COSC122, 2020 Semester 2
INTRODUCTION TO COMPUTER SCIENCE

Assignment 1—Testing times

6% of overall grade
Due 11:55pm on Friday 21 August 2020

1 Overview

1.1 Introduction

This assignment is based around matching quarantined people with their Covid-19 test results, if they have any. It is very loosely based on a Stuff article titled *50 people released early from quarantine without coronavirus test*, an excerpt is given below. Basically the Ministry of Health took a long time to answer the simple question, *how many of the people released from quarantined had been tested for Covid-19?* In this assignment you will look at a couple of ways that could be used to match records from two different databases.

> ...The director-general of health, Dr Ashley Bloomfield, has for the past week faced questions about how many people were released from quarantine without being tested for the virus, after he said health officials would test people twice before release from June 9.
> Last week, two women who travelled from the United Kingdom were confirmed as having Covid-19, after being granted early release from the mandatory two-week quarantine to visit a dying relative.
> They had not been tested before they left quarantine.
> A week later, the ministry had identified 2159 people who finished their two-week quarantine between June 9 and 16, when the problem at the border was discovered.
> It remained unknown how many were tested before leaving quarantine. Bloomfield on Tuesday said health officials continued to follow up and match the information held at quarantine facilities with testing data.
> Partly complicating the ministry's efforts was that officials at quarantine facilities were previously not required to collect the "National Health Index" (NHI) number of each person staying, meaning they could not easily connect people in quarantine to the database of people tested.
> "I still can't tell you that number, this is because the testing was done and no NHI was necessary on the information the hotel has got... I wish I could, what I can say is of course, they all completed the 14 days [quarantine]... We now know many had been tested on departure," Bloomfield said at a Tuesday press conference.
> Prime Minister Jacinda Ardern has expressed her frustration at the situation, saying on Monday it was "unacceptable" the tests were not done.
> "We want to know those who were tested... I do expect to see that information when [Bloomfield] has it. ..."
>
> Stuff, 24 June 2020
> Full article here.

In this assignment the basic task will be to take a list containing the names of quarantined people and generate a list that contains the test results for any people that can be found in the ministry's list

of tested people. The quarantined list will just have names as the quarantine hotels forgot to take down everyone's NHI (National Health Index) number whereas the ministry's list of tested people will contain the NHI, name, and result for each of those people.

The ministry of health had to be very careful to make sure they matched the records for quarantined people with their test records (eg, matching name, date of birth, etc) you will just need to look for matching names. You should be able to see that it's not too hard to expand to more precise matching methods.

1.2 Due date

The official due date is 11:55pm on Friday 21 August 2020.

Students can submit to the normal submission quiz up two days after this (ie, Sun 23 Aug) with no penalty (ie, normal submissions close on Sun 23 Aug). After this point submissions can be made until the drop dead date, which is Fri 28 Aug, via the *late* submission quiz. But *late* submissions incur a 15% absolute penalty, ie, if you submit to the *late* quiz then `your_mark = raw_mark - 15`. Please ask if you are unsure.

1.3 Submission

Submit via the quiz server. The submission page will be open about a week after the quiz is released—to emphasise the point that the quiz won't offer much help, your own testing and the provided unit tests are what you should be using to test your code. The submission quiz will not test your code properly until after submissions close, that is, it won't give you any more information than the provided tests so you can start work straight away! *This assignment is a step up from COSC121 so please make sure you start early so that you have time to understand the requirements and to debug your code.*

1.4 Implementation

All the files you need for this assignment can be found on the quiz server. Do not import any additional standard libraries unless explicitly given permission within the task outline. For each section there is a file with a function for you to fill in. You need to complete these functions, but you can also write other helper functions if you wish—this is recommended for the binary search method. All submitted code needs to pass *pylint* program checking—make sure you leave time for style adjustments when submitting.

1.5 Getting help

The work in this assignment is to be carried out individually, and what you submit needs to be your own work. You must not discuss code-level details with anyone other than the course tutors and lecturers. You are, however, permitted to discuss high-level details of the program with other students. If you get stuck on a programming issue, you are encouraged to ask your tutor or the lecturer for help. You may not copy material from books, the internet, or other students. We will be checking carefully for copied work. If you have a general assignment question or need clarification on how something should work,

please use the class forums on Learn, as this enables everyone to see responses to common issues, but **never** post your own program code to Learn (or make it available publicly on the internet via sites like GitHub)[1]. Remember that the main point of this assignment is for you to exercise what you have learned from lectures and labs, so make sure you do the relevant labs first, so don't cheat yourself of the learning by having someone else write material for you.

2 Looking up results

For each task of the assignment you will be required to write a function that takes a list of test results and a list of quarantined people, and returns a list containing the quarantined peoples' test results, or lack thereof. The functions will need to count the number of times `Name` objects are compared and return this count along with the list of results—as a measure of the work done.

You can assume that there are no duplicate names in either the *tested* list or the *quarantined* list, ie, within either list no name will appear more than once.

The file `tests.py` provides you with a suite of tests to help you check your implemented code before submission. These tests use the Python unit-test framework and you are not expected to fully understand how they work; you just need to know that the tests will check that your code generates the correct list of output, and that the number of comparisons that your code makes is appropriate. For the linear algorithm you should be able to match the exact number of comparisons, but for the binary algorithm the number of comparisons just needs to be in the right ballpark. If you think that you have an implementation that is very close to the test range, and works reliably, you can ask to have the expected number of comparisons range reconsidered. The tests used on the quiz server will be mainly based on these unit-tests, but with a small number of added secret test cases. Therefore passing the unit tests is not a guarantee that full marks will be given (however it's a good indication your code is working as required).

2.1 Provided classes

For this assignment the main class you will need to know about is the `Name` class. A `Name` object is basically a repackaged `str` object, that updates an internal counter every time it is compared with another `Name` object. You will be required to count each `Name` comparison made by your code and our tests will use the internal counter to check that you got it right.

Further information on `Name` objects is given below.

- `my_name = Name(name)` creates a new `Name` object with the given name.
- `print(my_name)` will print the Name object, in the form `<name>` so you don't confuse it with a simple `str` object.
- Every time two `Name` objects are compared the name comparison counter in the StatCounter class will be updated. For example, whenever a comparison such as `name1 < name2` is evaluated the internal counter is incremented. You will see that all the comparison operators work like this, ie, <, >, >=, <=, ==, !=.

[1] Check out section 3 of UC's *Academic Integrity Guidance* document. Making your code available to the whole class is certainly not discouraging others from copying.

- You shouldn't be using any of the double under-scored methods for `Name` objects directly. You should use the normal comparison operators, eg, name1 < name2 will be automatically translated into `name1.__lt__(name2)` in the background so you don't need to call `name1.__lt__(name2)` directly!
- `StatCounter.get_count(NAME_COMPS)` gives the actual number of `Name` comparisons that have been performed. If you read the definition for the `Name` class you will see that `Name` objects update the name comparisons counter each time they are compared. This is for your debugging only and using this in submitted code will cause your code to fail the submission tests. You must use the line `from stats import StatCounter, NAME_COMPS` if you want to check this counter.

The following example code should help you further understand the Name class:

```
>>> from classes import Name
>>> from stats import StatCounter, NAME_COMPS
>>> my_list = []
>>> name1 = Name('Paul')
>>> name1
Name('Paul')
>>> print(name1)
<Paul>
>>> my_list.append(name1)
>>> my_list.append(Name('Tim'))
>>> my_list
[Name('Paul'), Name('Tim')]
>>> name3 = Name('Zheng')
>>> name1 < name3
True
>>> my_list[0] == Name('Paul')
True
>>> StatCounter.get_count(NAME_COMPS) #This is allowed only for testing!
2
>>> my_list[1] >= my_list[0]
True
>>> StatCounter.get_count(NAME_COMPS) #This is allowed only for testing!
3
```

2.2 Provided tools and test data

Test files are given to you in the folder `test_data` to make it easier to test your own code. The test data files are named in the form `test_data-{i}{x}-{j}{y}-{k}-{m}.txt` and contain three lists of data. The first list, referred to as *tested*, contains (NHI, name, result) tuples for people that have been tested for the virus. The second list, referred to as *quarantined*, contains the names of people who are quarantined. The third list, referred to as *results*, contains (name, NHI, result) tuples for all the quarantined people, in the same order as they appear in the *quarantined* list. If a quarantined person isn't in the *tested* list then their NHI and result will be None.

2.2.1 Decoding data file names

The *tested* and *quarantined* data sections start with a line containing the number or records in that section. The third data section doesn't have this number as it will have the same number of records as the *quarantined* list. Note that lines starting with # are commented out and are not read.

- i = number of records in *tested* list
- x = 'i' if *tested* is sorted by NHI or 'n' if *tested* is sorted by name.

- j = number of records in *quarantined*
- y = 'n' if *quarantined* is sorted by name or 'r' if it's unsorted/randomly ordered.
- k = number of *quarantined* people that have a result, ie, can be found in the *tested* list.
- m = the random seed used to generate the test file. We may generate different random files for the quiz server tests.

For example, a file named `test_data-5i-5n-2-a.txt` will contain: 5 records for tested people that are sorted by NHI, 5 names of quarantined people that are sorted by name, 5 results records for the quarantined people, and two of the quarantined people will have have results in the tested list (in the example below Cissiee and Jon are the two).

```
# Number tested and their details:
5
2120000,Zorina Latin,True
2120002,Selime Sziladi,False
2120003,Vallipuram Kortekaas,True
2120005,Jon Klaudt,False
2120007,Cissiee Bednar,True
# Number quarantined and their names:
5
Andrzej Challice
Cissiee Bednar
Jon Klaudt
Meris Dendi
Rheal Wolfenbarger
# Expected result details for quarantined people:
Andrzej Challice,None,None
Cissiee Bednar,2120007,True
Jon Klaudt,2120005,False
Meris Dendi,None,None
Rheal Wolfenbarger,None,None
```

You will, of course, need to use files where the tested records are sorted by name when you are using a binary search for names—otherwise the binary search won't work! When tested records are sorted by NHI the names will effectively be in random order (as seen in the `test_data-10i-5n-2-a.txt` example above).

2.2.2 Reading data files

The `tools.py` module contains functions for reading data from test files and for making simple lists of `Name` objects, etc...The most useful function in the `tools` module is `read_test_data(filename)`, which reads the contents of the test file and returns a tuple containing the *tested, quarantined* and expected *results* lists respectively.

Suppose `test_data-2i-2r-1-a.txt` contains the following data, then the shell code below should give you an idea of how the data is read.

```
#Number tested and their details:
2
2120000,Sabina Starkes,False
2120001,Filippa Mau,False
#Number quarantined and their names:
2
Albert Willison
Sabina Starkes
#Result details for quarantined people:
Albert Willison,None,None
Sabina Starkes,2120000,False
```

Example data loading using `read_test_data`:

```
>>> import tools
>>> filename = "test_data/test_data-2i-2r-1-a.txt"
>>> tested, quarantined, expected_results = tools.read_test_data(filename)
>>> tested
[(2120000, Name('Sabina Starkes'), False), (2120001, Name('Filippa Mau'), False)]
>>> quarantined
[Name('Albert Willison'), Name('Sabina Starkes')]
>>> expected_results
[(Name('Albert Willison'), None, None), (Name('Sabina Starkes'), 2120000, False)]
>>> print(quarantined[0])
<Albert Willison>
```

2.2.3 Making your own tested/quarantined lists

The `make_name_list` function in the `tools` module can be used to make lists of `Name` objects from simple lists of strings or using the letters in a string. The `make_tested_list` function will make a list of `(nhi, Name, result)` tuples from a simple list of strings, or using the letters in a string. `make_tested_list` will generate the NHI numbers and test results for you and there are options for list sorting and the result you want everyone to have. Check out the function definitions and docstrings for more details. An example of each is given below.

```
>>> import tools
>>> my_quarantined = tools.make_name_list(['Bob', 'Abba', 'Faba'])
>>> my_quarantined
[Name('Bob'), Name('Abba'), Name('Faba')]
>>> my_tested = tools.make_tested_list(['Bingle', 'Zabba', 'Faba'], sort_order='name')
>>> my_tested
[(1, Name('Bingle'), True), (3, Name('Faba'), True), (2, Name('Zabba'), True)]
```

You will be able to use these functions to quickly generate simple lists that you can send to your functions for processing. This will help you verify and debug your code by allowing you to compare your hand-cranked results with the results returned by your function. For example, you could generate the lists above and then call your function to see if it returns the result you expect. If not then you need to work out whether your hand-cranking was wrong or your function was wrong...

2.3 Provided tests

The `tests.py` provides a number of tests to perform on your code. Running the file will cause all tests to be carried out. Each test has a name indicating what test data it is using, what it is testing for, and a contained class indicating which algorithm is being tested. In the case of a test case failing, the test case will print which assertion failed or what exception was thrown. The `all_tests_suite()` function has a number of lines commented out indicating that the commented out tests will be skipped; un-comment these lines out as you progress through the assignment tasks to run subsequent tests.

To get all the test results in a more manageable form in Wing101 you should make sure that *Enable debugging* is turned off in the options for the Python shell. You can get to the options by clicking on the Options drop down at the top right of the Shell window.

In addition to the provided tests you are expected to do your own testing of your code. This should include testing the trivial cases such as empty parameters and parameters of differing sizes. You will,

of course, want to start by testing with very small lists, eg, with zero, one or two items in each list. This will allow you to check your answers by hand.

NOTE: *The tests that are provided in `tests.py` aren't good for debugging!*

3 Tasks

3.1 Finding results using linear/sequential search [40 Marks]

This task requires you to complete the `linear_result_finder` function in `linear_module.py` using a sequential/linear search when looking up names in the `tested` list. This approach is designed to model the worst case scenario and isn't going to be very efficient, but it gives a good baseline. You should start with an empty list representing the results found so far. Go through each `Name` in `quarantined` and sequentially search `tested` to try to find that `Name`. If a match is found add a (Name, nhi, result) tuple to the `results` list and stop searching the rest of the `tested` list for that `Name` (as it can be assumed that no name will appear twice in the `tested` list). If a match isn't found then add a tuple with `(the_name, None, None)` to the results list as you don't have a NHI number or result for that person. You cannot assume that the `Names` will be in any particular order in either `tested` or `quarantined`. The returned list should be given in the same order that the names appear in the `quarantined` list. Your function should return a tuple containing the results list and the number of `Name` comparisons the function made, ie, a tuple in the form `(your_results_list, comparisons_used)`.

```
>>> import tools
>>> from linear_module import linear_result_finder
>>> filename = "test_data/test_data-2i-2r-1-a.txt"
>>> tested, quarantined, expected_results = tools.read_test_data(filename)
>>> tested
[(2120000, Name('Sabina Starkes'), False), (2120001, Name('Filippa Mau'), False)]
>>> quarantined
[Name('Albert Willison'), Name('Sabina Starkes')]
>>> expected_results
[(Name('Albert Willison'), None, None), (Name('Sabina Starkes'), 2120000, False)]
>>> results, comparisons = linear_result_finder(tested, quarantined)
>>> results
[(Name('Albert Willison'), None, None), (Name('Sabina Starkes'), 2120000, False)]
>>> # Think about why comparisons should be 3
>>> comparisons
3
```

3.1.1 Notes

- We recommend testing your function with very small lists first, eg, start out with one item in each list, then try with one in the first list and two in the second, etc, ...
- The provided tests are really just a final check — they won't help you very much when debugging your code.
- Your function shouldn't mutate the lists it is given in any way, eg, don't pop anything off the `quarantined` or `tested` list and don't insert anything into them either. You will, of course, need to append things to the results and this is fine.

3.2 Finding results using binary search [50 Marks]

This task requires you to complete the `binary_result_finder` file `binary_module.py`.

The `tested` list will be in alphabetical order by name (ie, the <= operator is true between any successive names in `tested`) but you may not assume anything about the order of `quarantined`. Given that the `tested` list is sorted you will be able to employ a binary search and get considerably better performance than the sequential search.

Again, you should start with an empty list representing the results for quarantined people. Then you should go through each `Name` in `quarantined` and perform binary search on the `tested` list to try find that `Name`. If a match is found, append a tuple containing `(Name, nhi, result)` to the results list. If a match isn't found then add a tuple with `(Name, None, None)` to the results list as you don't have a NHI number or test result for that person. The returned list should be given in the same order that the names appear in the `quarantined` list.

To get full marks in this question you must minimise the average number of comparisons made—assuming that not many quarantined people have been tested. Therefore, you must implement your binary search in such a way that only uses only one `Name` comparison per halving of the search area. That is, your search should use the equality (==) operator at most once per name that is being looked up. Basically, you shouldn't be comparing Name objects for equality (ie, with ==) inside the while loop of the binary search.

Your `binary_result_finder` should return the results list and the number of comparisons as a tuple in the same form as the linear function, ie, in the form `(your_results_list, comparisons_used)`. As for the linear function, your binary function shouldn't mutate the lists it is given in any way, eg, don't pop anything off the `quarantined` or `tested` list and don't insert anything into them either. You will, of course, need to append things to your results list, and this is fine.

Warning: fast binary search is difficult! Set aside a lot of time to get this right, and don't be put off if it doesn't work correctly straight away. You should test your function using some simple lists, eg, set the *tested* list to have names 'b', 'c', 'd', 'e', 'f' then make a *quarantined* list containing just 'b' and check that 'b' is reported in the results list. Then try with 'c', etc, until your are convinced that your function will find any name in the *tested* list. Then, try some names that aren't in the *tested* list, eg, 'a' or 'z'.

3.3 Analysis questions [10 Marks]

The submission quiz will also ask you to answer the following questions. For each question in this section you can assume there are no duplicates in `tested` or `quarantined`. You should also answer with respect to the overall task in each case, ie, with respect to generating the complete list of results for the quarantined people which is what each of your result finder functions is doing. For example, the first question below is basically asking what is the worst case big-O complexity for the total number of name comparisons used by your `linear_result_finder` function.

- What is the worst case big-O complexity for the number of comparisons made by the linear/sequential method given there are n items in the *tested* list and m items in the *quarantined* list. Explain how this would come about and give a small example of worst case input.
- What is the best case big-O complexity for the number of comparisons made by the linear/sequential method given there are n items in the *tested* list and m items in the *quarantined* list, AND $m < n$. Explain how this would come about and give a small example of best case input.

- Give the equation for the number of comparisons used by the linear/sequential method given there are n items in the *tested* list, n items in the *quarantined* list AND that all the names in the *quarantined* list are also in the *tested* list. NOTE: you are giving an equation for the exact number of comparisons made, NOT the big-O complexity.
- What is the worst case big-O complexity for the number of comparisons made by the binary search method given there are n items in the *tested* list and m items in the *quarantined* list. Explain how this would come about and give a small example of worst case input.
- What is the best case big-O complexity for the number of comparisons made by the binary search method given there are n items in the *tested* list and m items in the *quarantined* list. Explain how this would come about and give a small example of best case input.