CAB 301: Algorithms and Complexity

Assignment 1
Report and Analysis

Name : Yonathan Cahyadi

Student Number : 10149953

1. Algorithm for "Display the top 10 most frequently borrowed movie DVD"

In this function I decided to use *Merge Sort* as the sorting algorithm. How *Merge Sort* works, according to Wikipedia it works by using divide and conquer technique. The first step is divide the unsorted array into *n* sub-array containing only one element, then repeatedly merge those sub-array using the specified condition (in this case "how frequent the Movie is Borrowed") until there is only one array left. This will be the sorted array.

1.1 Pseudocode

1.1.1 MergeByFrequencyOrder

ALGORITHM MergeByFrequencyOrder*(L[i…j], R[c…d])*

// Merge and sort the 2 sub-array *L[i…j]* and *R[c…d]* into a single sorted array *T[i…d]*

// The sorting is based on "How many times the Movie has been rented"

// initialize the array X to store the result of the merge

resultLength ← *L.Length + R.Length*

T ← *T[resultLength]*

// initialize the *index* for array *A, B* and *T*

leftIndex ← *0* // index for array *L*

rightIndex ← *0* // index for array *R*

resultindex ← *0* // index for array *T*

// iterate through all the element in array *A* and *B*

while *leftIndex < L.Length* **and** *rightIndex < R.Length* **do**

    // Check if both array had an element

    if *leftIndex < L.Length* **and** *rightIndex < R.Length*

        // Sort the element by how many times it has been rented

        // if the *L* at *leftIndex* has been rented **less than or equal** *R* at *rightIndex*

        if *L[leftIndex].timesRented <= R[rightIndex].timesRented*

            // put the element of array *L* at *leftIndex* into array *T* at *resultindex*

            T[resultindex] ← *L[leftIndex]*

            leftIndex ← *leftIndex + 1*

            resultindex ← *resultindex + 1*

        else // array *L* at *leftIndex* has been rented **more than** *R* at *rightIndex*

```
                    // put the element of array R at rightIndex into array T at resultindex
                    T[resultindex] ← R[rightIndex]
                    rightIndex ← rightIndex + 1
                    resultindex ← resultindex + 1
        else if leftIndex < L.Length // put the remaining element of array L into array T
                T[resultindex] ← L[leftIndex]
                leftIndex ← leftIndex + 1
                resultindex ← resultindex + 1
        else if rightIndex < R.Length // put the remaining element of array R into array T
                T[resultindex] ← R[rightIndex]
                rightIndex ← rightIndex + 1
                resultindex ← resultindex + 1
    return T
```

1.1.2 GetSortedMovieByFrequencyOrder

```
ALGORITHM GetSortedMovieByFrequencyOrder(X[i…d])
    // Divide the array X into sub-array and do the Merge and Sorting algorithm
    // The result will be stored in array T
    if  X.Length <= 1  // if the array X only had one element return the array
            return X
    // get the mid point of the array X
    midPoint ← X.Length / 2

    // initialize array to store the Result
    T ← T[X.Length]

    // initialize 2 sub-array L and R
    L ← L[midPoint]
    If X.Length % 2 == 0 // if the Length of array X is even
            R ← R[midPoint]
    else // if the Length of array X is odd
```

```
        R ← R[midPoint + 1]

    // populate the left array using elements from X[i…midPoint]
    for index ← 0 to midPoint do
        L[index] ← X[index]
    // populate the right array using elements from X[midPoint…d]
    rightSubArrayIndex ← 0
    for index ← midPoint to X.Length do
        R[rightSubArrayIndex] ← X[index]
        rightSubArrayIndex ← + 1

    // recursively divide the array until there is 1 element left
    L ← GetSortedMovieByFrequencyOrder(L)
    R ← GetSortedMovieByFrequencyOrder(R)

    // merge L and R into one sorted array
    T ← MergeByFrequencyOrder(L, R)
    return T
```

1.1.3 AddToArray

```
ALGORITHM AddToArray(BinaryTreeNode, A[0…numberOfRegisteredDVD], index)
    // This algorithm will traverse through all the Node in Binary Tree and transfer all the Node
    // into an array A[0…numberOfRegisteredDVD]
    if BinaryTreeNode == null // if the current node is empty return the index
        return index
    A[index] ← BinaryTreeNode.data
    index ← index + 1

    // Traverse through all the Binary Tree Node and put the data into the array A
    if BinaryTreeNode.left != null
        index ← AddToArray(BinaryTreeNode.left, A, index)
    if BinaryTreeNode.right != null
```

```
        index ← AddToArray(BinaryTreeNode.right, A, index)
    return index
```

1.1.4 GetAllMovie

```
ALGORITHM GetAllMovie()
    // This algorithm will get all the Movie from Movie Collection Binary Tree
    if binaryTreeRoot == null
            return null
    R ← R[numberOfRegisteredDVD] // Initialize array with size of Registered DVD
    AddToArray(binaryTree, R, 0)
    return R
```

1.1.5 MemberGetTop10

```
ALGORITHM MemberGetTop10()
    // This will display the Top 10 most borrowed DVD in descending order
    A ← GetAllMovie() // Get array containing all registered the Movie

    if A != null // Check if there is any movie registered in movie collection
            T ← GetSortedMovieByFrequencyOrder(A) // Get the sorted version of array A
            // iterate through all the array T
            for index ← T.Length – 10 to T.Length do
                    // Check if the index is bigger or equals to 0
                    // To avoid index out of bound
                    if index >= 0
                            output T[index].title
                            output T[index].starring
                            output T[index].director
                            output T[index].genre
                            output T[index].duration
                            output T[index].releaseDate
                            output T[index].classification
                            output T[index].timesRented
```

index ← *index + 1*

else

output "*No DVD Registered*"

1.2 Code Implementation

1.2.1 MergeByFrequencyOrder

```
/// <summary>
/// Merge to 2 sub-array together into 1 array by Frequency
/// </summary>
/// <param name="left">Left Array</param>
/// <param name="right">Right Array</param>
/// <returns>Return Sorted an Merged Array</returns>
1 reference
private Movie[] MergeByFrequencyOrder(Movie[] left, Movie[] right) {
    int resultLength = left.Length + right.Length; // get the length of the reulted array from the 2 sub-array
    Movie[] result = new Movie[resultLength]; // init the result array for later use

    int indexLeft = 0, indexRight = 0, indexResult = 0;

    // itterate all element in left and right array
    while(indexLeft < left.Length || indexRight < right.Length) {

        //if both array had element
        if(indexLeft < left.Length && indexRight < right.Length) {
            // Sort the element by how many times it has been rented
            // If the left Movie array is has been rented less than the right Movie array
            // Put the left array movie into the Result array
            if (left[indexLeft].timesRented <= right[indexRight].timesRented) {
                result[indexResult] = left[indexLeft];
                indexLeft++;
                indexResult++;
            } else { // otherwise, put the right side Movie array into the Result array
                result[indexResult] = right[indexRight];
                indexRight++;
                indexResult++;
            }
        } else if (indexLeft < left.Length) { //if only the left array still has elements, add all its elements to the results array
            result[indexResult] = left[indexLeft];
            indexLeft++;
            indexResult++;
        } else if (indexRight < right.Length) { //if only the right array still has elements, add all its elements to the results array
            result[indexResult] = right[indexRight];
            indexRight++;
            indexResult++;
        }
    }
    return result;
}
```

1.2.2 GetSortedMovieByFrequencyOrder

```
/// <summary>
/// Get all DVD but sorted by how many time its been rented
/// The sorting algorithm used is Merge Sort
/// </summary>
/// <param name="allAvailableMovie"></param>
/// <returns>Sorted Movie Array, Sorted by borrowed Frequency</returns>
3 references
public Movie[] GetSortedMovieByFrequencyOrder(Movie[] allAvailableMovie) {
    Movie[] result = new Movie[allAvailableMovie.Length]; // Create an array for the Merge Sorting Result

    // Made 2 sub array for later use
    Movie[] left;
    Movie[] right;

    // Check the length of the inputed array
    // if the length is 1 then return it back
    if (allAvailableMovie.Length <= 1) return allAvailableMovie;

    // get the mid point of our array
    int midPoint = allAvailableMovie.Length / 2;
    // initialize the left sub-array
    // so the number of array in the left sub-array is [0 - midPoint]
    left = new Movie[midPoint];

    // initialize the right sub-array
    // so the number of array in the right sub-array is [0 - midPoint] if even
    // [0 - midPoint + 1] if odd
    if(allAvailableMovie.Length % 2 == 0) {
        right = new Movie[midPoint];
    } else {
        right = new Movie[midPoint + 1];
    }

    // populate the left array using left side array param as the value
    // the element is from [0 - midPoint]
    for(int i = 0; i < midPoint; i++) {
        left[i] = allAvailableMovie[i];
    }
    //poulate the right array using right side array param as the value
    // the element is from [midPoint - allAvailableMovie length]
    int rightSubArrayIndex = 0;
    for(int i = midPoint; i < allAvailableMovie.Length; i++) {
        right[rightSubArrayIndex] = allAvailableMovie[i];
        rightSubArrayIndex++;
    }

    // recursively divide the array until there is only 1 element left in the sub-array
    left = GetSortedMovieByFrequencyOrder(left);
    right = GetSortedMovieByFrequencyOrder(right);

    // merge the sub-array together into 1 sorted array
    result = MergeByFrequencyOrder(left, right);

    return result;
}
```

1.2.3 AddToArray

```
/// <summary>
/// Transfer Binary Tree Node into an array
/// </summary>
/// <param name="node">the node that will be traversed</param>
/// <param name="array">Array to store the Node</param>
/// <param name="i">current index of the array</param>
/// <returns></returns>
3 references
private int AddToArray(MovieNode node, Movie[] array, int i) {
    if(node == null) {
        return i;
    }

    array[i] = node.data;
    i++;

    // traverse through all the Binary Tree Node and Put the Node into an array
    if (node.left != null)
        i = AddToArray(node.left, array, i);
    if (node.right != null)
        i = AddToArray(node.right, array, i);

    return i;

}
```

1.2.4 GetAllMovie

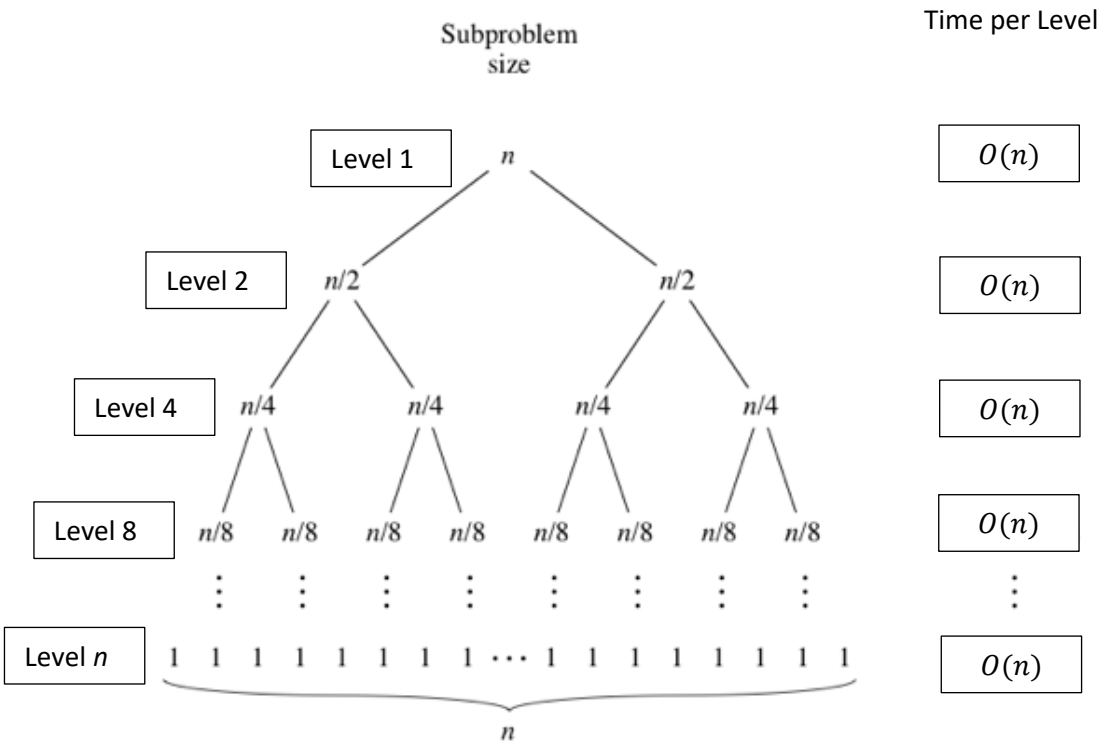
```
/// <summary>
/// Get all the Movie in the Binary Tree
/// </summary>
/// <returns>Array of all Movie Contained in the Binary Tree</returns>
6 references
public Movie[] GetAllMovie() {

    if (_root == null) return null;

    Movie[] result = new Movie[numberOfDVD];


    AddToArray(_root, result, 0);
    return result;

}
```

1.2.5 MemberGetTop10

```
/// <summary>
/// Show the top 10 most borrowed DVD, in descending order
/// </summary>
1 reference
private static void MemberGetTop10() {
    Movie[] allMovie = _movieCollection.GetAllMovie(); // get all the movie registered

    Console.Clear();
    Console.WriteLine("========== Top 10 DVD =========\n");

    if (allMovie != null) {

        // get the sorted version of the movie registered, sorted my frequency
        Movie[] sortedMovieArray = _movieCollection.GetSortedMovieByFrequencyOrder(allMovie);

        // Display the sorted array from the bottom up
        // this way we get the array to show in descending order
        for (int i = sortedMovieArray.Length - 10; i < sortedMovieArray.Length; i++) {
            // check if the i is bigger or equals 0, to avoid index out of bound.
            if (i >= 0) {
                Console.WriteLine("Title          : " + sortedMovieArray[i].title);
                Console.WriteLine("Starring       : " + sortedMovieArray[i].starring);
                Console.WriteLine("Director       : " + sortedMovieArray[i].director);
                Console.WriteLine("Genre          : " + sortedMovieArray[i].genre);
                Console.WriteLine("Duration       : " + sortedMovieArray[i].duration);
                Console.WriteLine("Release Date   : " + sortedMovieArray[i].releaseDate);
                Console.WriteLine("Classification : " + sortedMovieArray[i].classification);
                Console.WriteLine("Times Rented   : " + sortedMovieArray[i].timesRented);
                Console.WriteLine();

            }
        }

    } else {
        Console.WriteLine("No DVD Registered");
    }
    Console.WriteLine("\n==============================\n\n");
    Console.ReadKey();

    MemberMenu();
}
```

2. Analysis of the Time Complexity

2.1 In Theory



[Merge Sort Visualization]. https://cdn.kastatic.org/ka-perseus-images/5fcbebf66560d8fc490de2a0d8a0e5b1d65c5c54.png

From the images above we can see that Merge Sort works by dividing each problem into a smaller sub-problem. The sub-problem will be divided until there is only 1 element left, and every time the sub-problem divided it will take a ***time proportional to its number of element*** left in the sub-array so the time needed per level to divide the sub-problem into smaller ploblem is $\boldsymbol{O(n)}$. In addition the level needed to divide a problem size n into sub-problem size 1 is unknown so we can assume it need to keep dividing until n $level$. Therefore we can tell that the time needed for a problem with problem size n until it reach sub-problem with a size of 1 is $\boldsymbol{O(\log n)}$, because in each level we halve the problem size until it reach problem size of 1. So the total time needed for merge sort finished is:

$$The\ time\ needed\ by\ each\ level\ to\ finish\ an\ operation$$

$$\mathrm{X}$$

$$The\ time\ needed\ to\ divide\ the\ problem\ size\ n\ into\ problem\ size\ 1$$

and we know that,

$The\ time\ needed\ by\ each\ level\ to\ finish\ an\ operation\ is\ \boldsymbol{O(n)}$

$The\ time\ needed\ to\ divide\ the\ problem\ size\ n\ into\ problem\ size\ 1\ is\ \boldsymbol{O(\log n)}.$

Therefore we can do:

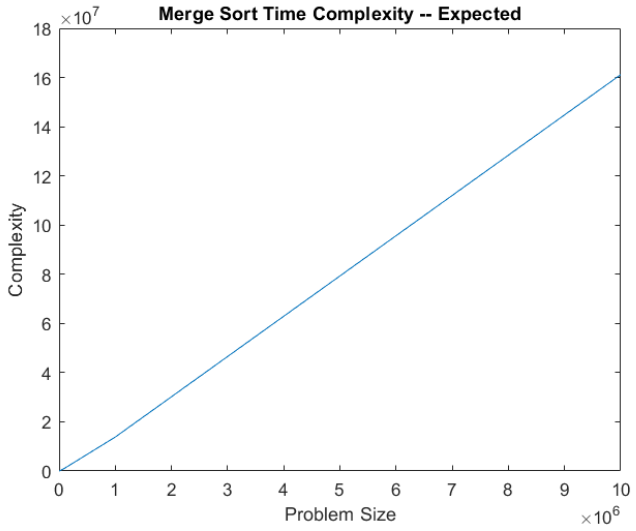
$$\boldsymbol{O(n)} * \boldsymbol{O(\log n)}$$

Which is

$$\boldsymbol{O(n \log n)}$$

For **best, average** and **worst-case time** complexity.

In Graph it will be shown as,



2.2 Implementation

2.2.1 MergeByFrequenceyOrder

`int resultLength = left.Length + right.Length;`	$O(1)$
`Movie[] result = new Movie[resultLength];`	$O(1)$
`int indexLeft = 0, indexRight = 0, indexResult = 0;`	$O(1)$
`while(indexLeft < left.Length || indexRight < right.Length)`	$O(n)$
`    if(indexLeft < left.Length && indexRight < right.Length)`	$O(1)$
`        if (left[indexLeft].timesRented <= right[indexRight].timesRented)`	$O(1)$
`            result[indexResult] = left[indexLeft];`	$O(1)$
`            indexLeft++;`	$O(1)$
`            indexResult++;`	$O(1)$
`         else`	$O(1)$
`            result[indexResult] = right[indexRight];`	$O(1)$
`            indexRight++`	$O(1)$
`            indexResult++;`	$O(1)$
`     else if (indexLeft < left.Length)`	$O(1)$
`        result[indexResult] = left[indexLeft];`	$O(1)$
`        indexLeft++;`	$O(1)$
`        indexResult++;`	$O(1)$
`    else if (indexRight < right.Length)`	$O(1)$
`        result[indexResult] = right[indexRight];`	$O(1)$
`        indexRight++;`	$O(1)$
`        indexResult++;`	$O(1)$
`return result;`	$O(1)$

From the table we can see that the time complexity of this function is $\boldsymbol{O(n)}$.

2.2.2 GetSortedMovieByFrequencyOrder

`Movie[] result = new Movie[allAvailableMovie.Length];`	$O(1)$
`Movie[] left;`	$O(1)$

Movie[] right;	$O(1)$
if (allAvailableMovie.Length <= 1)	$O(1)$
return allAvailableMovie;	$O(1)$
int midPoint = allAvailableMovie.Length / 2;	$O(1)$
left = new Movie[midPoint];	$O(1)$
if(allAvailableMovie.Length % 2 == 0)	$O(1)$
right = new Movie[midPoint];	$O(1)$
else	$O(1)$
right = new Movie[midPoint + 1];	$O(1)$
for(int i = 0; i < midPoint; i++)	$O(n)$
left[i] = allAvailableMovie[i];	$O(1)$
int rightSubArrayIndex = 0;	$O(1)$
for(int i = midPoint; i < allAvailableMovie.Length; i++)	$O(n)$
right[rightSubArrayIndex] = allAvailableMovie[i];	$O(1)$
rightSubArrayIndex++;	$O(1)$
left = GetSortedMovieByFrequencyOrder(left);	$O(\log n)$
right = GetSortedMovieByFrequencyOrder(right);	$O(\log n)$
result = MergeByFrequencyOrder(left, right);	$O(n \log n)$
return result;	$O(1)$

From this table we can see that the time complexity of this function is $\boldsymbol{O(n \log n)}$.

2.2.3 AddToArray

if(node == null)	$O(1)$
return i;	$O(1)$
array[i] = node.data;	$O(1)$
i++;	$O(1)$
if (node.left != null)	$O(1)$
i = AddToArray(node.left, array, i);	$O(n)$
if (node.right != null)	$O(1)$
i = AddToArray(node.right, array, i);	$O(n)$
return i;	$O(1)$

From this table we can see that the time complexity of this function is $O(n)$.

2.2.4 GetAllMovie

if (_root == null)	$O(1)$
return null;	$O(1)$
Movie[] result = new Movie[numberOfDVD];	$O(1)$
AddToArray(_root, result, 0);	$O(n)$
return result;	$O(n)$

From this table we can see that the time complexity of this function is $\boldsymbol{O(n)}$.

2.2.5 MemberGetTop10

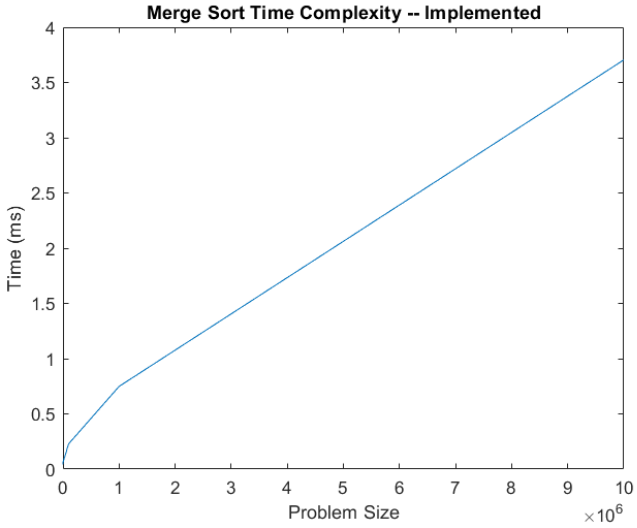
Movie[] allMovie = _movieCollection.GetAllMovie();	$O(n)$
Console.Clear();	$O(1)$
Console.WriteLine("========== Top 10 DVD =========\n");	$O(1)$

`if (allMovie != null)`	$O(1)$
`Movie[] sortedMovieArray = _movieCollection.GetSortedMovieByFrequencyOrder(allMovie);`	$O(n \log n)$
`if (i >= 0)`	$O(1)$
`Console.WriteLine("\tTitle : " + sortedMovieArray[i].title);`	$O(1)$
`Console.WriteLine("\tStarring : " + sortedMovieArray[i].starring);`	$O(1)$
`Console.WriteLine("\tDirector : " + sortedMovieArray[i].director);`	$O(1)$
`Console.WriteLine("\tGenre : " + sortedMovieArray[i].genre);`	$O(1)$
`Console.WriteLine("\tDuration : " + sortedMovieArray[i].duration);`	$O(1)$
`Console.WriteLine("\tRelease Date : " + sortedMovieArray[i].releaseDate);`	$O(1)$
`Console.WriteLine("\tClassification : " + sortedMovieArray[i].classification);`	$O(1)$
`Console.WriteLine("\tTimes Rented : " + sortedMovieArray[i].timesRented);`	$O(1)$
`Console.WriteLine();`	$O(1)$
`else`	$O(1)$
`Alert("No DVD Registered");`	$O(1)$
`Console.WriteLine("\n================================\n\n");`	$O(1)$
`Console.ReadKey();`	$O(1)$
`MemberMenu();`	$O(1)$

From this table we can see that this function time complexity is $\boldsymbol{O(n \log n)}$.

2.2.6 The Conclusion

From all the analysis above, we know that the function needed to get the top 10 most borrowed DVD have a Time Complexity of $\boldsymbol{O(n \log n)}$. In a graph it will be shown as:

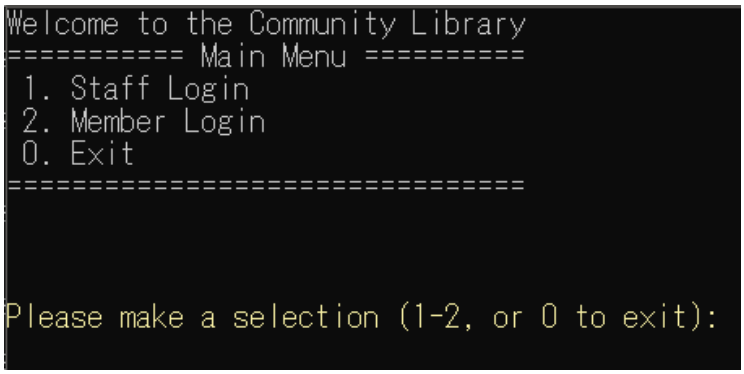


The graph looks almost the same with the graph discussed in the Theory Section [2.1]. Therefore, I can conclude that my Merge Sort implementation is correct and up to the expected time efficiency.

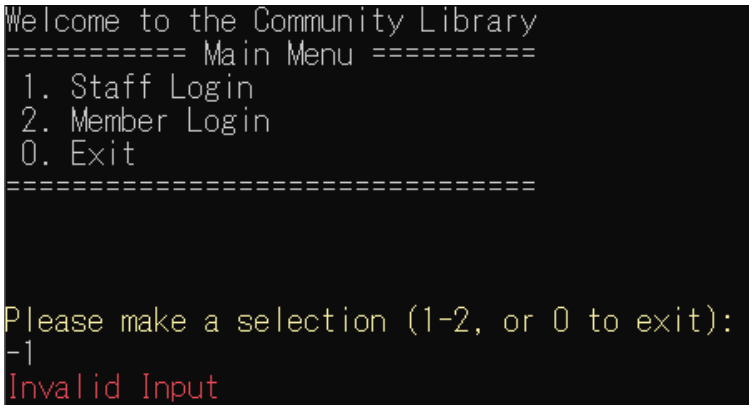
3. Screenshots of each Functional test

3.1 Main Menu

3.1.1 Main Menu Selection

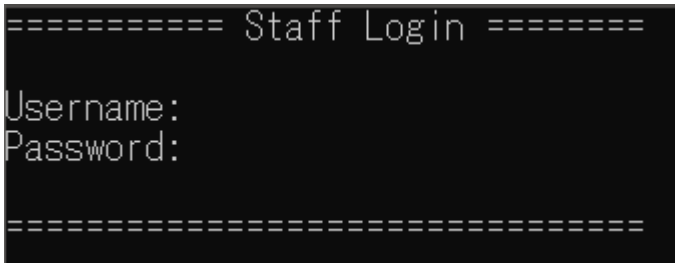


3.1.2 Main Menu Invalid Selection



3.2 Staff Login

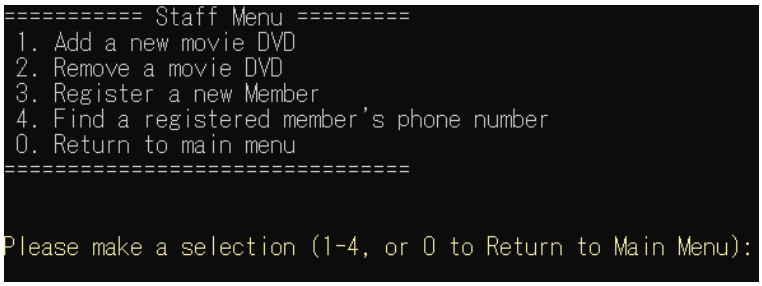
3.2.1 Staff Login



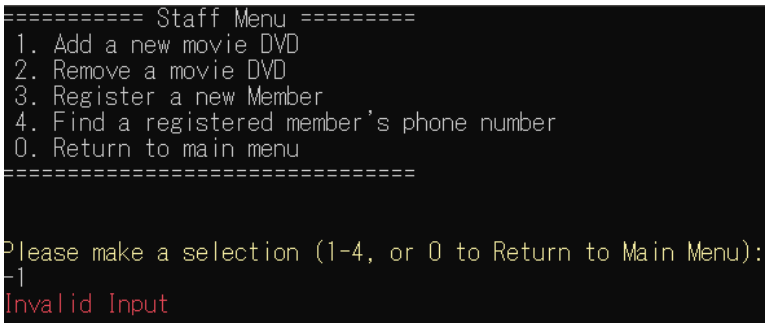
3.2.2 Staff Login with Username and Password

3.3 Staff Menu

3.3.1 Staff Menu Selection



3.3.2 Staff Menu Invalid Selection

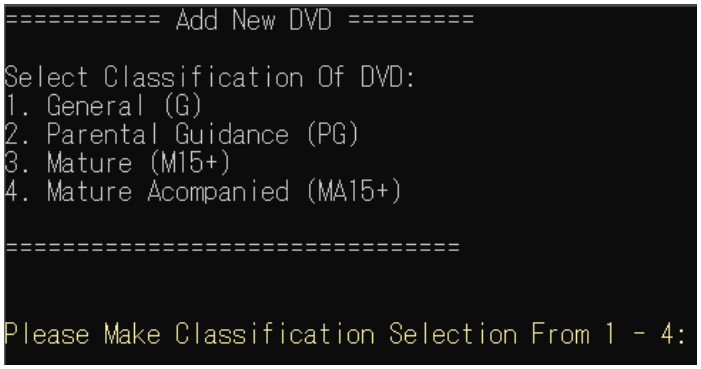


3.4 Add a new movie DVD

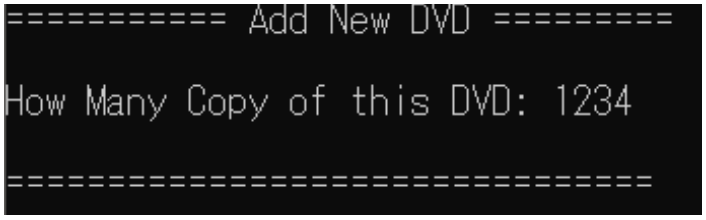
3.4.1 Add new DVD Filling the DVD Title, Director, Starring, Release Date and Duration

3.4.2 Select DVD Genre

3.4.3 Select DVD Classification



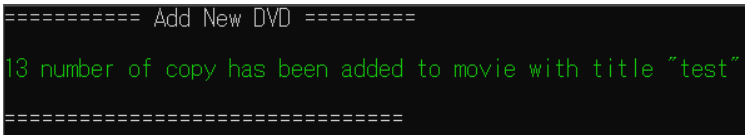
3.4.4 Add Number of Available DVD Copy



3.4.5 Add DVD into the Movie Collection

3.4.6 DVD Added to movie Collection Successfully

2.4.7 DVD With the Same Title in the Movie Collection is added



3.5 Remove a movie DVD

3.5.1 Remove DVD

3.5.2 DVD is Not in the Collection

3.5.3 DVD is Successfully Removed

```
========== Remove Movie =======

Movie Deleted.

================================
```

3.6 Register New Member

3.6.1 Add New Member

```
========== Register Member =======

First Name   :test
Last Name    :member
Phone Number :0123456
Address      :qwert
Password     :1234

================================
```

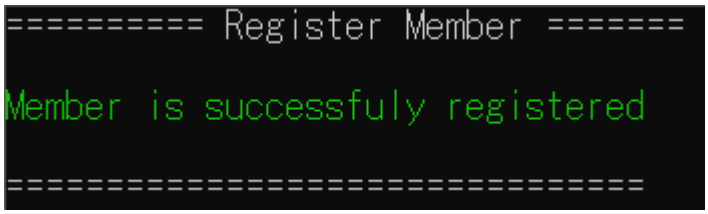
3.6.2 Password Less than 4 digit or Not an Interger

```
========== Register Member =======

First Name   :test
Last Name    :member
Phone Number :0123456
Address      :qwert
Password     :123s

==============================

Password is Invalid -- Must be atleast 4 digit Interger
```

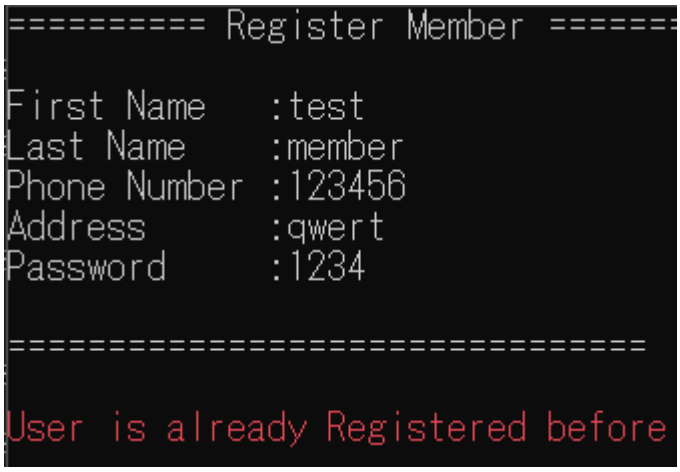
3.6.3 Adding New User into Member Collection and Generating Member Username

```
========== Register Member =======

First Name  : test
Last Name   : member
Phone Number: 123455
Address     : qwert

Username    : membertest
Password    : 1234

================================


Register New Member [y/n]:
y
```

3.6.4 New Member is Successfully Registered and added into the Member Collection



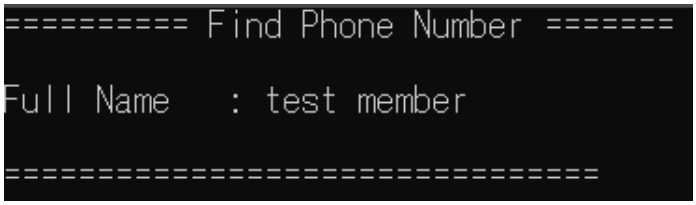
3.6.5 Member is Already Registered Before



3.6.6 Member Collection is Already Full

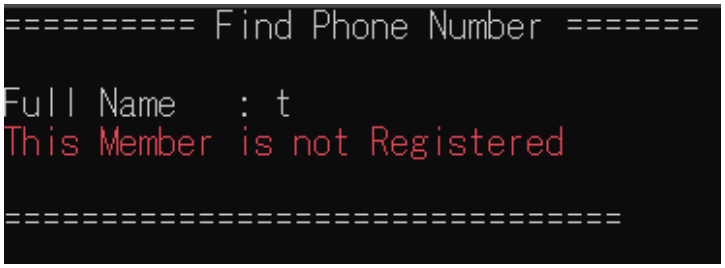
3.7 Find a Registered Member's Phone Number

3.7.1 Find Member Phone Number

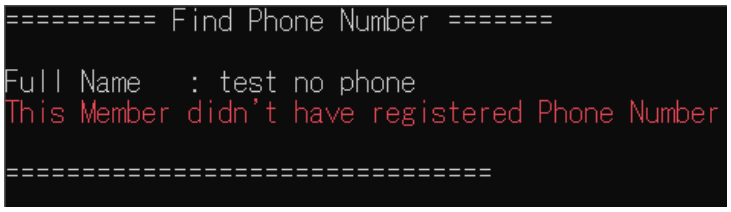


3.7.2 User Founded

3.7.3 User is Not Registered

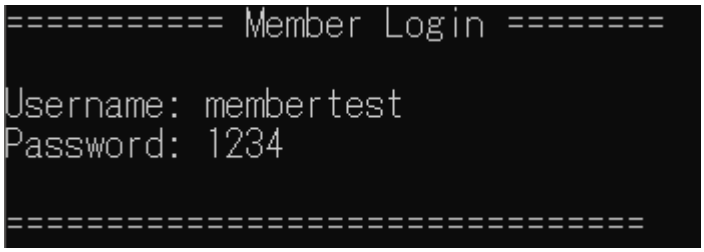


3.7.4 User didn't have Registered Phone Number



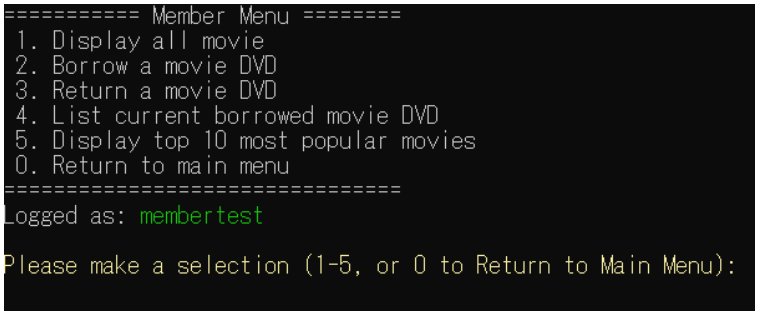
3.8 Member Login

3.8.1 Member Login

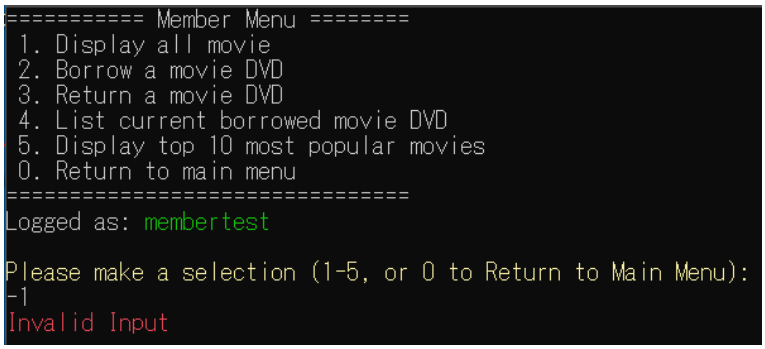


3.9 Member Menu

3.9.1 Member Menu Selection



3.9.2 Member Menu Invalid Selection



3.10 Display All Movie

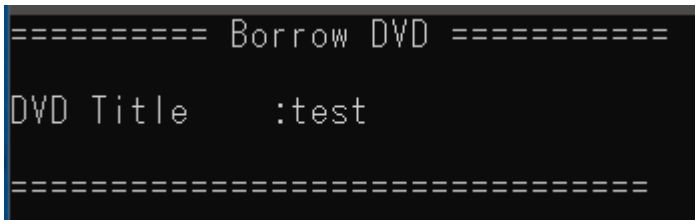
3.10.1 No Movie Registered

3.10.2 Display All Registered Movie

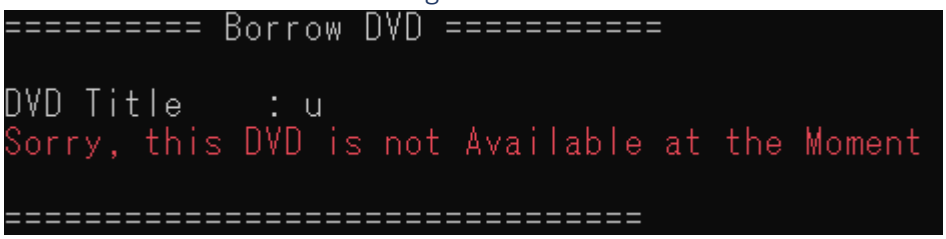
```
============ Movies ===========

1 ---   Title          : --
        Starring       : w
        Director       : r
        Genre          : Animated
        Duration       : 1234
        Release Date   : qw
        Classification : Mature
        Available Copy : 43
        Times Rented   : 0

2 ---   Title          : ??
        Starring       :
        Director       :
        Genre          : Adventure
        Duration       : 3
        Release Date   :
        Classification : ParentalGuidance
        Available Copy : 13
        Times Rented   : 0

3 ---   Title          : lw
        Starring       : 3
        Director       :
        Genre          : Animated
        Duration       : 13
        Release Date   : 2
        Classification : ParentalGuidance
        Available Copy : 32
        Times Rented   : 0

4 ---   Title          : qwer
        Starring       : w
        Director       : r
        Genre          : Drama
        Duration       : 132
        Release Date   : de
        Classification : Mature
        Available Copy : 135
        Times Rented   : 0

5 ---   Title          : rest
        Starring       : re
        Director       : e
        Genre          : Comedy
        Duration       : 13
        Release Date   : we
        Classification : MatureAccompanied
        Available Copy : 24
        Times Rented   : 0

6 ---   Title          : t
        Starring       :
        Director       :
        Genre          : Comedy
        Duration       :
        Release Date   :
        Classification : MatureAccompanied
        Available Copy : 24
        Times Rented   : 0

7 ---   Title          : test
        Starring       : test star
        Director       : test dir
        Genre          : Action
        Duration       : 100
        Release Date   : test rel
        Classification : Mature
        Available Copy : 23
        Times Rented   : 0

8 ---   Title          : test#
        Starring       : sd
        Director       : re
        Genre          : Animated
        Duration       : 134
        Release Date   : 13
        Classification : Mature
        Available Copy : 13
        Times Rented   : 0

9 ---   Title          : test1
        Starring       : s
        Director       : t
        Genre          : Comedy
        Duration       : d
        Release Date   : r
        Classification : Mature
        Available Copy : 43
        Times Rented   : 0


==================================
```

3.11 Borrow a Movie

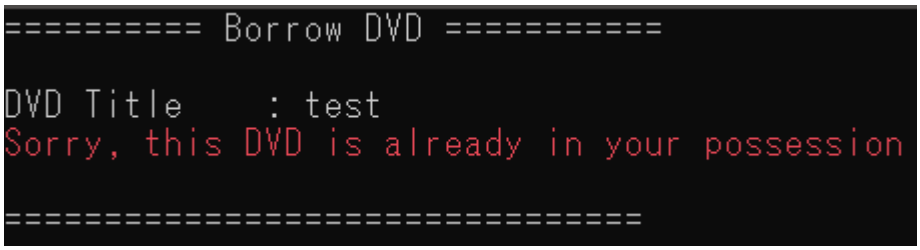
3.11.1 Borrow a Movie



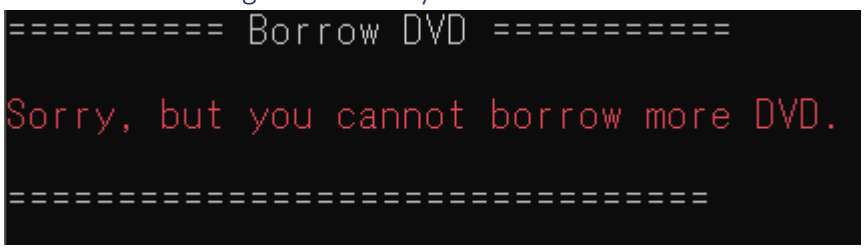
3.11.2 Movie is Not Available for Renting



3.11.3 Movie is Already in Possession

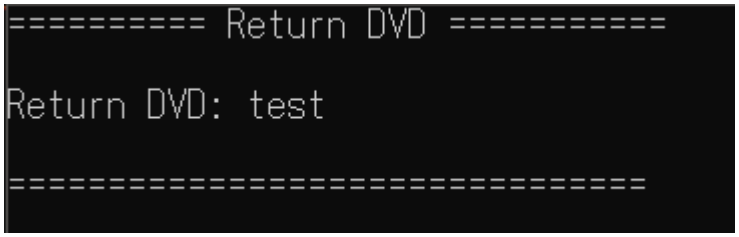


3.11.4 Member Borrowing Limit Already Reached

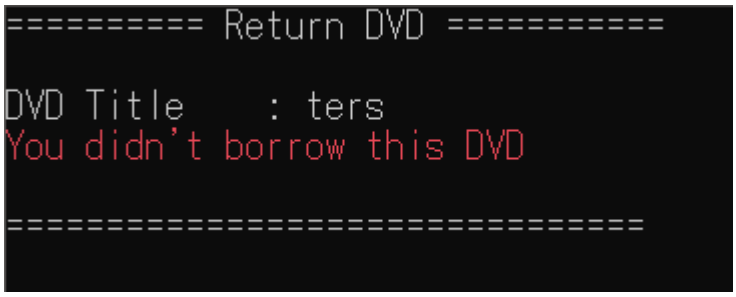


3.12 Return a Movie

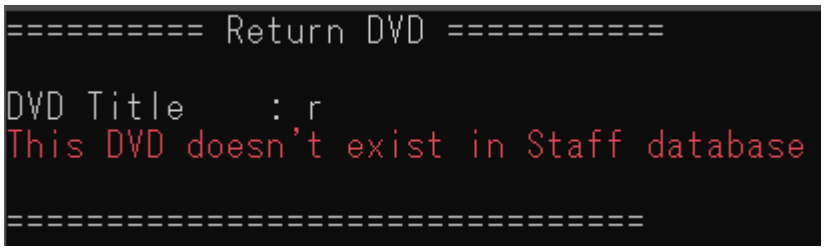
3.12.1 Return a Movie



3.12.2 Movie is Not Borrowed

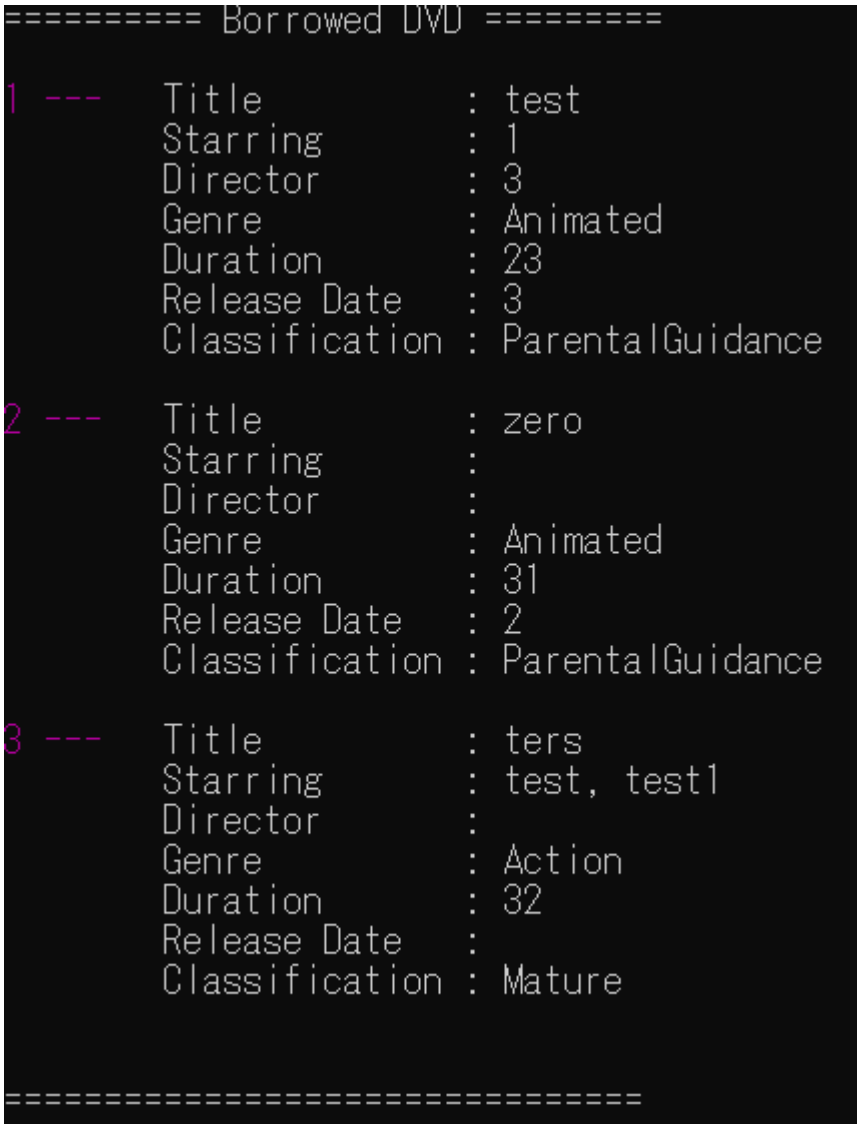


3.12.3 DVD is Deleted by Staff, but Still in Member Possession

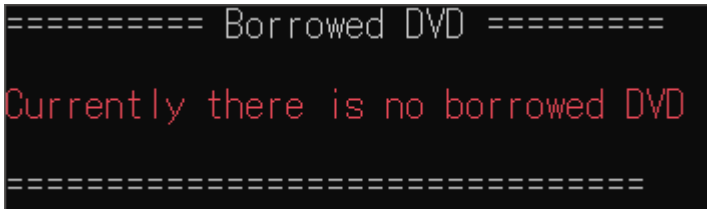


3.13 List All Borrowed Movie DVD

3.13.1 List All Borrowed Movie DVD



3.13.2 No DVD Borrowed



3.14 Display top 10 Most Popular Movie

3.14.1 List of top 10 Most Popular Movie in Descending Order

3.14.2 No Movie is Registered

4. Reference

Wikipedia. *Merge Sort*. Retrieved May 22, 2020, from https://en.wikipedia.org/wiki/Merge_sort