# Assignment-3

Task:1

Radix sort Revenge:

Time Complexity: O(C_I + C_Q + C_P ),

where C_I is the number of characters in data_file

C_Q is the number of characters in query_file

C_P is the number of characters in song_ids.txt

The task is about finding the song ids in which the corresponding word from the query_file appears. First, the file is read, and the words are taken out. In the algorithm, the data is processed and it's taking each word from the data_file and store it with the id inside tuple while query_file is processed to extract each word from it. The trie is implemented which inserts the tuples from the data file and it stores word and ids. However, while searching for the word from query_file, it searches for the word from trie as in the algorithm. The logic that I'm using in order to store the tuples inside trie is that at each node, I'm creating a node containing of 26 indexes.

Once, it searches the word from the tuple, it will return the id value of the corresponding word and output the id values on the same line as the word in the query file. If the word does not appear in any of these lyrics, the search function will output "Not found" instead of any ids.

So, it takes the complexity as follow. The time it takes will be going through each line and the characters in the file which will insert the words with characters into tries. Furthermore, search of words in query_file will take C_Q which is the number of characters in the file. So, each character from the query_file will be searched inside trie as it goes through all characters. Plus, the search function also looks for song_ids for corresponding word in the lyrics. Hence, it will take number of characters in the id file to find the id of the word.

Logic is also taken from the website below to get basic idea of insert and search:

https://www.geeksforgeeks.org/trie-insert-and-search/

Task:2

Most common lyric:

This task is about writing to file which contains the same number of lines as query_file. If the string on line i of query_file is the prefix of a word in any song, then line i of "most_common_lyrics.txt" will contain the word which is present in the most songs b) Has the string on line i of query_file as a prefix.

In the implementation, there is a different trie other than task1 where we are inserting the characters of data_file into insert function in a way such that trie stores all the characters joined in a link and at the end of the word, it returns the word. While, searching for the prefix, the search is implemented to search for all possible prefixes and once it reaches the end of word, it will search for the number of ids it's in using array defined above. Now, it will backtrack by each node and search for other possible words starting with the same prefix. Now, it will also count the number of times the word appears in the id stored above so the counter will keep storing values of the number of times each word appears. Therefore, it has stored the number of occurrences for all possible words being generated from the prefixes in the query_file. Now, the comparison of the counter will take place to see which word has appeared most for the given prefix in the query_file. The word having most counter will be returned from the search function and the most common lyric is returned. If no word is matched from the prefix given, "Not Found" will be returned on the screen.

It takes the time complexity of number of characters. The C_I is the number of characters in data_file so it goes through each characters in the file to insert into tries as it used the children link to insert into nodes and returns the end of word. So, insert function takes number of characters to insert into data_file. In addition, search function takes number of characters again to go through query_file. The prefix will be searched into tries and it will check for words as explained in the algorithm above. So, it takes all the number of characters in the query_file. Plus, it writes all these most appeared words in 'most_common_lyrics.txt' and it will also take this much complexity which includes in the total complexity to run this task.

Time Complexity: O(C_I + C_Q + C_M),

Where, C_I is the number of characters in data_file

C_Q is the number of characters in query_file

C_M is the number of characters in most_common_lyrics.txt

Task:3

Palindrome finding:

In this task, you need to find all palindromic substrings of S, with length at least 2. A palindromic substring is some S[i..j] such that S[j..i] = S[i..j]. For that, the function palindromic_substrings(S) has the time complexity of O(N^2) where N is the length of S string to be inputted.

As in the algorithm, each character is taken into consideration and you have to check for each character to find out where the substrings are matched and return the positions of the substrings. The substring has to be at least length 2. First, it goes from checking in the length of the string. The array is made to store the positions of the palindromic substrings inside the string. It returns the array from it.

Palindrome is searched from the mid of the list for both side of the string to check if the reverse is equal to the string in the order.

Time complexity: O(N^2)

where N is the length of S string to be inputted