**Lab 1 Boolean and Ranked Retrieval**

**Oreen Yousuf**

**Jae Eun Hong**

**1. How to run our code**

We've commented our code along the way for your (the grader's) convenience, and for us to review. We started by importing our data after completing the command line commands, leaving us with index files of before and after lower casing the data. We left a printout of the number of words before and after doing this for you to review. We had a decrease in 1,822 words. To get the top k (10) – most frequent words (from the lowerized test data, we traversed through the passed through data and sorted it by the length of its frequency in the dictionary (our variable name) and kept appending each word into a stop words list until our counter reached the desired top-k words.

**2. Inverted Index / Boolean queries**

For our inverted index function, we left very descriptive commenting to act as our explanation for that part. We then left a printout of how every possible outcome of the inverted index function would be depending on the parameters desired by the user. Next, our intersect functions we passed in 2 lists of documents containing document ID numbers of each word the user inputted to compare. We implemented the main chunk of this (and the intersect with skips) function using the ‘Boolean retrieval’ lecture from the professor. While the index of both lists being compared isn’t NoneType (inducing an Index Error), we see if the operator comparison between the two elements of the lists at 2 equal, but distinct variables, is equal to (==) each other, if list1[index1] is less than (<) list2[index2], or if list1[index1] is greater than (>) list2[index2]. Each of these comparisons ill trigger the respective actions: appendage of current list1[index1] onto a list to be returned later, increment of both index1 and index2, and a comparison counter, or an increment of index1 and the comparison counter, or an increment of index2 and comparison counter. And at the end of this function, we print out the number of comparisons to be viewed and the resultant list we’ve been appending to, which is the intersect of document IDs of the user’s words. Intersect with skip has the same structure, but we’ve self-defined our skip size as 3 for experimental purposes, which you can change if you feel inclined. We then created a lambda function to act as our condition later on. There are extra simple if-statement conditions to check if the current index has a skip, if that index plus (+) the skip size is still less than (<) the length of the list (this is to make sure you don’t cause an out of bounds indexing error), and that the current list’s index at the current index plus skip size is still less than list2 at its current index. And the second new conditional statement is functionally the same, but from list2’s perspective. And again, we print out the comparisons and return the intersects. For our query we left commentary for you. We also left an example printed out. The number of comparisons when using the skip functions makes a sizeable difference, that would be beneficial in this query. When searching for the intersection of 'school AND kids AND really', using the skip counter with size 2 gives 45 comparisons, a size of 3 gives 35 comparisons and without the skips it's 81 comparisons.

**3. Ranked queries**

We then moved on to our index ranking to complete the rest of the assignment. To calculate the TF-IDF we created a nested dictionary for the intersection of document IDs 72, 224, and 385, which represented the intersection of the keywords ‘school’, ‘kids’ and ‘really’. The inner values were initialized at 0 and we traversed our words after ranking and if the corresponding indices match the one of the previously mentioned words, and the document IDs, then we would increment the appropriate inner values. We then compute idf with the log method from the math library, and multiply the idf by each tf to get our final answer.

Our results were clear that ‘school’ was the most “important” word in this particular list of words to compare to. It can be inherently understood why because the word ‘school’ would most likely be involved in documents with ‘kids’. However, in one particular document, 385, the word ‘really’ appears relatively “much” more than in the other documents and so for this particular document/case, it bears with it more importance than ‘school’, yielding in a higher TF-IDF value, 2.07945.