CISC-235*
Test #2
February 14, 2020

Student Number (Required) _____

Name (Optional) _____

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but will not be re-marked under any circumstances.

The test will be marked out of 50.

Question 1	/16
Question 2	/16
Question 3	/15
Question 4	/3
TOTAL	/50

☐ By writing my initials in this box, I authorize the disposal of this test paper if I have not picked it up by April 15, 2020.

"Love is like a tree, it grows of its own accord"
~ *Victor Hugo*

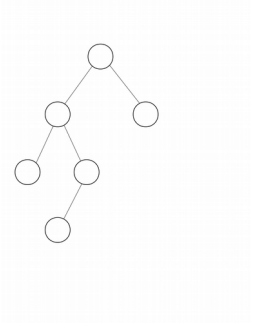
Here are three class definitions:

```
class BT_Vertex:

        # instance variables
        value : integer
        left_child  : BT_Vertex
        right_child : BT_Vertex

        # constructor
        def BT_Vertex(x: integer):
                this.value = x
                this.left_child = nil
                this.right_child = nil

class Binary_Tree:

        # instance variables
        root   : BT_Vertex
        count  : integer

        # constructor
        def Binary_Tree():
                this.root = nil
                this.count = 0

class Binary_Search_Tree:

        # instance variables
        root   : BT_Vertex
        count  : integer

        # constructor
        def Binary_Search_Tree():
                this.root = nil
                this.count = 0

        # instance methods
        def Insert(x: integer):
                this.count++
                this.root = rec_Insert(this.root,x)

        def rec_Insert(v : BT_Vertex, x : integer):
                if (v == nil):
                    return new BT_Vertex(x)
                else if (v.value >= x):
                    v.left_child = rec_Insert(v.left_child,x)
                else:
                    v.right_child = rec_Insert(v.right_child,x)
                return v
```

Question 1 (16 marks):

Write an iterative algorithm that uses a Stack to compute the number of levels in a Binary Tree. You do not need to write the Stack class, but you do need to specify the Type of data the Stack will hold (integer, Vertex, combination of objects, etc.)

Hint: the root is at level 1. A non-root vertex has level 1 higher than its parent's level. Your algorithm can proceed by computing the level of each vertex and keeping track of the highest level seen.



This tree has 4 levels

Solution:

I will use a Stack that holds objects in pairs, where the first element in each pair is BT_Vertex and the second element in each pair is an integer.

```
def count_levels():
        if this.root == nil:
                return 0
        else:
                S = new Stack()
                S.push (this.root, 1)
                int max_level = 1
                while not S.isEmpty():
                        pair = S.pop()
                        vertex = pair[0]
                        level = pair[1]
                        if level > max_level:
                                max_level = level
                        if vertex.left_child != nil:
                                S.push(vertex.left_child, level+1)
                        if vertex.right_child != nil:
                                S.push(vertex.right_child, level+1)
                return max_level
```

OR

```
def levels():
        if this.root == nil:
                return 0
        else:
                S1 = new Stack()          # stack of BT_Vertex objects
                S2 = new Stack()          # ditto
                S1.push(this.root)
                int max_level = 0
                while not S1.isEmpty():
                        max_level ++
                        while not S1.isEmpty():
                                v = S1.pop()
                                if v.left_child != nil:
                                        S2.push(v.left_child)
                                if v.right_child != nil:
                                        S2.push(v.right_child)
                        while not S2.isEmpty():
                                S1.push(S2.pop())
                return max_level
```

Marking:

Since pseudo-code solutions are accepted, syntax is not important. In particular the students are not required to use "def" etc which is Python-esque, nor "this" which is Java-esque. They are not required to specify the Type of their variables. The important issue is whether their solution is clearly expressed.

Students are not required to solve the problem in exactly the way(s) I did. They may decide to use two stacks (one for vertices, one for integers) or take another completely different (stack-based) approach.

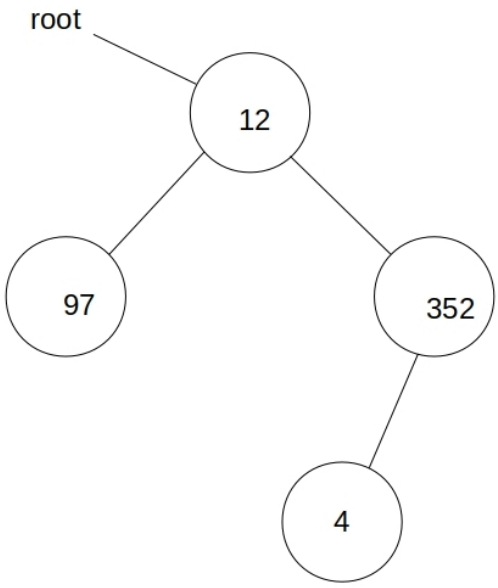
For a solution that correctly solves the problem	16
For a solution that incorrectly computes the number of levels	subtract 2
For a solution that neglects to check for the tree being empty	subtract 2
For a solution that only works on one side of the tree, or tries to access the children of a non-existent vertex	subtract 2
For other significant errors of logic	subtract 2
For a solution that shows poor understanding of how to manipulate binary trees	subtract 6
For a solution that shows poor understanding of how to manipulate stacks	subtract 6

Notwithstanding the previously listed deductions, a solution that shows a good understanding of the problem to be solved should not score less than 7

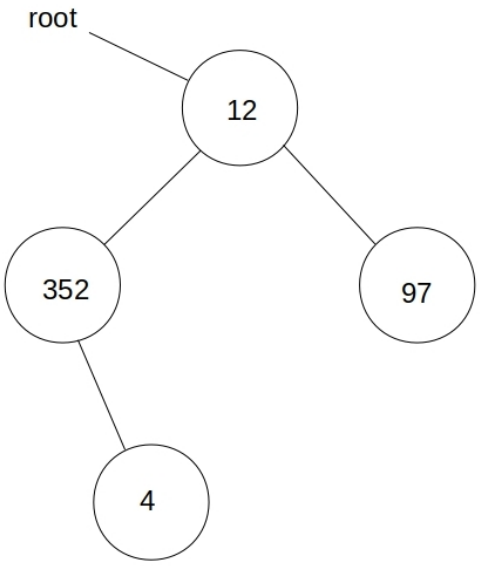
Question 2 (16 marks):

(a) [10 marks] Write a method (or combination of methods) in pseudo-code or in Java, Python, C or C++ that takes a Binary_Tree object as its parameter, and returns a new Binary_Tree containing the same values as the original tree, but as a mirror image. You are not required to use a Stack but you can if you want to.

For example if the original tree is



the new tree would be



Solution:

```
def mirror():
        this.root = rec_mirror(this.root)

def rec_mirror(current):
        if current != nil:
                temp = rec_mirror(current.left_child)
                current.left_child = rec_mirror(current.right_child)
                current.right_child = temp
        return current
```

OR

```
def mirror():
        if this.root != nil :
                S = new Stack()   # this stack holds BT_Vertex objects
                S.push(this.root)
                while not S.isEmpty():
                        v = S.pop()
                        if v != nil:
                                S.push(v.left_child)
                                S.push(v.right_child)
                                temp = v.left_child
                                v.left_child = v.right_child
                                v.right_child = temp
```

Marking:

Students do not have to structure their solution in the same way that I do, but the basic idea is that we just need to switch the left and right children of each vertex. Any solution that does this will give the correct result.

For a solution that correctly switches all the children	10
For a solution that neglects to see if the tree is empty	subtract 3
For a solution that only works on one side of the tree, or tries to reverse the children of a non-existent vertex	subtract 3
For other significant errors of logic	subtract 3
For a solution that shows poor understanding of how to manipulate binary trees	subtract 4

Notwithstanding the previously listed deductions, a solution that shows a good understanding of the problem to be solved should not score less than 4

(b) [6 marks] What can you say about the complexity of your algorithm? If possible, determine its Θ classification.

Solution:

This algorithm visits every vertex in the tree exactly once, and the amount of work that is done at each vertex is in O(1) (ie it is bounded above by a constant) so the algorithm is in O(n).

It is also in $\Omega(n)$ by the same observation – the work done is always $\geq$ c*n for some positive c.

Thus the algorithm is in $\Theta(n)$

Marking:

Their answer should apply to their algorithm.

Correct O() classification of their algorithm: 2

Correct $\Omega()$ classification of their algorithm 2

Correct $\Theta()$ classification (or correct statement that there is none) 2

Question 3 (15 marks) :

(a) [10 marks] Write a new method (or combination of methods) for the Binary_Search_Tree class that takes an integer parameter x and prints all values in the tree that are $\leq x$. The printed values do not have to be in any particular order. Make your method as efficient as possible in terms of its big-O complexity. You are not required to use a Stack but you can if you want to.

(b) [5 marks] What can you say about the complexity of your algorithm?

Solution to (a):

```
def all_le(x):
        rec_all_le(this.root, x)

def rec_all_le(current,x):
        if current != nil:
                if current.value > x:
                        rec_all_le(current.left_child, x)
                else:
                        print current.value
                        rec_all_le(current.left_child, x)
                        rec_all_le(current.right_child, x)
```

OR

```
def all_le(x):
        if this.root != nil:
                S = new Stack()          # stack of BT_Vertex objects
                S.push(this.root)
                while not S.isEmpty():
                        v = S.pop()
                        if v.value <= x:
                                print v.value
                                if v.left_child != nil:
                                        S.push(v.left_child)
                                if v.right_child != nil:
                                        S.push(v.right_child)
                        else:
                                if v.left_child != nil:
                                        S.push(v.left_child)
```

Marking:

For an algorithm that gives the correct result and examines just the vertices it needs to	10
For an algorithm that gives the correct result but doesn't take advantage of the lexical ordering in the tree to limit its work	8
For an algorithm that has the right idea but doesn't give the correct output	5
For an algorithm that shows weak understanding of the problem or poor understanding of binary trees	3
For an algorithm that shows very little understanding of the problem or binary trees	1

Solution to (b):

In the worst case, this algorithm looks at and prints the value from each vertex in the tree, and does no more than a constant amount of work at each vertex – so the algorithm is in O(n).

There are some trees and values of x for which the algorithm terminates after examining just one vertex (specifically, when the smallest value in the tree is > x and happens to occupy the root vertex). For these cases the algorithm terminates in constant time, so we can say the algorithm is in $\Omega(1)$ - this means the algorithm has no Θ classification.

This is a point of confusion because we can also say that the algorithm is *prepared* to examine all vertices, so it is in $\Omega(n)$ - this means the algorithm is in $\Theta(n)$

Either answer is acceptable.

Marking:

For a valid analysis of their algorithm	5
For an incorrect analysis of their algorithm, but a good understanding of what they were supposed to to	3
For an answer that shows very little understanding of how to address this question	1

Question 4: (3 marks)

(a) A binary tree with 15 vertices must have at least 6 leaves

True **False**

(b) A binary tree with 15 vertices cannot have more than 8 leaves

True **False**

(c) For a set of 15 distinct values, there are infinitely many different Binary Search Trees

True **False**

Solution:

False, True, False

Marking: 1 point for each right answer