README

inverted indexes are written to a file in the format of:
<list> term
fileA 2 fileB 5
</list>

variable reference for big-O:
f = number of files
n = total number of words
k = number of parameterized words in search query

The indexer begins by parsing all the files and throwing each word in a hash table. Each insert will be O(f) run time where f is the number of files to be parsed. This worst case will occur when many files have many of the same words and they all get hashed to the back end of the linked list holding file names for the corresponding word. With a total of O(n) inserts, the total run time of this module will be O(nf) and relatively equal amount of space required O(nf).

As the search takes over from the file, the initial pre-processing run time will be O(nf) to take all the words and files from the file into memory with the same memory cost of O(nf). Each word and it's corresponding files and frequencies will be held in a hash table as they were in the indexer.

Once the user fills in a query, the search module will parse through each of the k parameters and dump all the file lists for the words into another hash table which will hold <file_path, hits> pairs to count the number of parameterized words that occur in each file. These hash tables will exist only for the length of a single query, and they will be max as large as O(f). Each query, so and sa will have the same big-O run time. Each parameterized word that is found, will return a list containing the names of files that have that word. Iterating through a list will be in words cast O(f). Each insert of the file name and hit counts into the new temporary hash table will happen in O(1) run time. In total then, we have O(kf) run time, if we assume k will be relatively small, then O(f) run time for the query step.