Name: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

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: Nikolaas Bender
ID: 106977096

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. paragraphs:
extract alignments: This goes through the S matrix and finds the backwards path from the bottom right corner back to the top left corner. I used a lot of if statements to check for all the different conditions that may arise in the S matrix and to account for each condition.

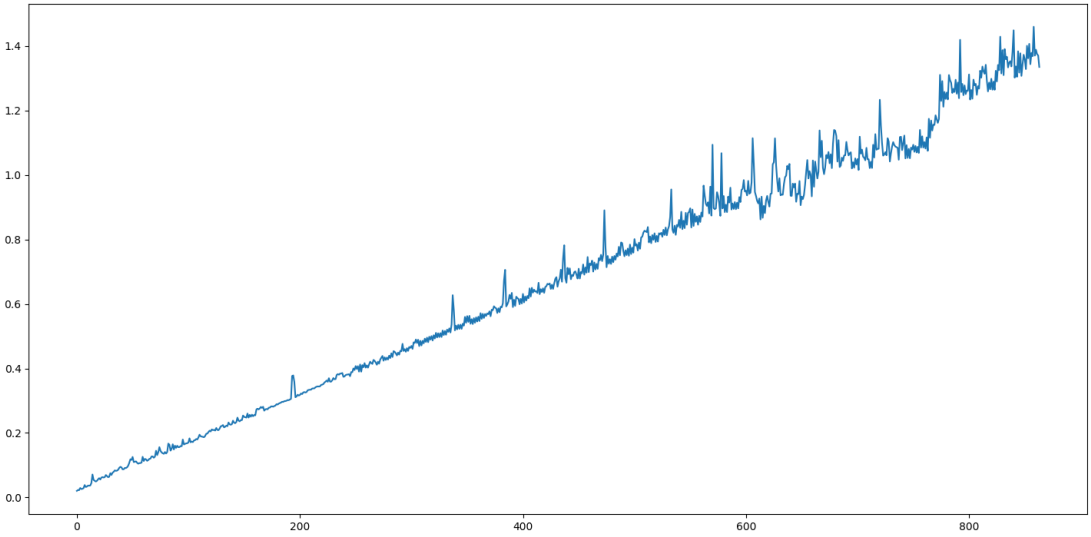
align strings: This creates the S matrix by going through and figuring out what operations need to be done and keeps track of all of the modifications. This was relatively straight forward as its almost the same algorithm as shown in lecture. The changes I made were to filling in the top and left edge and removing the creation of the P matrix because that was hard to implement.

common substrings: use the alignment list from extract alignments to go through x and extract a substring. The logic in there is to extract characters with "no-op" designations and to deal with insertions.
S matrix:

```
[[ 0  1  2  3  4  5  6  7  8  9 10 11]
     [ 1  0  1  2  3  4  5  6  7  8  9 10]
     [ 2  1  1  2  3  4  5  6  7  8  9 10]
     [ 3  2  2  2  3  4  5  6  7  8  9 10]
     [ 4  3  2  3  3  4  5  6  7  8  9 10]
     [ 5  4  3  2  3  4  5  5  6  7  8  9]
     [ 6  5  4  3  3  4  4  5  6  7  8  9]
     [ 7  6  5  4  4  4  5  5  6  7  8  9]
     [ 8  7  6  5  5  5  4  5  6  7  8  9]
     [ 9  8  7  6  6  6  5  5  6  7  8  9]
     [10  9  8  7  7  7  6  6  6  6  7  8]
     [11 10  9  8  8  8  7  7  7  7  6  7]
     [12 11 10  9  8  9  8  8  8  8  7  6]]
```

(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. Run time is along the y axis and x string length is along the x axis



You'll see that the run time is characteristic of DP as the trend of the run time is fairly linear with complexity 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. 74.66666666666667% plagerism

```
[19 |  I hear the train a]
[12 |  it's rollin]
[28 | round the bend  And I ain't ]
[26 |  since I don't know when  ]
[42 |     When I was just a baby my mama told me]
[34 |   When I hear that whistle blowin']
[19 |  I hang my head and]
[22 |  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]
```