Jonathan Lee (jl1424)
Niharika Nagar (nn162)

Our search reads the index file through the function readIndex() which takes the index file, opens it and reads each token that is not “<list>” or “</list>” as either a file or a word. If the word is “<list>” or </list>” pointer arithmetic is implemented to change the starting position of curr. Each word has a linked list of files associated with it which are use to search which files contain the word.

processOr() -
This function runs in O(n^3). It compares each word from the word linked list (wordLL wLL) to the tokens list (char *terms). If there is a match, we traverse the word’s files and create a fileLL. With the fileLL we add to the head of the linked list the name of the file and the word.
With the three while loops, the function runs in O(n^3) time.

processAnd() -
This function is simply a work of art. This function clocks in at a whomping O(n^4). First, it adds all of the file names of the words that match the tokens. This part of the function runs in O(n^3). Afterwards, we rearrange the list so that it is categorized by file rather than word. This is where the processAnd() is magical. We compare each search token to the words of wLL. We then compare the files that were compiled from the previous part of the function to the files of words of wLL. If the files match then we create a new fileLL and store the name of the file and the word. This has a staggering, yet beautiful, quadruple while loop and runs in O(n^4). The last part of this function determines if the files have the search terms together. This function creates sublists for different files. We send this list to goodBadFile() and compares each of the terms in the sublist to the search tokens. Combined with the beginning of this part of the code, the function also runs in O(n^4).
Altogether, this function is (n^3 + n^4 + n^2*n^2) = (n^3 + 2n^4) = O(n^4). Beautiful.

goodBadFile() -
This function determines the good and the bad of any given file that is being judged by processAnd(). It is a double while loop so it runs in O(n^2). If the word matches the search token, then it will increase the numCorrect count. If the numCorrect equals the number of search items then it will return true.

Also, our indexer is now functional.