Today's Content:

Problems
- → Inorder traversal (Iterative) } & Implement
- → Morris Inorder traversal (Iterative)
- → Diameter of Tree
- → BST → Sorted Circular Doubly Linked List

preorder → ✓

{ Inorder

postorder → ✓

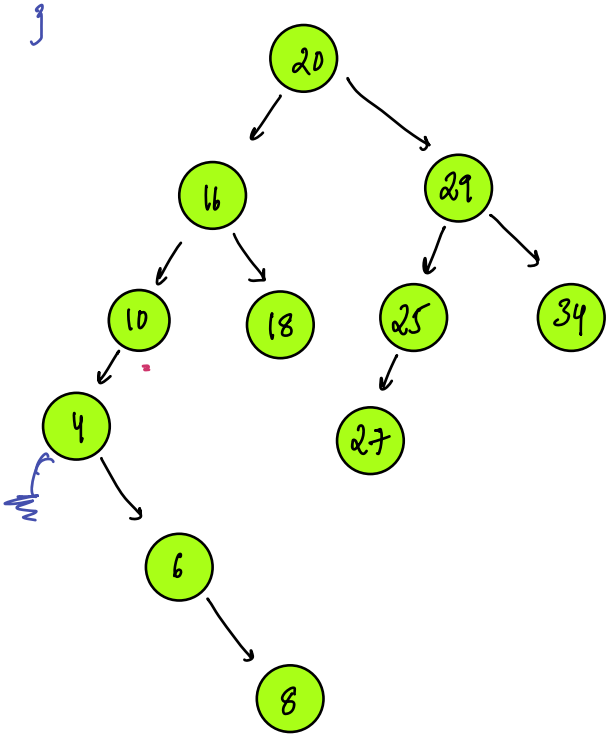
Inorder traversal : { Iterative }

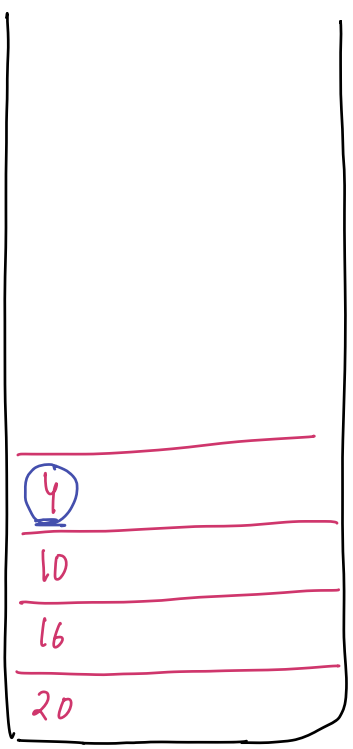
```
void inorder(root){
1   if(root == NULL){
        return;
    }
2   inorder(root.left)
3   print(root.val)
4   inorder(root.right)
}
```

```
// main(){
    // inorder(root)
}
```

Idea: 1) Till you get a null in left side, keep pushing

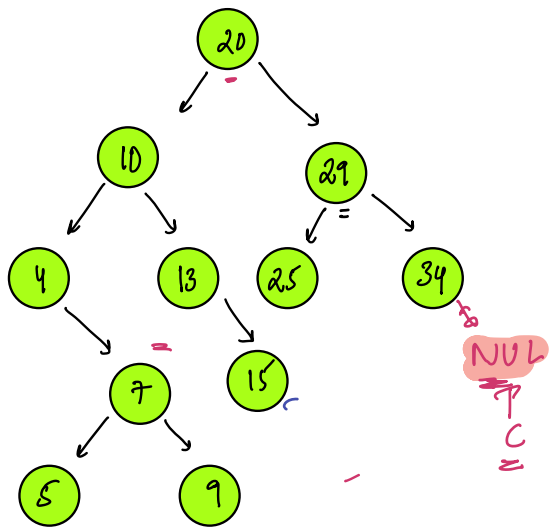
2) If root == NULL, get the top element of stack & print & goto right

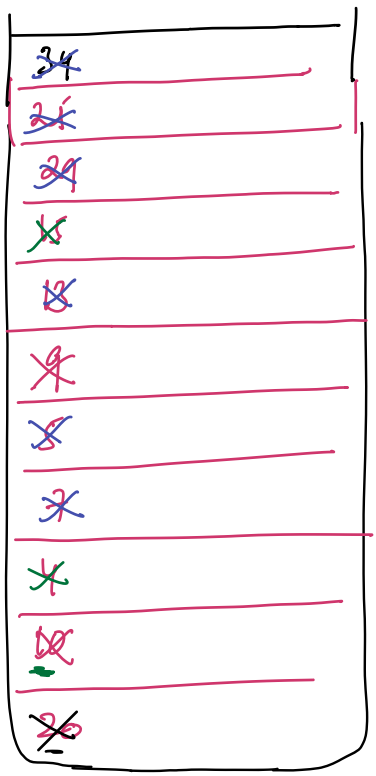




print : 4 6 8 16 18 20 . . . . .

print: 4 5 7 9 10 13 15 20 25 29 34





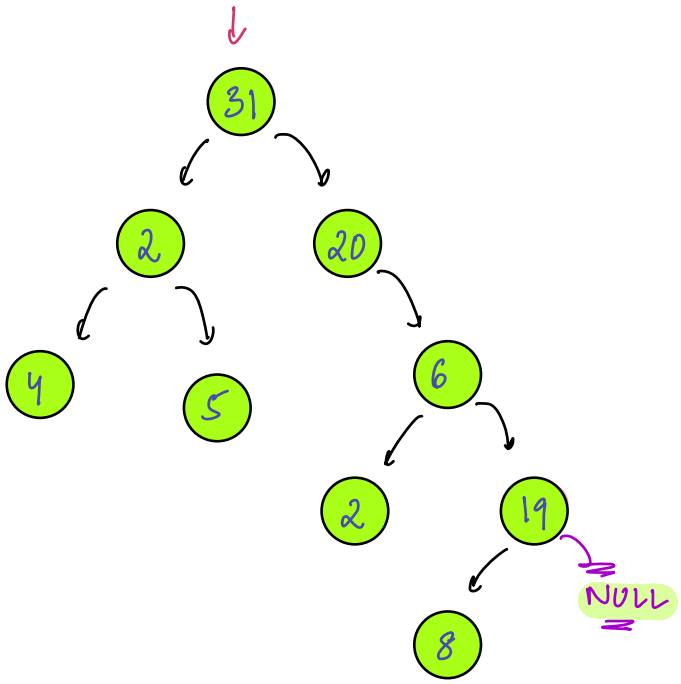

```
inorder( Node root){

    Stack<Node> st;
    Node cur = root;
    while ( cur != NULL || st.size() > 0 ){
        if ( cur != NULL){
            st.push( cur)
            cur = cur.left
        }
        else{ // cur == NULL
            Node t = st.top();
            st.pop()
            print(t.data)
            cur = t.right
        }
    }
}
```

TC: O(N)

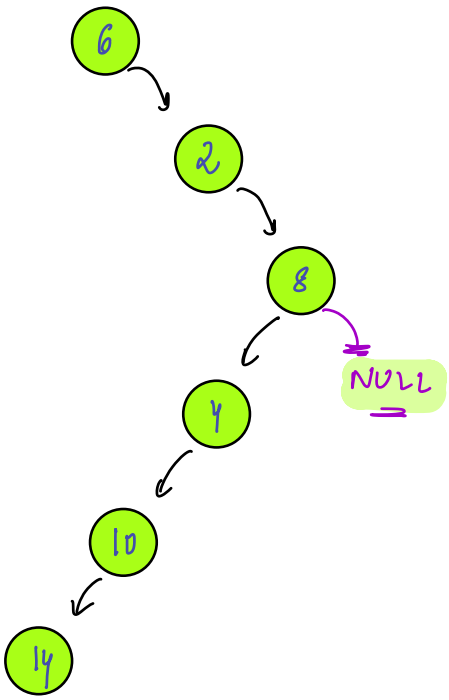
SC: O(H)

28): With Inorder Traversal in a Tree, last node we print?

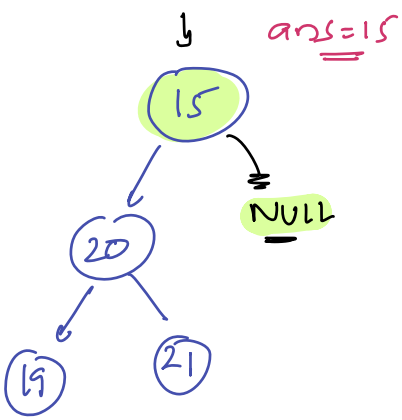
ans = 19



ans = 8

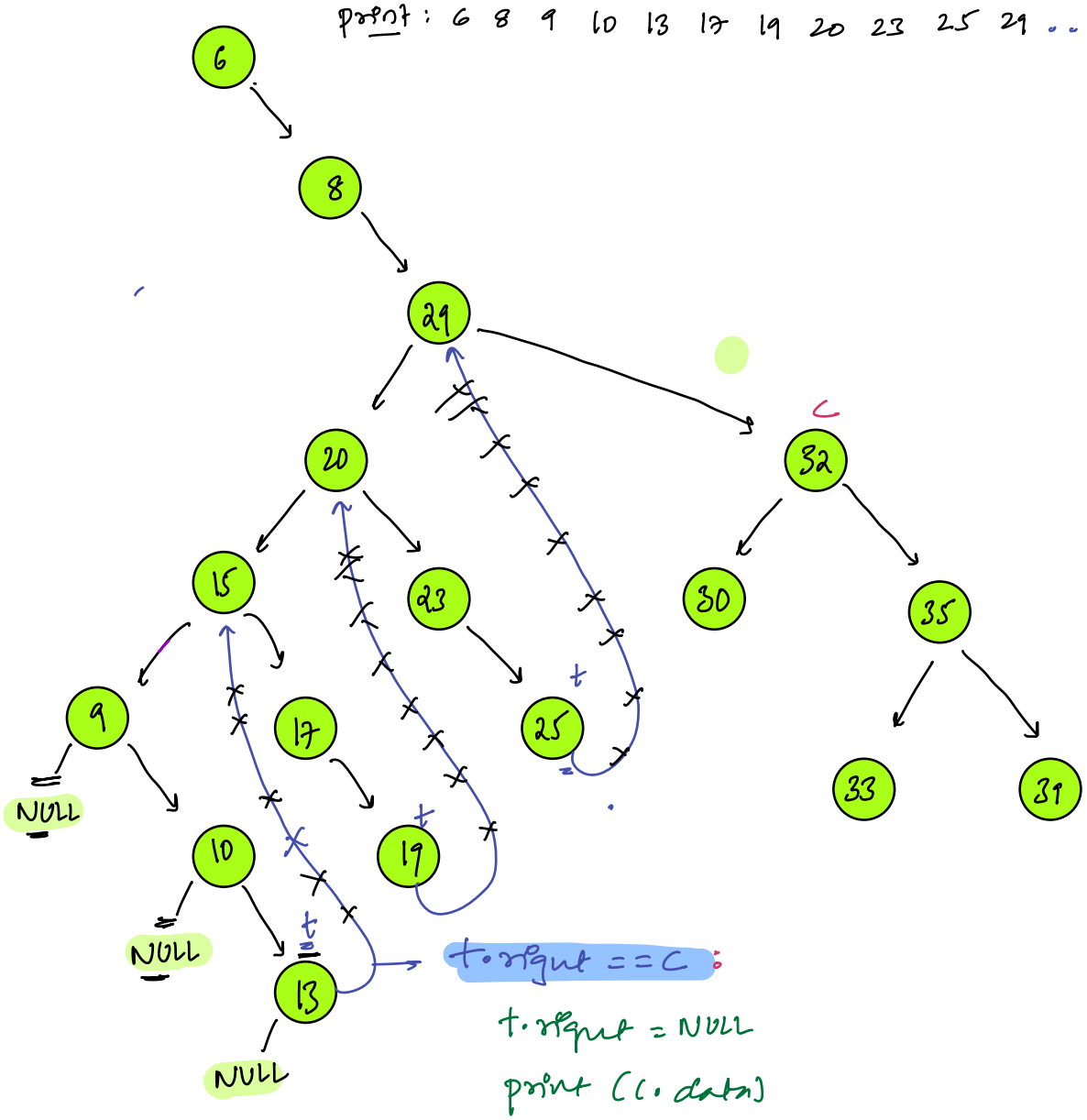


ans = 15



30) Morris Inorder traversal → { no extra space } { HARD }

print : 6 8 9 10 13 17 19 20 23 25 29 ....



Pseudocode:

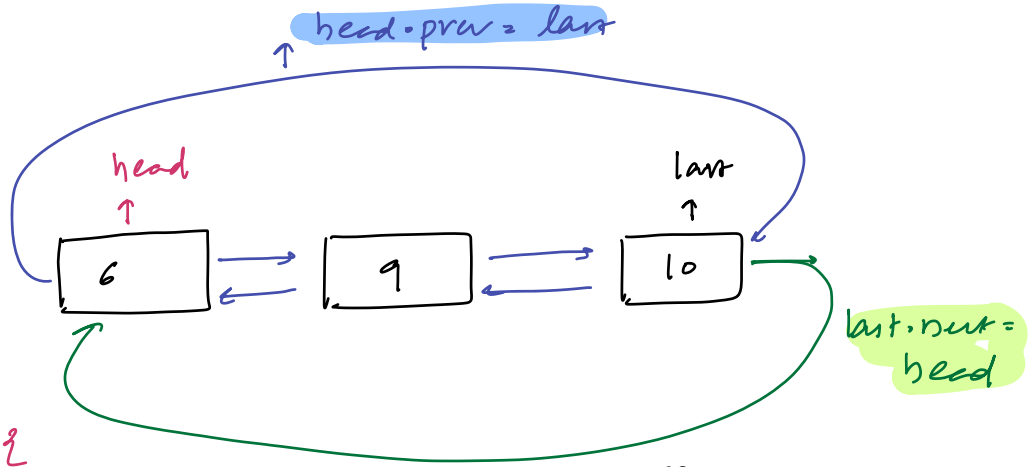
```
inorder(Node root){
    Node cur = root;
    while(cur != NULL){
        if(cur.left == NULL){
            print(cur.data)
            cur = cur.right
        }
        else{
            Node temp = cur.left
            while(temp.right != NULL && temp.right != cur){
                temp = temp.right
            }
            if(temp.right == NULL){ -> if visiting cur node 1st time
                temp.right = cur // update link
                curr = cur.left // update cur
            }
            else{
                // -> if visiting cur node 2nd time
                temp.right = NULL // remove link
                print(cur.data) //
                cur = cur.right
            }
        }
    }
}
```

TC: O(N)

SC: O(1)

temp.right == cur

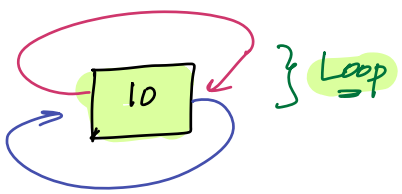
3rd: CDLL



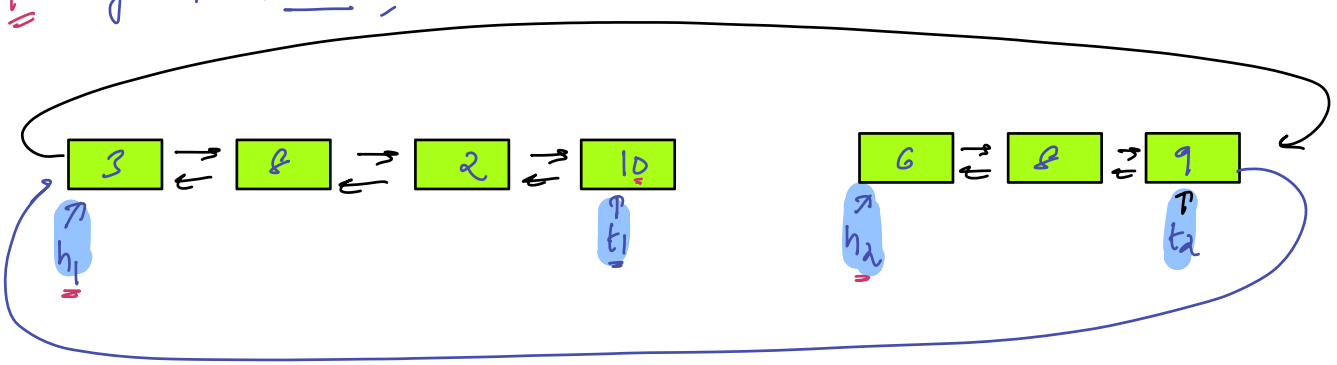
CDLL

```
class DLLnode {
    DLLnode pre
    DLL node next
}
```

Info



4th: given 2 CDLL, combined both of CDLL, & return head




```
__ Combine ( Node h1, Node h2) {
    if( h1 == NULL) return h2;
    if ( h2 == NULL) return h1;
    Node t1 = h1.prev, t2 = h2.prev
    t1.next = h2
    h2.prev = t1
    t2.next = h1
    h1.prev = t2
    return h1;
}
```

→ O(1)
SC:
O(1)

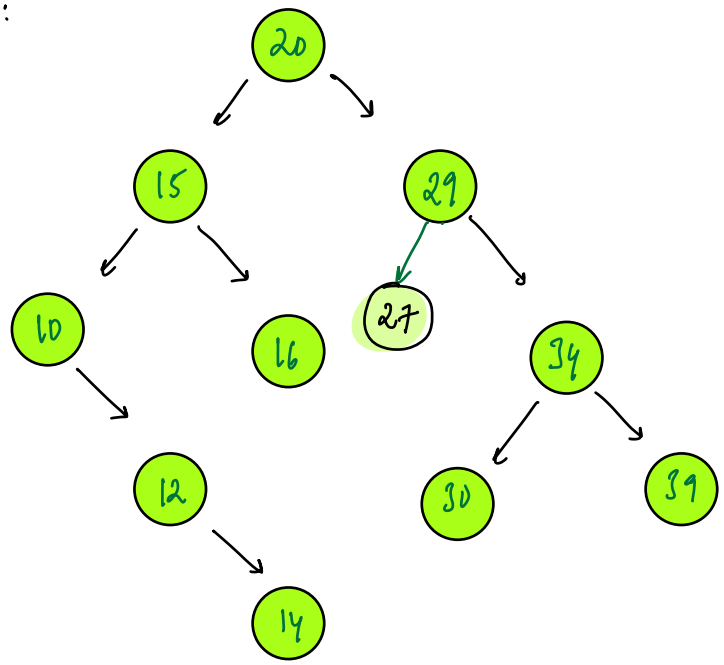
un
left of
right

48) given BST → Sorted Circular Double Linked List

→ SC: No Extra Space

→ Re-arrange BST Links to get CDLL

Ex:



// re-arrange links

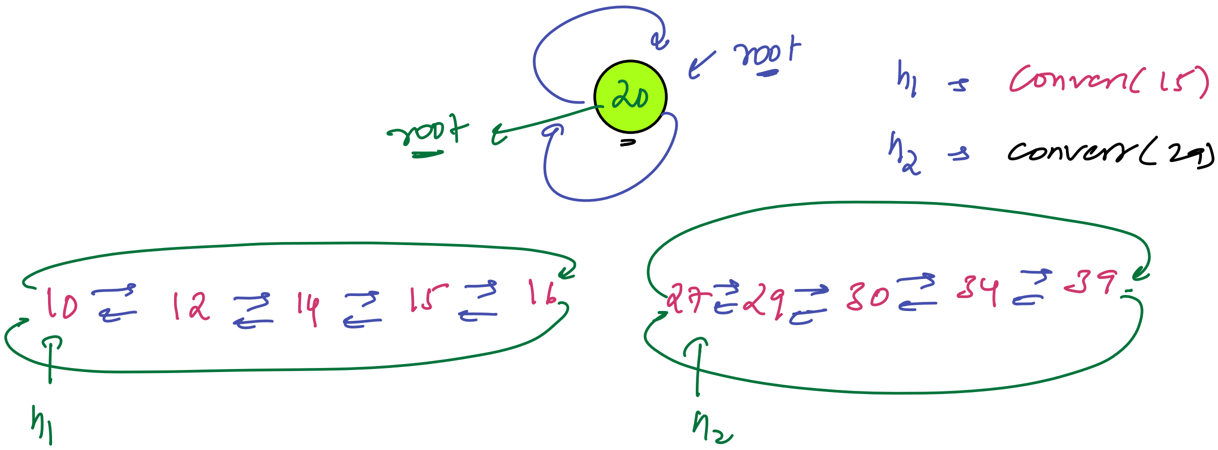
TC: O(N)

10 ⇄ 12 ⇄ 14 ⇄ 15 ⇄ 16 ⇄ 20 ⇄ 27 ⇄ 29 ⇄ 30 ⇄ 34 ⇄ 39

//

Ass: given a BST → SCDLL & return head

```
Node Convert ( Node root)
{
    if ( root == NULL) { return NULL }

    Node h1 = Convert (root.left)   // LST → CDLL

    Node h2 = Convert (root.right)  // RST → CDLL

    root->left = root, root->right = root //

    return combine ( combine (h1, root), h2 )
                    ↳ It will return head node of entire thing
}
```



root.left = root

root.right = root

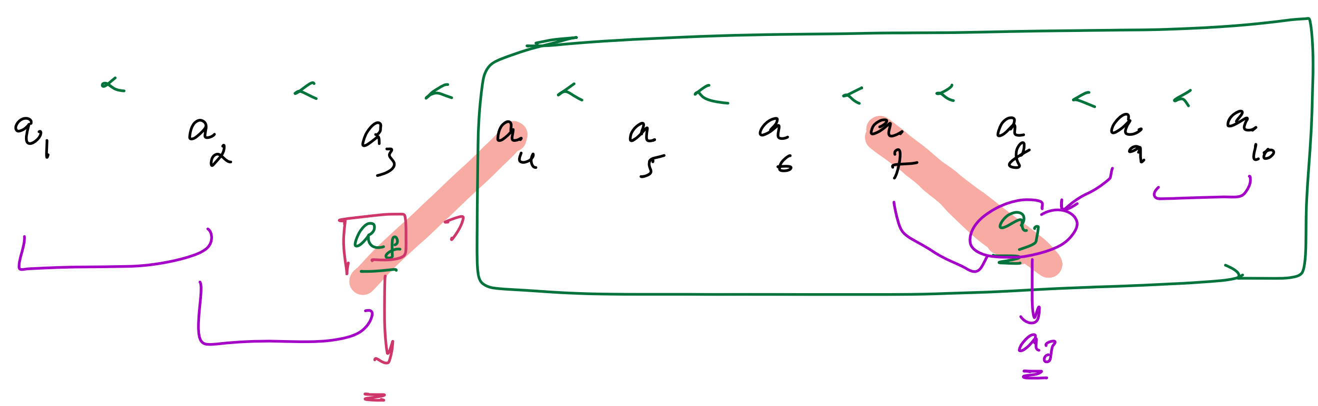
// BST → SDLL

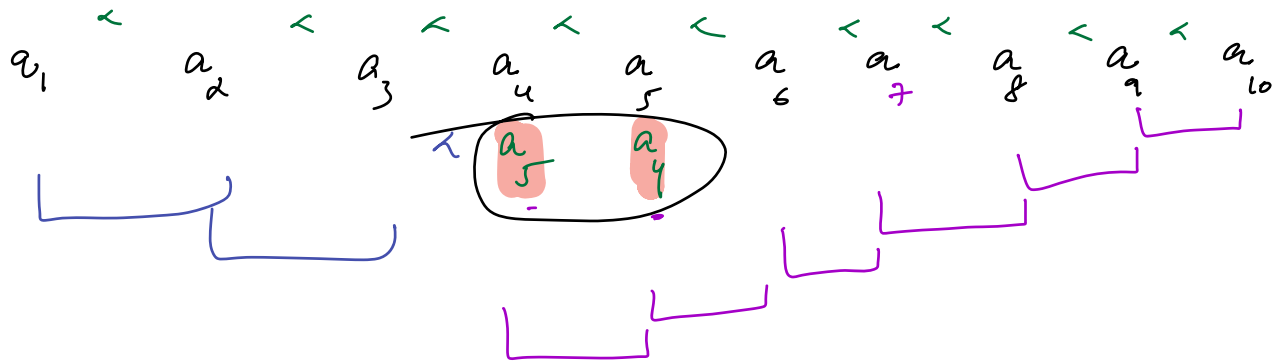
h3 ans: CDLL at last break loop ✓

→ { Left Tree → Subtree }

a_3 a_8

Idea: Sorted array.




a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10}
a_5 a_4