ZAEEM YOUSAF (l19-1196 3E2)

submitted to: Amina Khan (DS assignment 2)

Workout of Time Complexity

Contents

1 Q1/B Time complexity of iterative mirror

ans: **O(n)**

1.1 Detail

all right nodes were pushed and poped O(log n) each time, a poped nodes' children were swaped and total time O(n) max height of stack was the n/2. Tree was traversed from top to bottom while each time moving left and storing the right children to be treated afterward. *Total $O(\log_2 n)$ + O(n) = O(n)*p

2 Q2/B Time complexity of spiral order travesal

ans: **O(n)+O(n) = O(n)**

2.1 Detail

Two Stacks were used. 'left$_{\text{toright}}$' and 'right$_{\text{toleft}}$' traversal was made level vise to push all elements from left to right and then right to left. O(n) was for traversal as each node was visited and O(n) was for pushing and then poping each element.

3 Q4/B Time complexity of flattening a Binary Tree

ans: **O(n)**

3.1 Detail

each node was visited recursively preorder and that node was inserted in **linkList** ojbect. Note: Each node was inserted instead of data which was more difficult but it gives the real picture of how the tree was flattened. Therefore my flattened list looks like

4 Q5/B Time complexity of joining Two AVL Trees

ans: $\mathbf{O(h_1+h_2)}$ where h_1 is the hight of T_1 and h_2 is the height of T_2 Note: T_2 is not merged node by node, Whole tree is attached at once almost

4.1 Detail

T_2's smallest Element was broken and made root of new tree T_3. Then T_1 was traversed right until height becomes = h_2 Then T_1 was broken in two parts at that that root and was connected to left of T_3 T_2 was attached on the right side of T_3 Then that T_3 was attached to parent of broken node in T_1 Node breaking is constant and this algorithm is very efficient for AVL trees with greater hieghts. $O(h_1+h_2)$ is just due to traversal for the appropriate place where one T_2 could be attached.