FIT2004 S2/2020: Assignment 1

DEADLINE: Friday 28th August 2020 23:55:00 AEST

LATE SUBMISSION PENALTY: 10% penalty per day. Submissions more than 5 days late are generally not accepted. The number of days late is rounded up, e.g. 5 hours late means 1 day late, 27 hours late is 2 days late. For special consideration, please visit this page:
`https://www.monash.edu/connect/forms/modules/course/special-consideration`
and fill out the appropriate form. **Do not** contact the unit directly, as we cannot grant special consideration unless you have used the online form.

PROGRAMMING CRITERIA: It is required that you implement this exercise strictly using the **Python programming language** (version should not be earlier than 3.5). This practical work will be marked on the time complexity, space complexity and functionality of your program, and your documentation.
Your program will be tested using automated test scripts. It is therefore critically important that you name your files and functions as specified in this document. If you do not, it will make your submission difficult to mark, and you will be penalised.

SUBMISSION REQUIREMENT: You will submit a zipped file (named `studentId_A1.zip`, e.g. if your student id is 12345678, the name of zipped file must be `12345678_A1.zip`). It should contain a single python file, `assignment1.py`, and a single pdf file, `explanation.pdf`.

PLAGIARISM: The assignments will be checked for plagiarism using an advanced plagiarism detector. Last year, many students were detected by the plagiarism detector and almost all got zero mark for the assignment and, as a result, many failed the unit. "Helping" others is NOT ACCEPTED. Please do not share your solutions partially or completely to others. If someone asks you for help, ask them to visit a consultation for help.

Learning Outcomes

This assignment achieves the Learning Outcomes of:

- 1) Analyse general problem solving strategies and algorithmic paradigms, and apply them to solving new problems;
- 2) Prove correctness of programs, analyse their space and time complexities;
- 4) Develop and implement algorithms to solve computational problems.

In addition, you will develop the following employability skills:

- Text comprehension
- Designing test cases
- Ability to follow specifications precisely

Assignment timeline

In order to be successful in this assessment, the following steps are provided as a **suggestion**. This is an approach which will be useful to you both in future units, and in industry.

Planning

1. Read the assignment specification as soon as possible and write out a list of questions you have about it.
2. Watch the video on Moodle about numerical bases.
3. Clarify these questions. You can go to a consultation, talk to your tutor, discuss the tasks with friends or ask in the forums.
4. As soon as possible, start thinking about the problems in the assignment.
    - It is strongly recommended that you **do not** write code until you have a solid feeling for how the problem works and how you will solve it.
5. Writing down small examples and solving them by hand is an excellent tool for coming to a better understanding of the problem.
    - As you are doing this, you will also get a feel for the kinds of edge cases your code will have to deal with.
6. Write down a high level description of the algorithm you will use.
7. Determine the complexity of your algorithm idea, ensuring it meets the requirements.

Implementing

1. Think of test cases that you can use to check if your algorithm works.
    - Use the edge cases you found during the previous phase to inspire your test cases.
    - It is also a good idea to generate large random test cases.
    - Sharing test cases **is** allowed, as it is not helping solve the assignment.
2. Code up your algorithm, (remember decomposition and comments) and test it on the tests you have thought of.
3. Try to break your code. Think of what kinds of inputs you could be presented with which your code might not be able to handle.
    - Large inputs
    - Small inputs
    - Inputs with strange properties
    - What if everything is the same?
    - What if everything is different?
    - etc...

Before submission

- Make sure that the input/output format of your code matches the specification.
- Make sure your filenames match the specification.
- Make sure your functions are named correctly and take the correct inputs.
- Make sure you zip your files correctly

Documentation (3 marks)

For this assignment (and all assignments in this unit) you are required to document and comment your code appropriately. This documentation/commenting must consist of (but is not limited to)

- For each function, high level description of that function. This should be a one or two sentence explanation of what this function does. One good way of presenting this information is by specifying what the input to the function is, and what output the function produces (if appropriate)
- For each function, the Big-O complexity of that function, in terms of the input. Make sure you specify what the variables involved in your complexity refer to. Remember that the complexity of a function includes the complexity of any function calls it makes.
- Within functions, comments where appropriate. Generally speaking, you would comment complicated lines of code (which you should try to minimise) or a large block of code which performs a clear and distinct task (often blocks like this are good candidates to be their own functions!).

1 Numerical radix sort (9 marks)

Consider radix sort for integers. As discussed in lectures, it is possible to vary the base used for radix sort. If you are unsure what this means please rewatch the second half of lecture 2, where this concept is discussed.

In this task, you will be given a list of integers to sort, using a specific base. To do this you will write the function `numerical_radix_sort(num_list, b)`

1.1 Input

`num_list` is a list of positive integers. `b` is an integer (2 or more) which is the base you need to use to perform the radix sort. If you do not use the base `b` for your sort, you may receive no marks for this task.

Note: There is a video on Moodle in the **Assignments** section, which attempts to explain numerical bases and how they relate to this assignment. Please watch the video, and discuss with a tutor in consultation if it is unclear.

Note: Some large bases will cause a memory error. There is no way around this, and you will **not** lose marks for this memory error. The algorithm needs to create an array of the size of the base, and if the base is larger than the memory the program has access to, this allocation simply cannot be done.

1.2 Output

`numerical_radix_sort` will return a list of integers in ascending numerical order.

Example:
Numbers in this example are **written** in base 10, for clarity.

Input:

```
num_list = [123,
312,
1000,
76,
594,
100]
b = 3
```

Output:

```
[76,
100,
123,
312,
594,
1000]
```

Note that for a given list, the output is always the same regardless of `b`, since the base chosen does not affect the numerical order of the numbers.

1.3 Complexity

`numerical_radix_sort` should run in $O((n + b) * log_b M)$ time where

- n is the length of `num_list`
- b is the base
- M is the numerical value of the greatest element in `num_list`

2 Optimising radix sort (9 marks)

Now we consider the question of **choosing** a base for radix sorting numbers. To tackle this problem, you will write a function `test_bases(num_list)`

2.1 Input

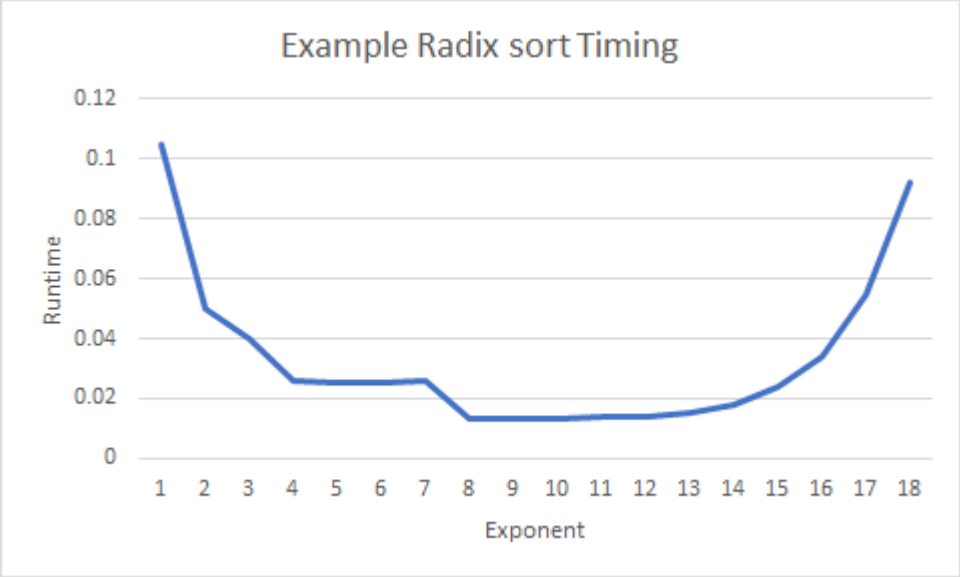
`num_list` is a list of positive integers. To test your program, it is recommended to generate a list of random integers of varying size, and of a significant length.

2.2 Output

`test_bases` will return a list of tuples as output. Each tuple consists of two elements. The first element is an integer, which represents a base. The second element is a number representing the time, in seconds, which it took to sort the input list using that base.

The bases your function should test are the powers of 2, starting with an exponent of 1 (i.e 2^1), and ending once the times become an order of magnitude larger than the time for 2^1. Once the times get much larger than this, the graphs become very flattened and hard to read.

A graph of your times, with the exponent on the x axis and time on the y-axis, should be trough-shaped. In other words, as the exponent increases, time should decrease for a while and then start to increase again.



2.3 Explanation

Now that you have written this function, you will use it to observe the runtimes of radix sort under various circumstances. Please include the line "`random.seed(0)`" in your code before you generate the lists below (do some Googling if you are not sure what this line of code does). Consider the following lists:

1. `data1 = [random.randint(0,2**8-1) for _ in range(10**4)]`
2. `data2 = [random.randint(0,2**8-1) for _ in range(10**5)]`
3. `data3 = [random.randint(0,2**(2**10)-1) for _ in range(10)]`
4. `data4 = [random.randint(0,2**(2**10)-1) for _ in range(20)]`

Note: For `data3` and `data4`, you may need a lower maximum base as opposed `data1` and `data2`. On my machine, reasonable maximum bases were around 2^{21} for the first two lists, and 2^{12} for the second two. These values will vary from machine to machine.

Run `test_bases` using these lists as the input, and graph the results (you may use a python library, excel, an online graphing tool etc.). In `explanation.pdf`,

- Include your graphs
- Identify interesting features of the graphs and explain why they occur
- Contrast graphs `data1` and `data2`, and explain the differences/similarities
- Contrast the graphs from `data3` and `data4`, and explain the differences/similarities
- Contrast the combined `data1` & `data2` graphs with the combined `data3` & `data4` graphs, and explain the differences/similarities

Be sure to base your explanations on your understanding of the radix sort algorithm and its complexity. Overall you should have at least 5 points of interest to discuss.

2.4 Complexity

The overall complexity for this task is not important, but the complexity for each individual radix sort should be the same as task 1 (in fact, you should be reusing your code from task 1 as much as possible).

3 Scrabble helper (9 marks)

In this task you will write a function which determines which words can be made using a given set of letters. To do this, you will write a function `scrabble_helper(word_list, char_set_list)` which takes as input two lists, and determines, for each collection of characters in the second list, which words from the first list can be made using those characters. In other words, if we think of the collections of characters in `char_set_list` as hands of scrabble tiles, we want to know which words we can make using **all** the tiles in that hand.

3.1 Input

Both inputs to this task are lists of strings. Each string will consist only of lowercase English alphabet characters (a-z), and the strings may be any length. They also may not be real words.

Example:

```
word_list = [pots, pot, stop, post, stops, stoop, sop, pos]
char_set_list = [sopt, otp, ppsto]
```

3.2 Output

For each string **s** in `char_set_list`, your function should determine which strings from `word_list` are anagrams of **s**. In other words, which strings from `word_list` consist of exactly the same letters (i.e. the same number of each letter) as **s**.

`char_set_list` should return a list of lists, where the i^{th} interior list contains the anagrams, **in alphabetical order**, of `char_set_list[i]`

Example:
Given the example above as input, `scrabble_helper` returns:

```
[[post, pots, stop], [pot], []]
```

Note that since there were no words in `word_list` which are anagrams of `"ppsto"`, the third interior list is empty.

3.3 Complexity

`scrabble_helper` must run in $O(nM + qMlog(n))$ time, where

- n is the length of `word_list`
- M is the number of characters in the longest word in `word_list`
- q is the length of `char_set_list`

Warning

For all assignments in this unit, you may **not** use python **dictionaries** or **sets**. This is because the complexity requirements for the assignment are all deterministic worst case requirements, and dictionaries/sets are based on hash tables, for which it is difficult to determine the deterministic worst case behaviour.

Please ensure that you carefully check the complexity of each inbuilt python function and data structure that you use, as many of them make the complexities of your algorithms worse. Common examples which cause students to lose marks are **list slicing**, inserting or deleting elements **in the middle or front of a list** (linear time), using the `in` keyword to **check for membership** of an iterable (linear time), or building a string using **repeated concatenation** of characters. Note that use of these functions/techniques is **not forbidden**, however you should exercise care when using them.

These are just a few examples, so be careful. Remember, you are responsible for the complexity of every line of code you write!