**Application for processing Log Files.**

**Main File name :**

com.amazon.logsprocssing.app.LogProcessingMainApp

**Approach followed and the complexity(Big O):**

1. The main file picks up the file input.txt from the classpath and reads every single line, parses it.
2. We will use buffer reader to read the file line by line I have placed the file at the classpath. If any other path has to be specified, it can be done by changing the input.txt to any other path
3. Once we read a line, we split it using ", " and get time, user action, ip address and user id.
4. Then we validate that the time is in the format HH:mm:ss and the action is either of LOGIN or LOGOUT and IP address is a valid IPv4 address and the user id is alphanumeric for userid.
5. We have written used strategy pattern for implementations of date validation, user id validation, IP address validation so that we have the flexibility to support new date formats, userids apart from being alphanumeric, IPv6 ip addresses as well in the future by just implementing a new strategy.
6. We have used a template pattern for validating the entire log line so that every implementing class validates date, userid, ip address and user action.
7. Once the validation is successful, we parse each valid line and populate the data we are interested in into following collections
8. For tracking “**theIP that received the most distinct User-Logins”,** we keep a map with key as IP address and value as a set of users logged into that IP. On every login action, we keep on adding the userid to the set. Along with it, we keep a class variable maxlogins and maxloginsIp which we populate when reading every line, so that we don’t have to iterate over the map again to get the result.

**COMPLEXITY :** Here the **Time** complexity of getting the ip that received the most distinct user logins is proportional to the number of lines in the log file. i.e. O(n), n being the number of lines in the log file, as we are scanning each log line to populate the map.

The **memory** complexity for it is proportional to the number of unique ip addresses \* number of distinct users that have logged in to that IP. As we are keeping a map with key as ip address and value as set of users that have logged in to that map. i.e. O(n^2).

1. For tracking “**User that at one point had the highest number of sessions open”,** we keep a map with userid as key and number of open sessions as the value. On every logout we keep on decrementing the open sessions and on every login, we keep on incrementing the open sessions. Along with every line, we keep a variable globalmaxopensessions and userwithMaxOpenSessions to keep track of the maximum open sessions and the user having maximum open sessions, so that we don’t have to iterate over the map again to get the final result.

**COMPLEXITY:** Here the **time** complexity will be directly proportional to the number of lines in the log file, as we are keeping a map of user id and the max open sessions and we are scanning each log line to populate the map. So the time complexity is O(n).

Here the **space** complexity is proportional to the number of unique users as we are keeping a map of user id’s . So the space complexity is also O(n).

1. For “**average session length in seconds (time between Login and Logout event for same User) per IP”,**  we again keep a map with IP address as the key and IPTracker object as the value. IP tracker object contains ip address, logged in users and their login times, total sessions and session times. Here we are assuming that if a user has multiple open sessions, the one that was most recently opened will be closed first. We are using a stack here. On reading every line, we populate this map with the ip address, and user details. On logout, we pop the value from the user time stack.

While presenting the final answer, we just iterate over the map and calculate the average session time by dividing total session time by total sessions.

**COMPLEXITY :** Here the **time** complexity is again O(n) as we are scanning each line of log file to populate this map.

The **space** complexity is also O(n) as we are keeping the map with ip address as the key and IPuserTracker as the value.

**ASSUMPTIONS:**

1. Any date/time which is in some format other than HH:mm:ss will be ignored.
2. If the user id is not alphanumeric, or if it doesn't start with an alphabet, it will be ignored.
3. If the IP address is not a valid Ipv4 address, it will be ignored
4. The LOGOUT can only happen after LOGIN. If the login line reads LOGOUT action before LOGIN, it will be ignored.
5. If the user has multiple login sessions and when the user logsout, the most recent session will be logged out. we are using the stack there.