PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

Term test #2

CSC 148H1, Section L0201
Duration — 50 minutes

aids allowed: 8.5" x 11" aid sheet, both sides

Student Number: |__|__|__|__|__|__|__|__|__|__|

Last Name: ______________________________

First Name: ______________________________

Do **not** *turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below.)

This test consists of 3 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy of the test is complete.*

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write "I cannot answer this question," on. You may earn substantial part marks for writing down the outline of a solution and indicating which steps are missing.

Good Luck!

Question 1. [8 marks]

Part (a) [4 marks]

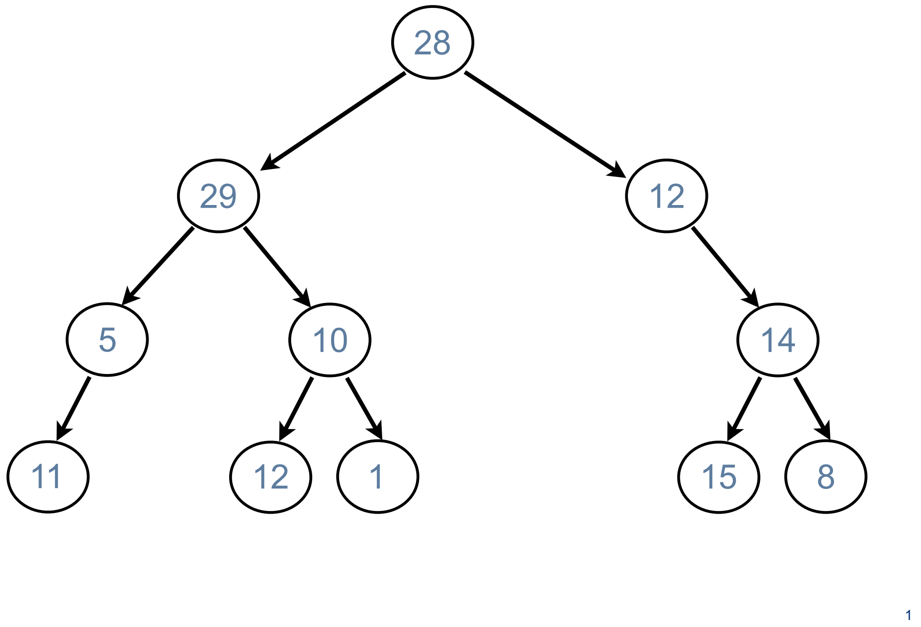
Recall that we defined the height of a tree in such a way that a tree consisting of just the root has a height of 1. Suppose we have a binary tree with 20 nodes.

(i) The greatest height this tree could have is: ___________. Use a diagram to justify your answer:

(ii) The least height this tree could have is: ___________. Use a diagram to justify your answer:

Part (b) [4 marks]

Here is a tree. Notice that it is *not* a binary search tree.



If we traverse the tree using pre-order traversal, in what order would the nodes be visited? Write the values below in the correct order.

Question 2. [10 MARKS]

Read over the declaration of class **BTNode** and the docstring for function **list_even_between**:

```
class BTNode:
    '''Binary Tree node.'''

    def __init__(self, data, left=None, right=None):
        ''' (BTNode, object, BTNode, BTNode) -> NoneType

        Create BTNode (self) with data and children left and right.
        '''
        self.data, self.left, self.right = data, left, right


def list_even_between(t, start, stop):
    ''' (BTNode, int, int) -> list

    Return a list of data in the tree rooted at t that
    (a) are even and (b) are between start and stop, inclusive.

    Assume that t is the root of a (possibly empty) binary
    search tree with integer elements. That is, assume:
        -- every non-empty node has integer data
        -- all data in any left sub-tree is less than the
           data in the root node
        -- all data in any right sub-tree is more than the
           data in the root node.

    >>> t = None
    >>> list_even_between(t, 5, 9)
    []
    >>> t1 = BTNode(4, BTNode(2), BTNode(5))
    >>> t2 = BTNode(9, BTNode(8), BTNode(10))
    >>> t3 = BTNode(7, t1, t2)
    >>> L = list_even_between(t3, 3, 8)
    >>> L.sort()
    >>> L
    [4, 8]
    '''
```

On the next page, implement (write the body for) **list_even_between**. For maximum credit, your implementation should use the binary search tree property to avoid visiting unnecessary nodes.

cont'd...

Question 3. [9 marks]

Read over the initializers below for classes **LLNode** and **LinkedList**, as well as the docstring for function **absorb_value**. You may assume that appropriate **LLNode._str_** and **LinkedList.append** methods have been defined.

```
class LLNode:
    '''Node to be used in linked list

    nxt: LLNode -- next node
                   None iff we're at end of list
    value: object --- data for current node
    '''

    def __init__(self, value, nxt=None):
        ''' (LLNode, object, LLNode) -> NoneType

        Create LLNode (self) with data value and
        successor nxt.
        '''
        self.value, self.nxt = value, nxt
```

```
class LinkedList:
    '''Collection of LLNodes organized into a
    linked list.

    front: LLNode -- front of list
    back:  LLNode -- back of list
    size:  int    -- size of list
    '''

    def __init__(self):
        ''' (LinkedList) -> NoneType

        Create an empty linked list.
        '''
        self.front, self.back = None, None
        self.size = 0
```

```
def absorb_value(lnk, value):
    ''' (LinkedList, int) -> NoneType

    Remove from lnk the node with the first occurrence of value, and add that
    value to the node that follows. If there is no node in lnk containing
    value, or if it occurs only in the back node, leave lnk unchanged.

    Assume that lnk contains only integer values.

    >>> lnk = LinkedList()
    >>> lnk.append(6)
    >>> lnk.append(7)
    >>> lnk.append(8)
    >>> lnk.append(7)
    >>> print(lnk.front)
    6 -> 7 -> 8 -> 7 ->|
    >>> absorb_value(lnk, 7)
    >>> print(lnk.front)
    6 -> 15 -> 7 ->|
    >>> absorb_value(lnk, 6)
    >>> print(lnk.front)
    21 -> 7 ->|
    '''
```

Now implement (write the body) of **absorb_value** You should probably **draw pictures!**

This page is left (mainly) blank for things that don't fit elsewhere.

1: ______/ 8

2: ______/10

3: ______/ 9

TOTAL: ______/27