Title: Binary Search Trees

Author: Arman Engin Sucu

ID: 21801777

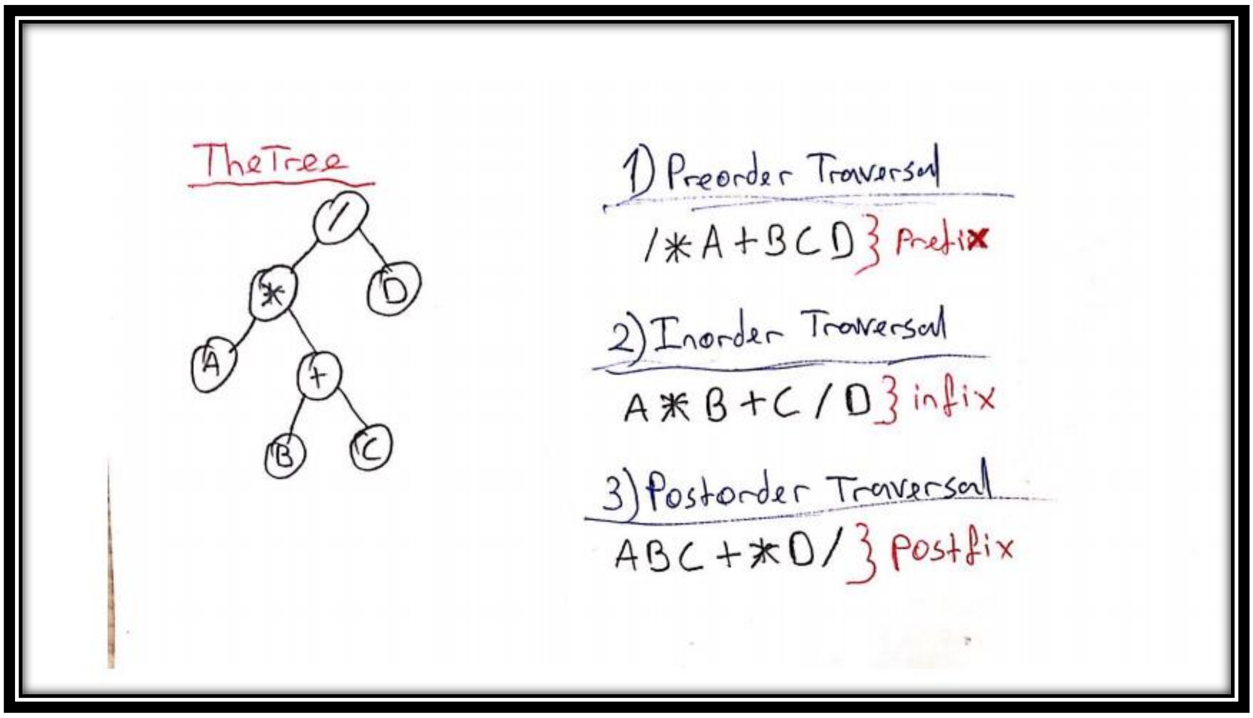
Section: 1

Assignment: 2

Description: Answers for Q1 and Q3

QUESTION 1

Part a)



1) Preorder Traversal

/ * A + B C D } Prefix

2) Inorder Traversal

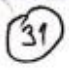
A * B + C / D } infix

3) Postorder Traversal

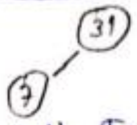
A B C + * D / } postfix

Part b)

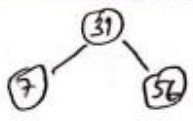
1) After 1st Insertion: 31



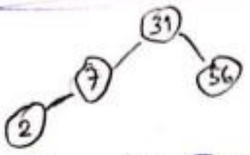
2) After 2nd Insertion: 7



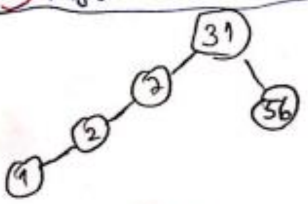
3) After 3rd Insertion: 56



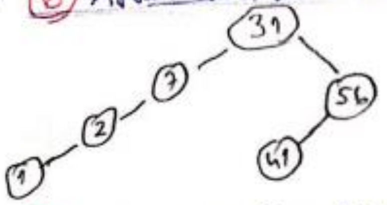
4) After 4th Insertion: 2



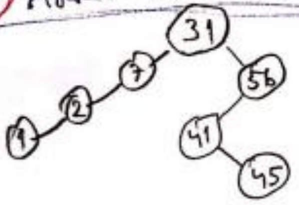
5) After 5th Insertion: 1



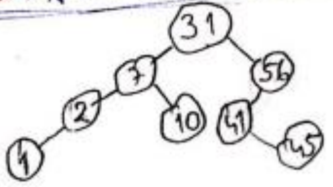
6) After 6th Insertion: 41



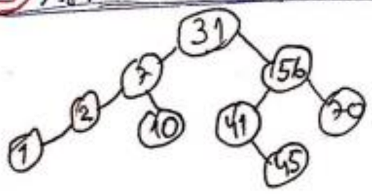
7) After 7th Insertion: 45



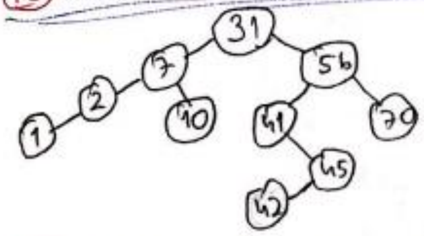
8) After 8th Insertion: 10



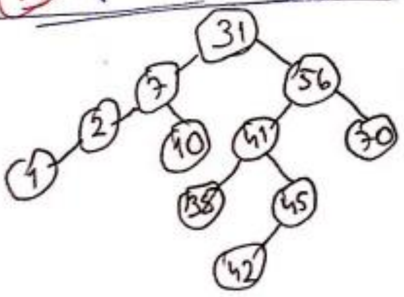
9) After 9th Insertion: 70



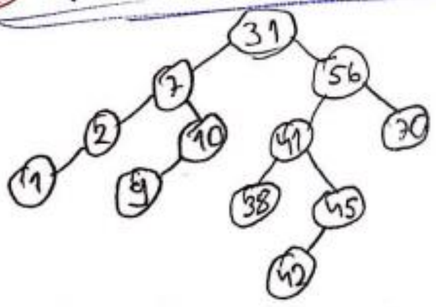
10) After 10th Insertion: 42

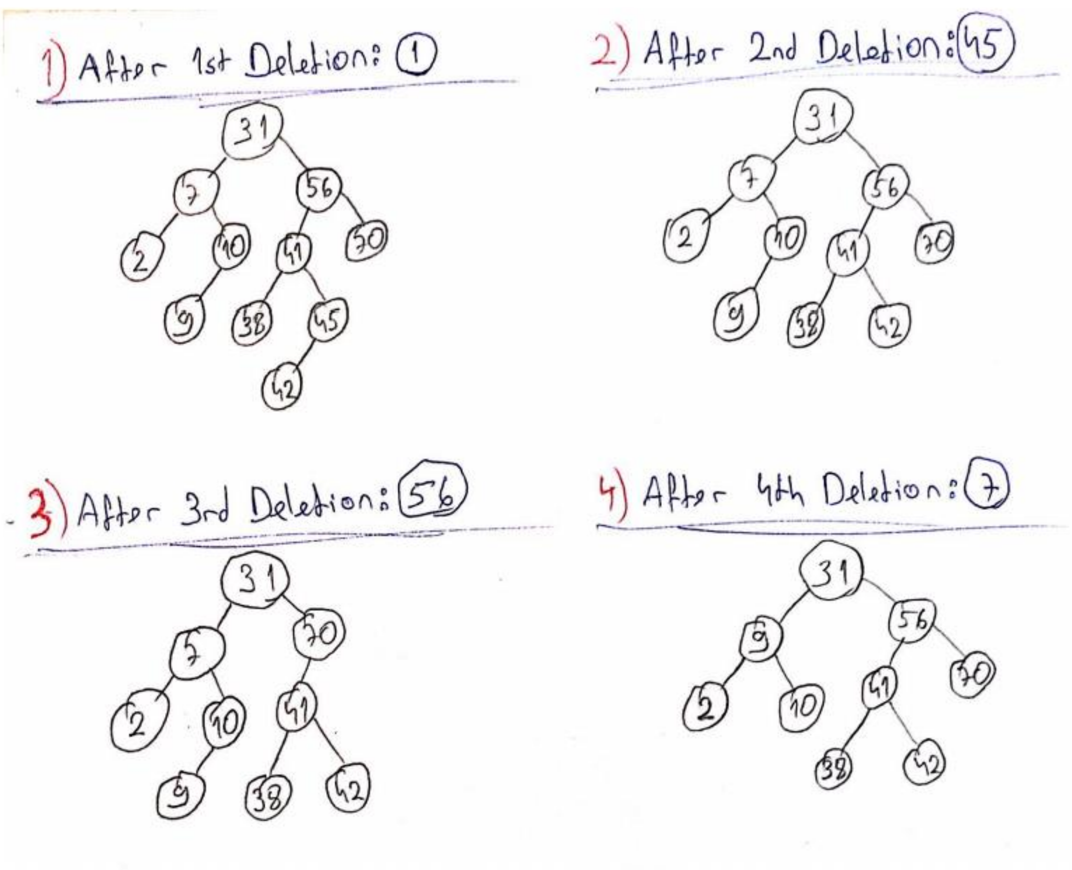


11) After 11th Insertion: 38



12) After 12th Insertion: 5

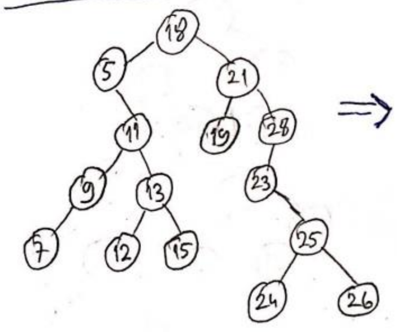




Part c)

The Preorder Traversal: 18, 5, 11, 9, 7, 13, 12, 15, 21, 19, 28, 23, 25, 24, 26

The Tree:



⟹ Postorder Traversal:

7, 9, 12, 15, 13, 11, 5, 19, 24, 26, 25, 23, 28, 21, 18

QUESTION 3

1) **levelorderTraverse**

- levelorderTraverse calls “levelorder” which gets the BinaryNode as a parameter, since levelorderTraverse can not get any parameter. Levelorder method finds the height of the tree by the method gethHeightHelper and prints the every level of the tree(by the help of printTheLevel function) in order until the height. “printTheLevel” is a recursive function. It gets two parameters one is BinaryNode and the other one is integer for the level. The method calls itself two times, one for left child and one for right child. In every recurison the method calls itself with one smaller level value and if the level is one, it exits the method.
- For getHeightHelper every node in the tree will be visited to find the longest path therefore it is O(n). For printTheLevel , first root will be visited, then root will be visited with its children, then root will be visited wit its cihildren and its children’s children. Therefore, $c(n + (n-1) + (n-2) + \cdots + 1) = O(n^2)$. Since all nodes will visited without any conditions Big O complexity does not depend any condition.

  Thus, as a result:

  Best Case: $O(n^2)$

  Average Case: $O(n^2)$

  Worst Case: $O(n^2)$

  Space Complexity: 0 since any extra memory is not used.

- The mothod can be implemented at $O(n)$ for any case by using Queue. However, this time space complexity will be $O(n)$ for avereage case and worst case.

2) **span**

- The span gets two integer parameters: a(smaller value) as well as b(bigger value) and it calls two recursive methods which are “findBigerThan” and “findSmallerThan”. findBigerThan gets two parameters a BinaryNode and an integer. The method finds the number of nodes that biger than the given number for the given root and vice versa for the findSmallerThan. The span method have 4 cases. If the given root is null return 0. If the given root item is equal to smaller value(a) increment result 1 and call the findSmallerThan for the right child with the value b. If the given root item is equal to biger value(b) increment result 1 and call the findBigerThan for the left child with the value a. If the given root item value is between a and b, increment result 1, then call both findBigerThan as well as findSmallerThan for left and right child respectively
- Span traverses every node item between the range a and b. Therefore, if the number of nodes between a and b is called “k”, it can be seen that on average and best case Big-O is $O(k)$. However, for worst case in addition to the k nodes, the function also needs to traverse number of heights of the tree. Therefore, if the height of the tree is h then, the worst case is $O(h + k)$.

Thus, as a result:

Best Case: $O(k)$

Average Case: $O(k)$

Worst Case: $O(h+k)$

Space Complexity: $O(1)$, since int result is used.

- $O(k)$ is the fastest time complexity for span, because k elements(all elements in the range) must be visited. Since this algorithm only traverse the k elements (except the worst case, for worst case: to reach the range elements some out of range elements has to be visited.), its time complexity can not be improved.

3) mirror

- The mirror calls one recursive function: “swap” which gets one parameter BinaryNode. The swap function resembles mergesort algorithm, since it first make recursive calls then calls the base case as merge sort. The swap first controls whether the given node is null, then if it is not, calls itself with left child and right child. Then it swaps the left child and right child.
- The function visits every node element one time, therefore it is Big O notation is $O(n)$. Since, independently from the cases all nodes will visited one time all case will have the same time complexity.
  Thus, as a result:

  Best Case: $O(n)$

  Average Case: $O(n)$

  Worst Case: $O(n)$

  Space Complexity: $O(1),$ 1 temp BinaryNode is used.

- The $O(n)$ is the fastest time complexity for mirror, since all elements needs to be visited at least one time in order to swap them with thier siblings. However, by using queue another implementation can be made with same time complexity $O(n)$, however this time space time complexity will be $O(n)$ for average case.