COMS3134 Data Structures and Algorithms
Spring 2017 Section 2 Midterm Solutions

1.

Let k be the height of a full binary tree.
Base case: if k = 0, N = 1. So the claim holds true for our base case.
Inductive hypothesis: assume the claim holds true for some full binary tree of height k0.
Inductive step: we need to show that the claim holds for a full binary tree T of height k = k0+1
The left and right subtree of T must also full, since every interior node of T has two children, and that the height of T.left and T.right must be at most k0.
According to the hypothesis, let 2m+1 and 2n+1 be the number of nodes in T.left and T.right respectively.
Then the number of nodes in T is:
(2m+1) + (2n+1) + 1 = 2(m+n)+3
which is an odd number.

2.

a)

1. O(N)
2. O(N)
3. O(2^N)
4. O(1)

b) O(1), O(N), O(N), O(2^N)

3.

a) This sequence is impossible. First, we need to print L, so push(P), pop()->p. Then, to print the only L, we must move A, R, S out of the way: push(A), push(R), push(S), push(L), then pop()->L. Now we would need to pop A next, but it is blocked by S and R on top of the stack.

b)

```
push(P),   pop()-> P,
push(A),   pop()-> A,
push(R),   pop()-> R,
push(S),
push(L),   pop()-> L,
push(E),   pop()-> E,
push(Y),   pop()-> Y,
           pop()-> S
```

4.

```
          F
       /     \
     A        B
   /  \    /
  E    D  C
```

Pre order traversal: FAEDBC

5.

```
int calcHeight(TreeNode root) {

    if (root == null) {
        return -1;
    }

    return (1 + Math.max(calcHeight(root.left), calcHeight(root.right)));
}
```

6.

```
public push(int x) {
    Node n = new Node();
    n.data = x;
    n.next = tail;
    n.prev = tail.prev;
    tail.prev.next = n;
    tail.prev = n;
}

public int pop() {
    if (tail.prev == head) {
        throw new EmptyStackException();
    }

    int result = tail.prev.data;
    tail.prev.prev.next = tail;
    tail.prev = tail.prev.prev;
    return result;
}
```

7) AVL Tree

```
4
```

```
4
 \
  5
```

```
  4
 / \
3   5
```

```
    4
  /   \
3       5
          \
            8
```

```
    4
  /   \
3       5*
          \
            8
          /
        7
```

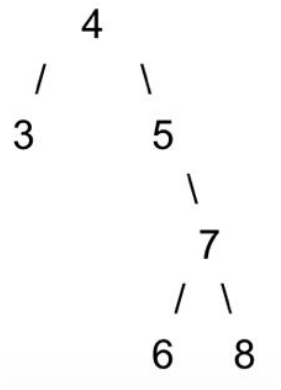
* Imbalance (double rotation)

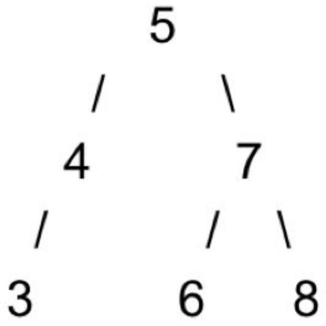
```
    4
  /   \
3       5
          \
            7
              \
                8
```

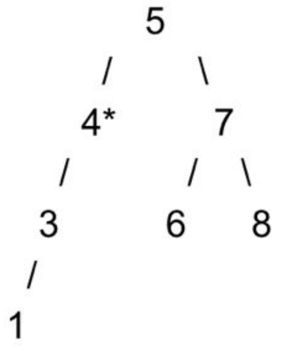
```
    4
  /   \
3       7
      /   \
    5       8
```

```
      4*
    /   \
  3       7
        /   \
      5       8
        \
          6
```

* Imbalance (double rotation)







* Imbalance (single rotation)