HW2

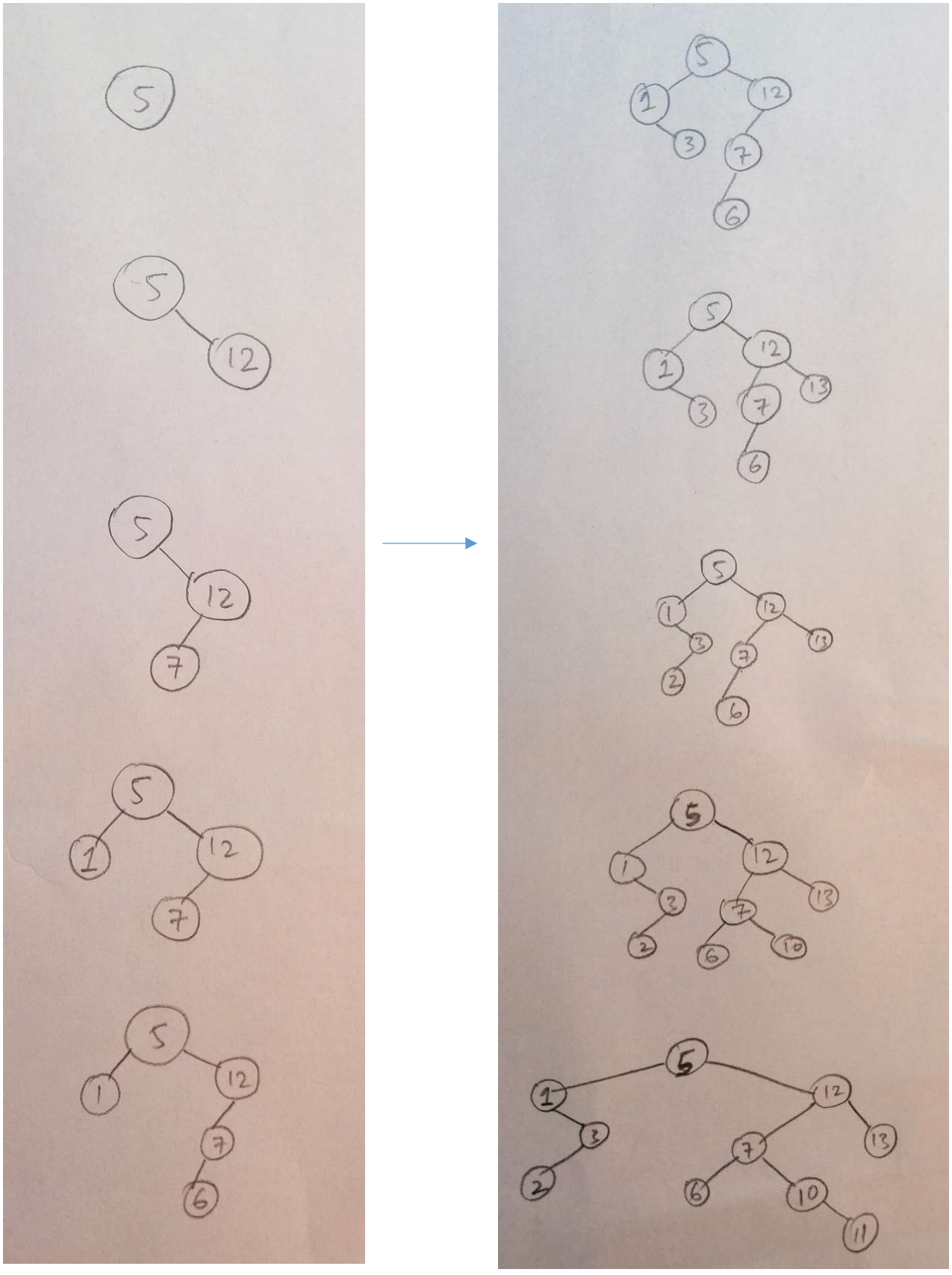
CS202–SECTION 3

Muhammad Arham Khan

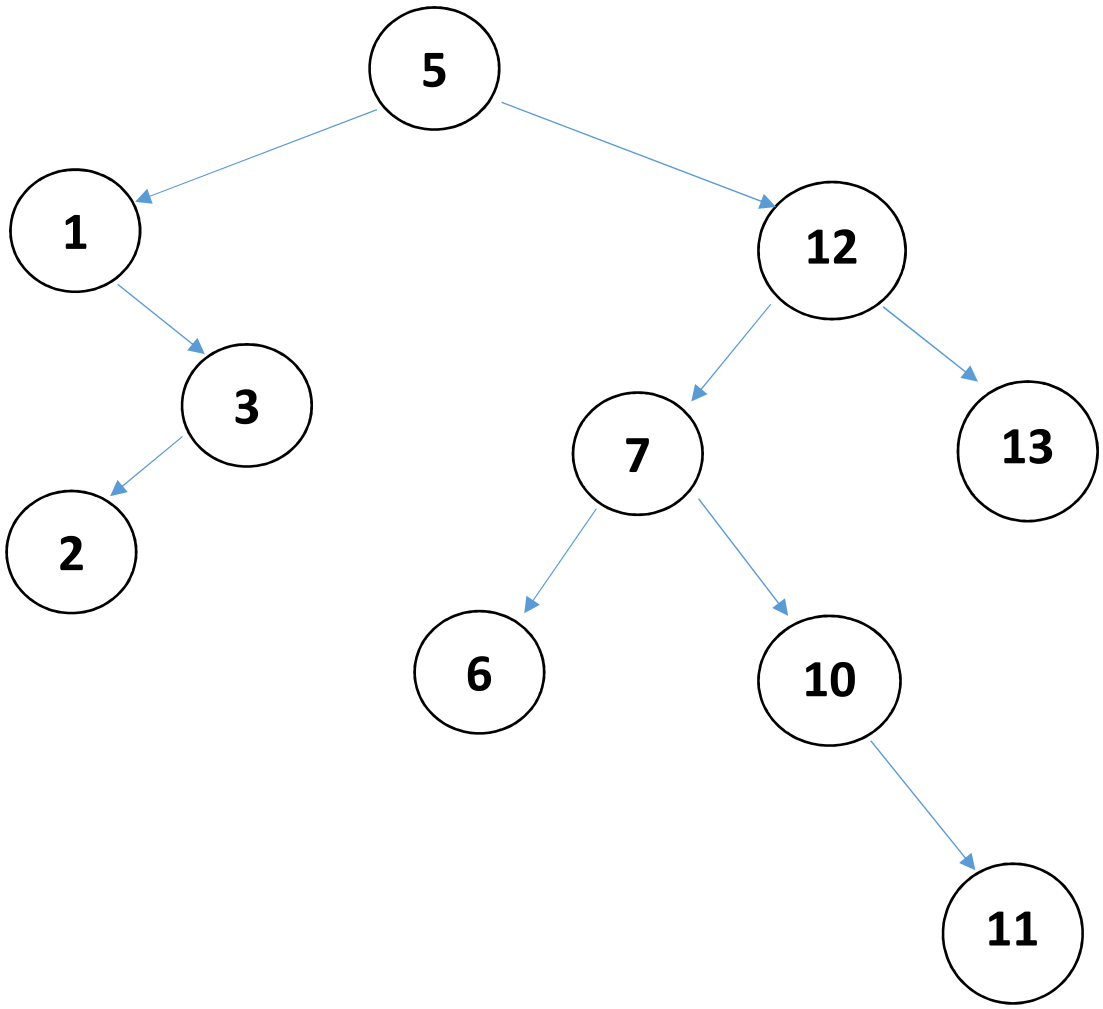
21701848 - CS

Dated: 22 March 2019

QUESTION 1 (a):


5
5
12
5
12
7
5
1
12
7
5
1
12
7
6
5
1
12
3
7
6
5
1
12
3
7
13
6
5
1
12
3
7
13
2
6
5
1
12
3
7
13
2
6
10
5
1
12
3
7
13
2
6
10
11

QUESTION 1 (b):

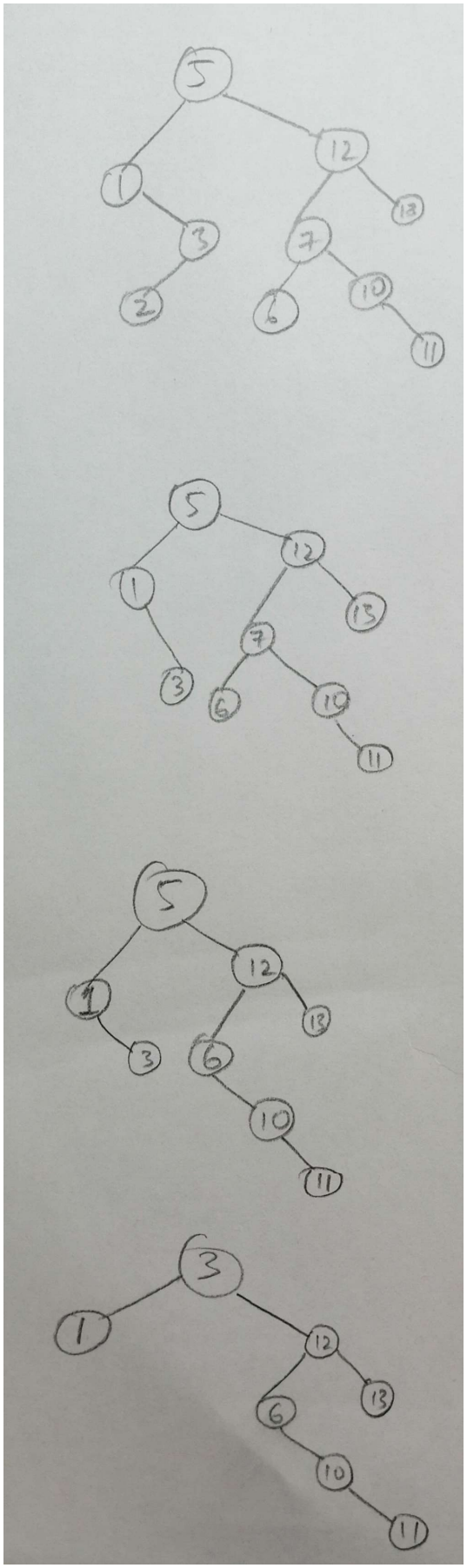


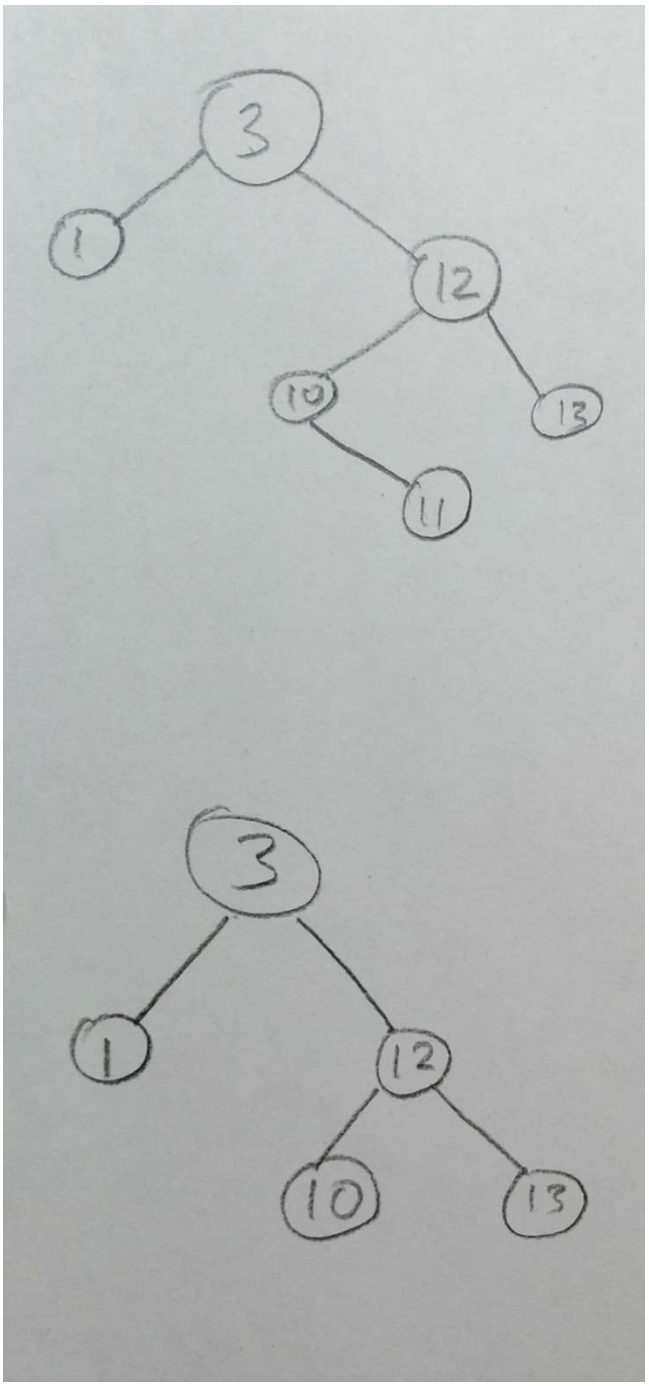
PREORDER: 5, 1, 3, 2, 12, 7, 6, 10, 11, 13

INORDER: 1, 2, 3, 5, 6, 7, 10, 11, 12, 13

POST-ORDER: 2, 3, 1, 6, 11, 10, 7, 13, 12, 5

QUESTION 1 (c):


5
12
1
13
3
7
2
6
10
11
5
12
1
13
7
3
6
10
11
5
12
1
13
3
6
10
11
3
1
12
13
6
10
11


3
1
12
10
13
11
3
1
12
10
13

QUESTION 3:

insertItem:

The function finds either left or right node according to the value of current node and key and recursively calls itself until the insertion position is reached.

The average runtime complexity of this function is O (log n) as it would traverse around half of the elements imbalanced tree to find the position of insertion. In a worst case, the complexity is O(n) as the tree would be imbalanced and appear to look like a linear linked list and hence, insertion might traverse through all nodes.

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.

deleteItem:

The function finds either left or right node according to the value of current node and key and recursively calls itself until the deletion position is reached. Then it identifies the case of the node being deleted and replaces relevant nodes with the deleted nodes (below left, below right or left most).

The average runtime complexity of this function is O (log n) as it would traverse around half of the elements imbalanced tree to find the position of deletion. In a worst case, the complexity is O(n) as the tree would be imbalanced and appear to look like a linear linked list and hence, insertion might traverse through all nodes. Furthermore, If that node has both nodes, then finding the left most node in tree would take additional time in a worst case.

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.

retrieveItem:

The function finds either left or right node according to the value of current node and loops until the end of list is reached or the item is found. Then returns it.

The average runtime complexity of this function is O (log n) as it would traverse around half of the elements imbalanced tree to find the position of the node being searched. In a worst case, the complexity is O(n) as the tree would be imbalanced and appear to look like a linear linked list and hence, finding node might traverse through all nodes.

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.

inorderTraversal:

The function recursively calls itself on its left node, itself, and right node adding them all to the array of elements.

The average case and worst case for this function would be the same as to traverse all nodes, the complexity is O(n) and hence, that it the answer.

The space complexity of this function is O(n) as the function utilizes a dynamically allocated array to store the n elements of the tree.

containsSequence:

The function recursively proceeds through the tree traversing it in order and beginning comparison as soon as the first sequence elements is reached. Skipping the node chains before it and after it, it tells if the sequence was found in the node chain.

The worst case for this program is O(n) as the sequence might lie at the end of the tree but even in the average case, the function will have to traverse through around half of the elements in the tree and hence, the average complexity is also O (n).

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.

countNodesDeeperThan:

The function recursively iterates through all nodes and keeps count of their current level and increments the node count if level is higher than or equal to some level.

The average case and worst case for this function are O(n) as it traverses through all nodes counting the ones that are after a height level.

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.

maxBalancedHeight:

recursively iterates through all nodes finding the minimum full height of the leaf nodes and returns it as it is the balanced height.

The average case and worst case for this function are O(n) as it traverses through all nodes the minimum height level that is common in all node chains to ensure that a balanced tree is found.

The space complexity of this function is O(1) as the function utilizes a constant memory for the process.