Data Structure Challenge

Eckel, TJHSST AI1, Fall 2022

Background & Explanation

A huge part of AI this year is understanding the data structures Python includes and when to use each one. We will need to process enormous amounts of data quickly, which means the difference between O(1) and O(n) can be the difference between our code finishing in a couple of minutes and our code taking all week to run. This is a series of exercises designed to give you practice with choosing and using the proper data structures in Python. Specifically, I'm looking for correct use of lists, heaps (which are implemented as utility functions on list objects in Python), sets, and dictionaries.

On the course website, download the zip file containing several sets of data files of varying sizes. Each file contains a random sequence of integers, one integer per line. You can check your work on this assignment by checking sample run output on each data set; these sample runs are given after the list of problems.

For this assignment (and most assignments this year) it has to be correct *and* run in a given time. So, your goal here isn't just to get right answers, it's to get them quickly! The sample runs at the end of the document give correct answers AND runtime targets; be sure to pay attention to both as you code.

Reading Files in Python

As I'm sure you expect at this point, file I/O in python is much simpler than Java. This code, for example, will read in every line of a file and store each line in a list of strings.

```
with open("some_file.txt") as f:
    line_list = [line.strip() for line in f]
```

The `.strip()` command is essential here. It removes any leading or trailing whitespace on a string. In this case, it will remove the "\n" from each line that has one, but if the last line of the file doesn't end in "\n" it will still work without removing a random final character from the last word.

One **absolutely necessary** guiding principle: when using any file, **you only want to read from the file ONCE.** Open it once, read it, and store the information in a data structure, then search or manipulate the data structure. Continually searching a file for information is **extremely slow** compared to searching data structures stored in RAM. Using this "with" construction means that the file is only open while the indented code block under the "with" statement is running. As soon as the code block completes Python closes the file automatically. This is the most efficient way to use files and you should use it every time.

The questions below should all be answered in one run of your code, which means you should pre-load the three sets of data before answering any of them so that the files only have to be opened once each.

Your code should have exactly three "with open" code blocks – exactly one per file – and no more.

Timing Your Code in Python

Since you'll need to time your code for this assignment, here's how to do that.

```
from time import perf_counter

start = perf_counter()

#All your code goes here, including reading in the files and printing your output

end = perf_counter()
print("Total time:", end - start)
```

Let me know if you have any questions!

Problems

Remember you'll take the names of three files. In your code you can temporarily write a line of code like this at the top of your file:

```
f1, f2, f3 = "10kfile1.txt", "10kfile2.txt", "10kfile3.txt"
```

…but be sure to change this to read in command line arguments instead when you're ready to submit (instructions on this are at the end of the document).

1) How many distinct values appear in both file 1 and file 2? (ie, a value is part of your count if it is in file 1 and it is also in file 2.)

2) Reading file 1 from top to bottom, find the 100th unique value in the file, then the 200th, then the 300th, etc, and add them all together. What is the sum of all of these numbers?

3) Print the total number of times any value in file 3 appears in files 1 and/or 2. (That is, for each **unique** value in file 3, find the count of how many times it appears in files 1 and 2. Then add together all those counts.)

4) Print a list of the 10 **smallest** numbers that appear **at least once** in file **1**. Print them as a single list in **increasing** order.

5) Print a list of the 10 **largest** numbers that appear **at least twice** in file **2**. Print them as a single list in **decreasing** order.

6) Generate a new sequence of numbers as follows. Read in file 1 and, every time you encounter a multiple of 53, add the smallest number you've seen so far that isn't already in your new sequence to your new sequence. (The multiples of 53 should be included when determining the smallest number you've seen so far.) What is the sum of every number in this new sequence?

7) At the bottom of your code, print the total amount of time your code took to run in seconds. This should include the time you took to read in the files and store their data as well as the time to solve all six prior problems. I will independently verify this with my grading script, so be sure to be honest!

Sample Runs

A couple of things to note before looking at this output:

- For the smallest data files, the ones marked "10k" because files 1 and 2 each have 10,000 numbers in them, I've included extra output for questions 2 and 6 so you can make sure you're generating the right sequences. You **SHOULD NOT INCLUDE** this output in your submission.
- My computer is very fast. **You should not aim for exactly the runtimes you see below.** If you're on a school laptop, **3x the runtime seen below** is a good target, and if you're just a little bit over that's probably still fine. If you're not on a school laptop, we might have to do some experimenting.

Sample run for the 10,000 number files (including extra info for debugging):

```
>python.exe data_challenge_solution.py 10kfile1.txt 10kfile2.txt 10kfile3.txt

#1: 3936


#2: 330168

Every 100th unique value in order: [9869, 7743, 363, 6571, 5741, 4731, 6569, 9237,
7624, 7795, 7165, 4172, 4781, 9189, 3848, 6834, 3704, 1407, 7271, 893, 9447, 3563,
3548, 481, 749, 9451, 4308, 827, 8063, 7906, 9536, 4025, 5817, 6835, 1980, 2755,
6538, 8825, 8109, 608, 4508, 2441, 9898, 3559, 2083, 7072, 4186, 4758, 2813, 6693,
4962, 6385, 56, 8202, 2808, 1093, 8423, 6161, 8042, 4037, 8568, 4542]


#3: 308


#4: [1, 2, 4, 5, 6, 8, 10, 11, 13, 15]


#5: [9998, 9995, 9994, 9987, 9977, 9975, 9974, 9972, 9949, 9947]


#6: 31073

The sequence generated for this question is [60, 203, 65, 82, 36, 50, 81, 93, 106,
133, 151, 156, 153, 10, 61, 162, 197, 226, 13, 77, 191, 194, 64, 214, 227, 275,
234, 286, 71, 115, 159, 78, 30, 2, 97, 129, 147, 189, 195, 86, 90, 72, 135, 150,
139, 152, 155, 163, 168, 188, 58, 83, 182, 198, 204, 21, 126, 4, 66, 136, 158, 206,
210, 211, 45, 213, 217, 39, 70, 75, 67, 26, 69, 8, 88, 146, 148, 174, 1, 29, 101,
185, 199, 84, 138, 89, 144, 201, 207, 218, 141, 220, 122, 196, 27, 55, 192, 231,
140, 108, 15, 23, 170, 53, 236, 237, 238, 239, 241, 243, 244, 117, 47, 6, 35, 245,
248, 249, 254, 255, 256, 20, 11, 57, 258, 166, 22, 230, 259, 263, 265, 113, 76, 5,
176, 127, 33, 222, 242, 266, 268, 271, 190, 273, 56, 44, 274, 276, 277, 40, 42,
110, 143, 280, 283, 284, 288, 41, 252, 289, 298, 299, 302, 305, 91, 306, 154, 229,
173, 219, 297, 107, 307, 46, 308, 309, 313, 99, 295, 314, 317, 80, 320, 323, 223,
325, 326, 114, 132, 142, 102, 172, 212, 260, 25]


Total time: 0.011101799999999995 seconds
```

Sample run for the 100,000 number files:

```
>python.exe data_challenge_solution.py 100kfile1.txt 100kfile2.txt 100kfile3.txt
#1: 40079

#2: 30282890

#3: 2881

#4: [1, 3, 4, 6, 8, 10, 11, 12, 13, 14]

#5: [99994, 99988, 99984, 99982, 99980, 99974, 99972, 99966, 99957, 99950]

#6: 2900849

Total time: 0.11956899999999998 seconds
```

Sample run for the 1,000,000 number files:

```
>python.exe data_challenge_solution.py 1mfile1.txt 1mfile2.txt 1mfile3.txt
#1: 399675

#2: 3158026662

#3: 29350

#4: [0, 2, 3, 4, 6, 9, 11, 13, 15, 16]

#5: [999998, 999994, 999991, 999987, 999980, 999979, 999977, 999973, 999969,
999966]

#6: 288836009

Total time: 1.7003063 seconds
```

Sample run for the 10,000,000 number files:

```
>python.exe data_challenge_solution.py 10mfile1.txt 10mfile2.txt 10mfile3.txt
#1: 3996136


#2: 316100646531


#3: 299076


#4: [0, 1, 2, 3, 4, 5, 6, 7, 11, 13]


#5: [9999995, 9999994, 9999993, 9999992, 9999989, 9999987, 9999982, 9999981,
9999978, 9999963]


#6: 27962828640


Total time: 19.2798091 seconds
```

REMINDER: If you're on a school laptop, anything less than 3x my runtime is probably fine. So, less than a minute on this largest data set is what you're aiming for.

Get Your Code Ready to Turn In

Don't forget to modify your code so it accepts command line arguments, then **test your code using command line arguments so you're sure that works** before you submit!

Specification

Submit a single Python script to the link given on the course website.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your code accepts **command line arguments** for the name of three files, then runs all six problems correctly on those three files.
- Your code **does not** print the additional output you see in my 10,000 value sample run below. Just the six answers and runtime.
- Your code runs quickly (see sample runs and multiply by 3 for runtime targets).