CS-UY 1134 - Fall 2017 midterm 2

Justin Lin

TOTAL POINTS

67 / 100

QUESTION 1

1 Question 1 18 / 18

✓ - 0 pts Correct

QUESTION 2

2 Question 2 12 / 12

✓ - 0 pts Correct

QUESTION 3

3 Question 3 19 / 20

✓ - 2 pts Did not increment size

+ 1 Point adjustment

QUESTION 4

4 Question 4 3 / 20

✓ - 20 pts Wrong

+ 3 Point adjustment

QUESTION 5

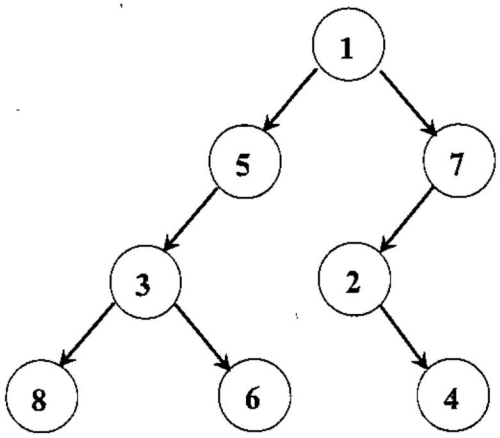
5 Question 5 15 / 30

✓ - 15 pts Failing constant time requirement

Loop in pop_dups violates O(1) time constraint.

Question 1 (18 points)

Give the preorder, postorder and inorder traversal sequences, for the binary tree given below.



CLR
Preorder: 1 5 3 8 6 7 2 4

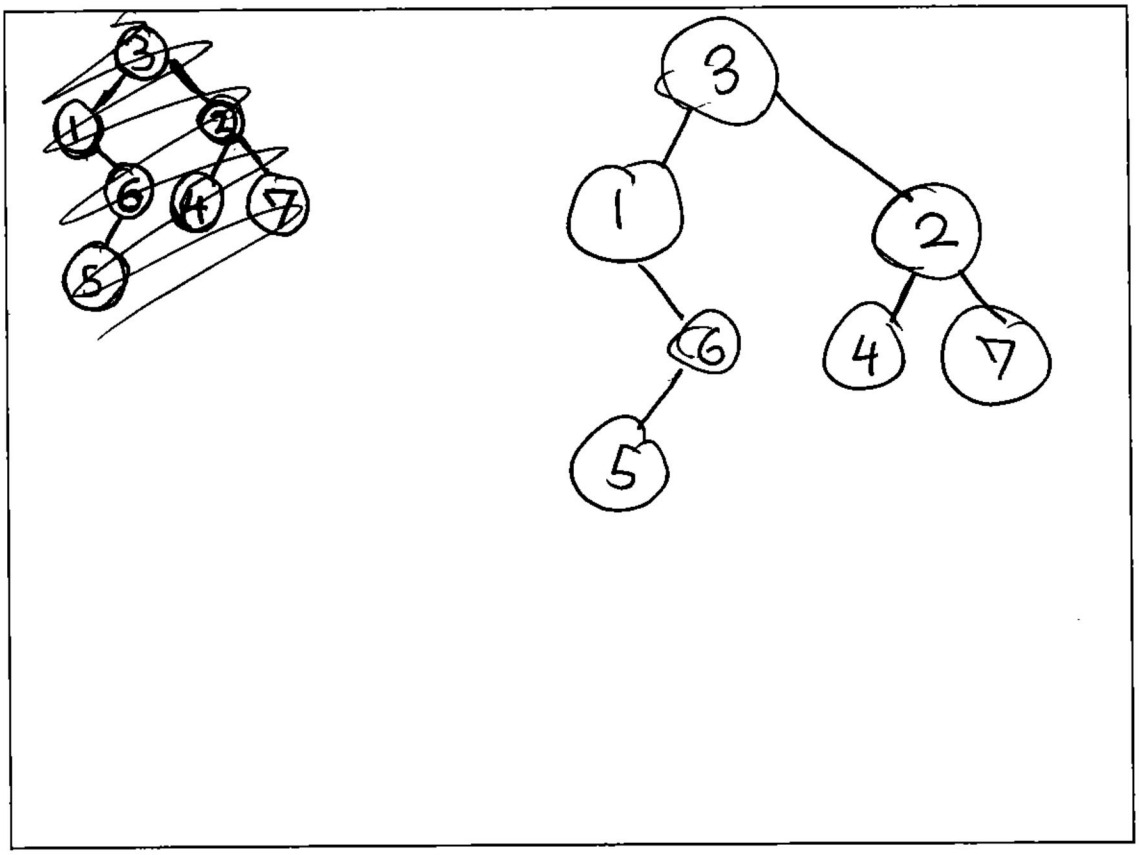
LRC
Postorder: 8 6 3 5 4 2 7 1

LCR
Inorder: 8 3 6 5 1 2 4 7

Question 2 (12 points)
Let T be a binary tree.

You are given the postorder and inorder sequences of T:
Postorder(T): 5, 6, 1, 4, 7, 2, 3
Inorder(T): 1, 5, 6, 3, 4, 2, 7

Draw T.



Question 3 (20 points)

Implement the following function:

```
def insert_sorted(srt_lnk_lst, elem)
```

This function is called with:

1. `srt_lnk_lst` – a `DoublyLinkedList` object containing integers, appearing in an ascending order.
2. `elem` – an integer

When called, it should add `elem` into its sorted place in `srt_lnk_lst`. That is, it mutates the list object, so that after the execution, it would also include `elem`, and remain sorted.

For example, if `srt_lnk_lst` is `[1<-->3<-->5<-->7<-->12]`,

after calling `insert_sorted(srt_lnk_lst, 6)`,

`srt_lnk_lst` should be: `[1<-->3<-->5<-->6<-->7<-->12]`

Implementation requirement: In this question, you are not allowed to use the `add_after`, `add_before`, `add_first` and the `add_last` methods of the `DoublyLinkedList` class.

Write your answer on the next page

Logic Bubble

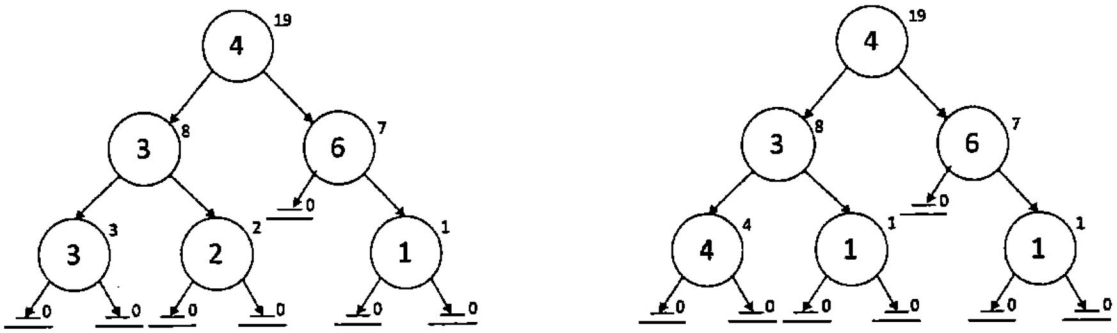
```
if ddl.isempty()
    ddl.header.next = elem
    elem.next = h
    succ = header.next
    elem = ddl.node(elem)
    header.next = ↑
    succ.prev = elem
    elem.next = succ
cursor = ddl.header
if elem < header.next:
        insert
else:
    cursor = cursor.next
```

```
def insert_sorted(srt_lnk_lst, elem):
    if srt_lnk_lst.is_empty():
        cursor = srt_lnk_lst.header  # cursor at header
    if srt_lnk_lst.is_empty():  # checks if empty
        succ = cursor.next
        elem = DoublyLinkedList.Node(elem)  # creates node
        cursor.next = elem  # Linking process
        elem.prev = cursor
        elem.next = succ
        succ.prev = elem
        srt_lnk_lst.size+=1
        # increase size
    else:
        while (cursor.next.data is not None):  # traverses
            if cursor.next.data > elem:  # checks if elem < cursor.next
                succ = cursor.next  # inserts elem below
                elem = DoublyLinkedList.Node(elem)
                cursor.next = elem
                elem.prev = cursor
                elem.next = succ
                succ.prev = elem
                return srt_lnk_lst  # if inserted return llist
            else:
                cursor = cursor.next
        prevprev = srt_lnk_lst.trailer.prev  # if the elem goes at
        elem = DoublyLinkList.Node(elem)  # the end
        prevprev.next = elem  # All this is outside while loop
        elem.prev = prevprev  # in case while loop doesn't return
        elem.next = srt_lnk_lst.trailer
        srt_lnk_lst.trailer.prev = elem
        srt_lnk_lst.size+=1
        # increase size
        return srt_lnk_lst
```

Question 4 (20 points)

Consider the following definition, of when is a binary tree considered to be sum-balanced. We say that a binary tree T satisfies the ***Sum-Balance Property*** if for every node p of T, the sum of all values in the subtrees rooted by the children of p, differ by at most 1.

For example, consider the following two trees. Note that in these figures we showed the sum of each subtree in a small font, to the right of each such root:



The tree on the left satisfies the sum-balance property, while the tree on the right does not (since the subtree rooted by the node containing 3 has one child with sum 4 and the second child with sum 1).

Notes:

1. An empty tree is sum-balanced.
2. A tree with a single node is always sum-balanced (since both its children are empty, hence their sum is 0).

In this question, we will implement the following function:

```
def is_sum_balanced(bin_tree)
```

The function is given `bin_tree`, a `LinkedBinaryTree` object, it will return `True` if the tree satisfies the sum-balance property, or `False` otherwise.

`is_sum_balanced` will call a **recursive** helper function:

```
def is_subtree_sum_balanced(subtree_root)
```

This function is given `subtree_root`, a reference to a node, that indicates the root of the subtree that this function operates on.

On the following page:

a. Complete the implementation of `is_sum_balanced`.
b. Implement the `is_subtree_sum_balanced` helper function

Implementation requirement: Your functions should run in **linear time**.

Hint: To meet the runtime requirement, you may want `is_subtree_sum_balanced` to return more than one value (multiple values could be collected as a tuple).

a.

```
def is_sum_balanced(bin_tree):
    result = is_subtree_sum_balanced(bin_tree.root)
    return result
```

b.

```
def is_subtree_sum_balanced(subtree_root):
    root = subtree_root
    depth = 0
    def helper(root, depth): # helper class define
        if root is None: # basc case
            return 0
        else:
            currData = root.data # Gets currNode data
            L = helper(root.left, depth
            depth += 1 # helps keep track of depth
            L = helper(root.left, depth)
            R = helper(root.right, depth)
        if depth > 0: # if depth not 0th level, sum it all
            return currData + L + R
        if depth == 0: # if depth is compare both sides
            return abs(L - R) L = 1
    result = helper(root, depth) # saves result
    return result # returns result
```

wrong algorithm

Check Extra Page for Rest # of Code

Question 5 (30 points)

Give a Python implementation for the *Duplicates Stack ADT*. A duplicates stack supports operations that look at consecutive elements with the same value.

The *Duplicates Stack ADT* supports the following operations:

- **`DupStack`**(): initializes an empty `DupStack` object
- `dupS.`**`is_empty`**(): returns `True` if `dupS` does not contain any elements, or `False` otherwise.
- **`len`**(`dupS`): returns the number of elements in `dupS`
- `dupS.`**`push`**(`e`): adds an integer element `e`, to the top of `dupS`.
- `dupS.`**`top`**(): returns the top most element from the top of `dupS`, without removing it; an exception is raised if `dupS` is empty.
- `dupS.`**`top_dups_count`**(): returns the number of consecutive times the top most element appears at the top of `dupS`; an exception is raised if `dupS` is empty.
- `dupS.`**`pop`**(): removes and returns the top element from the top of `dupS`; an exception is raised if `dupS` is empty.
- `dupS.`**`pop_dups`**(): removes all consecutive appearances of the top most element from the top of `dupS`. This method would return the common value, that was removed; an exception is raised if `dupS` is empty.

For example, your implementation should follow the behavior below:

```
>>> dupS = DupStack()
>>> dupS.push(4)
>>> dupS.push(5)
>>> dupS.push(5)
>>> dupS.push(5)
>>> dupS.push(4)
>>> dupS.push(4)
>>> len(dupS)
6
>>> dupS.top()
4
>>> dupS.top_dups_count()
2
>>> dupS.pop()
4
>>> dupS.pop()
4
>>> dupS.top()
5
>>> dupS.top_dups_count()
3
>>> dupS.pop_dups()
5
>>> dupS.top()
4
```

Implementation requirements:

1. Data members requirement:
   A `DupStack` object should have the following data-members:
   - A `Stack` object – You may use such object without implementing the `Stack` class. Assume that `Stack` supports the *Stack ADT* (*s=Stack()*, *len(s)*, *s.is_empty()*, *s.push(e)*, *s.pop()*, *s.top()*).
   - Constant number of additional data members, if needed
2. Runtime requirement:
   Assuming that all `Stack` operation run in $\theta(1)$ worst-case, in your implementation **ALL** `DupStack` **operations should run in $\theta(1)$ worst-case.**

Notes:

1. You should not assume anything about the inner implementation of the `Stack` objects. That is, you should use this class as a black box.
2. Make sure that your implementation of **`pop_dups` runs in constant time.**

Hint: You may want to store a tuple, as elements in the `Stack` data-member.

```
push ( self, elem)
    - check if empty:
        if its is:
          push ( elem, 1 )
    - if it is not:
        topElemC = stack.top( )[0]
            numElemC = stack.top( )[1]
            if elem == elem e:
                push [tuple](elemC, numElemC+1)
            else:
                push :(elem, 1)
```

```
class EmptyCollection(Exception):
    pass

class DupStack:
    def __init__(self):
        stack = Stack( )

    def __len__(self):
        return len(self.stack)

    def is_empty(self):
        return self.stack. is_empty( )

    def push(self, e):
        if self.stack.is_empty( ): # checks empty
            tuple =(elem,1)
            self.stack.push(tuple)
        else:
            var = self.stack.top( )[0] # checks variable
            num = self.stack.top( )[1] # num times
            if e== var: # if matches
                tuple = (var, (num +1)) # push with +1
                self.stack.push(tuple)
            else:
                tuple = (e, 1) # else new variable with 1 on top.
                self.stack.push(tuple)
```

```
def top(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
```

return self.stack.top()[0]

```
def top_dups_count(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
```

return self.stack.top()[1]

```
def pop(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
```

temp = self.stack.pop()

temp = temp[0]

return temp

```
def pop_dups(self):
    if (self.is_empty()):
        raise EmptyCollection("Duplicates Stack is empty")
    While self.stack.top()[1]>1: # pops till one dup.
        self.stack.pop()
    temp = self.stack.pop()  # pops last one
    return temp[0]  # returns common variable
```

Rest of Q.4

EXTRA PAGE IF NEEDED

Note question numbers of any questions or part of questions that you are answering here.
Also, write "ANSWER IS ON LAST PAGE" near the space provided for the answer.

```
if root is None: # if empty
    return True
elif root is not None and (root.left is None and root.right is None):
    return True # if 1 node
else: # if more than 1 node
    result = helper(root, depth)
    return result
```