Task 1: process(filename)

Step 1->Read in the file

The function process will take a text file as input and read in each line of the content. Each line is split into a list and append into an empty list. As each line (starting with a song id follow by a line of lyrics) is read through only once, I can say that the time complexity for it is O(K) where K is the total number of lines in the file.

*The list → [[song id, whole line of lyrics], [song id, whole line of lyrics], ….]

Step 2->Sort the content by song ids

Next, the list is sorted by its song id using counting sort which time complexity is O(N+K) where N is the longest song ID and K is the number of song ids.

Step 2 -> Sort the contents by length of individual words

Then, the lines of lyrics in each nested list is split into individual words, the time complexity for this procedure is O(T) where T is the number of words in the list.

*The list → [[song id, word, word, …], [song id, word, word, …], …]

The list is then sorted by the length of individual words using counting sort which result in a time complexity of O(T+M) where T is the total number of words and M is the longest word.

Step 3 -> Sort the words by alphabet

The item in list is sorted from the least significant alphabet (the last character) towards the most significant alphabet (the first character) by using radix sort. As the words has different lengths, the sorting start from i (the last character of the longest word) and decrease until 0 (the first character of the shortest word). Therefore the sorting goes by the time complexity of O(M)*O(h+T) which is equivalent to O(TM + Mh) where M is the longest word, T is the total number of words, and h is the maximum number of alphabet (in this case is 26 because it only includes lower case a-z)

Step 4 -> Write to file

The time complexity to write to file is in linear time O(U) where U is the total number of unique words. (U is less significant as number of unique words is less than total number of words)

Therefore, the total time complexity is O(K + N + K + T + T + M + TM + Mh + U) . In Big-O notation we would drop the constant and less significant terms, so we end up in O(TM).

Task 2: collate(filename)

This function is to read in a sorted file, collate the ids of each words and eliminate duplicates. The file is assumed to be sorted in alphabetic order and the stability for the song ids of duplicated words is maintain. The song ids of each word will be collated and the duplicates are removed. This process will read in each word and song id pair which will take O(T) where T is the total number of words. For each word we would compare them character by character to determine if we are to concatenate it to the list, this takes O(M)

where M is the longest words. So the resulting time complexity for it will be O(TM). Therefore, the resulting list: [[word, song id, song id, …], [word, song id, song id, …], …]. The list then wrote into an output file with linear time of O(U) where U is total number of items in list.

The total time complexity would be O(TM + U). In Big-O notation we would drop the constant and less significant terms, so we end up in O(TM).

Task3 -> lookup(collated_file, query_file)

This function takes in 2 input files, the content of collated_file is read in and stored in a list. This takes linear time O(U) where U is the number of lines in collated_file. Next, for each word in query_file (time complexity of O(q) where q is the number of words in quey_file, assuming each line only contains a word), we would find its match by using binary search and append the result of binary search to the resulting list. Each item in resulting list is then write into the output file with a time complexity of O(P) where P is the total number of IDs in the output.

Overall, this would take a total time complexity of O(q*Mlog(U)+P) where M is the length of the longest word in any song.