CSCA48 Winter 2015 Midterm Exam
Duration — 100 minutes
Aids allowed: none

Student Number: _____

Instructors: Anna Bretscher & Brian Harrington

Last Name: _____

First Name: _____

UtorID (Markus Login): _____

Please place a checkmark (✓) beside your tutorial session

Tutorial Number	Date/Time	TA Name	Check
TUT0001	MO 09:00 10:00 PO 101	Eric Wang	
TUT0002	MO 10:00 11:00 PO 101	Ryan Williams	
TUT0003	MO 11:00 12:00 IC 326	Nick Dujay	
TUT0004	MO 12:00 13:00 BV 361	Kenneth Ma	
TUT0005	MO 13:00 14:00 AA 209	Weining (William) Zhou	
TUT0006	MO 16:00 17:00 HW 402	Ben Cooper	
TUT0007	TU 09:00 10:00 MW 160	Kirisanth (Kiwi) Ganeshamoorthy	
TUT0008	TU 10:00 11:00 MW 160	Rohini Ragunathan	
TUT0009	TU 11:00 12:00 HW 408	Anastasios Exacoustos	
TUT0010	TU 12:00 13:00 HW 408	Ekin Ozcelik	
TUT0012	WE 09:00 10:00 MW 170	Lev Karatun	
TUT0013	WE 10:00 11:00 IC 326	Chengyu (Tyrone) Xiong	
TUT0014	WE 12:00 13:00 AA 112	Faisal Usmani	

*Do **not** turn this page until you have received the signal to start.*

This exam consists of 4 questions on 14 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Proper documentation is required for all functions and code blocks. If you use any space for rough work, indicate clearly what you want marked. Please read all questions thoroughly before starting on any work.

We have provided you with grids for your answers, this is simply to help you show the indentation of your code and you are not required to adhere to the grids in any specific way.

The University of Toronto's Code of Behaviour on Academic Matters applies to all University of Toronto Scarborough students. The Code prohibits all forms of academic dishonesty including, but not limited to, cheating, plagiarism, and the use of unauthorized aids. Students violating the Code may be subject to penalties up to and including suspension or expulsion from the University.

1: _____/10

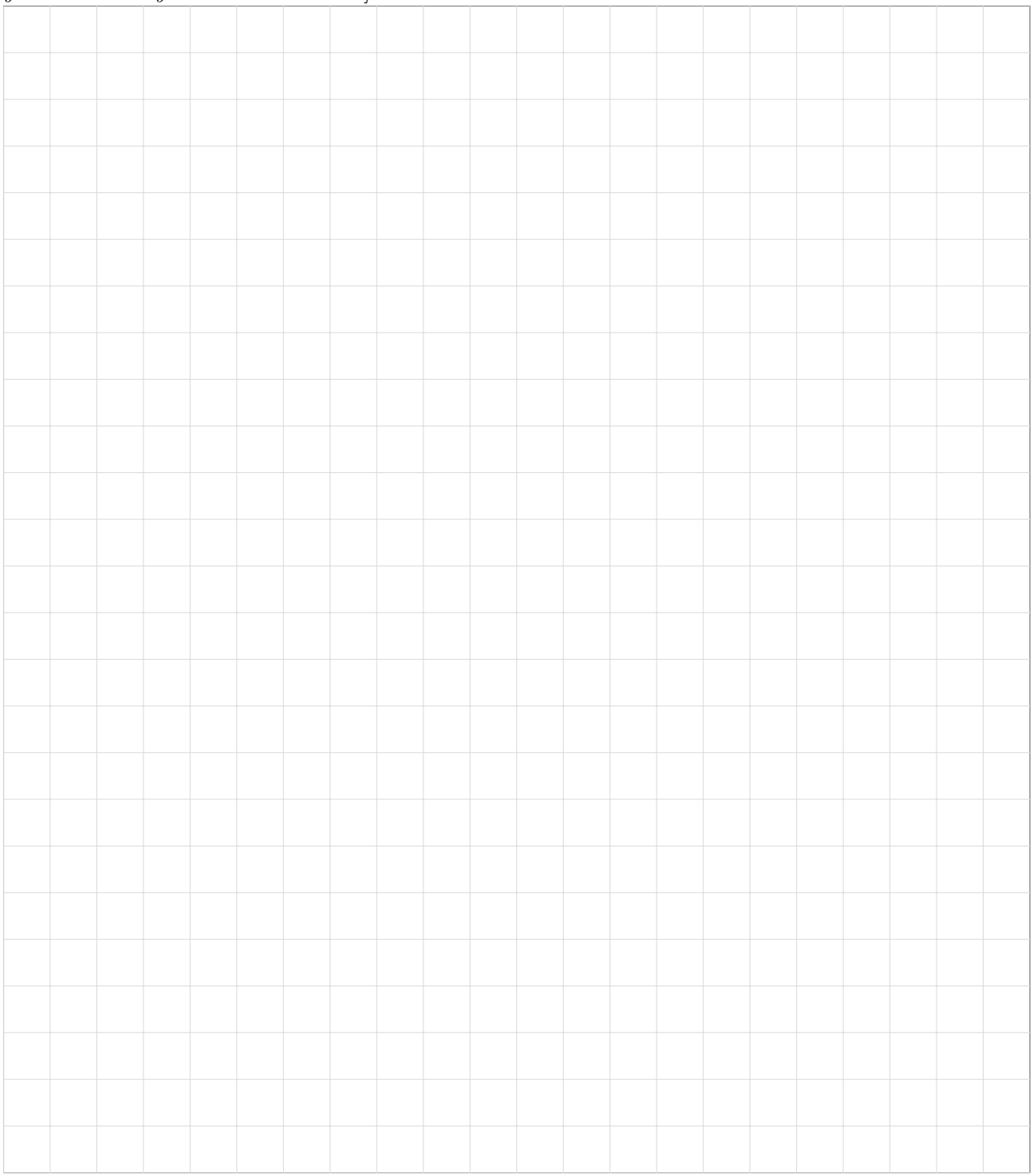
2: _____/10

3: _____/10

4: _____/15

TOTAL: _____/45

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Question 1. [10 MARKS]

In the space below, write the following function:

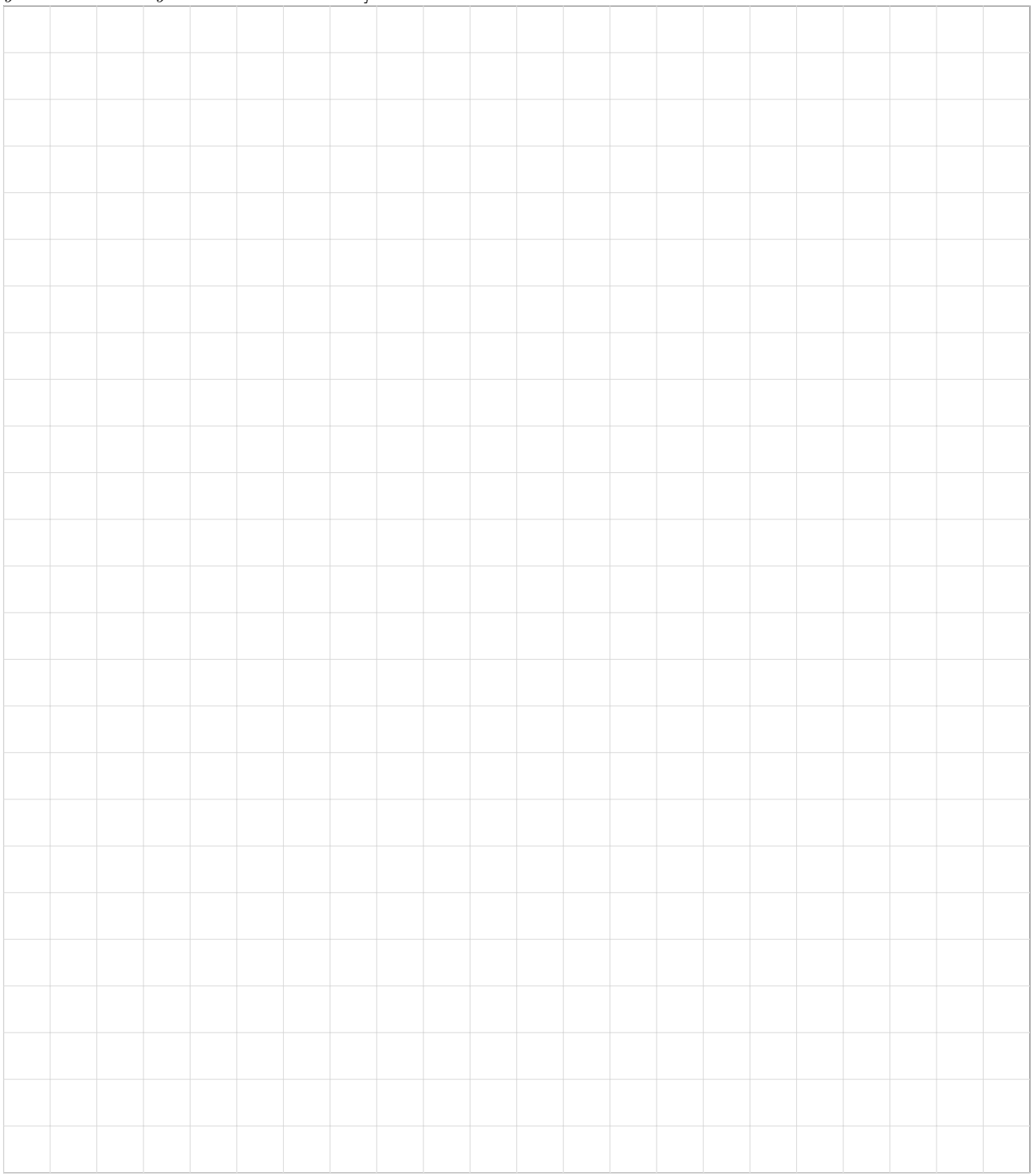
`edit_distance(s1, s2):` [1]

The edit distance of two strings `s1` and `s2` is defined as: `The minimum number of single character changes that would be needed to turn s1 into s2`.

You may assume that `s1` and `s2` are the same length, and the only character changes available are replacing one character with another. However, you can receive 2 bonus marks if you assume s1 and s2 may be different lengths, and we can also add and delete characters (you must clearly indicate in your comments if you are attempting the bonus mark. For full marks, your solution **must** be recursive.

[1] Gee... this function looks familiar... almost like it came from somewhere that we were told would be *good practice for the midterm...*

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Question 2. [10 MARKS]

In the space below, write the following function:

`all_perms(s1):`

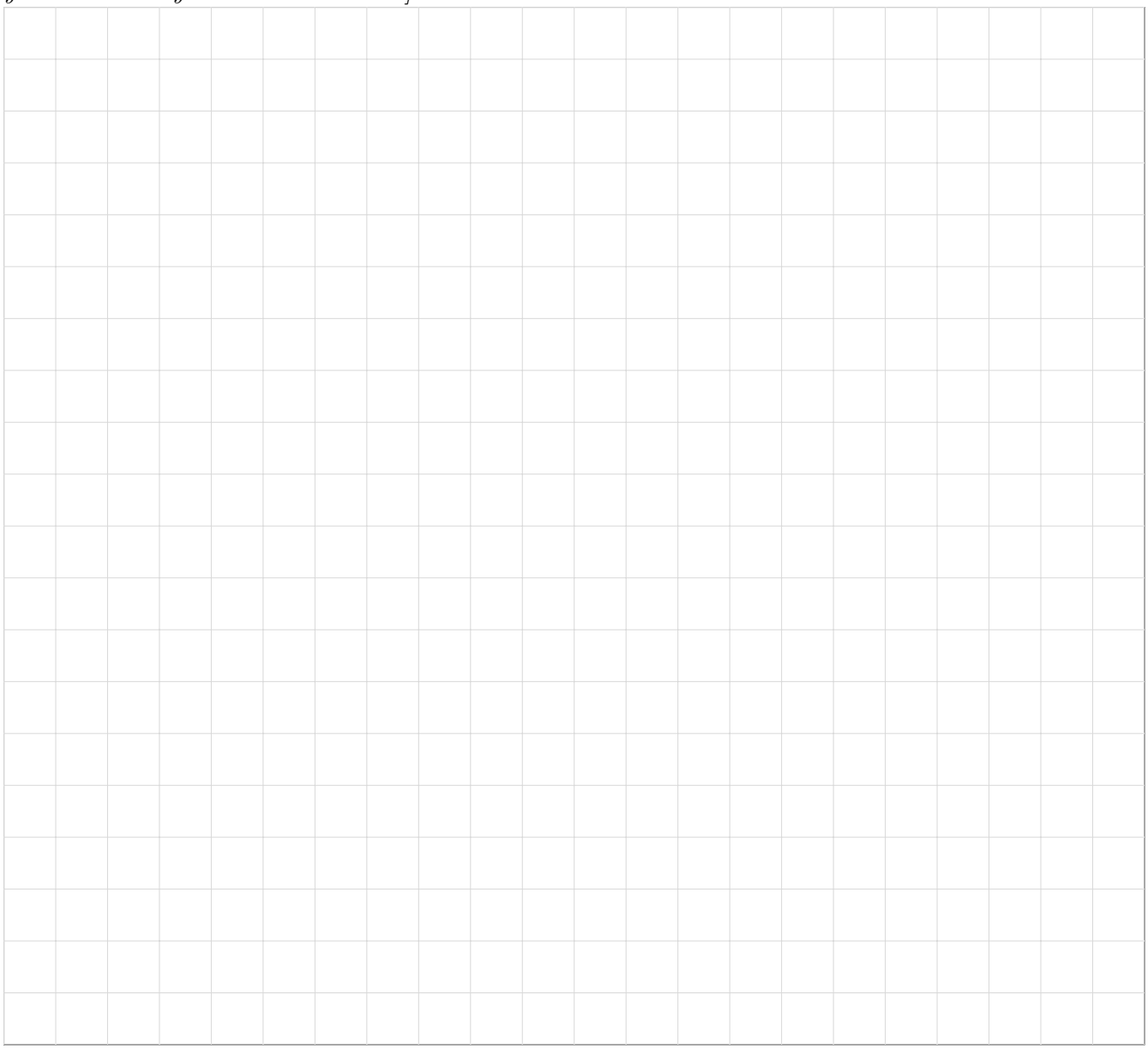
Given a string `s1`, return a list of all permutations of the letters in `s1`. For example: `all_perms('abc')`, might return something like: `['abc', 'bac', 'bca', 'acb', 'cab', 'cba']`

You may solve the problem in any way you wish, but for full marks, the solution must be at least partially recursive (i.e., it may use both loops and recursion). You will be assessed on the efficiency and simplicity of your solution.

```
from math import *

def f1(a, b, r):
    ''' (int, int, float) -> list of (int, int)
    REQ: 0 < r < 1.
    '''
    if a >= b:
        return []  # return empty list
    c = floor(a + (b-a)*r)
    return f1(a, c, r) + f1(c+1, b, r) + [(a, b)]  # concatenate 3 lists
```

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Question 3. [10 MARKS]

Consider the function `f1` on the previous page:

Part (a) [5 MARKS]

In the space below, write the output of the following code:

```
print("STEP 1":, f1(1, 7, 0.5))
print("STEP 2:", f1(10, 20, 0.2))
```

Part (b) [5 MARKS]

In the space below, write a function `f2` with the same type contract as `f1` such that `f2(a, b, r)` returns the same list that `f1(a, b, r)` returns except in reverse order. For full marks, your solution **must** be recursive, and cannot call any other functions besides `floor`. You are not required to provide external documentation for this function.

The CustomerNode Class

```
class CustomerNode:
    def __init__(self, ticket_num,  cust_name, next):
        '''(CustomerNode, int, str, CustomerNode) -> None
        Create a new CustomerNode with ticket number ticket_num,
        customer name cust_name, and pointing to next.
        '''
        self._ticket_num = ticket_num
        self._cust_name = cust_name
        self.next = next

    def get_ticket_num(self):
        return self._ticket_num

    def get_cust_name(self):
        return self._cust_name
```

The Mangled Code

```
# append any remaining nodes to new_list
# append corresponding node to new_list
# initialize list to return and its last node
# return merged list
# test which head of list has smaller ticket_num and
# while both lists not empty
else:
if(L1 == None):
if(L1.get_ticket_num() < L2.get_ticket_num()):
if(new_list == None):
L1 = L1.next
L2 = L2.next
last = L1
last = L2
last = None
last.next = L1
last.next = L2
new_list = L1
new_list = L2
new_list = None
return new_list
while(L1 != None and L2 != None):
```

Question 4. [15 MARKS]

Part (a) [10 MARKS]

Nick's Deli has just two servers, Anna and Brian. It's a small operation. To control their long queues of customers, they use a roll of numbered tickets. Each successive ticket's number is one greater than its previous ticket's number. On entry every customer takes the next available ticket and lines up either in Anna's queue or in Brian's queue. Sometimes Anna or Brian must leave to teach an A48 lecture. When this happens the two queues are merged into a single queue, and all customers are served in the order they arrived (i.e., ordered by their ticket number).

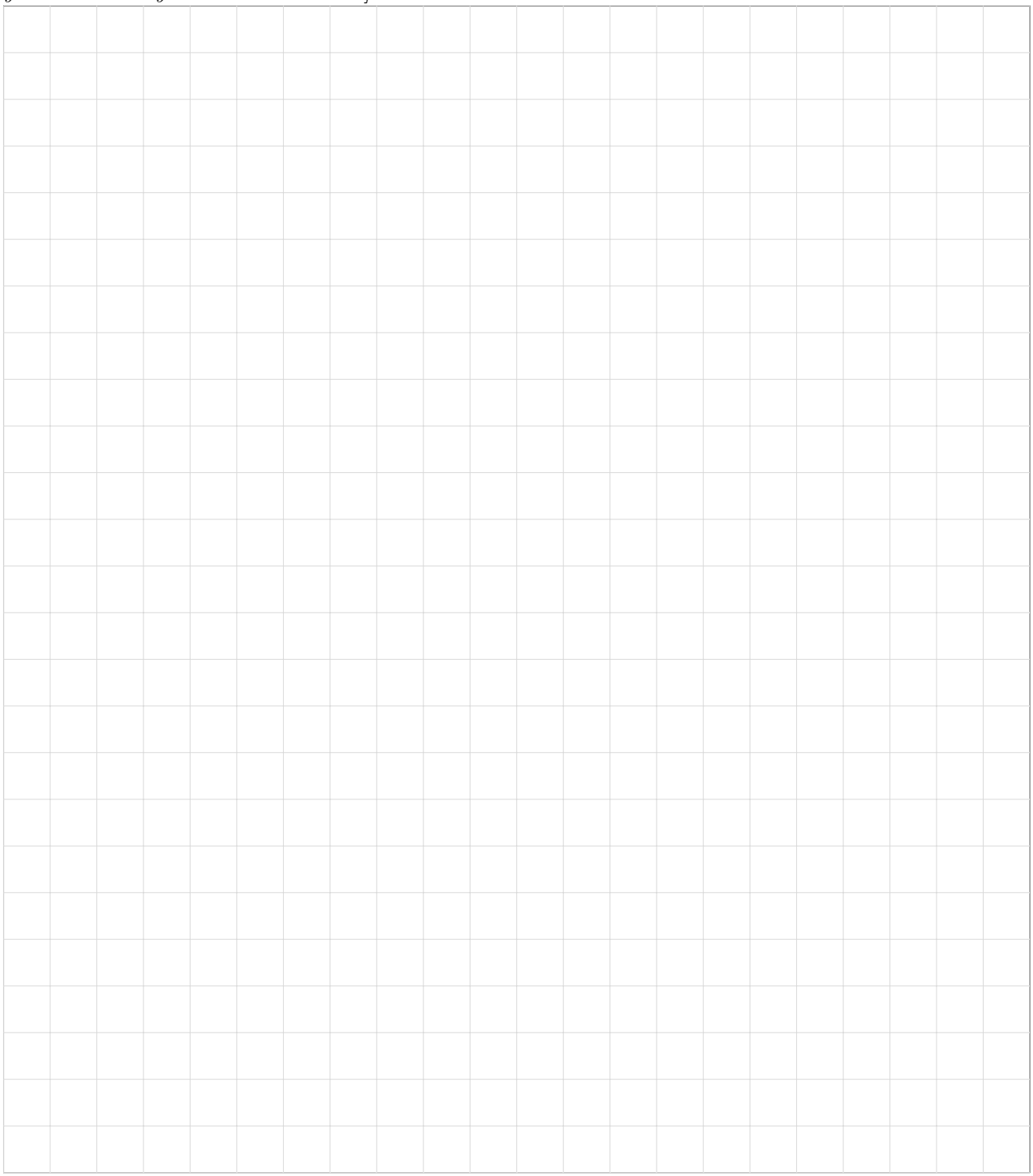
Nick decided to write some code to simulate the queues at his deli. He used linked lists to represent queues. First he created the class CustomerNode for the nodes in his linked list. Then he wrote a function called `listmerge()`, which takes two sorted (by ticket number) linked lists of customers, and returns a single sorted list of customers from the two original lists.

Then disaster happened (guess who...). The CODE MANGLER struck! The docstring was erased. All indentation was removed, and duplicate lines were deleted, then the remaining lines of code (including internal comments) were shuffled. The mangled code can be found on the previous page[2]. Please reconstruct Nick's `listmerge()` function.

```
def listmerge(L1, L2):
    '''(CustomerNode, CustomerNode) -> CustomerNode
    Merge the linked lists headed by L1 and L2 into a single list with ticket_num in
    increasing order. Return the head of the merged list.
    REQ: Lists headed by L1 and L2 are sorted by ticket_num, all ticket_num values are unique.
    '''
```

[2]You are not required to use the mangled code, but it is strongly recommended that you do

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



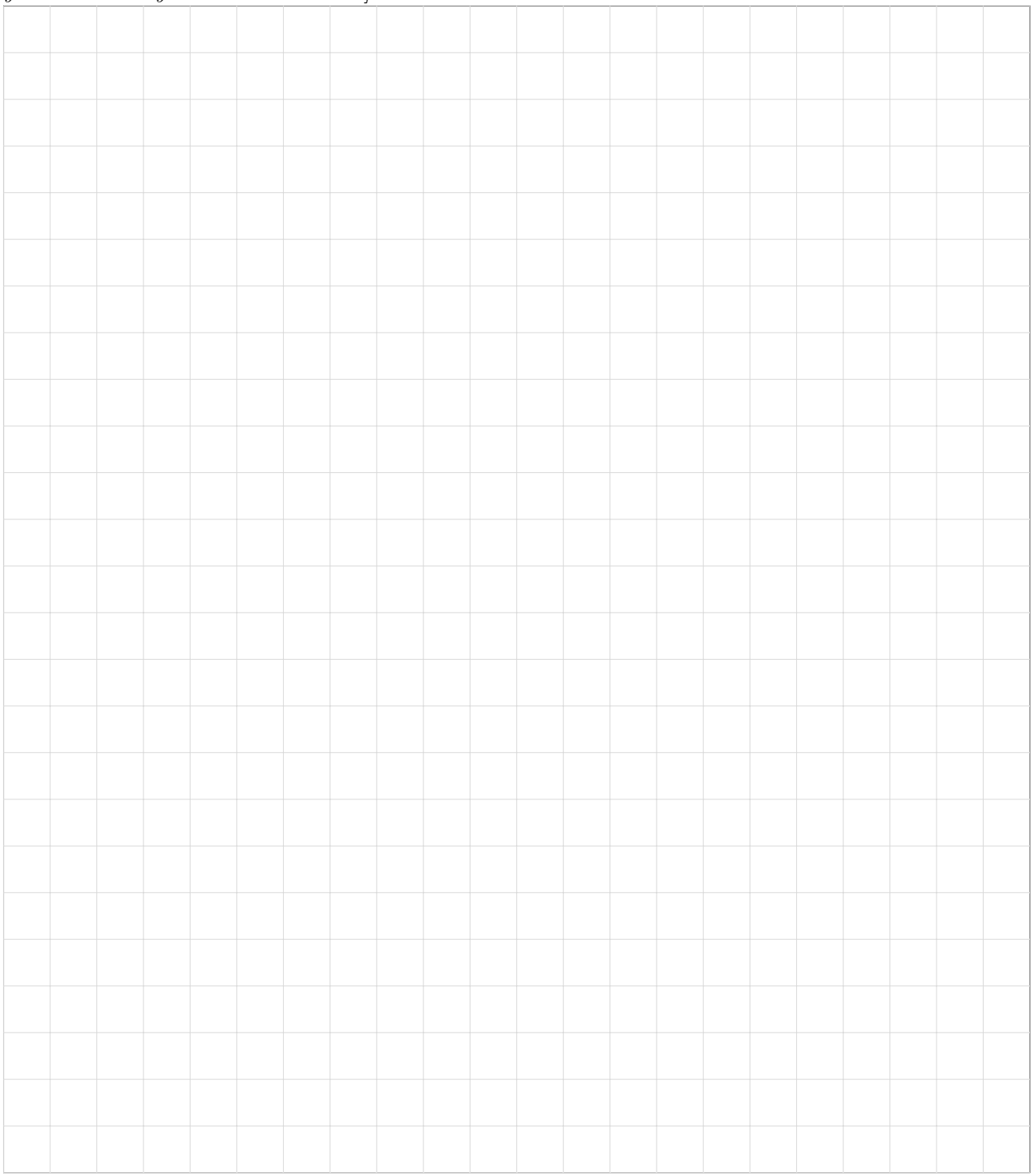
Part (b) [5 MARKS]

Provide a series of test cases (full UnitTest not required, just simple assertions), to test your `listmerge` function from part a. We've given you one test case to get you started.

Code	Explanation

`L1 = None`
`L2 = None`
`r = listmerge(L1, L2)`
`assertequal(r, None)`
	Merging 2 empty lists, should result in an empty list

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]



Short Python function/method descriptions:

You may tear this page off, but if you do so, you must not include any work on it (front or back) that you wish to have marked.

```
__builtins__:
  abs(number) -> number
    Return the absolute value of the given number.
  max(a, b, c, ...) -> value
    With two or more arguments, return the largest argument.
  min(a, b, c, ...) -> value
    With two or more arguments, return the smallest argument.
  isinstance(object, class-or-type-or-tuple) -> bool
    Return whether an object is an instance of a class or of a subclass thereof.
    With a type as second argument, return whether that is the object's type.
  int(x) -> int
    Convert a string or number to an integer, if possible.  A floating point argument
    will be truncated towards zero.
  str(x) -> str
    Convert an object into a string representation.

str:
  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.isalpha() --> bool
    Return True if and only if all characters in S are alphabetic
    and there is at least one character in S.
  S.isdigit() --> bool
    Return True if and only if all characters in S are digits
    and there is at least one character in S.
  S.islower() --> bool
    Return True if and only if all cased characters in S are lowercase
    and there is at least one cased character in S.
  S.isupper() --> bool
    Return True if and only if all cased characters in S are uppercase
    and there is at least one cased character in S.
  S.lower() --> str
    Return a copy of S converted to lowercase.
  S.replace(old, new) -> str
    Return a copy of string S with all occurrences of the string old replaced
    with the string new.
  S.split([sep]) -> list of str
    Return a list of the words in S, using string sep as the separator and
    any whitespace string if sep is not specified.
  S.startswith(prefix) -> bool
    Return True if S starts with the specified prefix and False otherwise.
  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.
```

```
list:
  append(...)
    L.append(object) -- append object to end
  count(...)
    L.count(value) -> integer -- return number of occurrences of value
  index(...)
    L.index(value, [start, [stop]]) -> integer -- return first index of value.
    Raises ValueError if the value is not present.
  insert(...)
    L.insert(index, object) -- insert object before index
  pop(...)
    L.pop([index]) -> item -- remove and return item at index (default last).
    Raises IndexError if list is empty or index is out of range.
  remove(...)
    L.remove(value) -- remove first occurrence of value.
    Raises ValueError if the value is not present.

set:
  pop(...)
    Remove and return an arbitrary set element.
    Raises KeyError if the set is empty.

dict:
  keys(...)
    D.keys() -> a set-like object containing all of D's keys
  get(...)
     D.get(k[,d]) -> returns D[k] if k is in D, otherwise returns d.  d defaults to None.

object:
  __init__(...)
    x.__init__(...) initializes x; called automatically when a new object is created
  __str__(...)
    x.__str__() <==> str(x)
```