Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 10b (34 points) **Fall 2019, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- If you don't have an account on it, sign up for one using your CU email. You should have gotten an email to sign up. If your name based CU email doesn't work, try the identikey@colorado.edu version.
- Gradescope will only accept **.pdf** files (except for code files that should be submitted separately on Gradescope if a problem set has them) and **try to fit your work in the box provided**.
- You cannot submit a pdf which has less pages than what we provided you as Gradescope won't allow it.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 10b (34 points) **Fall 2019, CU-Boulder**

Important: This assignment has 1 (Q1) coding question.

- You need to submit 1 python file.
- The .py file should run for you to get points and name the file as following -
  If Q1 asks for a python code, please submit it with the following naming convention - `Lastname-Firstname-PS10b-Q1.py`.
- You need to submit the code via Canvas but the table/plot/result should be on the main .pdf.

Name: Luna McBride
ID: 107607144

CSCI 3104, Algorithms **Profs. Hoenigman & Agrawal**
Problem Set 10b (34 points) **Fall 2019, CU-Boulder**

1. (34 pts total) Recall that the *string alignment problem* takes as input two strings x and y, composed of symbols $x_i, y_j \in \Sigma$, for a fixed symbol set Σ, and returns a minimal-cost set of *edit* operations for transforming the string x into string y.

   Let x contain n_x symbols, let y contain n_y symbols, and let the set of edit operations be those defined in the lecture notes (substitution, insertion, and deletion).

   Let the cost of *insert* be c_{insert} and *delete* be c_{delete}, and the cost of *sub* be c_{sub}, except when $x_i = y_j$, which is a "no-op" and has cost 0.

   In this problem, you will implement and apply three functions.

   (i) `alignStrings(x,y,` c_{insert}`,` c_{delete}`,` c_{sub}`)` takes as input two ASCII strings x and y, cost of the operations, and runs a dynamic programming algorithm to return the cost matrix S, which contains the optimal costs for all the subproblems for aligning these two strings.

   (ii) `extractAlignment(S,x,y,` c_{insert}`,` c_{delete}`,` c_{sub}`)` takes as input an optimal cost matrix S, strings x, y, cost of the operations, and returns a vector a that represents an optimal sequence of edit operations to convert x into y. This optimal sequence is recovered by finding a path on the implicit DAG of decisions made by `alignStrings` to obtain the value $S[n_x, n_y]$, starting from $S[0, 0]$.

   When storing the sequence of edit operations in a, use a special symbol to denote no-ops.

   (iii) `commonSubstrings(x,L,a)` which takes as input the ASCII string x, an integer $1 \leq L \leq n_x$, and an optimal sequence a of edits to x, which would transform x into y. This function returns each of the substrings of length at least L in x that aligns exactly, via a run of no-ops, to a substring in y.

   (a) (21 pts) From scratch, implement the functions `alignStrings`, `extractAlignment`, and `commonSubstrings`. You may not use any library functions that make their implementation trivial. Within your implementation of `extractAlignment`, ties must be broken uniformly at random.
   If you plan to create a version that saves the parent information in `alignStrings` itself, then you should break the ties randomly in `alignStrings` instead.
   Submit:
   - A brief paragraph for each function that explains how you implemented it (describe how it works and how it uses its data structures).
   - Your code implementation, with code (the code should be submitted on Canvas)

- The cost matrix S that your code produces on the strings x=`EXPONENTIAL` and y=`POLYNOMIAL` with $c_{insert} = 2$, $c_{delete} = 1$, $c_{sub} = 2$

Solution.
alignStrings: this one creates an array, sized by the length of each string plus 1 (to account for empty and give base cases). This is filled by two for loops, accounting for multiple main cases. Case 1: i and j are 0, so we are at the origin and thus have no cost yet. Case 2: we are on an edge (i or j are 0), so we fill it bounded by the cost of insert horizontally and delete vertically, as those are the operations done on each axis. Case 3: the letter at y and j are the same, so we do not need to add cost to them on either side (from inserting or deleting), so we just take the previous diagonal's value (would be sub, but is no-op because the values are the same). Case 5: They are not the same, so we pick the least weighted outcome (with the addition of the cost to do the operation) to pick the path of least resistance.

extractAlignment: this rebuilds the actions taken bottom up, then reverses the string (as recommended on Piazza). It uses a while loop to pick the path of least cost back to the start, making sure to put values that are the same as a 'no-op' instead of what it would be with costs.

commonSubstrings: this uses a for loop to count the number of no-ops in a row, holding the best one in a different array with its own count. That way, we can keep checking for a better one without deleting our current best.

```
[[ 0. 2. 4. 6. 8. 10. 12. 14. 16. 18. 20.]
[ 1. 2. 4. 6. 8. 10. 12. 14. 16. 18. 20.]
[ 2. 3. 4. 6. 8. 10. 12. 14. 16. 18. 20.]
[ 3. 2. 4. 6. 8. 10. 12. 14. 16. 18. 20.]
[ 4. 3. 2. 4. 6. 8. 10. 12. 14. 16. 18.]
[ 5. 4. 3. 4. 6. 6. 8. 10. 12. 14. 16.]
[ 6. 5. 4. 5. 6. 7. 8. 10. 12. 14. 16.]
[ 7. 6. 5. 6. 7. 6. 8. 10. 12. 14. 16.]
[ 8. 7. 6. 7. 8. 7. 8. 10. 12. 14. 16.]
[ 9. 8. 7. 8. 9. 8. 9. 10. 10. 12. 14.]
[10. 9. 8. 9. 10. 9. 10. 11. 11. 10. 12.]
[11. 10. 9. 8. 10. 10. 11. 12. 12. 11. 10.]]
['delete', 'delete', 'no-op', 'no-op', 'sub', 'sub', 'no-op', 'insert', 'sub', 'no-op', 'no-
op', 'no-op']
(['i', 'a', 'l'], 3)
```

(b) (7 pts) Using asymptotic analysis, determine the running time of the call `commonSubstrings(x, L, extractAlignment( alignStrings(x,y,`c_{insert}`,`c_{delete}`,`c_{sub}`),` `x,y,`c_{insert}`,`c_{delete}`,`c_{sub}` ) )`
Justify your answer.

Solution.
This uses the function calls as the inputs to other functions, having one finish before going to the other. As such, none of the actual functions are within another's loop and thus can be considered different parts rather than all within one big O (none are within another's loop). None are recurrances either, so we do not need to dabble with recurrance relations or the like. This leaves all to the worst cases of their loops.

commonSubstrings uses a for loop of the length of a, which is also the length of x by the concept of such programs. As we do not need to look at additions other than the variable, this has O(x)
extractAlignment uses a while loop instead of a for loop, which makes the worst case not as simple as "range(0,len(x)), so the worst is len(x)". However, by the code we can tell that one item (item representing the terms 'no-op','sub','insert','delete') is added to a each time we go through the loop. As such, we necesarily made as many loops as len(a) by proxy of the point being to build a, and as discussed before, that means it is also the same as len(x) (and thus the same as commonSubstrings).
alignStrings is different in that is uses two loops instead of 1. The loops use len(x) and len(y) as ranges, meaning that the overall usage comes to len(x)*len(y) (if they were the same, this would be similar to n^2, however, we cannot necessarily say they are both the same)

O(xy)+O(x)+O(x)
xy>x
O(xy)

(c) (6 pts) **Plagiarism detector** - String alignment algorithms can be used to detect changes between different versions of the same document (as in version control systems) or to detect verbatim copying between different documents (as in plagiarism detection systems).

The two song lyrics files for PS10b (see class Canvas) contain lyrics of two different songs in text format. Use your functions from (1a) with $c_{insert} = 1$, $c_{delete} = 1$, $c_{sub} = 1$ to align the text of these two documents. Utilize your **commonSubstrings** function for plagiarism detection. Present the results of your analysis, including a reporting of all the substrings in x of length $L = 10$ or more that could have been taken from y in two columns with the first being the length of the substring and the second being the actual common substring obtained via continuous 'no-op' run.
Briefly comment on whether these songs could be reasonably considered original works, under CU's academic integrity policy.

Solution.
length ...
—— ...
18 ...
12 ...
28 ...
26 ...
42 ...
32 ...
20 ...
22 ...
36 ...
44 ...
43 ...
substring

---

yI hear the train
yI hear the train ' it's rolli
yI hear the train ' it's rolliround the bend And I ain't
yI hear the train ' it's rolliround the bend And I ain't since I don't know when

yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet the
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet thee's rich folks eatin
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet thee's rich folks eatin They're probably dri
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet thee's rich folks eatin They're probably dri' coffee and smokin' big cigars But
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet thee's rich folks eatin They're probably dri' coffee and smokin' big cigars Butle farther down the line Far from Folsom Pr
yI hear the train ' it's rolliround the bend And I ain't since I don't know when en I was just a baby my mama told me son my head and I cry. I bet thee's rich folks eatin They're probably dri' coffee and smokin' big cigars Butle farther down the line Far from Folsom Pr's where I want to stay And I'd let that l

In terms of actual CU policy, I could not find a specific percentage to consider plagarism, however, due to the strong usage of key phrases and long stretches of common ground, I would personally say this is more likely plagarized than not.