1.) What does the following expression evaluate to? \_\_\_\_\_\_\_\_\_\_\_\_

12 5 7 - 5 10 + \* +

2.) What is the average case order of the following method? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public int search(LinkedList<Double> list, Double tgt) {

int low = 0;

int high = list.size() - 1;

int result = -1;

while(result == -1 && low <= high) {

int mid = (low + high) / 2;

int dir = tgt.compareTo(list.get(mid));

if(dir == 0)

result = mid;

else if(dir < 0)

high = mid - 1;

else

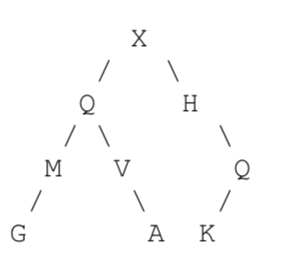
low = mid + 1;

}

return result;

}

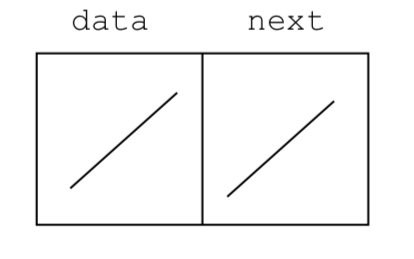
3.) Consider the following binary tree; “X” is the root of the tree.



1. What is the result of a pre-order traversal of the tree?
2. What is the result of an in-order traversal of the tree?

1. What is the result of a post-order traversal of the tree?

4.) Draw the variables, references, and objects that exist after the following code executes. Use arrows to show references and a forward slash to indicate variables that store null.



Node<Object> n1 = new Node<Object>(null, null); // data, next

Node<Object> n2 = new Node<Object>("ST", null);

n1.setData(n2.getData());

n1.setNext(n2);

n2.setNext(n1);

Node<Object> n3 = new Node<Object>(null, n2.getNext().getNext());

5.) What is output by the following code?

Queue314<Integer> q = new Queue314<Integer>();

for(inti=5;i>1;i-=2) {

q.enqueue(i);

q.enqueue(q.front() + i);

q.enqueue(q.front());

}

while(!q.isEmpty())

System.out.print(q.dequeue() + " ");

6.) Complete the removeBetween instance method for the LinkedList314 class. The method

removes all elements of the list between the first and second occurrence of a target value.

* You may not use any other methods in the LinkedList314 class unless you implement them yourself as a part of your solution.
* The LinkedList314 class uses singly linked nodes.
* The list has references to the first and last nodes in the chained structure of nodes.
* When the list is empty, first and last are set to null.
* None of the data in the list equals null.
* If the list is not empty the last node in the list has its next reference set to null.
* You may use the Node class and the Object equals method. You may not use other methods from LinkedList314 unless you implement them as part of your answer.
* You may not use any other Java classes or native arrays.
* Your method shall be as efficient as possible given the constraints of the question. Your method shall be O(1) space, meaning no matter how many elements are in the list your solution always uses the same amount of space.

public class LinkedList314<E> {

private Node<E> first;

private Node<E> last;

}

public class Node<E> {

public Node(E item, Node<E> next)

public E getData()

public Node<E> getNext(Node<E> n)

public void setData(E item)

public void setNext(Node<E> next)

}

Examples.

[].removeBetween(A) -> resulting list []

[A].removeBetween(A) -> resulting list [A]

[A, A].removeBetween(A) -> resulting list [A, A]

[A, A, A].removeBetween(A) -> resulting list[A, A, A]

[A, C, B, A].removeBetween(A) -> resulting list [A, A]

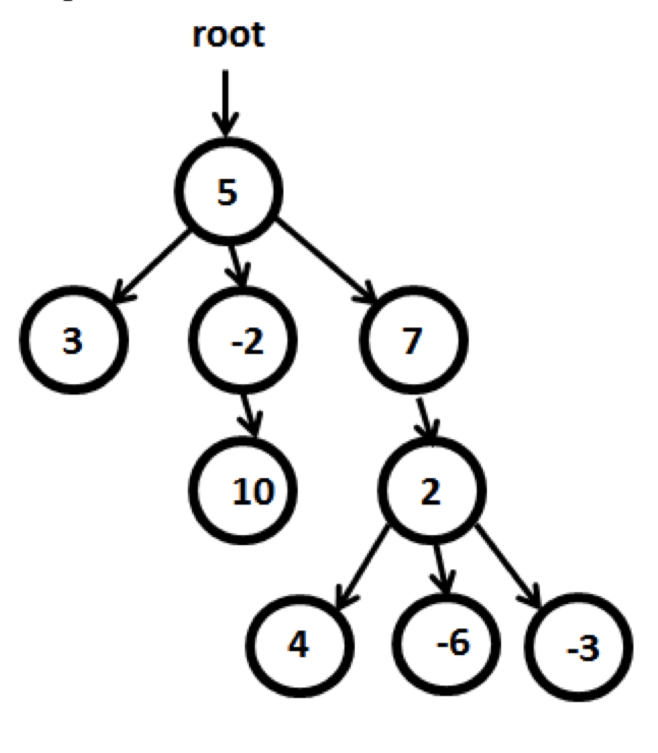
[C, C, C].removeBetween(A) -> resulting list [C, C, C]

[B, A, B, C, G, B, A, H, A, G].removeBetween(A) -> resulting list [B, A, A, H, A, G]

public void removeBetween(E tgt) {

7.) In this question you have a tree that stored ints in each node. The tree is NOT a binary tree. Nodes may have any number of children. Determine if there is a path in the tree from the root to descendant nodes so that the sum of the values in the nodes in the path equal some target value.

The path must start at the root and descend down the tree following the links that exist in the tree. The value of a node in the path must be included.

Given the tree to the left, the hasPath method would return the following:

hasPath(5) -> true: 5 in root hasPath(3) -> true: 5, -2 hasPath(13) -> true: 5, -2, 10 hasPath(18) -> true: 5, 7, 2, 4 hasPath(0) -> true (path with 0 nodes) hasPath(20) -> false

hasPath(1) -> false

hasPath(7) -> false (must use 5 in

root)

Assume the children of a node are numbered 0 to N - 1, left to right.

Complete the instance method hasPath(int tgt) for the IntTree class.

public class IntTree {

// if tree empty, root = null

private IntNode root;

private static class IntNode {

// value stored in this node

private int data;

// list of child nodes. If leaf node, children.size() == 0

private ArrayList<IntNode> children;

// rest of class not shown

}

Note, if the tree were empty hasPath(0) returns true. If the tree were empty, hasPath of any non- zero int returns false.

Complete the following instance method of the IntTree class. You may not use any classes other than the methods of the Java ArrayList. Do not use any auxiliary data structures.  
Add a recursive helper method.

// this IntTree is not altered as a result of this method call

public boolean hasPath(int tgt) {