COSC122 2025 Semester 2
INTRODUCTION TO ALGORITHMS

Assignment 1: Spotting Stolen Cars

6% of overall grade
Due : See *Due Dates* section for due dates

Contents

1 Assignment Overview

1.1 Introduction

Thanks to advances in computer and network technologies we live in an age of increasingly pervasive surveillance. As with most things, there are pros and cons to such widespread surveillance but we will leave the ethical discussion to other courses. This assignment isn't going to take sides in the debate—our aim is simply to give you some insight into how data from such surveillance systems might be used efficiently to meet whatever good/evil ends that one might have for the data.

1.2 Due dates

The due dates for the two parts of this assignment are as follows:

- Part A is due at 11:55pm on Friday 15 August 2025 (ie, end of week 5)
- Part B is due at 11:55pm on Friday 22 August 2025 (ie, end of week 6)

The deadline for Part A cannot be extended except in cases of illness etc. (this usually requires a medical certificate or equivalent). However, Part B may be submitted up to one week late—you will incur a 15% absolute penalty (in other words, your mark will be the raw mark less 15 marks, given the assignment is marked out of 100). Assignments cannot be submitted after these deadlines unless you've organised an extension with the course supervisor because of illness etc.

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 COSC121/131 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 program style 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. Do not use Generative AI tools to write code directly—use AIs only to guide your understanding. 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 Spotting stolen cars

In this assignment you will be working with the data from a roadside camera based Automatic Number Plate Recogniser (ANPR). You will be carrying out a simplified version of what might be done in reality—so you can focus on a basic implementation. You will be processing a stream of number plates from a single ANPR and making a list of the seen number plates that appear in a list of stolen car plates.

2.1 Task Overview

This assignment takes two different approaches to solving the same problem. Each task requires you to make a function that:

- Takes two lists of **unique** `NumberPlate` objects:
  1. `sighted_plates` – plates of vehicles sighted by a number plate camera
  2. `stolen_plates` – plates of stolen vehicles from a central database
- Goes through each *sighted* plate and searches for a matching plate in the *stolen* plates list, recording how many comparisons occurred between `NumberPlate` objects.
- Returns a list of plates *sighted* by the camera that belong to *stolen* cars, along with the number of comparisons used.

You will need to use `NumberPlate` objects, located in `classes.py`. You do not have to implement these data structures, you will just have to interact with them. Details of these data structures are in the Provided Classes section below.

[1] Check out section 2c, part 10 of UC's *Misconduct Procedures* document. The sharing or publication of assessment answers is not allowed.

You are expected to test your code using **both** your own tests and the provided tests. More information about testing your code is in the Testing your Code section.

2.2 Provided classes

2.2.1 NumberPlate

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

Further information on `NumberPlate` objects is given below.

- Every time two `NumberPlate` objects are compared the name comparison counter in the StatCounter class will be updated. For example, whenever a comparison such as `plate1 < plate2` is evaluated the internal counter is incremented. You will see that all the comparison operators work like this, ie, <, >, >=, <=, ==, !=.
- You shouldn't be using any of the double under-scored methods for `NumberPlate` objects directly. You should use the normal comparison operators, eg, `plate1 < plate2` will be automatically translated into `plate1.__lt__(plate2)` in the background so you don't need to call `plate1.__lt__(plate2)` directly!
- The `StatCounter` class holds all the comparison information that you might need to check your code against. We also use it to verify that you do perform the correct number of comparisons.
- `StatCounter.get_comparisons()` returns how many comparisons have been made between `NumberPlate` objects. This is for your debugging only and using this in submitted code will cause your code to fail the submission tests.
- You should be counting how many `NumberPlate` comparisons your functions do—in a similar fashion to what you did in Lab 1. This count should give the same value as StatCounter, which will be a check that you're counting comparisons correctly.

2.2.2 Example Code

The following example code should help you further understand the `NumberPlate` and `StatCounter` classes:

```
>>> from classes import NumberPlate
>>> from stats import StatCounter
>>> my_numberplate1 = NumberPlate('ABC123') # Create two NumberPlate instances
>>> my_numberplate2 = NumberPlate('XYZ789')
>>> my_numberplate1 # Looking at a single plate object
NumberPlate('ABC123')
>>> my_numberplate1 < my_numberplate2 # Comparing two NumberPlate objects
True
>>> StatCounter.get_comparisons() #This is allowed only for testing!
1
>>> my_list = [] # Make a list to put our plates in
>>> my_list.append(my_numberplate1)
>>> my_list.append(my_numberplate2)
>>> my_list
[NumberPlate('ABC123'), NumberPlate('XYZ789')]
>>> my_list[0] == NumberPlate('ABC123')
True
>>> StatCounter.get_comparisons() # See actual comparison count,this is allowed only for testing!
2
>>> my_list[0] > my_list[1]
False
>>> StatCounter.get_comparisons() #This is allowed only for testing!
3
```

Important: `StatCounter` will not work on the quiz server! You must do your own comparison counting to get marks.

3 Tasks [100 Marks Total]

3.1 Part A: Sequential/Linear search [40 Marks]

This task requires you to complete the `linear_stolen_plate_finder` function in the `linear_finder.py` module. This first method isn't going to be very efficient, but it's a starting point!

- You should start with an empty `list` representing the stolen number plates that have been sighted by the camera, which will collect the matches found.
- Go through each `NumberPlate` in the `sighted_plates` and sequentially search the `stolen_plates` to try find the `NumberPlate`.
- If a match is found add that number plate to the stolen plates sighted so far and stop searching the rest of the `stolen_plates` for that `NumberPlate` (it can be assumed that no number plate occurs twice in the stolen list).
- Your function should return a `tuple` containing the populated `list` of stolen number plates that were sighted `NumberPlates`, and the number of `NumberPlate` comparisons that the function made (see the example return line in the supplied code).

- The returned `list` of `NumberPlates` should be in the same order that the plates appear in `sighted_plates`.

3.1.1 General Notes

- You cannot assume that either the `stolen_plates` or the `sighted_plates` will be in any particular order.
- Think about when it's OK for your function to stop scanning through the *stolen* list before reaching the end of that list.
- You can assume that both lists provided to your function will only contain unique elements. You should use this knowledge to finish scanning the *sighted* list earlier in some cases.
- We recommend writing your own tests with very small lists first, eg, start out with one plate in each list, then try using one in the first list and two in the second list, etc, ...
- The provided tests are really just a final check — they won't help you very much when debugging your code. Remember you can do some initial test with simple lists of numbers and then switch to testing with lists of `NumberPlate` objects.
- Your function shouldn't mutate the lists it is given in any way, eg, don't pop anything off the `stolen_plates` or `sighted_plates` list and don't insert anything into them either. You will, of course, need to append things to the results list and this is fine.
- You do not necessarily have to do all of your testing using `NumberPlates`. It may be simpler to use integers as these can be quickly written and are easy to understand. An example of testing using integers is show below:

```
>>> from linear_finder import linear_stolen_plate_finder
>>> sighted_plates = [2]
>>> stolen_plates = [2, 3, 4, 5, 6]
>>> linear_stolen_plate_finder(stolen_plates, sighted_plates)
([2], 1)
```

Important: Part A is due at 11:55pm on Friday 15 August 2025 (ie, end of week 5)

3.2 Part A: Theory Questions [6 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 `stolen_plates` or `sighted_plates`. You should also answer with respect to the overall task in each case, ie, with respect to generating the complete list of sighted number plates that were stolen which is what each of your plate 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 number plate comparisons used by your `linear_stolen_plate_finder` function.

1. What is the worst case big-O complexity for the number of comparisons made by the linear/sequential method (to identify all stolen cars sighted by the camera) given there are n items in the *stolen* list and m items in the *sighted* list? Note, you can include both m and n in your answer. Explain how this would come about and give a small example of worst case input.

2. What is the best case big-O complexity for the number of comparisons made by the linear/sequential method (to identify all stolen cars sighted by the camera) given there are n items in the *stolen* list and m items in the *sighted* list, AND $n > m$? Note, you can include both m and n in your answer. Explain how this would come about and give a small example of best case input.

3. Give the equation for the number of comparisons used by the linear/sequential method (to identify all stolen cars sighted by the camera) given there are n items in the *stolen* list, n items in the *sighted* list AND that all the number plates in the *sighted* list are also in the *stolen* list. NOTE: you are giving an equation for the exact number of comparisons made, NOT the big-O complexity.

3.3 Part B: Binary Search [50 Marks]

In the first task, we made no guarantees about the order of the number plates. If we guarantee the *stolen* list is sorted (in lexicographic order) can we make a more efficient program?

This task requires you to complete the `binary_stolen_plate_finder` function in the `binary_finder.py` module. It should generate the list of *stolen* number plates that were *sighted* by the camera, using binary search to find each `NumberPlate` from the `sighted_plates` in the `stolen_plates`. You can assume the *stolen* list will be sorted.

- Again, start with an empty list to accumulate the stolen number plates that have been sighted by the camera.
- Go through each `NumberPlate` in the `sighted_plates` and perform a binary search, trying to find that `NumberPlate` in the list of stolen plates.
- A helper function that does a binary search for an item in a sorted list and returns True/-False along with the number of comparisons used could be helpful here.
- If a match is found add that number plate to the stolen plates sighted so far.
- Your function should return a `tuple` containing the populated `list` of stolen number plates that were sighted `NumberPlates`, and the number of `NumberPlate` comparisons that the function made (see the example return line in the supplied code).
- The returned `list` of `NumberPlates` should be in the same order that the plates appear in `sighted_plates`.

3.3.1 General Notes

- **Important:** Your binary search must only do one comparison per loop when it is searching the stolen list and should only compare number plates for equality once. This approach will basically narrow the search down to one number plate and then check if that is the one being searched for. This method will be discussed in lectures and labs; it is *not* the commonly given form of a binary search algorithm (which makes two comparisons per loop).
- Your `binary_stolen_plate_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 `sighted_plates` or `stolen_plates` 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 *stolen_plates* list to contain the numbers 2, 3, 4, 5, 6 then make a *sighted_plates* list containing just 2 and check that 2 is reported in the results list. Then try with 3, etc, until you are convinced that your function will find any number in the *stolen_plates* list. Then, try some numbers that aren't in the *stolen_plates* list, eg, 1 or 26. The example code below shows how this testing could be done:

```
>>> from binary_finder import binary_stolen_plate_finder
>>> sighted_plates = [2]
>>> stolen_plates = [2, 3, 4, 5, 6]
>>> binary_stolen_plate_finder(stolen_plates, sighted_plates)
([2], 3)
>>> sighted_plates = [3]
>>> binary_stolen_plate_finder(stolen_plates, sighted_plates)
([3], 3)
>>> sighted_plates = [1]
>>> binary_stolen_plate_finder(stolen_plates, sighted_plates)
([], 3)
```

Important: Binary search can be difficult to implement correctly. Be sure to leave sufficient time to solve this task.

3.4 Part B: Theory Questions [4 Marks]

1. What is the worst case big-O complexity for the number of comparisons made by the binary search method (to identify all stolen cars sighted by the camera) given there are *n* items in the *stolen* list and *m* items in the *sighted* list? Explain how this would come about and give a small example of worst case input.
2. 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 *stolen* list and *m* items in the *sighted* list? Explain how this would come about and give a small example of best case input.

4 Testing Your Code

There are two ways to test your code for this assignment (you are required to do **both**):

- Writing your own tests – useful for debuging and testing edge cases
  - This could also include loading and using some of the provided test data files
- Using the provided tests in `tests.py`– a final check before you submit

The tests used on the quiz server will be mainly based on the tests in `tests.py`, but a small number of secret test cases will be added, so passing the unit tests is not a guarantee that full marks will be given for that question (it is however a strong indication your code is correct; and importantly, failing the test indicates that something fundamental may be wrong). Be sure to test various edge cases such as when an input list is empty or when no number plates are sighted etc...

4.1 Writing your Own Tests

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 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.

4.1.1 Provided Data for Testing

The `utilities.py` module contains functions for reading test data from test files.

Test files are in the folder `test_data` to make it easier to test your own code. The test data files are named `stolen[s]-sighted-matches-seed.txt`, where:

- *stolen* = the number of stolen plates in the file
- *s* is optional and indicates that they are in sorted order
- *sighted* = the number of plates in the list that were sighted by the camera
- *matches* = the list of expected plates (in the order your functions should return them) which is used for testing
- *seed* = the seed value that was used when generating the test file.

For example `10-20-5-a.txt` was generated with `a` as the random key and contains 10 stolen plates and 20 sighted plates, where 5 of the sighted plates were in the stolen list. If the file name was `10s-20-5-a.txt` then the stolen plates would be provided in sorted order—obviously useful for testing your binary search function.

You should open some of the test data files and make sure you understand the format—remember to open them in Wing or a suitably smart text editor. If you are using windows

then try Notepad++ because the normal Notepad is pretty horrible. If you are using Linux (or a Mac) then almost any text editor will do.

The test files are generated in a quasi-random fashion and the seed suffix indicates the random seed that was used when generating the data—you don't need to worry about this. But, you do need to worry about the fact that we can easily generate different random files to test with.

4.1.2 Provided Tools for Testing

The `read_dataset` function in `utilities.py` reads the contents of a test file and returns three lists. The lists all contain `NumberPlate` objects. The first list contains the number plates of stolen vehicles, the second contains all the number plates that were seen by the camera and the third contains the stolen number plates that were seen by the camera (this third list *is* the plates that your functions should return).

The following example shows how to use the `utilities` module:

```
>>> from utilities import read_dataset
>>> filename = './test_data/5-5-2-a.txt'
>>> stolen, sighted, matches = read_dataset(filename)
>>> print(stolen)
['IQ1998', 'FS1860', 'JI9400', 'NN3705', 'AS0734']
>>> print(sighted)
['AR3546', 'QT0780', 'NN3705', 'PB2873', 'AS0734']
>>> print(matches)
['NN3705', 'AS0734']
>>> type(stolen[0])
<class 'classes.NumberPlate'>
>>> filename = './test_data/5s-5-2-a.txt'
>>> stolen, sighted, matches = read_dataset(filename)
>>> print(stolen) # the same list as before but sorted
['AS0734', 'FS1860', 'IQ1998', 'JI9400', 'NN3705']
```

4.2 Provided tests

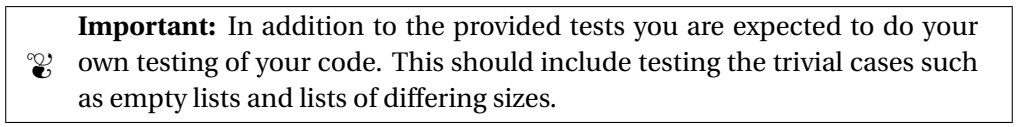
Off-line tests are available in the `tests.py` module:

- These tests are **not** very helpful for debugging–do your own smaller tests using hand workable examples when debugging!
- You should use them to check your implemented code before submission: the submission quiz *won't* fully mark your code until submissions close.
- The provided tests use the Python `unittest` framework.
  - You are not expected to know how these tests work, or to write your own `unittests`
  - You just need to know that the tests will check that your code finds the list of stolen number plates that were sighted, and that the number of comparisons that your code makes is appropriate.

4.2.1 Running the Provided Tests

The `tests.py` provides a number of tests to perform on your code:

- The `all_tests_suite()` function has a number of lines commented out:
  - The commented out tests will be skipped.
  - Uncomment these lines out as you progress through the assignment tasks to run subsequent tests.
- Running the `tests.py` file will cause all uncommented tests to be carried out.
  - You should run `tests.py` directly, rather than importing the tests into `linear_finder.py` or `binary_finder.py`.
- 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.
- 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.



— Have fun —