its "possible" but when you reconsturct it you cant know for certain its the same tree (this is just a guess)

Given a sorted list of integers from 1 to n, $n \in \mathbb{N} \setminus \{0\}$, how do you construct a balanced binary search tree (i.e: height $\log(n)$) from this list? Outline the order of insertion, and explain why this works.

Divide list into 3 parts: [0:n//2], n//2, and [n//2 + 1:]. Place n//2 as root value, then recursively divide the two sublists. Left child is root of [0:n//2], right child is root of [n//2 + 1]

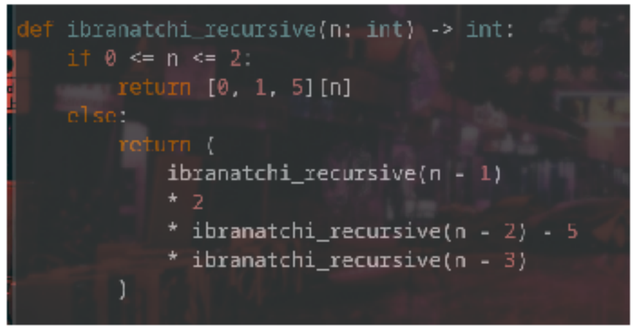
```
class BST: def __init__(self, lst): mid = len(lst)//2; self.root = lst[mid]; if len(lst) != 1: {self.left = BST(lst[:mid]); self.right = BST(lst[mid+1:]);}
```

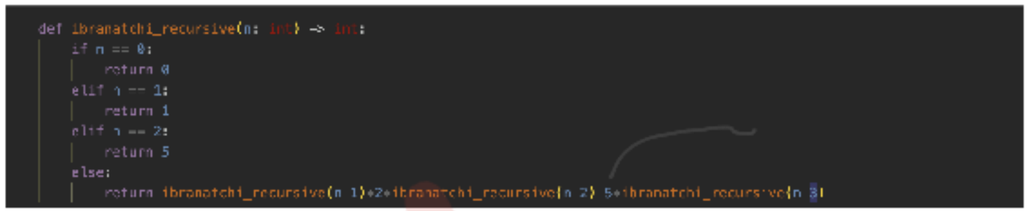
Convert the following iterative function to a recursive function:

```python
def ibranatchi_iterative(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 5

    sequence = [0, 1, 5]
    for i in range(3, n + 1):
        next_value = sequence[i - 1] * 2 * sequence[i - 2] - 5 *
        sequence[i - 3]
        sequence.append(next_value)

    return sequence[-1]

def ibranatchi_recursive(n: int) -> int:
    # TODO: Implement this recursively:
```

isnt it 0 for all n >= 2?

return ibranatchi_iterative(n-1)*2*ibranatchi_iterative(n-2)-5*ibranatchi_iterative(n-3)

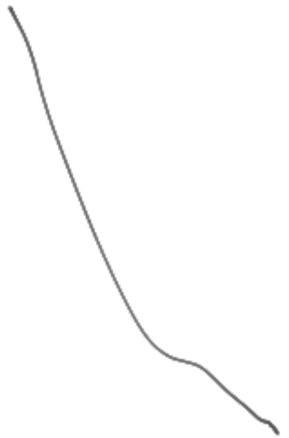


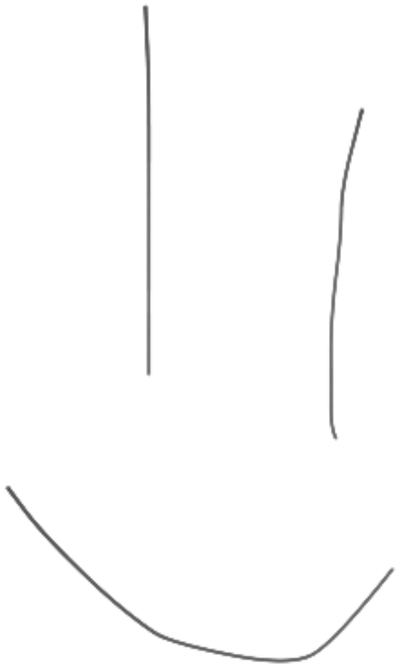


What is the output of the following list comprehension. If it throws an error, explain why:

```
[[x*y for x in range(4)] for y in range(7)]
```

what if we switch the `x` and `y` iterables?





secret
treasure
yeah