Sequence Alignment with Dynamic Programming Implementation

Group Names: Glen Anderson, Le-Chuan Justin Chang, Ronald Salinas

Pseudocode

```
function getCost(char1, char2):
     A gets index of char1
     B gets index of char2

     return costTable(A, B)

function editDistance(string1, string2):

     # SECTION 1: BASE CASES
     for(i = 0 to m)
          M[i,0] = getCost(insert “-” with string1[i]) + M[i-1,0]
          BT[i,0] = (M[i,0],”L”)
     for(j = 1 to n)
          M[0,j] = getCost(insert “-” with string2[j]) + M[0,j-1]
          BT[0,j] = (M[0,j],”U”)

     # SECTION 2: MIN COMPARISONS ALGORITHM
     for(i=1 to m)
          for(j=1 to n)
               if string1[i] == string2[j]: diff = 0
                    else: diff = getCost(string1[i], string2[j])
               M[i,j] = min(
                    M[i-1,j]+getCost(string1[i], -),
                    M[i,j-1]+getCost(-, string2[j]),
                    M[i-1,j-1]+diff(string1[i],string2[j])      )

               BT[i,j] = Direction based on the minimum value
                         M[i,j] selected above.
     #SECTION 3 Backtracing
     While i != 0 or j != 0
          i = Subtract if BT[i,j] == “D” or “L”
          j = Subtract if BT[i,j] == “D” or “U”
```

```
        tracei += string1[i] if BT[i,j] == “D”
            or “-” if BT[i,j] == “U”


        tracej += string2[j] if BT[i,j] == “D”
            or “-” if BT[i,j] == “L”

    reverse tracei and tracej #backtracing gives backwards strings

    return tracei,tracej,cost
```

Asymptotic Run Time Analysis

Section 1: Base Cases

This runs until all base cases on M[i, 0] and M[0, j] are filled. We know i is at most m and j is at most n.

$T(n) = c(m) + c(n) \quad = \quad O(m+n)$

Section 2: Min Comparisons Algorithm

This algorithm loops over each element in string2 which has n elements and does so until it finishes making comparisons for each i from 0 to m.

$T(n) = \quad (cn) + \ldots + (m)(cn) \quad = \quad O(mn)$

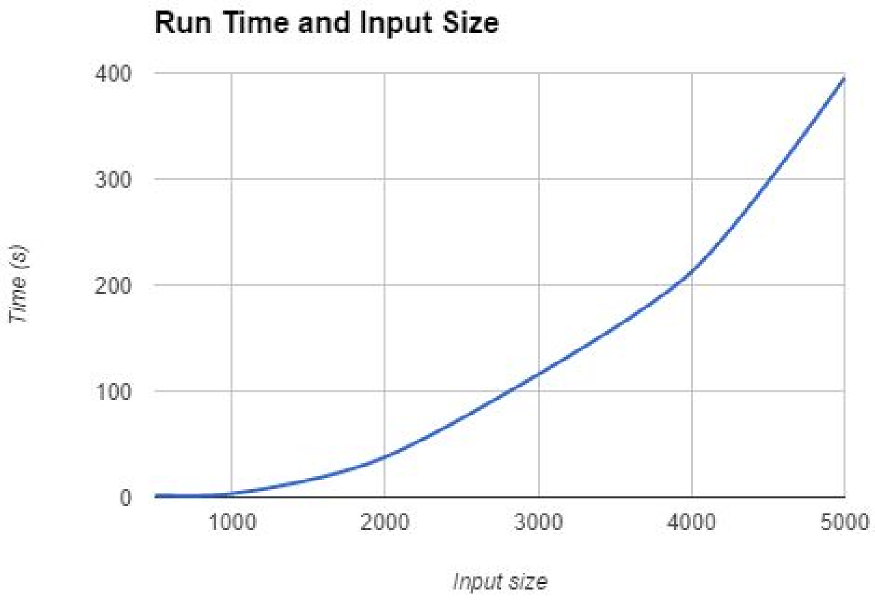
Section 3: Backtracing

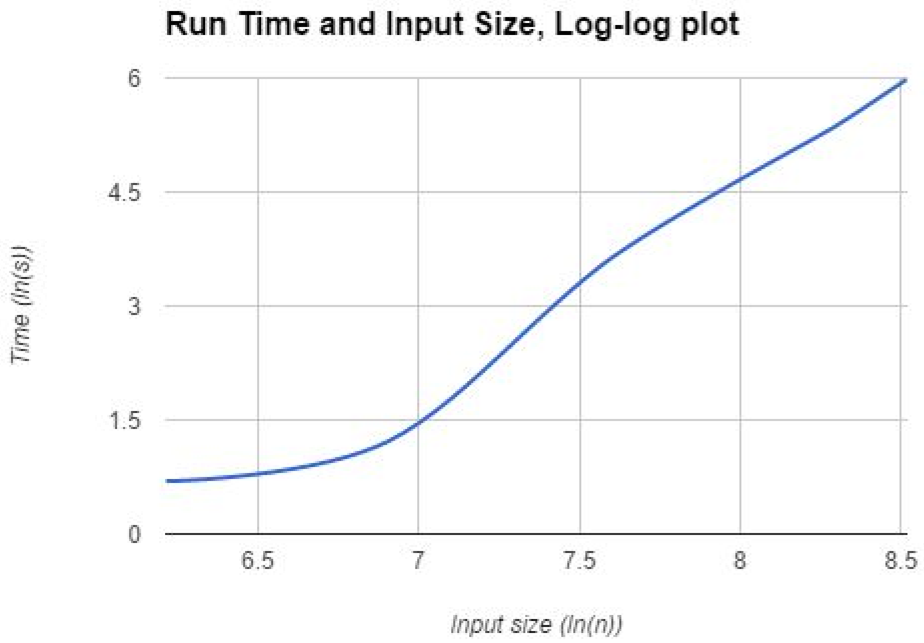
This algorithm continues to run while i (for m to 0) and j for (n to 0) and will decrement i, j, or both by 1. Therefore, this will run at most m+n times.

$T(n) = \quad c(m) + c(n) \quad = \quad O(m+n)$

Runtime Graph

For our data collection, we ran our program on the engineering servers at Oregon State on 5 different sizes of input. For each size of input, we ran 10 timed tests and plotted the average on the "Run Time and Input Size" graph. We did not exclude any operations during our testing.





The slope of this line is 2.439. We would likely get a steeper slope if we excluded smaller inputs, as these are more influenced by operations that are not O(nm), such as reading and writing to files or backtracing. Based on the slope, our runtime is as follows:

Based on the slope of this line, at any given point x, the runtime will be O(x^2.439).

Interpretation

The growth curve matches our expectations. While there are some factors that could have influenced our results, they ran roughly in O(nm) time. Since we ran inputs with 2 strings of equal length n, this can be represented as O(n^2) for this test. We attempted to run an input size of 10,000 per string, but it had taken over 20 minutes so it was not practical to test this input size.

We ran each input 10 times to get a reliable result for the time it took to complete a run for that given size of input, but some factors such as our connection speed, server load, and individual system could have had an impact on this. By running multiple tests for each size to determine an average, we try to minimize the impact these factors could have.

Other tests to specifically test the O(nm) runtime would be interesting, as we could ensure that the algorithm was actually running according to each input's size rather than the largest one.

Our log-log plot shows that there is a power relationship with our original data, although not as strong as we might expect. A perfect power relationship would be dictated by a straight line in the log-log plot, but ours does not have this, particularly for smaller input sizes. This could be because the runtime of smaller inputs are likely to be heavily influenced by additional processes such as backtracing and reading/writing to files. While these processes are not O(nm), they still add a significant amount of runtime to smaller input sizes. As our input sizes got larger, we saw more of a power relationship (a straighter line on the log-log plot) which we should expect.