I have written splitting scripts a couple times before, so I decided to use this homework assignment as an opportunity to improve on what I did in the past. The last script I wrote did not read in the entire input file, instead iterating over it with a `for` loop and using `random.choice` to choose which file to write to. However, it was not completely pseudorandom: while the initial choice of whether to write to the training, development or testing file was pseudorandom, if the file that was initially chosen was full, then the choice of what file would be chosen instead was predictable. Additionally, the code included a number of near-identical `if-else` suites, making it unnecessarily repetitive. Thus, my personal goal was to make a script that was (i) fully pseudorandom and (ii) clean that (iii) did not read the entire input file in (since such a strategy would not be scalable for larger files).

Making the script fully pseudorandom was not itself difficult; the challenge was to do so without making the script messier. The solution I came up with was to write a function, `write_sentences`, that would recursively decide which file to write to (still using `random.choice`. For example, suppose when we run `write_sentences`, `random.choice` decides to write to the development file, but the development file already has enough entries. Then `write_sentences` calls itself with only the training and test filestreams as inputs, and uses `random.choice` to choose between those two. Although this accomplished all of my goals to my satisfaction, it caused a number of typing problems with `mypy`. To be honest, I did not understand all of the typing errors it gave, but I managed to resolve the errors by changing the type of the first argument of `write_sentence` from `List[List[Union[IO, int]]]` to `List[Tuple[IO, int, int]]`, changing the rest of the file appropriately, and by explicitly declaring the type of the variable `outs` on line 77.

Update, 3/6

Kyle pointed out that I did not actually need recursion, so I removed the recursive element from the code. He also pointed out that the largest output file is also more likely to have sentences from the end of the file, and suggested that I use `random.choices` so that I can supply weights. I implemented his suggestions and compared this new algorithm to an algorithm that read the entire input file into memory and shuffled it, and found that my iterative algorithm was about 0.1 seconds faster (although I only spend about 10 minutes developing the in-place splitting code, so it is possible that I did not come up with the most information; for more information, see `split_in_memory.py` and `timing.txt` in the `time` branch — these were not part of the homework assignment, so I did not think it made sense to merge them into `master`.