First Name:
Last Name:
Student #:

NAME (PRINT): **Last/Surname** **First /Given Name**

STUDENT #: **SIGNATURE:**

UNIVERSITY OF TORONTO MISSISSAUGA
June 2015 FINAL EXAMINATION
CSC148H5F
Introduction to Computer Science
A.Attarwala
Duration - 3 hours
Aids: None

The University of Toronto Mississauga and you, as a student, share a commitment to academic integrity. You are reminded that you may be charged with an academic offence for possessing any unauthorized aids during the writing of an exam. Clear, sealable, plastic bags have been provided for all electronic devices with storage, including but not limited to: cell phones, tablets, laptops, calculators, and MP3 players. Please turn off all devices, seal them in the bag provided, and place the bag under your desk for the duration of the examination. You will not be able to touch the bag or its contents until the exam is over.

If, during an exam, any of these items are found on your person or in the area of your desk other than in the clear, sealable, plastic bag; you may be charged with an academic offence. A typical penalty for an academic offence may cause you to fail the course.

*Please note, you **CANNOT** petition to **re-write** an examination once the exam has begun.*

First Name:
Last Name:
Student #:

You must obtain a mark of at least 40% on this exam, otherwise, a course grade of no higher than 47% will be assigned.

'There will be a 40% reduction in marks for a non-recursive solution to any coding question requiring a recursive solution. So if you can't solve a problem recursively, submit a non-recursive solution for reduced marks.'

Questions	Mark:
Question1)	/8
Question 2)	/24
Question 3)	/25
Question 4)	/19
Total:	/76

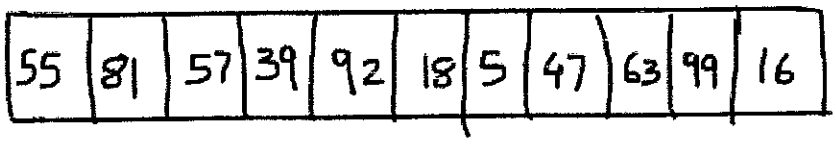
THIS EXAM CONTAINS A TOTAL OF 16 PAGES. PAGE 16/16 IS THE APPENDIX OF YOUR EXAM

First Name:
Last Name:
Student #:

Question 1) **[8]**

a) Describe the divide and conquer steps in the `quicksort`. i.e. describe in plain english how the *partition* function works AND how the *recursion* works in `quicksort`? [2]

b) *Partition* the following array using the first array element as the partition value in `quicksort`:



What is the index of value 55 after the first *partition*? [2]

First Name:
Last Name:
Student #:

c) **Multiple Choice Question:**

You are asked to sort a large list in ascending order:

c.1) Which of the basic sort algorithm is most efficient (in terms of time) when the list is already sorted in ascending order? [2]

- Selection Sort
- Insertion Sort
- Bubble Sort
- All of the above

Explain your answer:

c.2) Which of the basic sort algorithm is most efficient (in terms of time) when the list is initially sorted in descending order? [2]

- Selection Sort
- Insertion Sort
- Bubble Sort
- All of the above

Explain your answer:

First Name:
Last Name:
Student #:

Question 2) **[24]**

a) A binary string is a string that is just made of 1's and 0's. Given a binary string, write a *recursive* algorithm that returns back a `True` if the number of 1's in the binary string is greater or equal to the number of 0's. For example: [10]

- The binary string "000001" will return back `False`
- The binary string "1111100" will return back `True`
- The empty string will return back `True`

You are asked to write this *recursive* algorithm using two functions. The function `__countBinaryString(…)` is a helper function that will be called from the function `countBinaryString(…)`.

```
def countBinaryString(str):
'''str is a binary string. This non-recursive, function  returns back True if
the number of 1's in the binary string is greater or equal to the number of
0's'''
```

```
def _countBinaryString(str):
'''This is a helper function and is recursive. This function returns back a
tuple (x,y) where x is the number of 0's in the binary string and y is the
number of 1's in the binary string'''
```

First Name:
Last Name:
Student #:

b) A string is a palindrome if the string is equal to its reverse. Write a *recursive* function `palindromeStr(...)` that takes in as input a string and returns `True` if the string is a palindrome and returns `False` otherwise. A string of length 1 or length of 0 is considered as a palindrome. [5]

For example, the following two strings are palindrome:

```
'racecar'  i.e. 'racecar'==reverse('racecar')
'abba'     i.e. 'abba'==reverse('abba')
```

For example, the following string is not a palindrome:

```
'abbas'    i.e. 'abbas'!=reverse('abbas')
```

- Your function MUST BE *RECURSIVE.*
- Clearly mark your base case(s) and recursive case(s).
- No helper functions are allowed.

```
def palindromeStr(s):
'''s is a String. This recursive function returns True if s is a
palindrome      otherwise returns back a False'''
```

First Name:
Last Name:
Student #:

c) What is the *runtime* recurrence relation for your solution in part b) above from the following choices? Let `T(n)` be the number of steps taken by `palindromeStr` on inputs of size `n`. [2]

a) T(n) = T(n-1) + C (i.e. C is some constant)
b) T(n) = 2*T(n/2) + C (i.e. C is some constant)
c) T(n) = 2*T(n-2) + C (i.e. C is some constant)
d) T(n) = T(n-2) + nC (i.e. C is some constant)
e) T(n) = T(n-2) + C (i.e. C is some constant)

Explain your answer:

d) Following is the code that finds the *maximum* integer in a *binary search tree.*

```
def getMax(t):
    '''returns the max integer in the binary search tree represented
      by t. You can assume that t is in nodes and references form'''
    if t is None:
        raise EmptyTreeException('t is empty')
    elif t.right is None:
        return t.root
    else:
        return getMax(t.right)
```

d.1) Write the *runtime* recurrence relation (do not solve it) for the above code. [2]

First Name:
Last Name:
Student #:

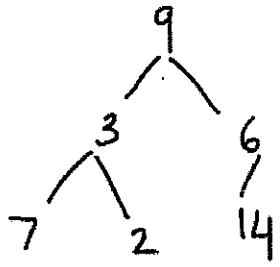
d.2) Solve the above recurrence relation assuming the binary search tree is complete and balanced, and clearly draw a box around the final *runtime* big O complexity. [3]

d.3) What is the *runtime* recurrence relation (just write it down, do not solve), for the recursive algorithm that finds the maximum integer in a complete and balanced *binary tree*? [2]

First Name:
Last Name:
Student #:

Question 3) **[25]**

a) Recall the list/array representation of a complete binary tree, where the following tree would be represented by the list `[9,3,6,7,2,14]`. **Note:** Do not assume the tree is a binary search tree. And do not confuse the *list of list* representation of a binary tree with the list/array representation of a binary tree. [10]



Recall that for a node 'n', at index i in the list:
- the left child of n is at index 2i+1
- the right child of n is at index 2i+2
- the parent of n is at index (i-1)/2

Write the recursive function `_to_node_repr(…)`, which takes a *list/array* representation of a complete binary tree and creates the *node and references* representation of that tree. You can assume that you have the following two already implemented helper functions at your disposal:

`left`:: The function left takes in an integer 'i' index position of node 'n' in list and returns back an integer that represents the index position of the node's left child in the list i.e. 2*i +1

`right`:: The function right takes in an integer 'i' index position of node 'n' in list and returns back an integer that represents the index position of the node's right child in the array i.e. 2*i +2

```
def to_node_repr(L):
'''L is a list representation of complete binary tree. This function returns
back a node/references version of the binary tree. Return the root of the
nodes/references version of the binary tree. This function calls the helper
function _to_node_repr'''

      return _to_node_repr(L)


def _to_node_repr(L,i=0):
'''complete this recursive function as per the specs outline above'''
```

First Name:
Last Name:
Student #:

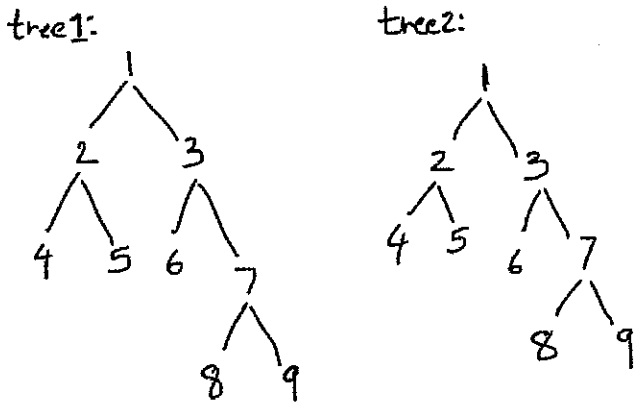
b) Given an empty ternary search tree, *draw* the ternary search tree after the string "**cscxxx**" is inserted into it. Just like your assignment, you can assume that *$* represents the end of string. [3]

c) Given the ternary search tree from b) above, *draw* the ternary search tree after the string "**cscrrr**" is inserted into it. Just like your assignment, you can assume that *$* represents the end of string. [3]

d) Given the ternary search tree from c) above, *draw* the ternary search tree after the string "**csc**" is inserted into it. Just like your assignment, you can assume that *$* represents the end of string. [3]

First Name:
Last Name:
Student #:

e) Write a *recursive* algorithm that compares two binary trees in *nodes and references form* and returns `True` if they are structurally identical *(they are made of nodes with the same values and arranged in the same way)* to each other. As an example the following two trees (`tree1` and `tree2`) are structurally identical to each other: [6]



- Your function MUST BE *RECURSIVE.*
- Clearly mark your base case(s) and recursive case(s).
- No helper functions are allowed.

```
def sameTrees(tree1,tree2):
'''Returns True if tree1 and tree2 are structurally identical to each other.
Returns False otherwise'''
```

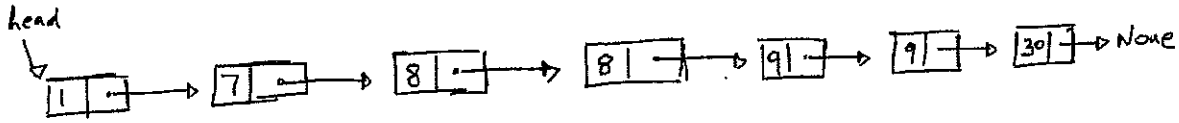
First Name:
Last Name:
Student #:

Question 4) Singly Linked List. **[19]**

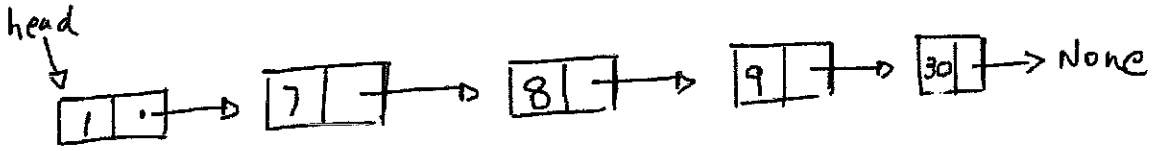
a) Write a function called `removeDuplicates(...)` that takes in a singly linked list in increasing order and deletes any duplicate nodes from the singly linked list. You can assume that the linked list passed in will ALWAYS be in increasing order. [12]

- Your function MUST BE *RECURSIVE.*
- Clearly mark your base case(s) and recursive case(s).
- No helper functions are allowed.

As an example if the following linked list is inputted in your function:



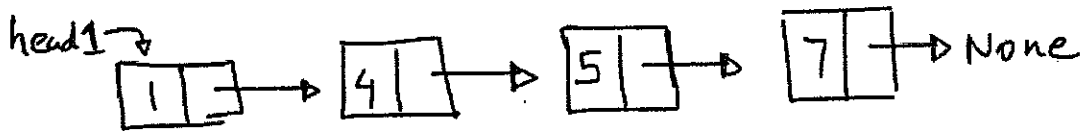
Your function MUST return the `head` of the same linked list with the duplicate nodes removed:

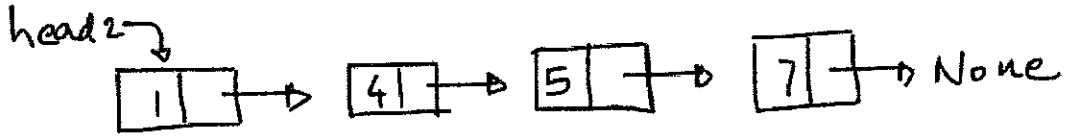



```
def removeDuplicates(head):
    '''head is the head of the singly linked list. This RECURSIVE
     function returns back the head of same linked list with duplicate
     nodes removed'''
```

First Name:
Last Name:
Student #:

b) Write a non-recursive function called `areLinkedListsEqual(...)` that takes in as input two linked list of integers and checks if they are equal to each other (i.e. have the same values in the same order) and have the same length. As an example, the following two linked list are equal to each other. [5]





- Your function MUST BE *NON RECURSIVE.*
- No helper functions are allowed.

```
def areLinkedListsEqual(head1,head2):
      '''returns True if linkedlist represented by head1 is equal to
      the linkedlist represented by head2 otherwise return False'''
```

First Name:
Last Name:
Student #:

c) What is the *runtime* complexity for your solution in part b) above from the following choices? [2]

a) O(n)
b) $O(n^2)$
c) O(1)
d) O(nlogn)
e) O(logn)
f) Something else

Explain your answer:

First Name:
Last Name:
Student #:

Extra Sheet.

*You **must** write your firstname, lastname and student# on this sheet if you like your any work on this sheet to be to considered for marking.*

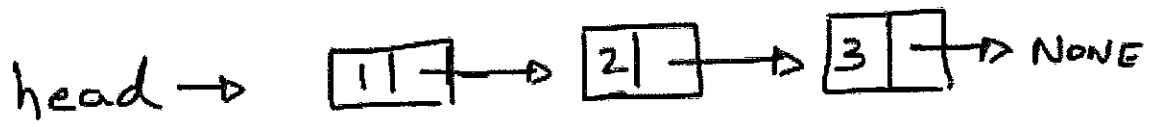
First Name:
Last Name:
Student #:

Appendix:

```
class Node:
    '''Node is a class that represents a single Node in a Linked List'''
     def __init__(self,element,next=None):
       '''Creates a Node instance. The _element of this instance is set to element and
       the _next of this instance is set to next'''
          self._element=element
          self._next=next
```

An example of creating a linked list using the Node class is as follows:

```
head=Node(1,Node(2,Node(3)))
```



==========

```
class BinaryTree:
 '''You can safely assume that this class has been
    correctly and fully implemented as per the above definition of Binary Tree.'''

    def __init__(self,rootValue):
        '''Creates a new BinaryTree where, rootValue refers to the integer value at
root            node. The left child is set to None, and the right child is set to
None'''
        self.root=rootValue
        self.left=None
        self.right=None
    .
    .
```

From the above code of BinaryTree:

`t.root` refers to the integer value at the root node of the tree `t`.
`t.left` is a reference to the left subtree of tree `t`.
`t.right` is a reference to the right subtree of tree `t`.

You can directly refer to the instance variables of the binary tree when writing your code.