

COMS3134 Data Structures and Algorithms
Spring 2017 Section 1 Midterm Solutions

1.

Let k be the height of the perfect binary tree.

Base case: if $k = 0$, $N = 1$. So the hypothesis holds true for our base case.

Hypothesis: assume the claim holds true for some k_0 .

Inductive step: we need to show that the claim holds for tree with height $k = k_0 + 1$

To increase the height of a tree by 1, we need to add two children to each leaf node of the old tree to make it a perfect binary tree.

If m is the number of leaf nodes in the old tree, we add $2m$ nodes, which is even, to the old tree.

Since the number of nodes in a tree of height k is odd according to the hypothesis, adding $2m$ nodes to the old tree makes the total number of nodes in the new tree odd (QED).

Fast, mathematical way:

We note that for a perfect tree of height h , # nodes = $2^{(h+1)} - 1$.

Assume it holds for a perfect tree of height h , i.e. $(2^{(h+1)} - 1)$ is odd.

Then for a tree of height $h+1$, # nodes = $2^{(h+1+1)} - 1 = 2(2^{(h+1)}) - 1$, and since $2^{(h+1)}$ is an integer, $2(2^{(h+1)}) - 1$ must be odd.

2.

a)

1. $O(N)$
2. $O(\log N)$
3. $O(2^N)$
4. $O(1)$

b) $O(2^N)$, $O(N)$, $O(\log N)$, $O(1)$

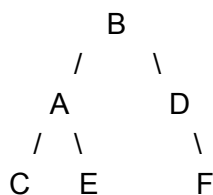
3.

a)

push(R),
push(E),
push(S), pop()-> S,
pop()-> E,
push(C), pop()-> C,
push(U), pop()-> U,
pop()-> R,
push(E), pop()-> E,
push(D), pop()-> D

- b) This sequence is impossible. To print R we must push(R), pop()->R. Next we need to print one of the two Es.
 Assume we decide to use the first E, so push(E), pop()->E. We need to print the D next, so we must push SCUE out of the way: push(S), push(C), push(U), push(E), push(D), pop()->D. Next we would need to pop U, but it is blocked by E on the stack.
 If we decide to use the second E, then we need to push ESCU out of the way: push(E), push(S), push(C), push(U), push(E), pop()->E. Then push(D), pop()->D. Next pop()->U, pop()->C, then we should pop E but it is blocked by S on the stack.
 (Only one of the two E cases was required for full credit).

4.



Post order traversal: CEAADB

5.

```

boolean isFull(TreeNode root) {

    if (root == null) {
        return true;
    }
    if (root.left != null && root.right == null){
        return false;
    }
    if (root.left == null && root.right != null){
        return false;
    }
    return (isFull(root.left) && isFull(root.right));
}
  
```

6.

```

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

```
public int dequeue() {  
    if (tail.prev == head) {  
        throw new EmptyQueueException();  
    }  
  
    int result = tail.prev.data;  
    tail.prev.prev.next = tail;  
    tail.prev = tail.prev.prev;  
    return result;  
}
```

7) AVL Tree

2

2
/
1

2
/ \
1 6

2
/ \
1 6
 \
 7

2
/ \
1 6*
 \
 7
 \
 8

* Imbalance (single rotation)

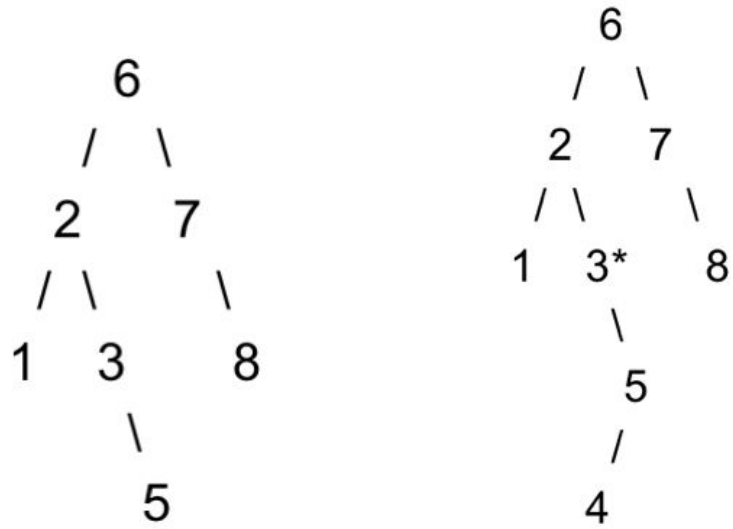
2
/ \
1 7
 /
 6
 \
 8

2*
/ \
1 7
 /
 6
 /
 3

* Imbalance (double rotation)

2
/ \
1 6
 /
 3
 \
 7
 \
 8

6
/ \
2 7
/ \
1 3
 \
 8



* Imbalance (double rotation)

