APRIL 2018 FINAL EXAM CSC148H5S

Last Name: _____ **First Name:** _____

Student #: |_|_|_|_|_|_|_|_|_| **Signature:** _____

UNIVERSITY OF TORONTO MISSISSAUGA
APRIL 2018 FINAL EXAMINATION
CSC148H5S
Introduction to Computer Science
Daniel Zingaro, Larry Zhang, Vincent Maccio
Duration - 2 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, smart devices, 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, once this exam has begun, you **CANNOT** *re-write it.*

You must earn 40% or above on the exam to pass the course; else, your final course mark will be set no higher than 47%.

MARKING GUIDE

This final examination consists of 6 questions on 14 pages (including this one). When you receive the signal to start, please make sure that your copy of the examination is complete.

If you need more space for one of your solutions, use the last pages of the exam and indicate clearly the part of your work that should be marked.

1: _____/ 7
2: _____/ 6
3: _____/ 6
4: _____/ 6
5: _____/ 6
6: _____/ 7

Good Luck! TOTAL: _____/38

Question 1. [7 MARKS]

Answer **exactly** 7 of the 10 true or false questions below. You **do not** need to justify your answer, simply write true or false. **Make it very clear which question(s) you're answering.** If you answer more than 7 questions, only the first 7 answers will be marked.

1. An inorder traversal of a binary search tree produces the keys of the tree in sorted order.

2. For binary search tree T, The preorder and postorder traversals of T could be the same.

3. If each internal node of a binary tree has two children, then the tree is complete.

4. Given a nonempty alphabet, Huffman's algorithm assigns a 1-bit code to at least 1 symbol.

5. For a typical English book, Huffman's algorithm would place the letter "e" relatively close to the root.

6. Given a linked list with front/back/size attributes, removing its last node takes $O(1)$ time.

7. Given the first node in a chain of `Node` objects (no front/back/size attributes), the worst-case time for searching for a given node value is upper-bounded by $O(n^2)$.

8. It is possible for Quicksort to run in $O(n)$ time.

9. Python built-in lists have fast prepend but slow append.

10. An n-ary tree can be used to represent a binary search tree.

Question 2. [6 MARKS]

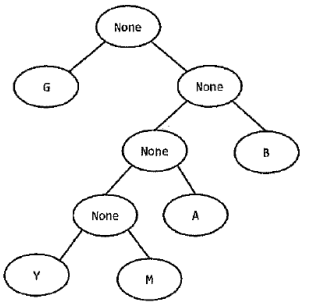
Part (a) [3 MARKS]

Below, draw the resulting Huffman tree built using the following frequency dictionary. Label each node of the tree with a symbol or `None`.

`{'h':23, 'u':21, 'f':11, 'm':9, 'a':50, 'n':5}`.

Part (b) [3 MARKS]

Given the following Huffman tree and the compressed text (in bits), fill in the blank with the **decompressed** text (in symbols).



Compressed text: `1001101101011011`

Decompressed text: _____

Question 3. [6 MARKS]

Explain in plain English the purpose of the following `mystery1` and `mystery2` functions. (Remember: this means that we want the overall purpose of the code, **not** a line-by-line description of what the code does.) For full marks, **be concise** and **clear**.

Part (a) [3 MARKS]

```
def helper(q):
    if q.is_empty():
        return 0
    item = q.dequeue()
    val = helper(q) + 1
    q.enqueue(item)
    return val

def mystery1(q):
    val = helper(q)
    val = helper(q)
    return val
```

Part (b) [3 MARKS]

```
def mystery2(n):
    # Precondition: n is an integer >= 0
    if n == 0:
        return False
    if n == 1:
        return True
    return mystery2(n-2)
```

Question 4. [6 MARKS]

Here is the binary tree node for this question.

```
class BT:

    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

Write the following function that returns the number of nodes in binary tree bt that have **exactly one child**.

```
def num_one(bt):
    """Return number of nodes in bt that have exactly one child."""
```

Question 5. [6 MARKS]

Here are the BTNode and BST classes for this question.

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

    def __init__(self: 'BTNode', data: object,
                 left: 'BTNode'=None, right: 'BTNode'=None) -> None:
        """Create BT node with data and children left and right."""
        self.data, self.left, self.right = data, left, right

class BST:
    """Binary search tree."""

    def __init__(self: 'BST', root: BTNode=None) -> None:
        """Create BST with BTNode root."""
        self._root = root

    def delete(self: 'BST', data: object) -> None:
        """Remove, if present, node containing data.

        self._root = _delete(self._root, data)
```

Complete each of the following functions.

Part (a) [3 MARKS]

```
def _find_max(node: BTNode) -> BTNode:
    """Find and return maximal node; assume node is not None"""
    # Your code here
```

APRIL 2018 FINAL E

Part (b) [3 MARKS]

```
def _delete(node: BTNode, data: object) -> BTN
    """Delete, if exists, node with data and r
    return_node = node
    if not node:
        pass
    elif data < node.data:
        node.left = _delete(node.left, data)
    elif data > node.data:
        node.right = _delete(node.right, data)
    elif not node.left:
        return_node = node.right
    elif not node.right:
        return_node = node.left
    else:
        # Your code here



    return return_node
```

Page 7 of

Question 6. [7 MARKS]

Consider the following classes `Node` and `LinkedList`. Complete the method `delete_all_occurrences` that removes **all of the nodes** whose `value` attribute equals the `value` parameter. There are **no** assumptions about the number or positions of such nodes in the linked list. Make sure to update all attributes of the `LinkedList` object appropriately.

```
class Node:

    def __init__(self, value):
        self.value = value
        self.next = None

class LinkedList:

    def __init__(self):
        self.front = None
        self.back = None
        self.size = 0

    def append(self, value):
        if self.size == 0:
            self.front = Node(value)
            self.back = self.front
        else:
            self.back.next = Node(value)
            self.back = self.back.next
        self.size = self.size + 1

    def delete_all_occurrences(self, value):
```

Short Python function/method descriptions:

```
__builtins__:
  input([prompt]) -> str
    Read a string from standard input; return that string with no newline. The prompt string,
    if given, is printed without a trailing newline before reading.
  len(x) -> int
    Return the length of the list, tuple, dict, or string x.
  max(iterable) -> object
  max(a, b, c, ...) -> object
    With a single iterable argument, return its largest item.
    With two or more arguments, return the largest argument.
  min(iterable) -> object
  min(a, b, c, ...) -> object
      With a single iterable argument, return its smallest item.
      With two or more arguments, return the smallest argument.
  range([start], stop, [step]) -> list-like-object of int
    Return the integers starting with start and ending with
    stop - 1 with step specifying the amount to increment (or decrement).
    If start is not specified, the list starts at 0.  If step is not specified,
    the values are incremented by 1.

dict:
  D[k] -> object
    Return the value associated with the key k in D.
  del D[k]
    Remove D[k] from D.
  k in D -> bool
    Return True if k is a key in D and False otherwise.
  D.get(k) -> object
    Return D[k] if k in D, otherwise return None.
  D.keys() -> list-like-object of object
    Return the keys of D.
  D.values() -> list-like-object of object
    Return the values associated with the keys of D.
  D.items() -> list-like-object of tuple of (object, object)
    Return the (key, value) pairs of D, as 2-tuples.

list:
  x in L -> bool
    Return True if x is in L and False otherwise.
  L.append(x) -> NoneType
    Append x to the end of L.
  L.index(value) -> int
    Return the lowest index of value in L.
  L.insert(index, x) -> NoneType
    Insert x at position index of L.
  L.pop(i) -> object
    Remove and return item at index i of L. Default to last index.
  L.remove(value) -> NoneType
    Remove the first occurrence of value from L.
  L.reverse() -> NoneType
    Reverse L *IN PLACE*.
  L.sort() -> NoneType
    Sort L in ascending order *IN PLACE*.
```


CONT'D...

```
str:
  x in S -> bool
    Return True if x is in S and False otherwise.
  str(x) -> str
    Convert an object into its string representation, if possible.
  S.count(sub[, start[, end]]) -> int
    Return the number of non-overlapping occurrences of substring sub in
    string S[start:end].  Optional arguments start and end are interpreted
    as in slice notation.
  S.find(sub[, i]) -> int
    Return the lowest index in S (starting at S[i], if i is given) where the
    string sub is found or -1 if sub does not occur in S.
  S.index(sub) -> int
    Like find but raises an exception if sub does not occur in S.
  S.isdigit() -> bool
    Return True if all characters in S are digits and False otherwise.
  S.lower() -> str
    Return a copy of S converted to lowercase.
  S.lstrip([chars]) -> str
    Return a copy of S with leading whitespace removed.
    If chars is given and not None, remove characters in chars from S.
  S.replace(old, new) -> str
    Return a copy of S with all occurrences of the string old replaced
    with the string new.
  S.rstrip([chars]) -> str
    Return a copy of S with trailing whitespace removed.
    If chars is given and not None, remove characters in chars from S.
  S.split([sep]) -> list of str
    Return a list of the words in S; use string sep as the separator and
    any whitespace string if sep is not specified.
  S.strip() -> str
    Return a copy of S with leading and trailing whitespace removed.
  S.upper() -> str
    Return a copy of S converted to uppercase.
```

b
binhan xiang
453
University of Toronto

Discover more


8

Tm trees - part of in assignment 1 part of i...

Introduction to... 100% (1)


1

Starter Code Architecture

Introduction to Computer... None


4

Application - part of in assignment 1 part ...

Introduction to Computer... None


3

Prep8 - part of in assignment 1 part of i...

Introduction to Computer... None

Recommended for you


8

Assignment: Lab 11 part 1 solutions codes

Introduction to... 100% (2)

8

Assignment: Lab 11 part 2 solutions codes

Introduction to...

100% (1)

2

Assignment: Lab 8 Question 7 solutions...

Introduction to...

100% (1)

1

Assignment: Lab 8 Question 13 solution...

Introduction to...

100% (1)