UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, DECEMBER 2023

DURATION: 2½ hours

ESC 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: |__|__|__|__|__|__|__|__|__|__|

UTORid: _____

UofT email: _____@mail.utoronto.ca

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 7 question on 16 page (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. Use the pages at the end of the exam for extra space. If you use extra pages, indicate that you have done so in the space under the question.

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.

MARKING GUIDE

7 # 0: _____/ **??**

TOTAL: _____/100

Good Luck!

OVER...

Question 1. [15 MARKS]

In this question, your function will take in a list of integers `L`. Some of the integers may appear in the list more than once. The function should return a list of all the integers that appear in the list `L` more than once. The returned list should be sorted in increasing order, and must not contain duplicate integers: all integers in the returned list must be unique.

```
def get_repeating_ints(L):
    """Return a list of the integers that repeat in L, sorted in
    increasing order, with no duplicates.
    >>> get_repeating_ints([6, 7, 6, 5, 1, 5, 6])
    [5, 6]
    >>> get_repeating_ints([1, 2, 3])
    []
    """
```

Question 2. [15 MARKS]

Part (a) [12 MARKS]

Without using Python's `sorted` or `list.sort` functions, write a function that finds the median of a list of an odd number of `float`s. The median of a list `L` of length n is a number such that at least $(n-1)/2$ elements of `L` are smaller or equal to it, and at least $(n-1)/2$ elements of `L` are larger or equal to it. For example, `my_median([5.0, 2.0, 4.0, 1.0, 3.0])` should return `3.0`. There are no restrictions on the runtime complexity of this function.

```
def my_median(L):
    """Return the median of the list of floats L. Assume
    len(L) is odd.
    """
```

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 do not need to justify your answer.

Question 3. [15 MARKS]

Santa received a list of gift requests, in the format of a Python list. An example list looks like

```
requests = ["socks", "calculus textbook", "calculator", "A+ in ESC180", "socks", ...]
```

Write a function that takes in a list like `requests` and returns the 10 most-requested items (i.e., the top 10 items that appear the most often in the list). You can assume that there are no ties in the numbers of times that items are requested. You can assume there are at least 10 different items in the list. The return list should be alphabetically sorted.

```
def top10requests(requests):
```


CONT'D...

Question 4. [15 MARKS]

Write a function that takes in a list `L`, and returns a list that consists of every third element of `L`. For example, `every_third([5, 6, 7, 12, 0, 4, 6])` should return `[7, 4]`. **You must use recursion**. You must not use global variables, for-loops, while-loops, or list comprehensions.

Question 5. [12 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) [3 MARKS]

```
def f(L):
  L = ["holidays"]

L = ["happy"]
f(L)
print(L)
```

Part (b) [3 MARKS]

```
L = [[[1, 2], 3], [4]]
L1 = []
for sublist in L:
  L1.append(sublist[:])
L[0][0][0] = 5
L[0][1] = 5
L[1][0] = 5
print(L)
print(L1)
```

Part (c) [3 MARKS]

```
def doubler(L):
  dL = L
  for index in range(len(dL)):
    dL[index] = dL[index] * 2
L = [1, 2, 3]
doubler(L)
print(L)
```

Part (d) [3 MARKS]

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

Question 6. [12 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 constant time.

Code	Complexity

`def power2(n):`
`  if n == 1:`
`  return 2.0`
`  else:`
`  temp = power2(n-1)`
`  return temp + temp`
`power2(n)`	

`def f(n):`
`  i, j, sum = 1, 1, 0`
`  while i < n:`
`  while j < i:`
`  sum = sum + j`
`  j += 1`
`  i *= 2`
`f(n)`	

`def g(n):`
`  if n == 0:`
`  return 1`
`  return g(n//2) + g(n//2) + g(n//2)`
`g(n)`	

`def h(n):`
`  total = 0.0`
`  for i in range(n):`
`  for j in range(n):`
`  if i == j:`
`  break`
`h(n)`	

Question 7. [16 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 *friendship chain* is a chain of people who are connected by friendship, with no repetitions allowed. For example, the following is a friendship chain of length 5:

Carl Gauss → Isaac Newton → Charles Babbage → Ada Lovelace → Michael Faraday

Write a function which takes in a dictionary in the format above, and returns the length of the longest friendship chain in the data in the dictionary.

Extra space for solutions

CONT'D...

Extra space for solutions

CONT'D...

Extra space for solutions

Extra space for solutions

CONT'D...

Extra space for solutions

PLEASE WRITE NOTHING ON THIS PAGE