CSC 120 EXAM 2 REVIEW GUIDE

Exam: Friday April 01, 2022

Review Session: March 30, 2022; 5:30-7:00 @ BIOW 301

TOPIC LIST:

1. Classes
2. Recursion
3. Trees
4. Debugging

WHERE TO STUDY:

- Slide-decks
- 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 questions, and do the ones you think will help you the most. Ask questions on Discord, and attend the review session for additional clarification.

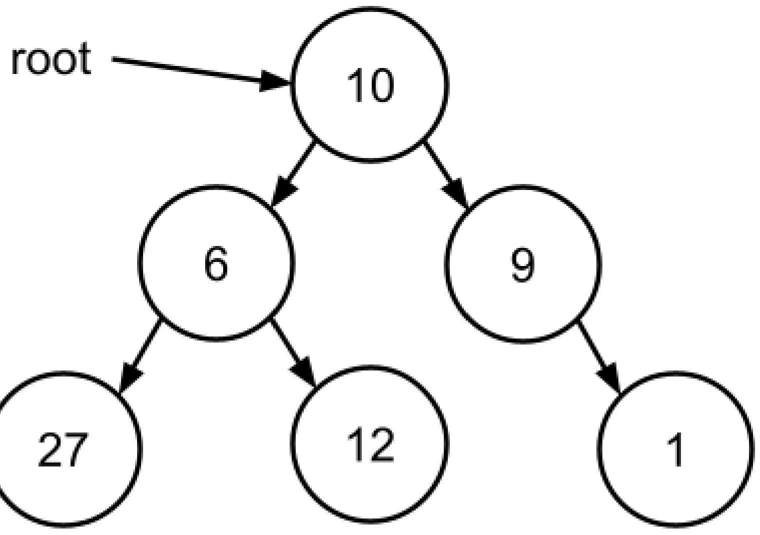
REVIEW PROBLEMS:

1. Given a linked list where all the values are integers, write a **recursive** function `remove_odd(head)` that takes in the head of a linked list. The function should remove all the nodes where the value of the node is odd and then return the list. This was done in the first study guide, but now do it recursively!

   Ex:
   Given: 4 -> 3 -> 2 -> 6 -> 1 -> 45, return: 4 -> 2 -> 6
   Given: 2 -> 22 - > 34 -> 6, return: 2 -> 22 - > 34 -> 6
   Given: 5 -> 7 -> 11 -> 167, return: None

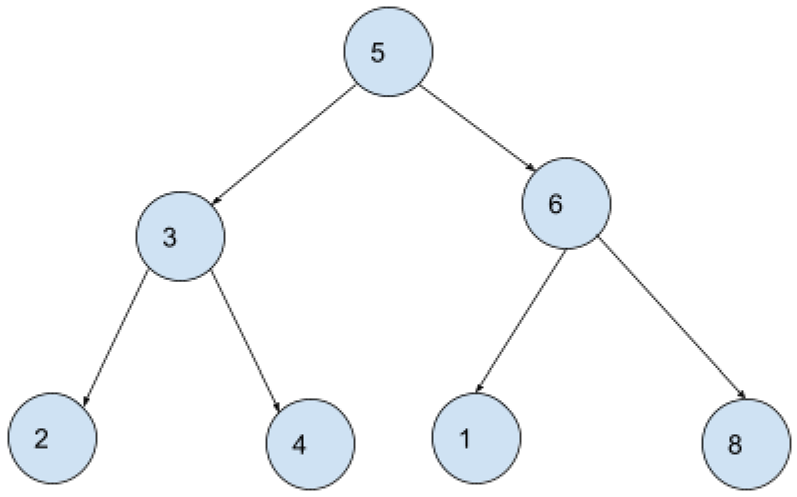
2. Given the root of a binary tree, write a **recursive** function `count_even_parents(root)` which returns the number of nodes in the binary tree which are 1) even and 2) are parents of at least one node.

   Ex:
   Given:



   return: 2

3. Give the preorder, inorder, and postorder traversal of the following tree:



4. What is the output of the following code?

```
def mystery(num, num_array):
     mystery_helper(num, num_array)
     return num, num_array

def mystery_helper(num, num_array):
     num +=10
     num_array[1] = 40

num = 5
num_array = [10, 20, 30]
num, num_array = mystery(num, num_array)
print(num)
print(num_array)
```

5. Create a `MarchMadness` class which has an `init` with `String name`, `int num_players`, `int cur_seed`, and `float avg_hei` as attributes; add getters for the variables

6. Create a `MarchMadness` object called `Arizona` where there are 18 players, they are seeded at number 1, and the average player height is 6.66 (6 feet and 8 inches).

7. Explain why Trees and/or Linked Lists are recursive by nature.

8. Write a function called `binary_tree_to_linked_list(root)` which takes the root of a binary tree and returns a linked list of all the values (order does not matter).

9. Which of the following is considered a BST?
   On the one that isn't a BST, which value do we need to change for it to become a BST?

```
tree1 = TreeNode(48)
tree1.left = TreeNode(22)
tree1.right = TreeNode(58)
tree1.left.left = TreeNode(19)
tree1.left.right = TreeNode(36)
tree1.right.left = TreeNode(27)
tree1.right.right = TreeNode(99)
```

```
tree2 = TreeNode(40)
tree2.left = TreeNode(25)
tree2.right = TreeNode(55)
tree2.left .left = TreeNode(17)
tree2.left .right = TreeNode(38)
tree2.right.left = TreeNode(47)
tree2.right.right = TreeNode(97)
```

10. In what situations would you use a binary search tree over an array or linked list? (Hint: think about time complexity)

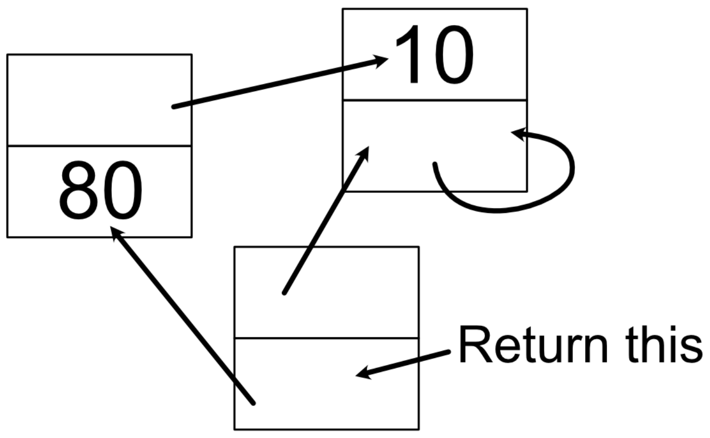
11. Find the bug in the following code that is attempting to find and return the index for a specified value in an array. You can assume that `arr` is an array that contains integers, and that val is an integer. (Hint: There are 2 bugs!)

```
def find_index(arr, val, index = 0):
    #index defaults to 0 if not given as a parameter.

    if arr[0] == None:
        return -1
    elif arr[0] == val:
        return index
    else:
        find_index(arr[1:], val, index + 1)
```

12. Define the following terms, in the context of Data Structures:
    -Parent
    -Child
    -Root
    -Leaf
    -Subtree
    -Head

13. Write the function `create_shape()` that returns the following data structure. Each object is an array



14. What is the `x = change(x)` style? Why is it helpful?

15. List the pros and cons of the following data structures:
    - Arrays (Python Lists)
    - Linked Lists
    - Trees
    - Dictionaries
    - Sets

16. What is the difference between a Binary Tree and a Binary Search Tree?

17. What is the Big-O value for finding a value in a Binary Tree? What about a Binary Search Tree?

18. Write a class that is a bare-bones version of QueueBot called `QueueBot` that stores an **array** of strings where each string represents a student. When the user calls `add(val)`, the student should be added to the end. When the user calls `remove()`, it should remove the student who has been waiting the longest. `remove()` should also assert there is at least one student within the waiting list. Lastly, you should implement a `list()` method which returns a single string with all the students in it (from longest waiting to shortest waiting) with a comma separating each one.

19. What is the difference between the `==` and `is` operators?

20. Suppose you have a variable pointing to `None`. What does it represent?

21. Create the function `remove_second(head)` which takes a head to a linked list and removes the second-to-last element. (Hint: Remember to test your function on any corner cases).

22. Write a function `bst_remove(root, val)` that inserts a node with the specified value into a BST.

23. Given a binary search tree containing [1,12,5,66,-5,0,3], what would the In-Order, Pre-Order, and Post-Order traversals print. If there is not enough information, write "Unknown".

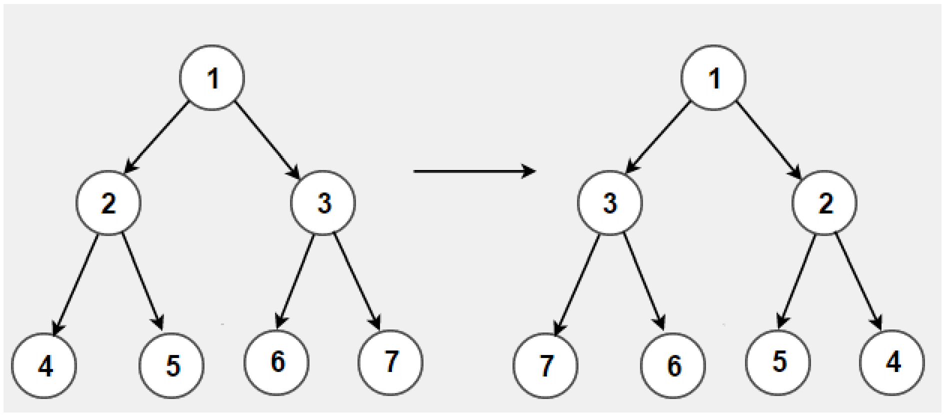
24. Given a **balanced** binary search tree containing [1,12,5,66,-5,0,3], what would the In-Order, Pre-Order, and Post-Order traversals print. If there is not enough information, write "Unknown".

25. I like to climb stairs. I also like to know how many stairs I have climbed every day. Write me a class that will keep track of how many stairs I have climbed. I want all of the fields to be private. I also want to be able to add one step at a time, subtract a step (when I walk down stairs backwards), and reset the amount of stairs that I have climbed that day. I would also like to be able to add two stairs in one method for when I do a super stair step (2 stairs at once).

26. What is the maximum possible height of a binary search tree if we insert N elements, explain? What is the minimum possible height of a binary tree if we insert N elements, explain? Does the height of a binary search tree play a role in the operations (search, insert, delete), and if so, how and why?

27. If you are given a sorted array of integers from 1 to 10 (e.g. [1,2,3,...,9,10]), how would you pick these integers to insert into a binary search tree to yield the greatest height possible? How would you pick these integers to create a balanced search tree?

28. Challenge: Given a binary tree, invert the tree. That is, the right and left children of each node must be swapped, as shown in the example below. The function will be called invertTree(root) where root is the tree to be inverted.



SOLUTIONS:

1.
```
def remove_odd(head):
      if head is None:
            return head
      if head.val % 2 != 0:
            head = head.next
      head.next = remove_odd(head.next)
      return head
```

2.
```
def count_even_parents(root):
      if root is None
            return 0
      elif root.left is None and root.right is None:
            return 0

      if root.val % 2 == 0:
            return 1 + count_even_parents(root.left) +
                    count_even_parents(root.right)
      else:
            return count_even_parents(root.left) +
                     count_even_parents(root.right)
```

3. Preorder - 5, 3, 2, 4, 6, 1, 8

   Inorder - 2, 3, 4, 5, 1, 6, 8

   Postorder - 2, 4, 3, 1, 8, 6, 5

4. 5

   [10, 40, 30]

5.
```
class MarchMadness:
     def __init__(self, name, num_players, cur_seed, avg_hei):
          self._name = name
          self._num_players = num_players
          self._cur_seed = cur_seed
          self._avg_hei = avg_hei

     def get_name(self):
          return self._name

     def get_num_players(self):
          return self._num_players

     def get_cur_seed(self):
          return self._cur_seed

     def get_avg_hei(self):
          return self._avg_hei
```

6.
```
arizona = March_Madness("Arizona", 18, 1, 6.66)
```

7. For trees, the root has multiple children, and each child node is either None or another TreeNode. In the case where a child is another TreeNode, you can treat the child as the new root in recursive calls. Each tree holds in itself multiple other trees, each their own root in their own recursive function call.

   The same can be applied to LinkedLists. The next field of a head node is either None or another ListNode. In the case where next is another ListNode, you can treat the next as the current head in recursive calls. Each linked list holds in itself a smaller linked list, which is its own head in its own recursive function call.

8.
```
def binary_tree_to_linked_list(root):
    if root is None:
        return None

    head = ListNode(root.val)
    head.next = binary_tree_to_linked_list(root.left)

    cur = head
    while cur.next is not None:
        cur = cur.next

    cur.next = binary_tree_to_linked_list(root.right)

    return head
```

9. Tree 2 is a BST.

   (Either one is correct down below)

   On line 5 for tree 1, change the value so that it 22 < val < 27

   OR

   On line 6, change the value so that it is 36 < val <5

10. Since most BST operations are O(logN), if an algorithm needs to do frequent element accessing along with some combination of searching, insertion, or deletion, then it makes sense to use a BST instead of a linked list or array.

Time Complexity (Average case)	BST	Linked List	Unsorted Array	Sorted Array
Access	O(logN)	O(N)	O(1)	O(1)
Searching	O(logN)	O(N)	O(N)	O(logN) (binary search)
Insertion	O(logN)	O(1)	O(N)	O(N)
Deletion	O(logN)	O(1)	O(N)	O(N)

11. 1) The function needs to return the output of the function call. In its current state, when the function's return value isn't stored, the code will return `None` instead

    2) The code will break if val is not in arr, since the base-case `if arr[0] == None` checks whether the first element is `None`, instead of whether the array is empty. Instead, something like `if arr == []` would work.

12.

Parent: The node that is the immediate predecessor of a node is the parent of that node.
Child: The node that is the immediate successor of a node is the child of that node.
Root: The node at the top of the tree. (Also, the node that doesn't have parents *unless* it's the root of a subtree.)
Leaf: The node(s) that do not have any children nodes. (The nodes at the bottom of the tree)
Subtree: A tree that is stored in another tree.
Head: The first node of a linked list.

13.
```
def create_shape():
    # It is impossible to point to a specific slot in an array.
    # It is only possible to point to a stored object or the
whole array.
    # Therefore, each arrow points to the *whole array*.
    left = [None, 80]
    right = [10, None]
    bottom = [right, left]
    left[0] = right
    right[-1] = right

    return bottom
```

14. `x = change(x)` is a style used to help with updating data structures. With this style, we have some structure x (this could be a tree, linked list, etc.) This is then passed into some function which changes x and returns the updated data structure. This allows for very clean code as we don't need to account for empty data structures (see previous ICAs for examples).

15.

Data Structure	Pros	Cons
Arrays (Python Lists)	- Very simple data structure - Allows for O(1) indexing - Items have specific ordering	- O(n) to insert in middle
Linked Lists	- O(1) insertion (assuming you have a reference to a node) - Ordered	- Not built into Python - No indexing
Trees	- Allows for hierarchical data storage - Insertions generally O(lg n)	- Not built into Python - More difficult to traverse than arrays - Unbalanced trees can have O(n) performance instead of O(lg n)
Dictionaries	- Maps keys to values - O(1) access via keys	- Hard to get keys with just values - Unordered
Sets	- Removed duplicates - Easy to check if a set contains an item	- Doesn't have an ordering - Can't index

This table is not exhaustive.

Ordered/Unordered and Duplicates/No Duplicates can be a pro or con depending on the viewpoint.

16. A Binary Tree is a recursive data structure where each node can have up to two children; each of those children being binary trees as well. A Binary Search Tree is a specific type of Binary Tree where given a root, the *entire* left subtree has values less than the root while the *entire* right subtree has values greater than the root.

17. Because a Binary Tree is not sorted, we would have to do a brute-force search (look at every node) - aka `O(n)`. A Binary Search Tree has a specific structure that allows us to do this in `O(lg n)`. This is for the same reason Binary Search is `O(lg n)`. For each node we visit, we exclude half the nodes at that level (See PlayPosit Trees - Part 2).

18.
```
class QueueBot:
    def __init__(self):
        self._list = []

    def add(self, val):
        self._list.append(val)

    def remove(self):
        assert len(self._list) > 0
        return self._list.pop(0)

    def list(self):
        return ", ".join(self._list)
```

19. The `==` operator checks the value of the two objects. The `is` operator checks the reference of the two objects. `==` will return true if the values (things they are storing) are the same. `is` checks to see if the objects are the exact same object.

For example:
```
a = [1, 2, 3]
b = [1, 2, 3]
a == b #returns True
a is b #returns False
```

20. This variable can represent many things. `None` is often used to represent "nothingness." `None` is also used to represent an empty Linked List as well as an empty Binary Tree.

21.
```
def remove_second(head):
    # List of length 0
    if head is None:
        return None
    # List of length 1
    if head.next is None:
        return head
    # List of length 2
    if head.next.next is None:
        return head.next

    curr = head
    # We need to stop on the 3rd to last node
    while curr.next.next.next is not None:
        curr = curr.next

    # Remove the node in front of the third-to-last node
    curr.next = curr.next.next
    return head
```

22.
```
def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val == key:
           return root
        elif root.val < key:
            root.right = insert(root.right, key)
        else:
            root.left = insert(root.left, key)
    return root
```

23. [1,12,5,66,-5,0,3] - Binary Tree
    a. In-Order: -5, 0, 1, 3, 5, 12, 66
    b. Pre-Order: Unknown
    c. Post-Order: Unknown

24. [1,12,5,66,-5,0,3] - Balanced Binary Tree
    a. In-Order: -5, 0, 1, 3, 5, 12, 66
    b. Pre-Order: 3, 0, -5, 1, 12, 5, 66
    c. Post-Order: -5, 1, 0, 5, 66, 12, 3

25.
```
class CountStairs:
	def __init__(self):
		self._count = 0
	def step(self):
		self._count += 1
	def superStep(self):
		self._count += 2
	def backwardStep(self):
		self._count -= 1
	def getStairCount(self):
		return self._count
	def resetCount(self):
		self._count = 0
```

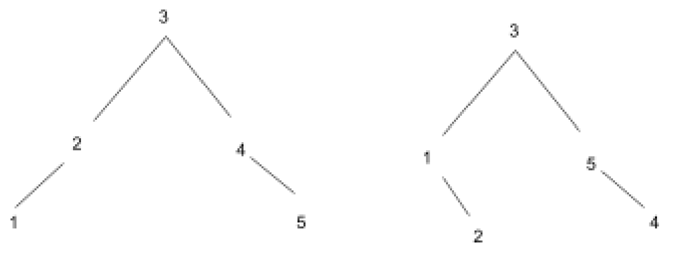
26. If we insert N elements into a binary search tree, the maximum possible height is (N - 1) (NOT N, because height is based on the number of edges from the root to the lowest leaf): If every node inserted into the tree is inserted as the parent's right child or left child, e.g. insert 1, then insert 2, then insert 3, etc..

    If we insert N elements into a binary search tree, the minimum possible height is LogN : It was discussed in lecture/slides that a balanced binary search tree will have O(LogN) runtime for each operation (insert, delete, search), because the height of the tree is LogN

    The height of a binary search tree does play a role in the operations. These operations are based solely on how many nodes we have to traverse to complete the operation (finding the node to delete, finding where to insert the node, finding the node).

27. The maximum possible height is obtained when every node goes to the right or left child of its parent. So, the order to insert these integers to create the maximum possible height is inserting 1, then 2, then 3, etc… or inserting 10, then 9, then 8, etc…

    To approach this problem, start with the array [1,2,3]. In this case, the balanced tree is 2 as the root, 1 as the left child, 3 as the right child. What about [1,2,3,4,5]? 3 as the root, then either 1 or 2 as its left child, then either 4 or 5 as its right child, then either 1 or 2 as 1 or 2’s left/right child, then either 4 or 5 as 4 or 5’s left/right child, like so:



    (Note there are 4 possible balanced trees with array [1,2,3,4,5])

    The approach seems to be to set the middle element as the root (if there is no middle pick left or right), then divide the array into two sub arrays and pick the middle element of those sub arrays to be the left and right children of the root, and so on for the rest of the tree. One possible balanced tree that can be created from the array [1,2,...,9,10], is made inserting in this order:
    5->3->2->1->4->8->7->9->10->6

28.
```
def invertTree(self, root):
        if root is None:
             return None
         root.left, root.right = \
             invertTree(root.right), invertTree(root.left)
         return root
```