CSC 120 EXAM 2 REVIEW GUIDE

WHERE TO STUDY:

- Weekly Videos
- ICAs
- This Study Guide

QUESTIONS?

- Ask Your UGTA
- Discord
- Office Hours

Note: These are not necessarily the questions that will be on your test. These questions were written collectively by the TAs (who haven't seen the test yet) and resemble what we think you need practice with for the test. Do not use this as your only resource for studying.

That being said, feel free to jump between questions, and do the ones you think will help you the most. Ask questions on Discord, and attend office hours for additional clarification.

For all problems, you may refer to the following `TreeNode` class definition:

```python
class TreeNode:
    def __init__(self, value, left, right):
        self.value = value
        self.left = left
        self.right = right
```

Review Problems

1. Write a recursive function named `triangle_pattern(n)` that generates a pattern resembling an inverted triangle using `*`, such that each row contains decreasing number of asterisks. You can assume `n` is a positive integer.

   Example:

   ```
   >>> triangle_pattern(7)

   * * * * * * *
   * * * * * *
   * * * * *
   * * * *
   * * *
   * *
   *
   ```

2. What will be printed by the following code:

   ```
   def foo(n):
       if n == 0:
           return
       foo(n // 2)
       print(n)

   foo(64)
   ```

3. Draw the binary search trees that result from the following insertion orders:

   a)

```
60, 15, 34, 8, 61, 4, 9, 0, 16, 7, 63
```

   b)

```
0, 15, 8, 63, 34, 7, 60, 4, 16, 61, 9
```

4. Given a binary tree, write a recursive function `invert(root)` to invert the tree (the left sub-tree becomes the right sub-tree, and vice versa)

   Example:

```
      9                                9
     / \                              / \
    4   8                            8   4
   /   /              =>              \   \
  11  13                              13  11
 / \                                     /  \
7   2                                   2    7
```

5. Given the following binary tree, what are the pre-order, in-order, and post-order traversals?

```
       4
    /     \
   3       7
 /   \    /
1     4  9
 \
  6
```

6. Write a recursive function `reverse_array(data)` that gets a list (array) as input and returns the reversed of that list.

   Example:

```
>>> reverse_array([0,1,2,3])
[3,2,1,0]
```

7. Write a recursive function named `power(base, n)` that gets a natural number `n` and `base` and returns the value of base ^{n} as the output.

   Example:

```
>>> power(2, 2)
4
>>> power(5, 3)
125
```

8. What aspect of BSTs allows a search to be ran in O(logn) time if it is a balanced tree?

9. Write a recursive function named `count_elements(arr)` to count the number of elements in an array without using the `len` function.

10. Write a recursive function named `tree_height(root)` to find the height of a binary tree. The height of a tree is the number of links from the root to the deepest leaf.

11. Write a recursive function named `count_odd_values(arr)` to count the number of odd values in an array of integers.

12. Explain the steps to insert the value 15 into the following BST and draw the final BST.

```
      20
     /  \
    10   30
   / \
  5   12
```

13. What will be the output of the following code when the function `sum_leaf_nodes(root)` is called with the root of the given tree?

```python
def sum_leaf_nodes(root):
    if root is None:
        return 0
    if root.left is None and root.right is None:
        return root.value
    return sum_leaf_nodes(root.left) + sum_leaf_nodes(root.right)

# Creating the binary tree
root = TreeNode(10)
root.left = TreeNode(5)
root.right = TreeNode(15)
root.left.left = TreeNode(3)
root.left.right = TreeNode(7)
root.right.right = TreeNode(18)

# Calling the function
print(sum_leaf_nodes(root))
```

14. Consider the following code that finds the maximum value in a binary tree. What will be the output when the function `find_max(root)` is called with the root of the given tree?

```python
def find_max(root):
    if root is None:
        return float('-inf')
    left_max = find_max(root.left)
    right_max = find_max(root.right)
    return max(root.value, left_max, right_max)

# Creating the binary tree
root = TreeNode(10)
root.left = TreeNode(20)
root.right = TreeNode(30)
root.left.left = TreeNode(40)
root.left.right = TreeNode(50)
root.right.left = TreeNode(25)
root.right.right = TreeNode(5)

# Example usage
print(find_max(root))
```

Solutions

1.

```
def triangle_pattern(n):
    if n == 0:
        return
    else:
        print('* ' * n)
        triangle_pattern(n - 1)
```

2.

```
1
2
4
8
16
32
64
```

3.

a)

```
              60
            /    \
          15      61
        /   \       \
       8     34     63
     /  \    /
    4   9  16
   / \
  0   7
```

b)

```
        0
         \
         15
       /    \
      8      63
     / \      /
    7   9  34
   /      /  \
  4     16   60
              \
               61
```

4.

```python
def invert(root):
    if root == None:
      return None
    else:
      invert(root.left)
      invert(root.right)
      temp = root.left
      root.left = root.right
      root.right = temp
```

5.

Pre-order:

```
4, 3, 1, 6, 4, 7, 9
```

In-order:

```
1, 6, 3, 4, 4, 9, 7
```

Post-order:

```
6, 1, 4, 3, 9, 7, 4
```

6.

```
def reverse_array(data):
    if data == []:
        return []
    return [data[-1]] + reverse_array(data[:-1])
```

7.

```
def power(base, n):
    if n == 0:
        return 1
    return base * power(base, n-1)
```

8. Since BSTs are in a way sorted, BSTs can run binary search and through a quick comparison, divide the problem in half by knowing whether the value is in the left or right sub-tree if the search has to continue. By cutting the problem in half each time, it runs much quicker than having to check every node's value.

9.

```python
def count_elements(arr):
    if arr == []:
        return 0
    else:
        return 1 + count_elements(arr[1:])
```

10.

```python
def tree_height(root):
    if root is None:
        return -1  # Height of an empty tree is -1
    else:
        left_height = tree_height(root.left)
        right_height = tree_height(root.right)
        return max(left_height, right_height) + 1
```

11.

```python
def count_odd_values(arr):
    if arr == []:
        return 0
    elif arr[0] % 2 != 0:
        return 1 + count_odd_values(arr[1:])
    else:
        return count_odd_values(arr[1:])
```

12.

1. Start at the root node (20).
2. Since 15 is less than 20, move to the left child (10).
3. Since 15 is greater than 10, move to the right child (12).
4. Since 15 is greater than 12 and there is no right child, insert 15 as the right child of 12.

   The tree after insertion:

   ```
       20
      /  \
     10   30
    / \
   5   12
         \
         15
   ```

13. The output will be 28 (3 + 7 + 18 = 28).

14. The output will be 50.

    Tree Structure:

    ```
          10
        /    \
      20      30
     / \     /  \
    40  50 25    5
    ```