UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, DECEMBER 2016

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 10 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:	_____/ 15
# 4:	_____/ 5
# 5:	_____/ 10
# 6:	_____/ 8
# 7:	_____/ 8
# 8:	_____/ 7
# 9:	_____/ 10
# 10:	_____/ 7
TOTAL:	_____/100

Good Luck!

OVER...

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

Question 1. [15 MARKS]

Part (a) [12 MARKS]

Write a function with the signature `insert(L, e)` which takes in a list of `floats` `L`, which is sorted in non-decreasing order, and returns a new list which is also sorted in non-decreasing order, and contains all the elements of `L` as well as the `float` `e`. Here are some examples:

`insert([3.0, 4.0, 5.0], 3.5)` should return `[3.0, 3.5, 4.0, 5.0]`.
`insert([2.0, 5.0], 7.0)` should return `[2.0, 5.0, 7.0]`.
`insert([], 42.0)` should return `[42.0]`.

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.

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

Question 2. [15 MARKS]

Santa wants to select gifts for an EngSci student. Santa has two dictionaries: a dictionary that records the rating of how good a gift would be for the student (on a scale of 1-5), and a dictionary that records the rating of how much the student wants the gift (on a scale of 1-5). **The rating of a gift which is not in either of the dictionaries is considered to be 0**. Santa wants to select the gifts with the maximal possible combined rating, where the combined rating is the sum of the rating of how good the gift would be for the student and the rating of how much the student wants the gift. For example, the dictionaries can be:

```
good_ratings = {"Calc textbook": 5, "iPhone": 1, "Alarm clock": 4, "Notebooks": 4}
want_ratings = {"iPhone": 4, "A+ in CSC": 5, "Calc textbook": 4, "Notebooks": 5}
```

Here, the gifts Santa wants to select are `"Calc textbook"` and `"Notebooks"`, since the combined rating for them is `5+4=9`, larger than any other one. The combined rating of `"Alarm clock"` is `4+0=4`.

Write a function with the signature `select_gifts(good_ratings, want_ratings)` that returns a list of all the gifts which have the highest combined rating of all the gifts, sorted in alphabetical order.

For example, for `good_ratings` and `want_ratings` as defined above,
`select_gifts(good_ratings, want_ratings)` should return `["Calc textbook", "Notebooks"]`.

```
def select_gifts(good_ratings, want_ratings):
```

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

CONT'D...

Question 3. [15 MARKS]

In Python, you can use a *list of lists* to store a matrix, with each inner list representing a row. For example, you can store the matrix

$$\begin{pmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{pmatrix}$$

by storing each row as a list: `M = [[5,  6, 7], [0, -3, 5]]`.

Complete the following function. The function takes in a matrix `M` in a list-of-lists format. The matrix returns the transposed version of `M`, in a list-of-lists format. For example,

`transpose([[5,  6, 7], [0, -3, 5]])` should return `[[5, 0], [6, -3], [7, 5]]`.

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

Question 4. [5 MARKS]

Write a recursive function with the signature `max_rec(L)` which takes in a list of `int`s `L`, and returns the largest element in the list. You may not use loops, global variables, or Python's `max()`, `sorted()` and `sort()` functions. You may use slicing.

For example, `max_rec([103, 180, 101, 102, 180])` should return `180`.

```
def max_rec(L):
```

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

Question 5. [10 MARKS]

Write a recursive function with the signature `is_fib(L)` which takes in a list of `int`s `L`, and returns `True` if `L` is the start of the Fibonacci sequence, and `False` otherwise. For example:

`is_fib([1, 1, 2, 3, 5])` should return `True`.
`is_fib([1, 1, 2, 3, 5, 8, 13])` should return `True`.
`is_fib([5, 8, 13])` should return `False`.
`is_fib([1, 1, 1])` should return `False`.
`is_fib([])` should return `True`.
You may not use helper functions, loops or global variables. You may use slicing.
Reminder: $fib(n+2) = fib(n+1) + fib(n), fib(1) = fib(2) = 1$.

```
def is_fib(L):
```

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

Question 6. [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]

```
A = [[1, 2], [3, 4]]
A[0] = A[1]
B = A[:][0]
B[0] = 5
print(A)
```

Part (b) [2 MARKS]

```
def f():
  L[0] = 5

L = [1, 2]
print(f(L))
print(L)
```

Part (c) [2 MARKS]

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

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

Part (d) [2 MARKS]

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

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

Question 7. [8 MARKS]

The left-hand column in the table below contains different pieces of code that work with 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. Assume that arithmetic operations such as `+` and `**` take contstant time.

Code	Complexity

`total, i = 0.0, 0`
`for i in range(n):`
`  for j in range(i//2):`
`  total += i`	

`i, j, sum = 1, 1, 0`
`while i < n**3:`
`  while j < n:`
`  sum = sum + i`
`  j += 1`
`  i += n`	

`def f(n):`
`  if n == 0:`
`  return 1`
`  return f(n//2) + f(n//2)`
`if __name__ == "__main__":`
`  f(n)`	

`def f(n):`
`  i, total = 0, 0.0`
`  while (i < n) and ((i % 10000) != 0):`
`  total += i`
`  i += 1`
`if __name__ == "__main__":`
`  f(n)`	

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

Question 8. [7 MARKS]

Consider the following code

```
def mystery_helper(L, k):
    p = max(L[0], L[-1])
    L1 = []
    L2 = []

    for e in L:
        if e < p:
            L1.append(e)
        else:
            L2.append(e)

    if len(L1) > k:
        return mystery_helper(L1, k)
    elif len(L1) < k:
        return mystery_helper(L2, k-len(L1))
    else:
        return p

def mystery(L):
    return mystery_helper(L, len(L)//2)
```

Part (a) [4 MARKS]

State clearly and concisely what `mystery_helper(L, k)` returns.

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

Part (b) [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.

Question 9. [10 MARKS]

A timestamp is a tuple consisting of two integers, with the first one denoting the hour in the day (between 0 and 23), and the second one denoting the minute (between 0 and 59). The timestamp `(5, 10)` corresponds to 5:10AM, the timestamp `(13, 25)` corresponds to 1:25PM, and so on. Write a function with the signature `sorted_timestamps(timestamps)` that takes in a list of timestamps, and returns a sorted version of that list, with the sorting done from earlier to later timestamps. **The function must run in $O(n)$ time, where** $n = \text{len(timestamps)}$. For example,

`sorted_timestamps([(5, 10), (2, 40), (22, 59), (5, 10)])`
should return `[(2, 40), (5, 10), (5, 10), (22, 59)]`.

```
def sorted_timestamps(timestamps):
```

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

Question 10. [7 MARKS]

We can use a dictionary to record who is friends with whom by recording the lists of friends in a dictionary. For example:

```
friends = {"Carl Gauss": ["Isaac Newton", "Gottfried Leibniz", "Charles Babbage"],
           "Gottfried Leibniz": ["Carl Gauss"],
           "Isaac Newton": ["Carl Gauss", "Charles Babbage"],
           "Ada Lovelace": ["Charles Babbage", "Michael Faraday"]}
           "Charles Babbage": ["Isaac Newton", "Carl Gauss", "Ada Lovelace"],
           "Michael Faraday"" ["Ada Lovelace"] }
```

Here, Carl Gauss is friends with Isaac Newton, Gottfried Leibniz, and Charles Babbage. Assume that friendships are symmetric, so that if X is friends with Y, then it's guaranteed that Y is friends with X. A clique is defined as a group of friends where everyone is friends with everyone. For example, Carl Gauss, Isaac Newton, and Charles Babbage form a clique in the example above, since all three are friends with each other. Ada Lovelace and Michael Faraday also form a clique.

Write the function `max_clique(friends)`, which takes in a dictionary in the format above, and returns the largest clique that can be found, as a list. (If there are several such cliques, return one of them.) For example, the largest clique in the example above is `["Carl Gauss", "Isaac Newton", "Charles Babbage"]`, since there is no clique of size larger than 3.

```
def max_clique(friends):
```

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

Extra space for solutions

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

Extra space for solutions

PLEASE WRITE NOTHING ON THIS PAGE