UECS2083 Past Year 2016 Answer

1(a)

Tail recursion is a special kind of recursion where the recursive call is the *very last* thing in the function (means there is no pending operation to computed). It's a function that does not do anything *at all* after recursing.

1(b)(i) f2

1(b)(ii) 9 times

1(b)(iii) 36

1(c)(i) No. No.

1(c)(ii)

According to https://stackoverflow.com/questions/12359660/difference-between-complete-binary-tree-strict-binary-tree-full-binary-tre

Full binary is a tree where every node have 2 children, except for leaves (which are nodes that don't have children).

In another words, no node can have only 1 children.

```
public static boolean isFullBST(BinaryTree tree){
	return isFullNode(tree.root);
}

public static boolean isFullNode(TreeNode node) {
	if(node == null) {
		return true;
	} else {
		if(node.left == null && node.right == null) {
			// no children
			return true;
		} else if(node.left != null && node.right != null) {
			// recursion
			return isFullNode(node.left) && isFullNode(node.right);
		} else {
			// only 1 children
			return false;
		}
	}
}
```

2(a)(i)

First, using Array.
Secondly, using LinkedList.

2(a)(ii)

Refer https://eddmann.com/posts/implementing-a-queue-in-java-using-arrays-and-linked-lists/

LinkedList, because LinkedList has unlimited size (excepted bounded by physical constraint such as RAM size). For Array, you need to constantly resize it, thus is less efficient.

4(b)

```
protected void nonRecursiveInorder(TreeNode<E> root) {
    Stack<TreeNode<E>> stack = new Stack<TreeNode<E>>();

    TreeNode<E> current = tree;
    while(true) {
        if(current != null) {
            stack.push(current);
            current = current.left;
        } else {
            if(stack.size() > 0) {
                current = stack.pop();
                System.out.println(current.element);
                current = current.right;
            } else {
                break;
            }
        }
    }
}
```