CSE2050 Sample Test

Notes:

- The midterm may include more questions with different levels of difficulty.
- The midterm is closed book and closed notes.

Section 1:

- True or False:
  In terms of computational complexity, an algorithm that is O(n^2)
  is better than an algorithm that is O(2^n)?

- Fill in the blank: What is the worst case complexity of the quicksort algorithm?___________,

Section 2:

2.1

Read the following program and answer the questions that follow:

```
class ListNode:
    def __init__(self,  data, link=None):
        self._data = data
        self._link = link

class LinkedList :
    def __init__(self):
        self._head = None
    def addFirst(self, data):
        self._head = ListNode(data, self._head)
    def unknown(self):
        d = []
        current = self._head
        while current != None:
            if current in d:
                return True
            else:
                d.append(current)
            current = current._link
        return False

l = LinkedList()
l.addFirst(5)
l.addFirst(8)
l.addFirst(2)
print(l.unknown())
l._head._link._link._link = l._head
print(l.unknown())
```

2.1.1

What will be the output after the execution of the above program?

True

2.1.2

What does the **unknown** function do?

Returns True if the linkedlist contains a cycle, False otherwise.

2.1.3

What is the worst-case time complexity of the unknown function?

Leave it for you!

2.1.4

Write the unknown method without using any helper collection (without list, tuple , dictionary, or any other collection).

Idea: initialize two pointers to the first node in the list, let's call them slow and fast. Within a loop the slow one makes one step and the fast makes two. At some point they will point to the same node which means we have a cycle. Try it!

2.2 O(n)

Give the Big-O performance of the following code:

```
i = n
while i > 0:
   x = i + 4
   i = i - 1
```

2.3 O(log(n))

Give the Big-O performance of the following code:

```
i = n
while i > 0:
   k = 2
   i = i // 2
```

Section 3:

3.1

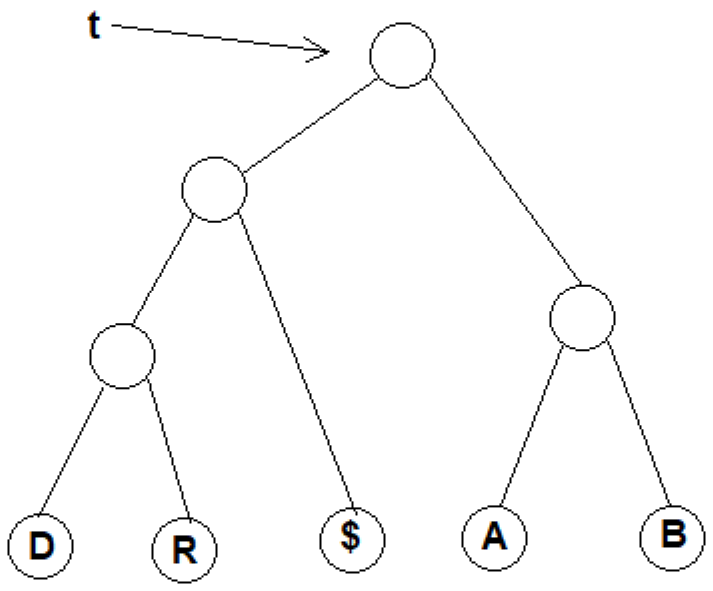
Given the following program:

```
class BinaryTree:
    def __init__(self,data, left = None, right = None):
        self.data = data
        self.left = left
        self.right = right

t1 = BinaryTree('', BinaryTree('D'), BinaryTree('R'))
t1 = BinaryTree('', t1, BinaryTree('$'))
t2 = BinaryTree('', BinaryTree('A'), BinaryTree('B'))
t = BinaryTree('', t1, t2)
print(t.getCode('R'))
```

3.1.1

Draw the binary tree that this program creates



3.1.2

Notice that the tree in 3.1.1 contains empty strings in the internal nodes and one-character string in the leaves. A path from the root node to one of the leaves creates a binary code for the character in the corresponding leaf. The code is a string of zeros and ones that is built by traversing the path from the root to the leaf. Every time we go to the left in the tree we append 0 to the string and every time we go to the right in the tree we append one. For example, the code for the character 'R' should be '001' and for the character 'A' is '10'.

Write a recursive method for the class BinaryTree called getCode. This method receives a character as a parameter and returns the code of that character. After adding your method to the code above and executing that code, the output should be '001'.

Note: This kind of tree is called Huffman tree used in compression algorithms.

```python
class BinaryTree:
    def __init__(self,data, left = None, right = None):
        self.data = data
        self.left = left
        self.right = right
    def getCode(self, char):
        if self.left == None and self.right == None:
            if self.data == char:
                return ''
            else:
                return None
        if self.left != None:
            LT = self.left.getCode(char)
            if LT != None:
                return '0' + LT
        if self.right != None:
            RT = self.right.getCode(char)
            if RT != None:
                return '1' + RT
        return None

t1 = BinaryTree('', BinaryTree('D'), BinaryTree('R'))
t1 = BinaryTree('', t1, BinaryTree('$'))
t2 = BinaryTree('', BinaryTree('A'), BinaryTree('B'))
```

```python
t = BinaryTree('', t1, t2)
print(t.getCode('R'))
```