(a) Basic working: First we use a stack to print out the inorder traversal of the binary tree (say in a linked list where we keep inserting on the tail). Then we check if the elements in dynamic array are arranged in strictly increasing order or not. If they are, then the binary tree is valid BST else not.

Note: we can replace D by another stack & push it instead of inserting in D. this way, we only use stacks.

Problem 6: [15 marks]

Given a binary tree with integer keys, given an iterative algorithm using stacks to check if the tree is valid a binary search tree. (a) First explain the basic working of your algorithm and the data structures used in a few sentences (b) Write the pseudo-code for the algorithm. (c) Explain in a paragraph why your code works correctly.

(b) Pseudo code:

```
Stack S = ∅                                          1
Dynamic Array D = ∅                                  2
Node curr = root;                                    3
while (S ≠ ∅ or curr ≠ NIL) do                       4
    if (curr ≠ NIL)                                  5
        push(S, curr)                                6
        curr = curr→left                             7
    else                                             8
        curr = top(S)                                9
        insert(D, curr)                              10
        pop(S)                                       11
        curr = curr→right                            12
integer n = length(D).                               13
boolean flag = True                                  14
for i from 1 upto n-1 do                             15
    if (D[i]→key > D[i+1]→key)                       16
        flag = False                                 17
return flag.                                         18
```

(c) Explanation: Lines 1-3 do some initializations. Then lines 4-12 perform an inorder traversal of the tree with root root & store the elements in an dynamic array (takes O(1) time). The loop invariant for line 4 is that in D, for all elements, the ones occurring in the left subtree occur before that element & then the ones in the right subtree. (we go to right only if cant go to left). Then we check if D is sorted in increasing order. This will check for validness of BST since for the BST, for each node, the keys of all nodes in left subtree are less than node's key & keys of all nodes in right subtree are more than node's key. So

Valid BST iff inorder traversal is strictly increasing.

The boolean flag keeps track whether there are any inversions in the key of the array.

Space complexity: $\Theta(n)$

Time complexity: $\Theta(n)$

Note: space complexity can be reduced to O(1) if we just check whether top of stack is less than curr→key in line 6. If no then return false. After whole while loop return true

& for traversal of tree using stack & then once traversing the array