UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, December 2015

DURATION: 2½ hours

CSC 180 H1F — Introduction to Computer Programming

Calculator Type: None
Exam Type: D

Aids allowed: reference sheet distributed with the exam
Examiner(s): M. Guerzhoy

Student Number: |_|_|_|_|_|_|_|_|_|_|

Family Name(s): ______________________________

Given Name(s): ______________________________

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This final examination paper consists of 9 questions on 28 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy is complete, and fill in the identification section above.*

Answer each question directly on this paper, in the space provided, and use the reverse side of the previous page for rough work. If you need more space for one of your solutions, use the reverse side of a page or the pages at the end of the exam and *indicate clearly the part of your work that should be marked.*

Write up your solutions carefully! Comments and docstrings are *not* required to receive full marks, except where explicitly indicated otherwise. However, they may help us mark your answers, and part marks *might* be given for partial solutions with comments clearly indicating what the missing parts should accomplish.

When you are asked to write code, *no* error checking is required: you may assume that all user input and argument values are valid, except where explicitly indicated otherwise.

Use the Python 3 programming language. You may not `import` any module except `math`, unless otherwise specified.

A mark of at least **40%** (after adjustment, if there is an adjustment) on this exam is required to obtain a passing grade in the course.

Marking Guide

# 1:	______/ 15
# 2:	______/ 15
# 3:	______/ 20
# 4:	______/ 15
# 5:	______/ 8
# 6:	______/ 6
# 7:	______/ 5
# 8:	______/ 6
# 9:	______/ 10
TOTAL:	______/100

Good Luck!
May the Force be with you!

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 1. [15 MARKS]

Part (a) [12 MARKS]

Write a function with the signature `is_sorted(L)` which takes in a list of `int`s `L`, and returns `True` iff the list `L` is sorted, in either non-increasing or non-decreasing order. For example,

`is_sorted([4, 5, 6, 1, 2, 3, 7])` should return `False`,
`is_sorted([4, 5, 5, 6])` should return `True`,
`is_sorted([6, 3])` should return `True`,
`is_sorted([])` should return `True`.

Part (b) [3 MARKS]

What is the tight asymptotic bound on the worst-case runtime complexity of the function you wrote in Part (a)? Use Big O notation. You should assume that all the `int`s are smaller than $32,000$.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D. . .

Question 2. [15 MARKS]

Write a function with the signature `euc_distance(u, v)` which computes the Euclidean distance between the endpoints of the two sparse vectors `u` and `v`. Reminder: we store sparse vectors using dictionaries, with only the non-zero entries being stored. For example, `[4, 5, 0, 10, 0]` is stored as `{1:4, 2:5, 4:10}`. The Euclidean distance between the endpoints of the vectors $[u_1, u_2, u_3, ..., u_k]$ and $[v_1, v_2, v_3, ..., v_k]$ is

$$\sqrt{\sum_{i=1}^{k}(u_i - v_i)^2}.$$

Reminder: you can assume that the input to the function is valid.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 3. [20 marks]

Write a function with the signature `movies_by_release_date(movies)` which takes in a dictionary whose keys are movie names and whose values are release dates, and which returns a list of movie names, in order from the most recent release date to the earliest release date. The release dates are either in the format `"<year>, in <location>"`, or in the format `"a long time ago, in <location>"`. Any movies released `"a long time ago"` were released before the movies for which the year is indicated. Movies released at the same time can be placed in the list in any order.

For example, if `movies` equals

```
{"Dude, Where's My Death Star": "a long time ago, in a galaxy far far away",
 "Star Wars: The Force Awakens": "2015, in Los Angeles",
 "Star Wars": "1977, in Los Angeles",
 "Sleepless in Aldera":  "a long time ago, in Alderaan City",
 "Jurassic World": "2015, in New York"},
```

`movies_by_release_year(movies)` can return

```
["Jurassic World", "Star Wars: The Force Awakens", "Star Wars",
 "Sleepless in Aldera", "Dude, Where's My Death Star" ].
```

Part marks may be given for clearly-documented helper functions accompanied by clear and concise explanations of how they would help with the overall solution. However, as with the other questions, full marks can be given for correct solutions that are not commented or separated into functions.

Use this page for rough work—clearly indicate any section(s) to be marked.

cont'd...

Extra space for solutions

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 4. [15 MARKS]

Write a recursive function with the signature `merge(L1, L2)` which takes in two lists sorted in non-decreasing order, and returns a list that contains all the elements from both `L1` and `L2`, and is itself sorted. **You may not use loops, global variables, or helper functions, and the function signature must be exactly as specified (i.e., you may not add additional parameters)**. You may not use Python's `sorted()` and `sort()` functions. You may use slicing.

For example, `merge([4, 8, 10], [2, 5])` should return `[2, 4, 5, 8, 10]`.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 5. [8 MARKS]

Each of the subquestions in this question contains a piece of code. Treat each piece of code independently (*i.e.*, code in one question is not related to code in another), and **write the expected output for each piece of code**. If the code produces an error, write down the output that the code prints before the error is encountered, and then write "ERROR." You do not have to specify what kind of error it is.

Part (a) [2 MARKS]

```
def f():
  print("Solo shot first")

def g():
  print("Greedo shot first")

def h(g):
  g()

h(f)
```

Part (b) [2 MARKS]

```
L = [[5, 6], [7, 8]]
M = L[:][:]
M[1] = [3, 4]
M[0][1] = 2
print(L)
```

Part (c) [2 MARKS]

```
def f(L, M):
  global L
  L = M
  L[0] = 3

M = [1, 2]
L = [3, 4]
f(L, M)
print(M[0])
```

Part (d) [2 MARKS]

```
s1 = "Happy Holidays!"
s2 = s1
s1 = "HO HO HO"
print(s2)
```

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 6. [6 MARKS]

The left-hand column in the table below contains different pieces of code that work with list `L`, string `s` and integer `n`. In the right-hand column, give the asymptotic tight upper bound on the worst-case runtime complexity of each piece of code, using Big O notation.

Code	Complexity
`# L is a list of floats with n = len(L)`	
`total = 0.0`	
`for i in range(len(L)):`	
`   if L[i] > 0.0:`	
`      total += L[i]`	
`# L is a list with n = len(L)`	
`a = 5.0`	
`for i in range(n):`	
`   for j in range(i % 2):`	
`      a += 1`	
`# L is a list with n = len(L)`	
`def f(L, i, j):`	
`  if j-i <= 1:`	
`    return L[i]`	
`  if L[i] == 0:`	
`    return f(L, i, i + (j-i)//2)`	
`  else:`	
`    return f(L, i + (j-i)//2, j)`	
`f(L, len(L)//5, len(L)//4)`	

Use this page for rough work—clearly indicate any section(s) to be marked.

cont'd...

Question 7. [5 MARKS]

What is the tight asymptotic upper bound on the runtime complexity of the Forward Step of the Gaussian Elimination algorithm, applied to a matrix of `float`s of size $m \times n$ (i.e., m rows and n columns)? Use Big O notation. Justify your answer (a formal proof is *not* required; we should simply be able to understand how you got the answer). An example of a run of the Forward Step is provided on the reference sheet.

Answer:

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 8. [6 marks]

Consider the following code

```
def mystery_helper(L1, L2):
    L = L1 + L2
    for i in range(len(L)):
        m, loc_m = L[i], i
        for j in range(i, len(L)):
            if L[j] > m:
                m, loc_m = L[j], j
        L[i], L[loc_m] = L[loc_m], L[i]
    return L

def mystery(L):
    step = 1
    while step <= len(L):
        for partition_start in range(0, len(L), 2*step):
            start1 = partition_start + 0 * step
            end1   = partition_start + 1 * step

            start2 = end1
            end2   = partition_start + 2 * step


            L[start1:end2] = mystery_helper(L[start1:end1], L[start2:end2])
        step *= 2
```

Part (a) [1 mark]

State clearly and concisely what mystery_helper(L1, L2) returns.

Part (b) [2 marks]

State clearly and concisely what mystery(L) does.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Part (c) [3 MARKS]

What is the tight asymptotic upper bound on the worst-case runtime complexity of `mystery(L)`, where `n = len(L)`? Use Big O notation. Explain how you got your answer to this subquestion. You may assume that `L` is a list of `float`s.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Question 9. [10 MARKS]

Write a function with the signature `x_can_win(board)` which takes in a 3×3 tic-tac-toe board, stored as a list of lists, and returns `True` iff the player who plays `"X"` has a winning strategy starting from the state of the board, assuming it is `"X"`'s move. That is, `x_can_win(board)` should return `True` if and only if `"X"` can always eventually win starting from the state of the board if `"X"` plays correctly, no matter how `"O"` plays.

Santa is giving you the function `x_won(board)`, which returns `True` iff there is a (vertical, horizontal, or diagonal) row of 3 `"X"`s somewhere on the board. You can use the function.

The list of lists `board` is in the following format (which is identical to what was used in the lab.)

`board == [["O", "  ", "  "], ["X", "O", "X"], ["X", "O", " "]]` means that the position is

```
 O |   |
---+---+---
 X | O | X
---+---+---
 X | O |
```

For example, `x_can_win([[" ", " ", " "], [" ", "X", "O"], [" ", " ", " "]])` should return `True` and `x_can_win([["O", "  ", "  "], ["X", "O", "X"], ["X", "O", " "]])` should return `False`.

For this question, you will receive 2 marks if you answer "I don't know." Part marks will only be given for making substnatial progress toward a solution.

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Extra space for solutions

Use this page for rough work—clearly indicate any section(s) to be marked.

CONT'D...

Extra space for solutions

PLEASE WRITE NOTHING ON THIS PAGE