InvertedIndex

To create this index we utilized a linked list that sorted itself upon insertion, based on the tokens in ascending order. The program would access a file, take in the file's contents as one large string, remove all non-alphanumeric characters, lowercase every uppercase letter, and then tokenize the string using a space as a delimiter. The function removeNums is called on each token to remove any numbers at the beginning of tokens. The frequencies of the tokens in the list are then counted for each token and stored in each node. The removeDoops function is then called to remove duplicate tokens in the list from the same file. If there is more than one file taken in as input, this process is done for every file and then one final list is created by merging these lists in the mergeLists function. Once we have a final list, an additional sorting operation is performed to ensure that if the same token appears in more than one file in the list, it is stored in descending order by frequency first, and then in ascending order by file name if the frequencies are the same.

In order to get these list of words from the directory or file path, the recursive directories function was implemented to attempt to open a directory, and upon error, if errno is set to 2, it is declared that the file does not exist, and if errno is set to 20, then the file path is not a directory, and therefore is treated as a file, being passed through the above stated functions. However, if the directory is successfully opened, a loop will run through the directory to test every file in the sub directory, and its own sub directories, and so forth. After this is directory is created, the output file name is tested through the checkExist function to ensure that the name of the output file is not already being used by another file in the directory, and if it is, the user will have the option to either overwrite the current saved file or rename the file to anything they desire. Then, the list will be properly sorted and created, and passed through along with the desired output file name to the outputFile function to write the file in XML format.

Taking in the contents of the file is in O(m) time where m is the number if characters in the file. Removing non-alphanumeric values from the input string and lowercasing all uppercase values happens in O(m+m) time. Tokenizing and inserting the string into a sorted linked list takes place in O(n) time where n is the number of tokens. Finding the frequencies of the tokens and storing them in the linked list happens in O(n+n) time. Removing duplicate nodes happens in O(n) time. Merging lists is done in O(n) time where n is the total number of nodes in both lists. Sorting the nodes based on frequency and file name when the tokens are the same is done using a bubble sort algorithm, so the running time is $O(n^2)$ but majority of the list is already properly sorted when these functions are called so the runtime on average should not get out of hand. Writing the tokens to the output is done in a worst case of $O(n^2)$. Overall our program runs in O(n^2).