Due: Tuesday 11/1/2022

For this homework you are doing 3 questions on **complexity** which are a mix of pencil and paper (showing the O[1] of the runtime of your algorithm), implementation, graphing, and comparing the runtime you came up with to the experimental results; and also 1 pencil and paper question on **correctness**.
This means that for each of the first three problems below you should:

- Devise and implement (in Python) an algorithm for that problem—this part will get submitted separately as the implementation portion. Note that for each Python program you write, if we run it from the command line, *e.g.* :

  ```
  python3 rmdup.py
  ```

  It should measure the runtimes (as described below) and print out the data in a comma separated format, with the size first and the values for each type of data as separate columns, such as (note these numbers are just made up):

  ```
  4096,12345,12456,13987
  8192,24690,24900,30012
  16384,50073,52045,61875
  etc..
  ```

  If we load your file into another with `import` (*i.e.*, to test its behavior and grade correctness) your performance experiments should **NOT** run. Please use

[1] Your bounds should be tight, so really Θ

```
if __name__ == "__main__":
```

to make your performance experiments run only if we run your python file directly.

- Analyze the runtime of your algorithm as we did in class: for each line, state how many times that line happens, and how long it takes. Then give an overall runtime. Please be sure to include the code in the writeup so that the TAs can grade this while looking only at the pencil and paper part (your pencil and paper submission should be "self contained").

- Measure the runtime for various data sizes. Include in your writeup both a graph, and a table of the raw data (with units!). Note that for each problem below we will say a bit more about the range of input sizes you should measure. We will also give you some different kinds of data that you should measure on for each problem. These different types should all be put into one graph (separate lines) and one table (a column for each). Please also include a brief description of any experimental methodology of note—*e.g.*, if you did multiple runs, if so how many and how you combined the results (average, median, etc). If you present error bars, please describe what they mean, etc.

- Analyze your results. If theory (2 above) and practice (3 above) diverge, please note such, and give any thoughts you have on why this divergence may happen. If different categories of inputs exhibit different behaviors, give your thoughts at to why those difference happen.

1. The first problem you should implement is in `rmdup.py`. Your goal is to write a function which removes duplicates from the

data, keeping the **last** occurrence of each element and preserving order. The function should return its answer. You may modify `data` if you wish, but do not need to.

You should measure the performance of this code with data sizes from 4,096 (4K) to 4,194,304 (4M) elements, in steps of 2x (so 4K, 8K, 16K,...) on three types of random data:

Many duplicates Each data item is chosen by
`random.randrange(0,size/2048)`

Moderate duplication Each data item is chosen by
`random.randrange(0,size/16)`

Rare duplication Each data item is chosen by
`random.randrange(0,size*4)`

Note: your data creation should be *outside* your time measurement (not included in the time).

We note that for us, the worst of these took about 1.5 seconds on the 4M size. If you have a significantly slower algorithm, you can stop at whatever size takes 10 seconds for one call to of the function at that size, but start at correspondingly smaller size (so if you go up to 1M, start at 1K).

For us, the O of the runtime was the same for the three categories of data, but we observed drastically different constant factors. If your approach has significantly different factors (or other behavior between the three categories), please give some thoughts as to why.

2. The second problem you should implement is in `matrixmult.py`. Here, you are going to write matrix multiplication. Note that there are many fancy matrix multiplication algorithms with improvements to the runtime over the "standard" approach (that

you would do by hand). You are welcome to do a fancy algorithm if you want to learn such—however, please note that what you write should reflect your understanding of that algorithm and not copying code. If you do learn a fancy algorithm, please cite any relevant sources.

Also please note that the standard algorithm you would do by hand/what you would have learned in school is fine, and you will not receive more or fewer points for it than a fancy algorithm.

For this problem, you should consider data sizes with N ranging from 4 to 512 (as before, doubling at each step: 4, 8, 16...). You will also consider three categories of the "shapes" of the matrix:

Many rows by few columns (N*4)xN * Nx(N/4)

Square NxN * NxN

Few rows by many columns (N/4)xN * Nx(N*4)

Note that for all three you can consider N to be the x-value in your plots.

3. For this problem, you are going to work in `eqsubstr.py`, and write The `matching_length_substrs` function. This function takes a string (`s`) and two characters (`c1` and `c2`). The goal of this function is to find consecutive sequences of c1 and consecutive sequences of c2 that are the same length. It should return a *set* of three-tuples where each element of the tuple is (c1-index, c2-index, length). For example, an element in the answer with value (5, 12, 3) means "there are 3 c1s at index 5 and also 3 c2s at index 12". In this problem, we only consider the total sequence length (if you see 4 "a"s in a row, you only call it a sequence of length 4, not also a sequence of length 3), and we only consider lengths > 0.

Your function should return a set containing *all* pairings of equal length sequences. If there are 2 places that 4 of c1 appear and 3 places that 4 of c2 appear, then your answer should have all 6 pairings of those. Likewise, if c1 has a length 5 sequence but c2 does not have any length 5 sequences, then no tuples with length 5 will appear in the output.

There may be characters other than c1 and c2 in the input string.

For this problem, you should consider three categories. However, two of them you need to describe in more detail—namely, give a regular expression which describes those cases.

Best Case (all c1 and c2): What is the best case for the runtime of your algorithm on a string that only has c1 and c2 in it (no other characters)? Describe the *entire* category with a **regular expression**. Note that we restrict it to no other characters as an input of d^* when $c1 = a$ and $c2 = b$ is likely to be trivial.

Worst Case: What is the worst case? Again, please write a **regular expression** that describes all strings that give this worst case behavior.

Random Input: Use the `rndstr` function we have provided. This function takes an number `n` and produces a string of length `n` which is approximately `3/7` "a"s, `3/7` "b"s and `1/7` a capital letter. You should use this with c1=a and c2=b.

For this problem, you should consider inputs sizes of 512 to 16K (16,384) in steps of 2x. If your algorithm has significantly worse runtimes than ours, you can stop at whatever input size takes about 1 minute for the worst case, but as before, start corre-

spondingly earlier.

Hint: you can make a list of functions in Python, so I wrote `best(n)` and `worst(n)` and then made the list `strgens=[best,worst,rndstr]` which could iterate over like any other list:

```
strgens=[best,worst,rndstr]
for sg in strgens:
    #....
    s=sg(size)
    #....
```

4. Consider the following code in our tiny programming language from class (note: we have added line numbers for later):

```
1:  fun reverse(data : array of int) {
2:    n : int := data.length / 2; # rounds down
3:    i : int := 0;
4:    while (i < n)
5:      do
6:        j : int := data.length - i - 1;
7:        temp : int := data[i];
8:        data[i] := data[j];
9:        data[j] := temp;
10:       i := i + 1;
11:     done
12:  }
```

This function reverses an array of integers, and your job is to show that it does so correctly. Note that this function has no preconditions for either total or partial correctness.

(a) Give a formal definition of what it means to reverse an array. Specifically, define the predicate `reversed(data)` which is a formal mathematical statement (hint, it should have a $\forall$ in it) that `data` has been reversed. As in class, you should reference $\text{data}_{\text{orig}}$ to talk about the original data before `reverse` is called. Note also that `reversed(data)` should be the postcondition of the `reverse` function.

(b) You will find it quite helpful to define another predicate `revregion(data, i)` which means "the first `i` elements of data have been reversed". Give a formal mathematical definition of this predicate below (again. this should have a $\forall$ in it). Note: if you do not make this definition strong enough, you will run into trouble on the proofs below. Hint: you probably want to to look something like this:

$$\begin{aligned}\forall k \in \mathbb{N}.\ & ((0 \leq k < i) \Rightarrow ....) \wedge \\ & ((i \leq k < \text{data.length} - i) \Rightarrow ...) \wedge \\ & ((\text{data.length} - i \leq k < \text{data.length}) \Rightarrow ...)\end{aligned}$$

(c) **Prove**

$$\forall \text{data.revregion}(\text{data}, \lfloor \text{data.length}/2 \rfloor) \Rightarrow \text{reversed}(\text{data})$$

Hint: data.length $\in \mathbb{N}$ and you can use the lemma $\forall n \in \mathbb{N}.(\text{odd}(n) \vee \text{even}(n))$

(d) State "what you know" after each of the following lines in the program: 3, 5, 6, 7, 8, 9, 10, 11 (give one set of propositions for each line number). Note that after line 11 you should know `reversed(data)` by virtue of knowing `revregion(data, data.length/2)`, and what you proved above.

To submit:

Pencil and paper Q1–3 Big-O analysis, graphs, tables, analysis of results. Q4 Answers to all parts above

Implementation Your modified eqsubstr.py, matrixmult.py, and rmdup.py files.