Name: ___________________________ Net ID: _________________

NYU, Tandon School of Engineering
CS-1134: Data Structures and Algorithms — Spring 2019

CS-1134 – Final Exam

Wednesday, May 15, 2019

- You have two hours.

- There are 5 questions all together, with 100 points total.

- The exam has **TWO Parts**:
  - The first part of the exam contains:
    - This cover page.
    - Documentation of the interface of the `ArrayStack`, `ArrayQueue`, `DoublyLinkedList`, `LinkedBinaryTree` and `HashTableMap` classes we implemented in the lectures.
      You may use these classes and methods without implementing them, unless explicitly stated otherwise**.**
    - A page for scratch work. **What you write in this page will not be graded**, but you must hand it in with your exam.
  - The second part of the exam contains the questions you need to answer, and a space for you to write your answers at. Write your answers clearly and concisely, in those spaces.

- **YOU MAY NOT USE THE BACKSIDE OF THE EXAM PAPERS**, as they will not be looked at. Also, try to avoid writing near the edge of the page.
  If you need extra space for an answer, use the **extra page at the end of the exam** and **mark it clearly**, so we can find it when we're grading.

- If you write with a pencil, press hard enough so that the writing will show up when scanned.

- Write your Name and NetID at the head of each page.

- Calculators are not allowed.

- Read every question completely before answering it.

- For any questions about runtime, give an asymptotic analysis.

- You do not have to do error checking. Assume all inputs to your functions are as described

- Cell phones, and any other electronic gadgets must be turned off.

- Do not talk to any students during the exam. If you truly do not understand what a question is asking, you may raise your hand when one of the CS1134 instructors is in the room.

Name: ___________________________ Net ID: _________________

```
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
        (out of all the items currently in the stack),
        or raises an Exception, if the stack is empty"""

    def top(self):
        """returns (without removing) the item that entered the stack
        last (out of all the items currently in the stack),
        or raises an Exception, if the stack is empty"""
```

Name: ___________________________ Net ID: _________________

```
class ArrayQueue:

    def __init__(self):
        """initializes an empty ArrayQueue object.
        A queue object has the following data members:
        1. data – an array, holding the elements currently in the
           queue in the order they entered the queue. The elements
           are stored in the array in a “circular” way (not necessarily
           starting at index 0)
        2. front_ind – holds the index, where the (cyclic) sequence
           starts, or None if the queue is empty
        3. num_of_elems – holds the number of elements that are
           currently stored in the queue"""

    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
        (out of all the items currently in the queue),
        or raises an Exception, if the queue is empty"""

    def first(self):
        """returns (without removing) the item that entered the queue
        first (out of all the items currently in the queue),
        or raises an Exception, if the queue is empty"""

    def resize(self, new_cap):
       """resizes the capacity of the self.data array to be new_cap,
       while preserving the current contents of the queue"""
```

Name: ___________________________ Net ID: _________________

```
class DoublyLinkedList:

    class Node:
        def __init__(self, data=None, prev=None, next=None):
            """initializes a new Node object containing the
            following attributes:
            1. data - to store the current element
            2. next - a reference to the next node in the list
            3. prev - a reference to the previous node in the list """

        def disconnect(self):
            """detaches the node by setting all its attributes to None"""


    def __init__(self):
        """initializes an empty DoublyLinkedList object.
        A list object holds references to two "dummy" nodes:
        1. header - a node before the primary sequence
        2. trailer - a node after the primary sequence
        also a size count attribute is maintained"""

    def __len__(self):
        """returns the number of elements stored in the list"""

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

    def first_node(self):
        """returns a reference to the node storing the
        first element in the list"""

    def last_node(self):
        """returns a reference to the node storing the
        last element in the list"""

    def add_after(self, node, data):
        """adds data to the list, after the element stored in node.
        returns a reference to the new node (containing data)"""

    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"""

    def add_before(self, node, data):
        """adds data to the list, before the element stored in node.
        returns a reference to the new node (containing data)"""
```

```
def delete_node(self, node):
    """removes node from the list, and returns the data stored in it"""

def delete_first(self):
    """removes the first element from the list, and returns its value"""

def delete_last(self):
    """removes the last element from the list, and returns its value"""

def __iter__(self):
    """an iterator that allows iteration over the
    elements of the list from start to end"""

def __repr__(self):
    """returns a string representation of the list, showing
    data values separated by <--> """
```

```
class LinkedBinaryTree:
    class Node:
        def __init__(self, data, left=None, right=None, parent=None):
            """initializes a new Node object with the following attributes:
            1. data - to store the current element
            2. left - a reference to the left child of the node
            3. right - a reference to the right child of the node
            4. parent - a reference to the parent of the node"""

    def __init__(self, root=None):
        """initializes a LinkedBinaryTree object with the structure
        given in root (or empty if root is None). A tree object holds:
        1. root - a reference to the root node or None if tree is empty
        2. size - a node count"""

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

    def is_empty(self):
        """returns True if"f the tree is empty"""

    def subtree_count(self, curr_root):
        """returns the number of nodes in the subtree rooted by curr_root"""

    def preorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in a preorder order"""
    def subtree_preorder(self, curr_root):
        """generator allowing to iterate  in a preorder order
        over the nodes of the subtree rooted with curr_root"""

    def postorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in a postorder order"""
    def subtree_postorder(self, curr_root):
        """generator allowing to iterate  in a postorder order
        over the nodes of the subtree rooted with curr_root"""

    def inorder(self):
        """generator allowing to iterate over the nodes of
        the (entire) tree in an inorder order"""
    def subtree_inorder(self, curr_root):
        """generator allowing to iterate  in an inorder order
        over the nodes of the subtree rooted with curr_root"""

    def breadth_first(self):
        """generator allowing to iterate over the nodes of the
        (entire) tree level by level, each level from left to right"""

    def __iter__(self):
        """generator allowing to iterate over the data stored in the
        tree level by level, each level from left to right"""
```

```
class HashTableMap:
    class MADHashFunction:
        def __init__(self, N, p=40206835204840513073):
            """initializes a new hash function object. This function uses
            the build in hash function for coding, and the MAD method for
            compression.
            The function is for mapping to an array with N
            slots. That is the function’s range is {0, 1, …, N-1} """

        def __call__(self, key):
            """returns the index (in the range {0, 1, …, N-1}) to where
            key is mapped to """

   class Item:
       def __init__(self, key, value):
           """initializes a new Item object with the following attributes:
           1. key - to store a key
           2. value - to store the value (associated to the key) """


   def __init__(self):
       """initializes an empty HashTableMap (hash-table) object"""

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

   def is_empty(self):
       """returns True if"f the table is empty"""

   def __getitem__(self, key):
       """returns the value associated to key, or raises a KeyError
       exception if key is not in the table.
       runs in O(1) average time"""

   def __setitem__(self, key, value):
       """adds the value associated by key, to the table. If key is
       already associated to an old value, it replaces it with value.
       runs in O(1) average time"""

   def __delitem__(self, key):
       """removes the value associated to key from the table, or raises
        a KeyError exception if key is not in the table.
        runs in O(1) average time"""

   def __iter__(self):
       """generator allowing to iterate over the keys in the table"""
```

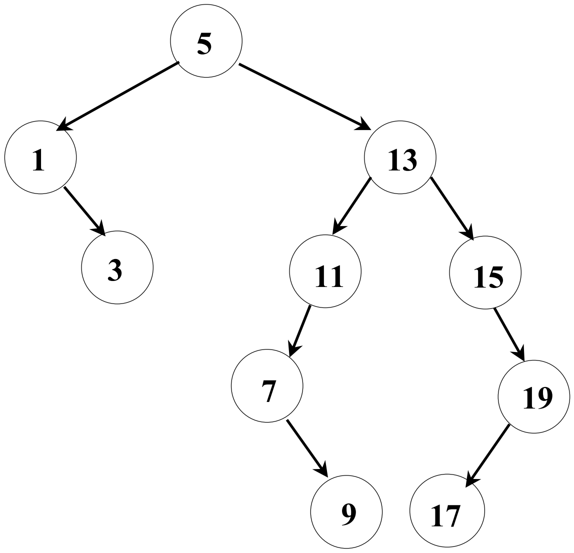
Name: ____________________________ Net ID: __________________

Scratch
(This paper will not be graded)

Name: ____________________________ Net ID: __________________

Question 1 (15 points)

a. Given the following (unbalanced) binary search tree:



We are executing the following two operations **on the tree above (not one after the other)**:

- Inserting 6
- Deleting 13

For each one of these operations, **apply the algorithm described in class** for these operations, and draw the resulting tree.

1. After inserting 6 to the tree above:	2. After deleting 13 from the tree:

b. Let *T* be a **binary search tree**. If we traverse *T* in postorder, we get the following sequence:

Postorder(T): 1, 3, 4, 2, 5, 8, 7, 9, 11, 10, 6

Draw *T*:

Name: ____________________________ Net ID: __________________

Question 2 (15 points)

For each of the following expressions (given in infix, postfix and prefix notations), Draw the corresponding expression tree.

Note: For Infix expressions, remember order of operations.

a. Infix: `2 + 3 * (4 + 5)`

b. Postfix: `4 3 2 ** +`

c. Prefix: `* + 2 3 - 4 5`

Name: ____________________________ Net ID: __________________

Question 3 (25 points)

In this question, you should implement the following function:

```
def remove_duplicates(lnk_lst)
```

The function is given `lnk_lst`, a `DoublyLinkedList` object. When called, it will mutate the object, and remove all the duplicate values, keeping only the first occurrence of each unique value.

For example:
if `lnk_lst=[1 <--> 7 <--> 3 <--> 3 <--> 1 <--> 5 <--> 7]`,
after calling: `remove_duplicates(lnk_lst)`, `lnk_lst` should be:
`[1 <--> 7 <--> 3 <--> 5]`.

Notes:

1. Your function must run in **average linear time**. That is, if `lnk_lst` has n items, the **average** run time for the call `remove_duplicates(lnk_lst)` should be $\theta(n)$.
2. You may use objects of any type learned in class (`ArrayStack`, `ArrayQueue`, `DoublyLinkedList`, `HashTableMap`, etc.). You should **use the interface of these types as black boxes**. That is, you may not assume anything about their inner implementation.

Write your implementation on the next page.

Name: ______________________________ Net ID: ___________________

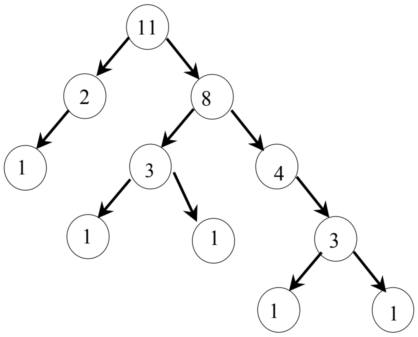
```python
def remove_duplicates(lnk_lst):
```

Name: ____________________________ Net ID: _________________

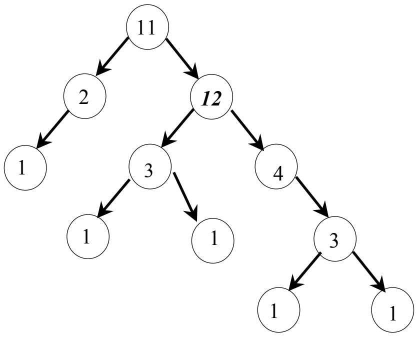
Question 4 (20 points)

Consider the following definition of a *size-tree*:
Let T be a binary tree. We say that T is a size-tree if the data of **each** node *n* in T is equal to the size (number of nodes) of the subtree rooted by *n*.

For example, the tree on the left is a valid size-tree, however the tree on the right is not (the node with the data 12 is not the root of the subtree of size 12).



A valid size-tree



Not a size-tree

In this question, we will implement the following function:

```
        def is_size_tree(bin_tree)
```

The function is given `bin_tree`, a non-empty `LinkedBinaryTree` object, it will return `True` if `bin_tree` is a valid size-tree, or `False` otherwise.

The implementation of `is_size_tree` uses a **recursive** helper function:

```
        def is_size_tree_helper(root)
```

This function is given `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_size_tree`.
b. Implement the recursive `is_size_tree_helper` helper function.

Implementation requirements:

1. Your implementation should run in **linear time**.
2. You should give a **recursive** implementation for the helper function.
3. You are **not allowed** to add parameters to the functions' header lines, set default values to any parameter, nor use global variables.

Note:
You may (though it's not necessary) have `is_size_tree_helper` return more than one value (multiple values could be collected as a tuple).

a.

```
def is_size_tree(bin_tree):

    _____________ = is_size_tree_helper(bin_tree.root)

    return __________
```

b.

```
def is_size_tree_helper(root):

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________

    ________________________________________________________
```

Name: ___________________________ Net ID: _________________

Question 5 (25 points)

An ***Extended Parties-Queue*** is a variation of a *Queue*. It is used to apply a first-in-first-out order, but instead of storing individual items as elements, it stores parties (collection of items) as elements. Each party is identified by a name. An Extended Parties-Queue also supports an additional operation, that allows to add to an existing party.

Note: In this question, for simplicity, **instead of storing each party as a collection, we will only store its size**.

An *Extended Parties-Queue* has the following interface:

- ***pq = ExtendedPartiesQueue()*****:** creates a new *ExtendedPartiesQueue* object, with no parties in it

- ***len(pq)*****:** returns the number of parties in *pq*.

- ***pq.enq_party(party_name, party_size)*****:** inserts a new party, by the name *party_name* and of size *party_size*, to the end of the line.

- ***pq.first_party()*****:** returns the **size** of the party that is first in line.

- ***pq.deq_first_party()*****:** removes the party that is first in line, and returns its **size**.

- ***pq.add_to_party(party_name, size_to_add)*****:** mutates the object to reflect an addition of *size_to_add* guests to the party by the name *party_name*, or raises an *Exception* if there is no party by the name *party_name* currently in *pq*.

For example, you should expect the following interaction.
For clarity, we commented to the side of each instruction, the parties (name and size) in the order they are currently in the queue:

```
>>> pq = ExtendedPartiesQueue()
>>> pq.enq_party(“Jeff”, 3) # <Jeff, 3>
>>> pq.enq_party(“Mike”, 5) # <Jeff, 3>, <Mike, 5>
>>> pq.enq_party(“Nick”, 2) # <Jeff, 3>, <Mike, 5>, <Nick, 2>
>>> pq.deq_first_party()    # <Mike, 5>, <Nick, 2>
3
>>> pq.enq_party(“Jessica”, 4) # <Mike, 5>, <Nick, 2>, <Jessica, 4>
>>> pq.add_to_party(“Nick”, 2) # <Mike, 5>, <Nick, 4>, <Jessica, 4>
>>> pq.deq_first_party()       # <Nick, 4>, <Jessica, 4>
5
```

Complete the implementation of the `ExtendedPartiesQueue` class.

Runtime requirement:

EACH `ExtendedPartiesQueue` operation should run in $\theta(1)$ **average**-amortized time. That is, any sequence of n `ExtendedPartiesQueue` operations should run in $\theta(n)$ average time

Notes:

1. For simplicity, assume that the parties' names, that are in the queue at any given time, are unique. That is when a new party is added, there is no party by that same name already in line.
2. You may use any combination of objects of the types learned in class (`ArrayStack, ArrayQueue, DoublyLinkedList, HashTableMap`, etc.). You should **use the interface of these types as black boxes**. That is, you may not assume anything about their inner implementation.

```
class ExtendedPartiesQueue:

    def __init__(self):

        __________________________________________________

        __________________________________________________

        __________________________________________________


     def __len__(self):

        __________________________________________________

        __________________________________________________


     def enq_party(self, party_name, party_size):

        __________________________________________________

        __________________________________________________

        __________________________________________________

        __________________________________________________

        __________________________________________________

        __________________________________________________

        __________________________________________________
```

```
def add_to_party(self, party_name, size_to_add):

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________


def first_party(self):
    if(len(self) == 0):
        raise Exception("ExtendedPartiesQueue is empty")

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________


def deq_first_party(self):
    if(len(self) == 0):
        raise Exception("ExtendedPartiesQueue is empty")

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________

    _____________________________________________________
```

Name: ____________________________ Net ID: __________________

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.