UserActivityLogManager

Design Proposal

**Brendan Viscount**

**CSC316: Data Structures & Algorithms**

**bpviscou@ncsu.edu**

**North Carolina State University**

**Department of Computer Science**

**8 October 2024**

# **System Test Plan**

| **Instructions.** In this section, you must provide your system test plan with at least 5 test cases. **If you want to provide more than 5 test cases, add an appendix at the end of this document with the 6th, 7th, etc. test cases so that page numbers for all sections match the required template! Only the first 5 test cases will be graded.**  Make sure:   * You provide your sample test data * Test IDs are uniquely identified and descriptive * Test descriptions are fully specified with complete inputs, specific values, and preconditions   + Be sure to provide SPECIFIC INPUTs and VALUEs so that your test cases are repeatable * Expected results are fully specified with specific output values * All tests cover scenarios based on the problem statement * All tests cover unique scenarios for the system * All strategies for system testing are demonstrated in the tests (testing equivalence classes, testing boundary values, testing exceptions/unexpected inputs) |
| --- |

**Test Data:**

In *LOGIN1.txt*

hqcooney, 03/02/2015 07:17:42PM, print, office visit OV02132

hqcooney, 10/02/2015 01:04:38AM, print, office visit OV02132

gphsu, 11/13/2015 06:36:48AM, sort, HL7 Code 422

gphsu, 02/22/2016 11:09:06PM, sort, HL7 Code 422

gphsu, 04/18/2016 03:18:42AM, sort, HL7 Code 422

gphsu, 02/27/2017 05:26:50AM, sort, HL7 Code 422

hqcooney, 02/19/2017 06:16:58PM, sort, HL7 Code 422

gphsu, 01/04/2016 12:44:52PM, sort, HL7 Code 422

hqcooney, 11/08/2016 10:43:29AM, sort, HL7 Code 422

gphsu, 07/06/2015 04:17:06PM, sort, HL7 Code 422

hqcooney, 12/18/2017 03:02:54AM, sort, HL7 Code 422

hqcooney, 09/11/2015 09:14:44PM, sort, HL7 Code 422

hqcooney, 04/15/2017 10:59:13PM, sort, HL7 Code 422

hqcooney, 10/06/2016 04:58:44AM, sort, HL7 Code 422

hqcooney, 01/23/2017 12:05:22AM, sort, HL7 Code 422

hqcooney, 09/15/2017 05:44:15AM, unmerge, notification NX1115

hqcooney, 01/04/2016 01:12:22AM, view, HL7 Code 422

In *LOGIN2.txt*

[EMPTY DOCUMENT]

(test data continued)

| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| --- | --- | --- | --- |
| **Test #1**  **testID:**  **testLogUI**  **Strategy:** | **Steps:**   1. Run program | LogUI opens and displays options screen | LogUI Runs and options 1-5 are displayed with a prompt to choose |
| **Test #2**  **testID:**  **testLoadLogData**  **Strategy:** | Preconditions:   * Log UI has opened   Steps   1. Select [1] 2. Type ‘*LOGIN1.txt’* | The UI is populated with the data from the logFile *LOGIN1.txt*, prints message | Displays “Log data loaded successfully” |
| **Test #3**  **testID:**  **Test Top Activities**  **Strategy:**  **valid** | Preconditions:   * Log UI has opened   Steps   1. Select [2] to view most frequent activities 2. Type in ‘10’ to view the 10 most frequented activities | The valid LogEntry is added to the list and appears on the UI |  |
| **Test #4**  **testID:**  **Get Date Logs**  **Strategy:**  **Invalid number** | Preconditions:   * Log UI has opened * Test loadLogData has passed   Steps   1. Select [3] 2. Type in ‘10/20/2022’ | Code displays all logs from the date 10/20/2022 | The proper message displays |
| **Test #5**  **testID:**  **Test getHourLogs**  **Strategy:**  **valid** | Preconditions:   * LogUI has opened * Test loadLogData has passed   Steps   1. Select [4] 2. Type ‘12’ into the hour slot | The UI is populated with all LogEntry objects that have the hour ‘12’ associated |  |

**Algorithm Design**

| **Instructions.** In this section, you should provide pseudocode for the specified algorithms (see the project writeup).  Make sure:   * The algorithms use the abstract data type operations as much as possible * The algorithms contain all necessary pseudocode components to fully describe the algorithm * The algorithms include in-line comments to explain what the algorithm is doing |
| --- |

**Algorithm:**

| Algorithm getMostFrequentActivities(L)  Input L, a list of log entries that are contained in the input file  Output a list of the user activities in L ordered by frequency (most to least)  //Storage queues to count frequency  S <- empty queue  P <- empty queue  U <- empty queue  V <- empty queue  // Result list to store frequencies and activities  resultList <- empty list  //Enqueue each entry in given frequency queue  For i ← 0 to L.size() - 1  ACTION ← logEntry.getAction() // Extract the ACTION  RESOURCE ← logEntry.getResource() // Extract the RESOURCE  activity ← ACTION + " " + RESOURCE // Concatenate  if ACTION == 'sort'  S.enqueue(activity)  else if ACTION == 'print'  P.enqueue(activity)  else if ACTION == 'unmerge'  U.enqueue(activity)  else if ACTION == 'view'  V.enqueue(activity)  // Add the frequencies to the result list in order  if S.size() > 0  resultList.add((S.size(), "sort " + S.getResource()))  if P.size() > 0  resultList.add((P.size(), "print " + P.getResource()))  if U.size() > 0  resultList.add((U.size(), "unmerge " + U.getResource()))  if V.size() > 0  resultList.add((V.size(), "view " + V.getResource()))  // Sort the result list by frequency in descending order  resultList.sortByDescending(frequency)  // Create the result queue  O <- empty queue  // Add the sorted activities to the result queue  For each (frequency, activity) in resultList  O.add(frequency + ": " + activity)  Return O |
| --- |

# 

# **ADTs, Data Structures, & Sorting**

| **Instructions.** You must determine which data structure(s) would be the best choice for implementing an efficient solution to the problem. You must:   * Describe all of the abstract data type(s) you are using * Describe each of the data structure(s) you will use for each ADT. * Briefly justify why you chose the data structure(s) in terms of runtime efficiency. * If you need to sort any data, explain which sorting algorithm you will use, why you chose the specific sorting algorithm, and how you will sort your data structure(s) |
| --- |

**ADTs:**

For this project, I will be using the following abstract data types:

* List ADT
* Queue ADT
* Dictionary
* LogEntry
* Date
* Activity

**Data Structures:**

I will use the following data structures:

* Array-Based List - In order to contain all general LogEntry objects a Array-Based List
* Array-Based Queue - In order to implement the getMostFrequentActivities algorithm there are several queues used, because the only queue operations that are used are enqueue and size, and array-based queue with the front at index 0 will be sufficient

Both Array-Based structures will have dynamic sizing capabilities to grow the structure as needed

In order to provide the proper functionalities for this project the following ADT operations will be implemented:

* add(LogEntry) - add a LogEntry to the list
* getDateLogs(Date) - gets all LogEntry objects with specified date, month, or hour
* getHourLogs(Hour) - gets all LogEntry objects with the specified hour
* getMostFrequentActivities(List) - returns an array of the most frequent activity entries in order

**Sorting:**

The sorting for this project will be done in chronological order, so calendar dates will be compared using mergeSort for consistency and stability. Beyond this times within the same date will be separated ordered by hour using military hours 0-23.

# **Algorithm Analysis**

| **Instructions.** In this section, you should analyze the algorithm you designed earlier (see the project writeup) based on the data structures you selected.  Make sure:   * An estimated running time is provided (based on your selected data structure(s)) and matches the algorithm * All work is shown to justify the estimated running time * The Big-Oh growth rate is provided and is correct |
| --- |

**Algorithm Analysis:**

| **Algorithm:** | **Analysis/Runtime Rationale** |
| --- | --- |
| Algorithm getMostFrequentActivities(L)  Input L, a list of log entries that are contained in the input file  Output a list of the user activities in L ordered by frequency (most to least)  //Storage queues to count frequency  S <- empty queue  P <- empty queue  U <- empty queue  V <- empty queue  // Result list to store frequencies and activities  resultList <- empty list  //Enqueue each entry in given frequency queue  For i ← 0 to L.size() - 1  ACTION ← logEntry.getAction() // Extract the ACTION  RESOURCE ← logEntry.getResource() // Extract the RESOURCE  activity ← ACTION + " " + RESOURCE // Concatenate  if ACTION == 'sort'  S.enqueue(activity)  else if ACTION == 'print'  P.enqueue(activity)  else if ACTION == 'unmerge'  U.enqueue(activity)  else if ACTION == 'view'  V.enqueue(activity)  // Add the frequencies to the result list in order  if S.size() > 0  resultList.add((S.size(), "sort " + S.getResource()))  if P.size() > 0  resultList.add((P.size(), "print " + P.getResource()))  if U.size() > 0  resultList.add((U.size(), "unmerge " + U.getResource()))  if V.size() > 0  resultList.add((V.size(), "view " + V.getResource()))  // Sort the result list by frequency in descending order  resultList.sortByDescending(frequency)  // Create the result queue  O <- empty queue  // Add the sorted activities to the result queue  For each (frequency, activity) in resultList  O.add(frequency + ": " + activity)  Return O | For my getMostFrequentActivities method, the for loop iterates over the size of the parameter list, so that’s O(n) iterations. Because I am using an array-based queue with the front at index 0 to store these, each enqueue operation will only be O(1).  In this case T(n) = O(1) \* n, so the overall runtime for this algorithm is O(n). |

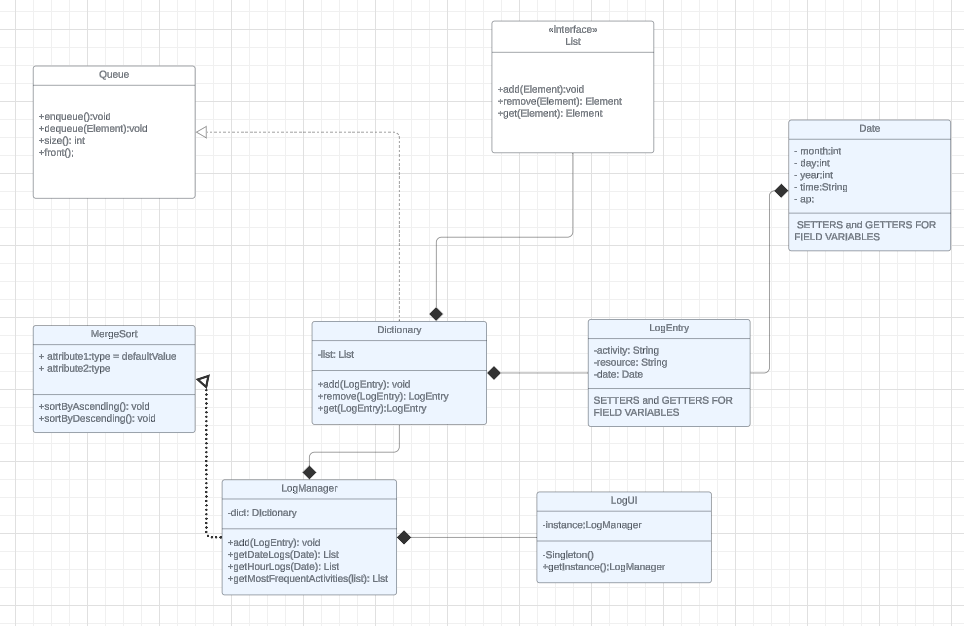
# **Software Design**

| **Instructions**. In this section, you must present your software design for implementing your proposed algorithm. You must provide a UML class diagram.  Make sure:   * All UML notation is correct * All relationships required to implement the system are present * All classes demonstrate high cohesion * The full data structure(s) is/are included in the UML class diagram * The UML class diagram clearly demonstrates or follows a design pattern described by the student |
| --- |

**Brief Description:**

My UML class diagram represents the components of a log analysis system that tracks and organizes user activities by frequency. The main class, LogManager, aggregates LogEntry objects in a List and processes them to find the most frequent activities using a Queue. Each log entry consists of an Action (a combination of action and resource) and a Date for sorting chronologically. The Array-Based List and Array-Based Queue provide dynamic storage for log entries and activities. The system implements functionality to retrieve logs by date, hour, and most frequent activities, using merge-sorting for chronological and frequency ordering.

**UML Class Diagram:**



# Appendix

* If any of your responses from the previous pages overflow the area we have provided in this template for you (for example, if you need more than 2 pages to present your test data from pages 1-2), then continue your responses below in this appendix. This will help ensure your PDF submission still aligns with the template that Gradescope expects.
* If you use this appendix for overflow, be sure to reference the appendix from the relevant sections within your proposals document (for example, include ‘see the appendix for additional test data’ at the end of page 2)
* If all of your responses from the previous pages fit within the areas provided within this template, your appendix here will be empty.