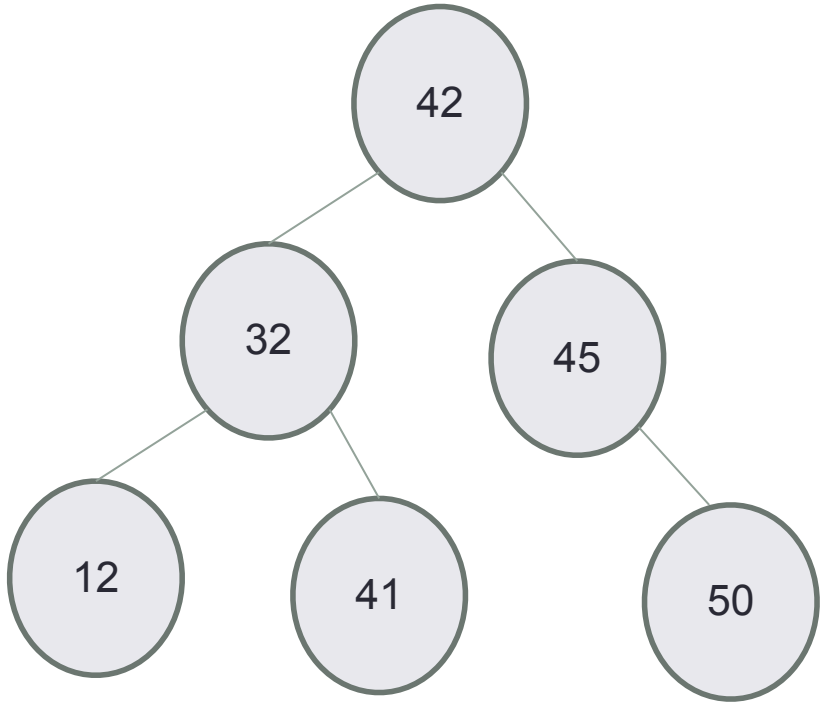
# BINARY (SEARCH) TREE TRAVERSALS

Problem Solving with Computers-II

# In order traversal: print elements in sorted order

```
Inorder(r : pointer to current node):
    if r is null, return
    Inorder(r->left)
    process r (e.g., print r->val)
    Inorder(r->right)
```



# Post-order traversal: use to recursively clear the tree!

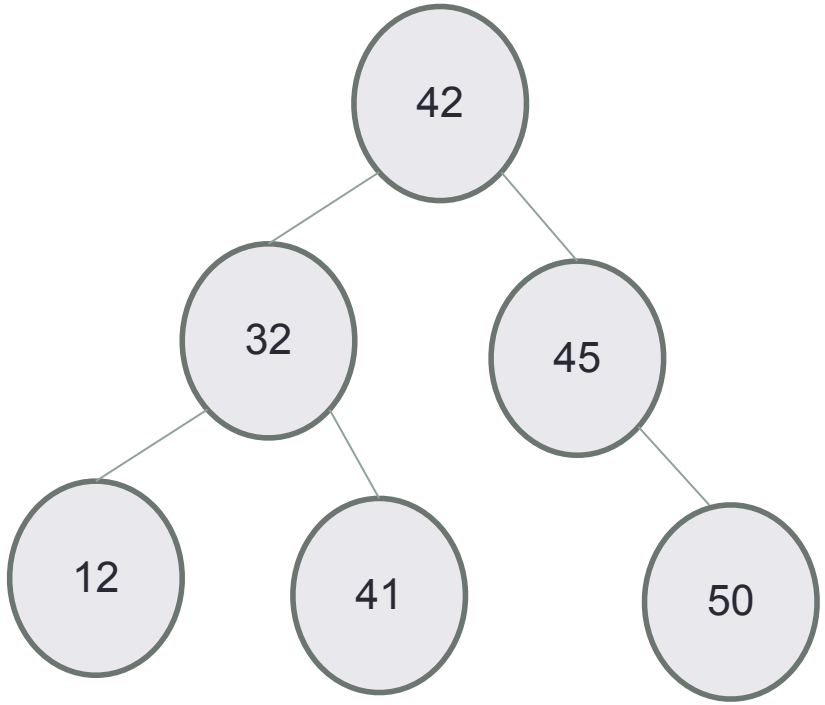
postorder(r : pointer to current node):
    if r is null, return
    postorder(r->left)
    postorder(r->right)
    process r (e.g., print r->val)

```
int bst::getHeight(Node *r) const{
    if (!r)
        return -1;
    int hleft = getHeight(r->left);
    int hright = getHeight(r->right);
    return max(hleft, hright) + 1;
}
```

```
void bst::clear(Node *r){
    if (!r)
        return;
    delete r->left;
    delete r->right;
    delete r;
}
```

# Pre-order traversal: nice way to linearize your tree!

```
preorder(r : pointer to current node):
    if r is null, return
    process r (e.g., print r->val)
    preorder(r->left)
    preorder(r->right)
```



# Pre-order traversal Game!

**1. Draw a BST:** Individually, draw a BST with 6 distinct keys (e.g., integers 1–10). Ensure it's a valid BST (left child < node < right child). Keep your drawing secret until prompted.

**2. Trace Preorder Sequence:** Trace the preorder traversal (root, left, right) of your BST and write the sequence of node values.

**(1) Your BST (draw secretly):**

**Tip:** Double-check your BST is valid (left < node < right) before tracing.

# Pre-order traversal Game!

**3. Pair Up and Exchange:** Partner with a neighbor (left/right or front/back). Share your preorder sequence by writing it for your partner. Receive their sequence. Don't share your tree drawing.

**4. Reconstruct the BST:** Using your partner's sequence, rebuild their BST by inserting nodes in the given order, respecting BST properties.

**5. Compare Trees:** Show your reconstructed tree to your partner. Compare it with their original drawing. Do they match? Discuss any differences and how you rebuilt the tree.

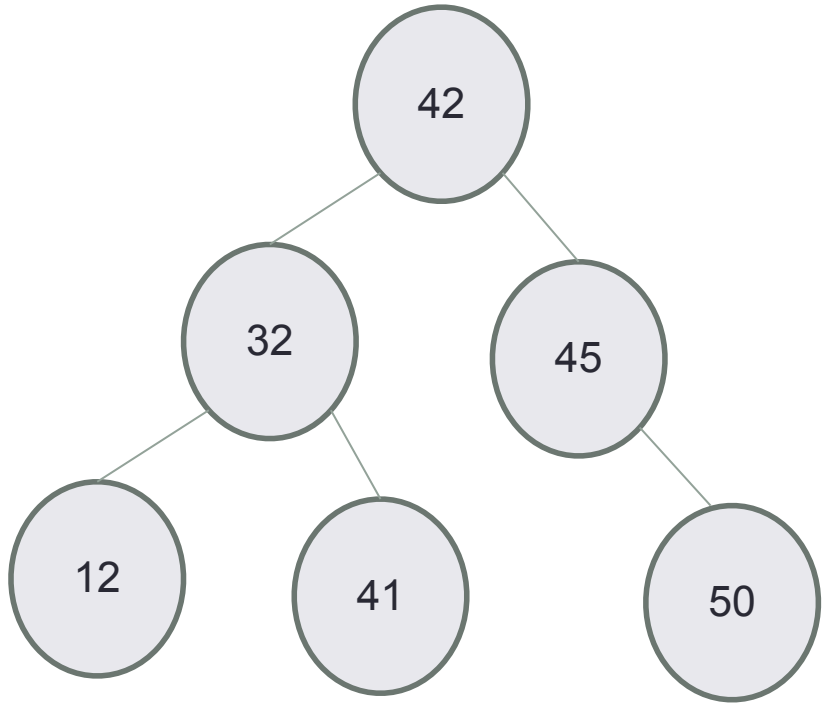
**(2) Your Preorder sequence (share with partner):**_____

**(3) Partner's sequence:**_____

**(4) Your reconstruction of partner's BST:**

# Interview question

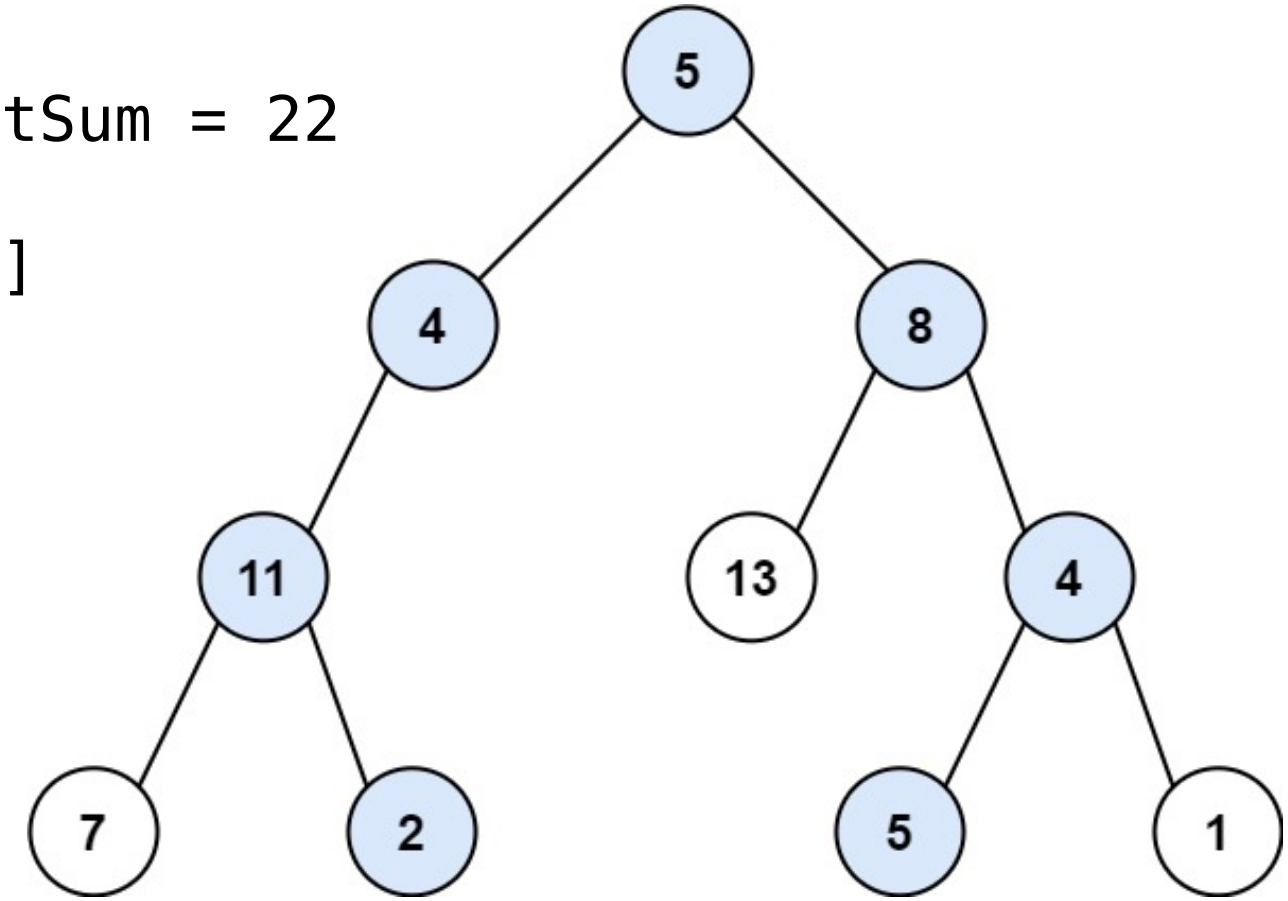
Write a function to linearize the BST into a vector of keys.
Constraint: the vector should be constructed such that the original BST can be recovered by inserting the keys of the vector in the order provided into a new (empty) BST



# Practice problem: Path Sum II (medium)

- https://leetcode.com/problems/path-sum-ii/description/?envType=problem-list-v2&envId=binary-tree

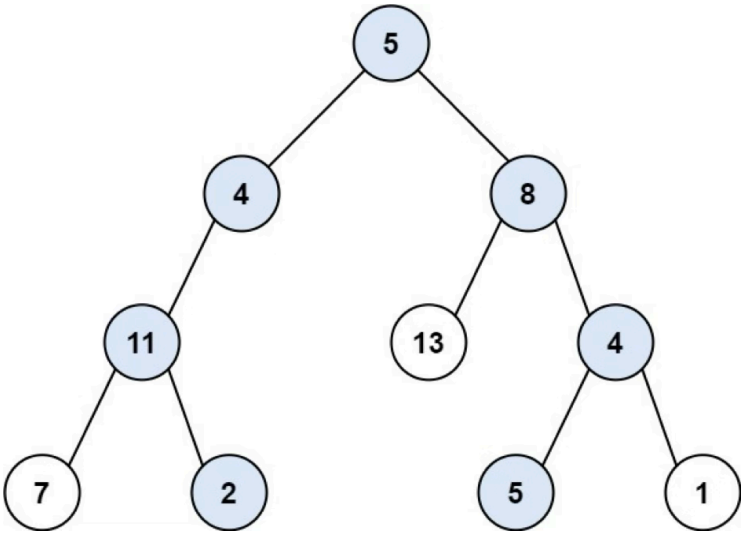
```
Input:tree on the right,targetSum = 22

Output: [[5,4,11,2],[5,8,4,5]]
```



Answer the following questions to plan and trace a solution for Path Sum II (10 mins):

1. Which traversal should you use to explore all root-to-leaf paths, and why is it suitable?
2. When should you record a path and check its sum, and how do you identify this point in the tree?
3. What data structures should you use to track the current path and store all valid paths?
4. Using your answers, trace the values in your data structures for a complete run on the example tree.
    - Work individually or with your game partner.
    - Write your answers in the space below.
    - Focus on planning and tracing, not coding.

1. **Traversal**: _____
2. **Path Recording Point**:_____
3. **Data Structures**: _____
4. **Trace of your approach on sample tree**



## Tips

- Recall the preorder game's root-first tracing to choose the traversal.
- Identify leaves by checking for no children (null left and right).
- Use dynamic data structures (e.g., vectors) that can grow/shrink for paths.
- For tracing, follow preorder order, updating your path at each step and resetting after leaves.
- Discuss with your partner to clarify ideas.