**Project 4**

**Ryan Cook**

**Project:** Approximate Pattern Matching

**Test Case Group Descriptions:** I had two different test cases that I used, the first was a basic test document for easy to search words to verify that the program worked accordingly. The second was a text file with random strings that way I could run a time complexity case on the algorithm. I then added a section where a size of the text can be changed, doubling each time.

**Testing Plan / Test Results:** The way the algorithm works is that it works through each character of both the pattern and the text at the same time, through a nested for loop. This would give the time complexity of n^2 if each for loop goes through n times. Since the pattern and the text are not always the same size this complexity looks like m\*n. To start testing I used a small text and a small pattern that I knew was in the text in order to make sure that each case worked. Then once I was sure this worked properly I began testing on larger sizes of texts to make sure that the proper string was still returned. Finally, I did the timing on large patterns and large texts to find the order of the algorithm. For the large size text, I searched for items that weren’t in the list and were in the list to test time complexity, both of which came back with the same order time.

**Actual Testing Results:** See attached file.

**Problems:** A few problems I had were trying to create the data structure that would hold all the information I needed to be held. To do this I created a two-dimensional vector that holds a node, each node holds a value and a pair, the pair holds the previous spot that the algorithm used to get its answer (row and column numbers). I got the pseudocode from the Levenshtein distance algorithm and modified it to be able to reproduce the matched string at the end. Another problem I had was keeping track of variables. There was a lot of temporary variables that needed to be used in order to calculate and change positions through the array, especially when reproducing the string. I solved this problem by creating separate variables to hold the old position and the new position, then when moving to the new position would load the new into the old and find a calculation for the new position. The final problem I had was initializing the 2-dimmensional structure to hold the values that would lead to the right answer. I realized that the first row and column would need to be a place holder in order to do this so I pad the strings with a space that is not in the calculation but in the array to be a place holder. I then set the first-row value to be 0 and the first column value to be the row number. This would insure that the matrix would be properly initialized for the next step in finding an approximate match.