Polytechnic Tutoring Center

Final Exam Review - CS 1134, Spring 2026

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

```
class ArrayStack:
    def __init__(self):
        """Initializes an empty ArrayStack object. A stack object has:
        - data: an array storing the elements currently in the stack
          in the order they entered the stack."""

    def __len__(self):
        """Returns the number of elements stored in the stack."""

    def is_empty(self):
        """Returns True if and only if the stack is empty."""

    def push(self, elem):
        """Inserts elem to the stack."""

    def pop(self):
        """Removes and returns the item that entered the stack last,
        or raises an Exception if the stack is empty."""

    def top(self):
        """Returns (without removing) the item that entered the stack last,
        or raises an Exception if the stack is empty."""
```

```
class ArrayQueue:
    def __init__(self):
        """Initializes an empty ArrayQueue object. A queue object has:
        - data: an array holding the elements currently in the queue
          (stored in a circular way).
        - front_ind: index where the sequence starts, or None if empty.
        - num_of_elems: number of elements currently stored."""

    def __len__(self):
        """Returns the number of elements stored in the queue."""

    def is_empty(self):
        """Returns True if and only if the queue is empty."""

    def enqueue(self, elem):
        """Inserts elem to the queue."""

    def dequeue(self):
        """Removes and returns the item that entered the queue first,
        or raises an Exception if the queue is empty."""

    def first(self):
        """Returns (without removing) the item that entered the queue first,
        or raises an Exception if the queue is empty."""

    def resize(self, new_cap):
        """Resizes the data array to new_cap while preserving queue contents."""
```

```
class DoublyLinkedList:
```

```
    def delete_node(self, node):
```

```
    class Node:
        def __init__(self, data=None, prev=None,
next=None):
            """Initializes a new Node object with:
            - data: the element value
            - next: reference to next node
            - prev: reference to previous node"""

        def disconnect(self):
            """Detaches the node by setting all
attributes to None."""

    def __init__(self):
        """Initializes an empty DoublyLinkedList
object with:
        - header: dummy node before first element
        - trailer: dummy node after last element
        - size: count of elements"""

    def __len__(self):
        """Returns the number of elements in the
list."""

    def is_empty(self):
        """Returns True if and only if the list is
empty."""

    def first_node(self):
        """Returns reference to the node storing the
first element."""

    def last_node(self):
        """Returns reference to the node storing the
last element."""

    def add_after(self, node, data):
        """Adds data after the given node and returns
new node."""

    def add_before(self, node, data):
        """Adds data before the given node and
returns new node."""

    def add_first(self, data):
        """Adds data as the first element of the list."""

    def add_last(self, data):
        """Adds data as the last element of the list."""
```

Continued in next column

```
        """Removes node from the list and returns its data."""

    def delete_first(self):
        """Removes the first element and returns its data."""

    def delete_last(self):
        """Removes the last element and returns its data."""

    def __iter__(self):
        """Iterator over elements from start to end."""

    def __repr__(self):
        """Returns string representation of list with <--> between
values."""
```

class LinkedBinaryTree:	class HashTableMap:

```
    class Node:
        def __init__(self, data, left=None,
right=None, parent=None):
            """Initializes a Node with:
            - data: element value
            - left: left child reference
            - right: right child reference
            - parent: parent reference"""

    def __init__(self, root=None):
        """Initializes a LinkedBinaryTree with:
        - root: root node reference
        - size: number of nodes"""

    def __len__(self):
        """Returns the number of nodes in the tree."""

    def is_empty(self):
        """Returns True if and only if the tree is
empty."""

    def subtree_count(self, curr_root):
        """Returns number of nodes in the subtree
rooted at curr_root."""

    def preorder(self):
        """Generator to iterate over all nodes in
preorder."""

    def subtree_preorder(self, curr_root):
        """Generator to iterate subtree rooted at
curr_root in preorder."""

    def postorder(self):
        """Generator to iterate over all nodes in
postorder."""

    def subtree_postorder(self, curr_root):
        """Generator to iterate subtree rooted at
curr_root in postorder."""

    def inorder(self):
        """Generator to iterate over all nodes in
inorder."""

    def subtree_inorder(self, curr_root):
        """Generator to iterate subtree rooted at
curr_root in inorder."""



    def __iter__(self):
        """Generator to iterate data level by level, left
to right."""
```

```
    class MADHashFunction:
        def __init__(self, N, p=4020683520484051307 3):
            """Initializes MAD hash function for mapping to an
array of N slots."""

        def __call__(self, key):
            """Returns index in [0, N-1] where key maps."""

    class Item:
        def __init__(self, key, value):
            """Initializes an Item with:
            - key: key value
            - value: data associated with the key"""

    def __init__(self):
        """Initializes an empty HashTableMap."""

    def __len__(self):
        """Returns number of entries in the table."""

    def is_empty(self):
        """Returns True if and only if the table is empty."""

    def __getitem__(self, key):
        """Returns value associated with key or raises
KeyError."""

    def __setitem__(self, key, value):
        """Adds or updates key with value."""

    def __delitem__(self, key):
        """Removes key from table or raises KeyError."""

    def __iter__(self):
        """Generator to iterate over keys in the table."""
```

Question 1 (10 Points)

Given the postorder and inorder traversal of a binary tree, reconstruct (draw out) this tree:

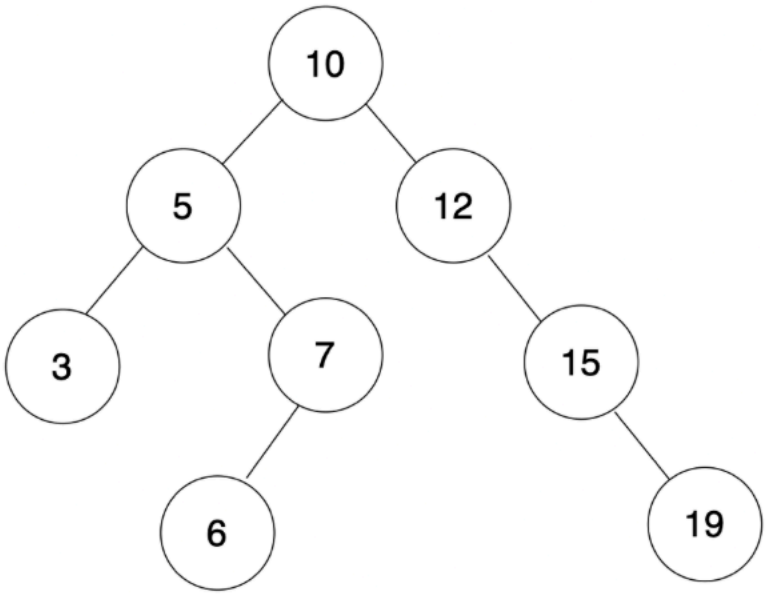
```
postorder: 0, 8, 6, 9, 7, 5, 3, 4, 2, 1

inorder: 3, 6, 8, 0, 5, 7, 9, 2, 4, 1
```

Draw the described tree.

Question 2 (5 points each, total 20 points)

From the following binary search tree, perform the following operations and show the output for the traversal operations



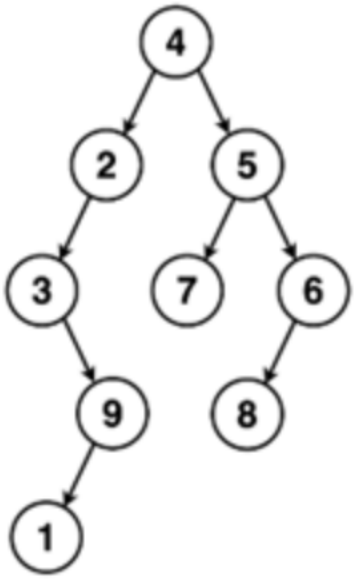

```
>> Insert 3
>> Delete 15
>> Insert 18
>> Delete 5
```

Traversal	Your Answer
Preorder	
Inorder	
Postorder	
Breadth First Search	

Question 3 (15 Points)

Write a function that, given a LinkedBinaryTree object, the function recursively finds the longest branch in that tree and returns that length. The function takes only one parameter/argument

For example, if the tree looks like this with starting root containing the value ‘4’, the function returns 5, since the longest branch (down the path of 4 - 2 - 3 - 9 - 1) is 5-node-long.



Question 4 (15 Points)

Write a method for LinkedBinaryTree. The method **recursively** checks if every single node in the tree contains the same data value. If so, return True. Otherwise, return False (If the tree is empty, return True) The method **must take no argument**, such as:

```
def mono(self):
```

Question 5 (15 Points)

We have learnt the Multiply-Add-Divide (MAD) method for hash map. If you need a little reminder, the mathematical formula looks like this:

```
Address = [(ai+b)% p]% N
```

```
where:
a is an integer within the range of [1, p-1]
b is an integer within the range of [0, p-1]
p is a prime number
N is the total capacity of the array used to store our data
```

```
In addition, let's say we handle collision with linear probing.
```

What is the **worst case runtime** to store a total number of n entries (n data points) into the hash table map with a hashing function as described above?

Suppose n=3, which means we have 3 entries. And suppose these 3 entries are the worst runtime case, can you propose three possible i values? You can use any of the letters in the hashing equation, a, b, p, and N, and any constant number. You do not need to simplify your answer.

Question 6 (25 Points)

In this question you will implement a class, BrowserHistory.

The **BrowserHistory** class has four methods we need (but that doesn't mean you can only define four methods! You can define as many as you want, as long as the four desired methods are supported!)

visit(website)

which takes the argument of a website, you can assume it is always a string ending with ".com" and add it to our browser history. This method must be **constant runtime** because you want a fast browser!

deleteHistory(website)

And, perhaps the most important method of our BrowserHistory class, the deleteHistory(website) method removes the given website from our history! This method must also be **constant runtime** because, well, sometimes you just want to get rid of a particular history as quickly as possible!

checkHistory()

which will print out all websites we have visited, and the total number of visits, from the oldest website we have visited in the past to the most recent website we have checked out. This method must be no worse than **linear runtime**.

len()

Lastly, this must return the total number of **different** websites we have visited. It must have a constant **runtime**.

To make your life as a programmer easier, our BrowserHistory is incredibly amnesiac and it **only remembers the most recent 3 websites** we have visited in the past! That is to say, if we visit more than 3 website, only the most recent 3 websites we have visited show up when we call checkHistory()

example:	You should see:
myBrowser = BrowserHistory() myBrowser.visit('gitHubbub.com') myBrowser.visit('gitHubbub.com') myBrowser.visit('lipgen.com') myBrowser.visit('catPics.com') myBrowser.visit('gitHubbub.com') myBrowser.visit('CS1134isEASY.com') print("Length:", len(myBrowser))	Length: 3 catPics.com: 1 visits. gitHubbub.com: 3 visits. CS1134isEASY.com: 1 visits. catPics.com: 1 visits. gitHubbub.com: 3 visits.

myBrowser.checkHistory() myBrowser.deleteHistory('CS1134isEASY.com') print("\n") myBrowser.checkHistory()	CS1134isEASY.com: 1 visits.

```
class BrowserHistory:
    def __init__(self):



    def __len__(self):



    def visit(self):



    def checkHistory(self, elem):



    def deleteHistory(self):
```