Name: Kerry Zheng Student ID: 28794346

FIT2004: Movie Buddies Assignment

Task 1: Finding the top-k users

Let U be the total number of users, k be the number of users that we want to find that has the most time spent on the app, where k is less than or equal to U.

The algorithm first reads the user data from the file into a list of size U. This has O(U) time complexity and O(U) space complexity.

Then it heapifys the list into a max heap using the sift down method, where we start from the last parent node and swap it with the largest child if it has less time spent on the app or has a higher user id number if the time spent is equal to the largest child of the parent node until that sub-heap structure satisfies the heap property. It does this starting from last non-trivial sub-heap to the root. This has O(U) time complexity and O(1) space complexity.

Finally, it gets the max node of the heap, which is at the root, and swaps it with the last node of the heap. It prints the user id and the time spent of the max node. The node at the root is then sifted down using the sift down method until the data structure satisfies the heap property. This has O(logU) time complexity and O(1) space complexity. Because it must print the top-k users, this method is called k so it has an overall time complexity of O(klogU).

So the time complexity of this algorithm is O(U + klogU) = O(klogU). However, because k is less than or equal to U, that means klogU is less than or equal to Ulogk, so it has a time complexity of O(Ulogk) and has space complexity of O(U) as required.

Task 2: Movie Buddy Recommendation

Let U be the total number of users, C be the maximum number of characters in any movie and K be the maximum number of movies liked by a user.

The algorithm reads in the file into a list of size U storing each user’s favourite movies. This has O(UCK) time complexity and O(UCK) space complexity.

Then for each user, it pads the movie’s string with @ at the end so that each movie has the same string length as the movie with the highest number of characters. This is done so that we can perform radix sort. This has O(UCK) time complexity and O(UCK) space complexity.

Next, we perform a least significant digit (LSD) radix sort using a counting sort for each user’s movies so we sort each user’s movies alphabetically. Counting sort has time complexity O(27+K) which is O(K), and space complexity O(CK). Because we sort the entire string, radix sort has O(CK) time complexity and O(CK) space complexity.

Furthermore, because we do it for every user, the overall time complexity is O(UCK) time complexity with O(UCK) space complexity.

After that, we remove the padding from each user’s movie, in O(UCK) time complexity and O(1) space complexity.

Then we concatenate each user’s favourite movies into one big string and pad the string with @ so that each concatenated movie string has the same length, so that we can perform radix sort on it. This has O(UCK) time and space complexity.

After that, we perform radix sort using counting sort (like when we sorted each user’s movies alphabetically) on the concatenated movie string’s so that it is sorted alphabetically. This has O(UCK) time and space complexity.

Finally, we do a linear scan on the sorted list and group user’s together if they have the same concatenated movie string and print accordingly. This has O(UCK) time and space complexity.

Overall, because time and space complexity are represented by big O notation where it ignores constants and lower order terms, this algorithm has O(UCK) time and space complexity as required.