Name: Tristan Thomas
ID: 105771156

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: Tristan Thomas
ID: 105771156

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: Tristan Thomas
ID: 105771156

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.

Name: Tristan Thomas
ID: 105771156

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

i. `alignStrings(x,y,` c_{insert}`,` c_{delete}`,` c_{sub}`)`

For this function, I used a dynamic programming algorithm to produce the cost matrix S, which contains the optimal costs for all the subproblems for aligning the two input strings, x and y. The cost matrix is represented by a two dimensional array. This matrix is of size: $length(x)+1 \times length(y)+1$. The $0th$ row and the $0th$ column of the matrix are first initiated as they are trivial. The function then utilizes a nested for loop in order to calculate each index of the matrix in accordance with the recurrence defined as:

$$cost[i][j] = min(\begin{matrix} S[i-1][j-1] + \text{cost of sub}, \\ S[i][j-1] + \text{cost of insert} \\ S[i-1][j], + \text{cost of delete} \end{matrix}$$

The value stored in $S[i][j]$ is thus equal to the optimal cost to align the strings $x[0 : i]$ and $y[0 : j]$. Upon termination of the for loop, the function will return the completed cost matrix S.

ii. `extractAlignment(S,x,y,` c_{insert}`,` c_{delete}`,` c_{sub}`)`

For this function, I again utilized the above recurrence in order to produce a vector, a, which stores an optimal sequence of edit operations in order to convert string x into string y. Iterators i and j are initialized to zero before each iteration of the while loop examines the indices $S[i+1][j+1]$, $S[i+1][j]$, and $S[i][j+1]$. It will choose the optimal operation, append that operation to a list, and move both i and j in accordance with the operation chosen. The loop exits once it has traversed from $S[0][0]$ to S[length of x][length of y]. The result to be returned is a list containing the sequence of operations in order to convert string x into string y.

iii. `commonSubstrings(x,L,a)`

This function takes the list of operations produced from extractAlignment(), and produces the set of substrings of length $\geq L$, in x, that align exactly to a substring in y. To do this, the function iterates through a and checks if elements in a[i]-a[i+L] are all 'no-op' operations.

If it finds a sequence which satisfies this condition, it will store the respective indices of string x in a list set to hold the set of all common substrings. Upon termination, the function will return the set of common substrings.

```
------- TESTING WORDS -------
Source = EXPONENTIAL
Target = POLYNOMIAL
       P  O  L  Y  N  O  M  I  A  L
  [ 0  2  4  6  8 10 12 14 16 18 20]
E [ 2  2  4  6  8 10 12 14 16 18 20]
X [ 4  3  4  6  8 10 12 14 16 18 20]
P [ 6  4  5  6  8 10 12 14 16 18 20]
O [ 8  5  4  6  8 10 10 12 14 16 18]
N [10  6  5  6  8  8 10 12 14 16 18]
E [12  7  6  7  8  9 10 12 14 16 18]
N [14  8  7  8  9  8 10 12 14 16 18]
T [16  9  8  9 10  9 10 12 14 16 18]
I [18 10  9 10 11 10 11 12 12 14 16]
A [20 11 10 11 12 11 12 13 13 12 14]
L [22 12 11 10 12 12 13 14 14 13 12]
```

(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.

The running time of the function call:
`commonSubstrings(x, L, extractAlignment(`
`alignStrings(x,y,`c_{insert}`,`c_{delete}`,`c_{sub}`), x,y,`c_{insert}`,`c_{delete}`,`c_{sub}` ) )`
can be determined by summing each of the 3 function calls.

Let m = length of x and n = length of y.
commonSubstrings() will have runtime- $O(m+n)$. The worst case for this function is a sequence of length $(n+m)$ consisting of 0 consecutive no-ops.

extractAlignment() will have runtime- $O(m+n)$. The worse case for this function is generating a sequence consisting of n inserts and m deletes. This can be done in $m+n$ time.

alignStrings() will have runtime- $\Theta(m \times n)$. This function runs through a nested for loop consisting of n and m iterations.

Thus, the runtime of the function call will be $= (m+n)+(m+n)+(m \times n)$.
Asymptotically, the function **alignStrings()** will dominate runtime.
The asymptotic runtime is $\Theta(m \times n)$.

(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.

```
|---- Plagarism Results -----|
|Length |   Common Sub String|
|-----------------------------------------------------------------------|
18        I hear the train a
12         it's rollin
27        ound the bend  And I ain't
26         since I don't know when
41           When I was just a baby my mama told me
32        When I hear that whistle blowin'
19        I hang my head and
21        rich folks eatin' in
36         fancy dining car  They're probably
44         if that railroad train was mine  I bet I'd
43        n a little farther down the line  Far from
62         to stay  And I'd let that lonesome whistle blow my blues away
-----------------------------------------------------------------------|

Length of source: 874
Plagarism Detected: 507
Results = 0.580091533180778
```

The above image is the report generated by checking the provided songs for plagiarism. Under CU's academic integrity policy, these songs could not be reasonably considered original works. The song "Folsom Prison" was found to be approximately 58% similar to "Crescent City Blues".