Lab 09 Notes

CS 61BL
Summer 2019

Lab 9
July 11, 2019

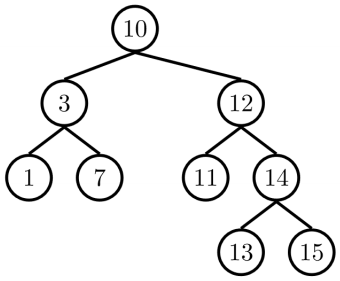
Name:

SID:

Please complete this worksheet during your lab, and turn it in to your TA by the end of your section. You are encouraged to work with your neighbors collaboratively.

Section Number: (01) (02) (03) (04) (05) (06) (07) (08) (09) (10) (11) (12)

1 Recursive Traversals



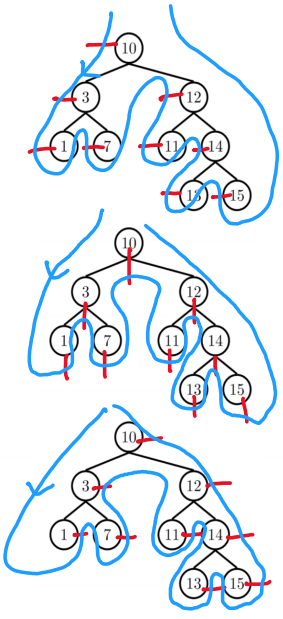
Run this below pseudocode on the binary tree above in order to write its pre-order, in-order, and post-order traversals. When evaluating `preorder(node)`, print the node's value to the pre-order list; when evaluating `inorder(node)`, print the node's value to the in-order list; and when evaluating `postorder(node)`, print the node's value to the post-order list.

```
1  binaryTreeTraversal(node):
2      preorder()
3      if left != null:
4          binaryTreeTraversal(left)
5      inorder()
6      if right != null:
7          binaryTreeTraversal(right)
8      postorder()
```

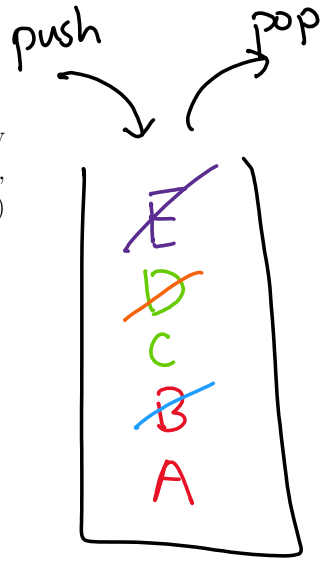
Pre-order: 10, 3, 1, 7, 12, 11, 14, 13, 15

In-order: 1, 3, 7, 10, 11, 12, 13, 14, 15

Post-order: 1, 7, 3, 11, 13, 15, 14, 12, 10



2 Stacks and Queues

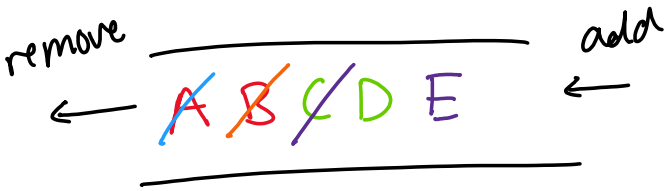


2.1 Suppose that the following sequence of operations is executed using an initially empty **stack**. What ends up in the stack? (List the values separated by spaces, where the bottom of the stack is to the left and the top of the stack is to the right.)

```
1 push A
2 push B
3 pop
4 push C
5 push D
6 pop
7 push E
8 pop
```

A, C

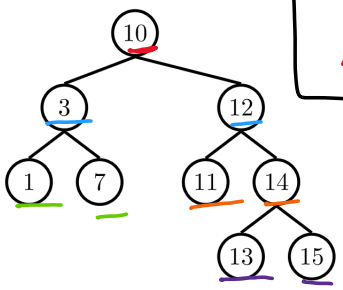
2.2 Suppose that the following sequence of operations is executed using an initially empty **queue**. What ends up in the queue? (List the values separated by spaces, where the front of the queue is to the left and the back of the queue is to the right.)

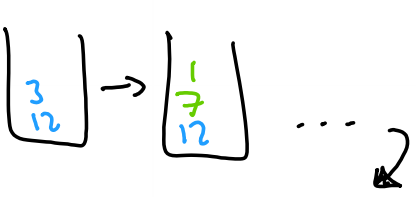


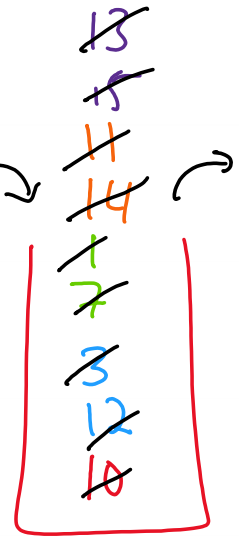

```
1 add A
2 add B
3 remove
4 add C
5 add D
6 remove
7 add E
8 remove
```

D, E

3 Iterative Traversals







3.1 Run this below pseudocode on the binary tree above in order to write its DFS traversal. When evaluating "Process removed node", print the node's value. When adding multiple children to the fringe at once, add the right child into the stack first, and then add the left child.

LIFO
last in first out

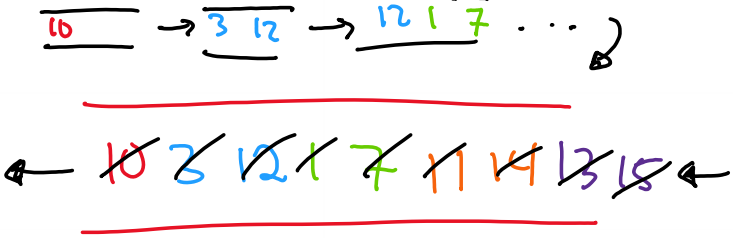
```
Use STACK as fringe.
Add root to fringe.

while fringe not empty:
  Remove next from fringe.
  Add its children to fringe.
  Process removed node.
```

DFS: 10 3 1 7 12 11 14 13 15

3.2 Now run this below pseudocode on the binary tree above in order to write its BFS traversal. When adding multiple children to the fringe at once, add the left child into the queue first, and then add the right child.

FIFO
first in first out




```
Use QUEUE as fringe.
Add root to fringe.

while fringe not empty:
  Remove next from fringe.
  Add its children to fringe.
  Process removed node.

```

BFS: 10 3 12 1 7 11 14 13 15

4 Follow-up Questions

4.1 Notice that the DFS traversal is the same as the pre-order traversal. In fact, pre-order, in-order, and post-order are all forms of DFS. How and why are they related? (Hint: Our iterative DFS uses a stack as its fringe, is there a stack used in pre/in/post-order?)

Call stack

4.2 Do all of these traversal algorithms we've learned today have the same runtime? If so, what is it, or if not, what are they?

$\Theta(N)$ - touch each node once!

4.3 If a binary tree has the following pre-ordering and in-ordering, what is its post-order? (Drawing the tree and thinking about what must be true about where each node is based upon the definitions will help you solve this.) Write your final answer on the line provided.

Pre-order: christone

In-order: rhctsoine

Post-order: rhtosenic