COMMON INTERVIEW QUESTIONS
A COMPILED LIST

JOHN WANG

1. Tree Questions

1.1. **Successor Nodes. Problem:** Given a binary tree, find the successor for any element x.

Solution: Notice that the next larger element of x in a binary tree will either be 1) in an x's right subtree or 2) x will be in the left subtree of its successor. Moreover, we know the exact positions for each of these cases of the successor. In the first case we know that the successor has to the be smallest element in the right subtree (the left-most element). In the second case, the successor will be the smallest element whose left subtree contains x.

We now have enough information to code this up:

```
def get_successor(node)
  if node.right
    get_min(node.right)
  else
    get_parent_successor(node)
  end
end

private

def get_parent_successor(node)
  return nil unless node.parent

  if node.parent.left == node
    node.parent.left
  else
    get_parent_successor(node.parent)
  end
end

def get_min(node)
  if node.left
    get_min(node.left)
  else
    node
  end
end
```

The runtime is $O(h)$ where h is the height of the tree. However, note that if we get each successor, starting from the minimum element, it will only take $O(n)$ time (where n is the number of nodes in the tree). This is because we only traverse each edge twice: once going down the edge and once coming back up the edge. Therefore, we only make $2m$ traversals, where $m = n - 1$. □

1.2. **Printing out Sequential Levels. Problem:** Given a tree (not necessarily a binary tree), print out the levels of the tree sequentially (with nodes ordered from left to right within each level).

Solution: There are a couple of different ways to do this. One manner of doing this is to use a BFS, making sure to order the nodes so they are traversed by the left-most child first. Let's assume that each

node has a ordering property which ensures that if $x.ordering < y.ordering$ then x will be further left in the tree than y if x and y are at the same distance away from the root.

```
def print_tree(root)
  puts "#{root}\n"
  current_children = root.children.sort_by { |child| child.ordering }

  while current_children
    puts "#{current_children.join(',')}\n"
    current_children = current_children.map { |child| child.children }.flatten
  end
end
```

□

1.3. **Lowest Common Ancestors. Problem:** Find the lowest common ancestor of two nodes on a tree (not necessarily a binary tree).

Solution: First, let's solve a simpler problem. If we know that two nodes are at the same height in the tree, can we find the lowest common ancestor? Yes, it's pretty easy, we just move up parent pointers until the pointers end up at the same node. Now, we just need to solve this for the full problem.

It's easy to see that we can reduce our problem to the simple case. For each node, traverse up the tree until you reach the root, keeping track of the number nodes it took to reach the root. This will calculate the distance to the root for each root. Now, move up parent pointers from the node with the greater distance from the root until the distances are the same. We've now reduced our problem to the simple case.

Let's code this up:

```
def get_lowest_common_ancestor(first_node, second_node)
  first_distance = distance_to_root(first_node)
  second_distance = distance_to_root(second_node)

  if first_distance > second_distance
    (first_distance - second_distance).times { first_node = first_node.parent }
  elsif second_distance > first_distance
    (second_distance - first_distance).times { second_node = second_node.parent }
  end

  while first_node != second_node
    first_node = first_node.parent
    second_node = second_node.parent
  end

  first_node
end

private

def distance_to_root(node)
  distance = 0
  while node.parent
    node = node.parent
    distance += 1
  end

  distance
end
```

The solution has runtime of $O(h)$ where h is the height because we never traverse the height of the tree more than 4 times (twice to find the height for each node, then twice to proceed to the lowest common ancestor). $\square$