CAB301- Assignment Report

Name: Hieu Nghia Huynh

Student ID: n10315071

1) Algorithm

- To solve the “display top 10 most frequently borrowed movie” situation, I add some additions to my tree-traversal code. A movie collection is built upon a BST structural, hence I use In-order traversal to go through the tree.
- Before the traversal, I check how many movies are there in the MovieCollection by calculating the tree’s height. If the height is smaller than 10, I’ll display top ***H*** most frequently borrowed movies. Let ***N*** is either 11 or ***H***
- Create a Movie array to store the result (we call it resultArray: array of ***N*** elements), initializing them with null then start the tree-traversal.
- For each node we are checking during tree-traversal, if the node is null, we return the movie array, else we compare the borrowedTimes of the current node to each element of the resultArray. If the comparing element in the resultArray is null, we add the current Node to the position, else we find the first element in the resultArray to be smaller than the current Node then we insert the current Node to that position. In the end, the resultArray will contain top 10 movies with most borrowed times.
- Pseudo code:

--- Calling out top 10 method:

```
void top10()
  h = getHeight(Root);
  if (h < 10)
      n = h
  else
      n = 10
  resultList = mostBorrow(Root, ref resultList);
  for i <- 0 to n do
      print(resultList[i])
```

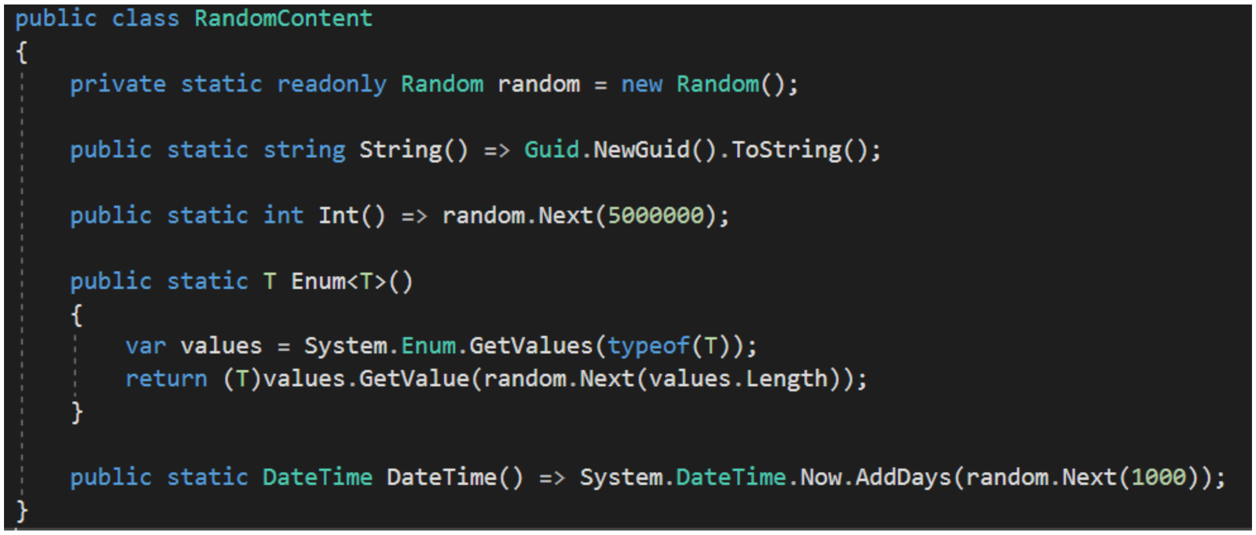
--- Get the current tree's height function:
```
 int getHeight(Node root)
      if (root = null) return 0;
      lh <- getHeight(root.left);
      rh <- getHeight(root.right);
      return 1 + Max(lh, rh);
```
--- Most borrow function:
```
mostBorrow(Node root, ref Movie[] mvList)
      if (root = null)
            return mvList
      else
          mvList <- mostBorrow(root.left, ref mvList);
            curr <- root.movie.borrowTimes;
            for i <- 0 to mvList.Length do                {
                if (mvList[i] = null)
                    mvList[i] = root.movie;
                    break;
                else
                    if (mvList[i].borrowTimes < curr)
                    for  j <- mvList.Length - 1 downto i+1 do
                         mvList[j] <- mvList[j - 1];
                    mvList[i] <- root.movie;
            mvList = mostBorrow(root.right, ref mvList);
      return mvList;
```
The mvList returned will be the top 10 borrowed movies.

2) Algorithm Analysis

- For calculating the tree's height, the complexity would be O(n) with n is the number of nodes.
- Foreach node of the BST, I used 2 nested for-loops : 1 to go through the result array to check for the correct position, 1 to move all the elements on the right of the correct position's element (including that element) to the right 1 unit, then add the current node to the correct position. The worst case for this 2 loops would be O(11^2). The worst case for the mostBorrow function would be O(n * 11^2) = O(n). Since the 2 loops are too small compare to N, the complexity for this function is O(n).
- The last *for* loop would have worst case complexity of O(10) .
- Hence: The total complexity for the worst case of this algorithm would be O(n) + O(n) + O(n) = O(n)

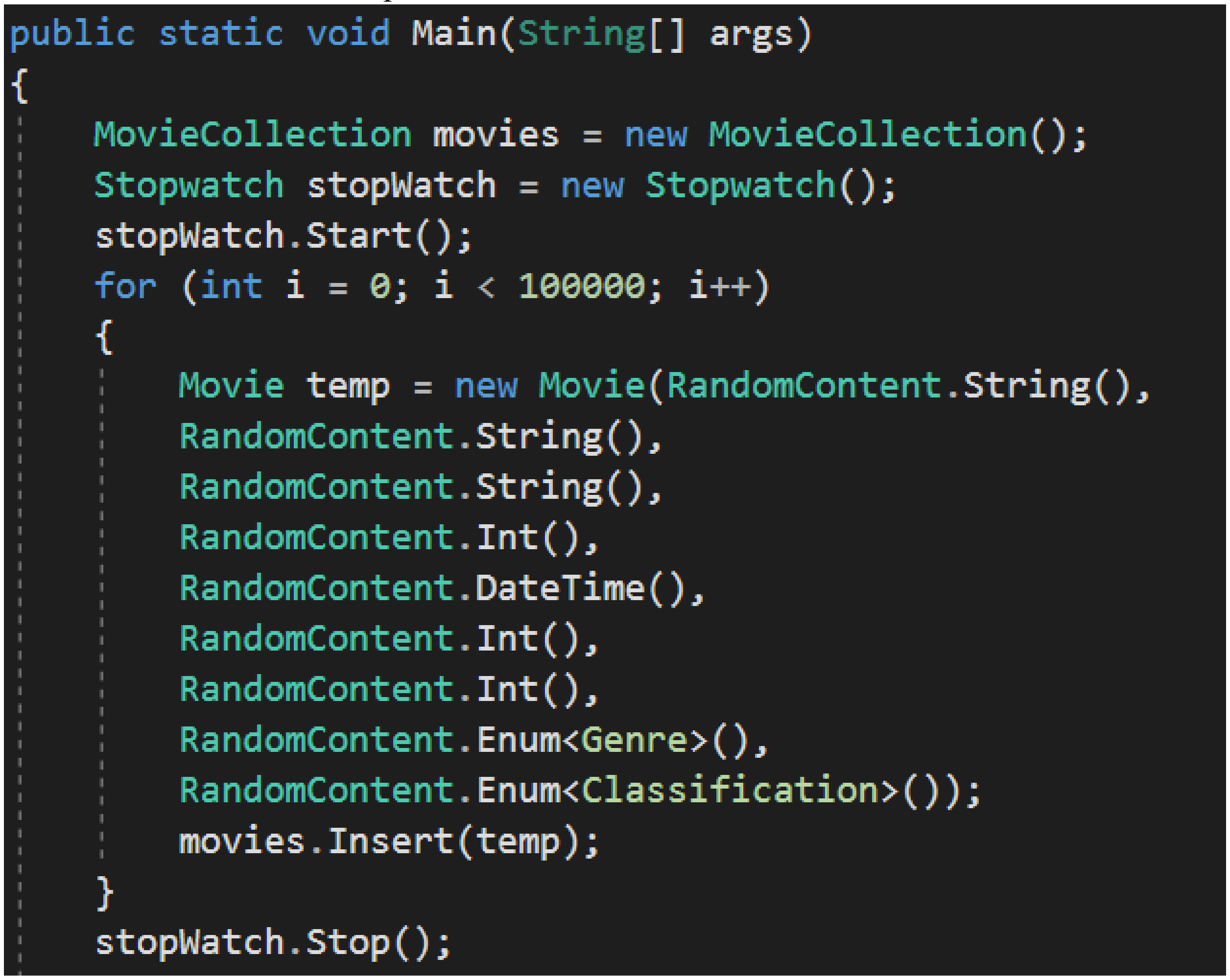
3) Functions testing results

First I made a class called "RandomContent" to generate a random value for a Movie:



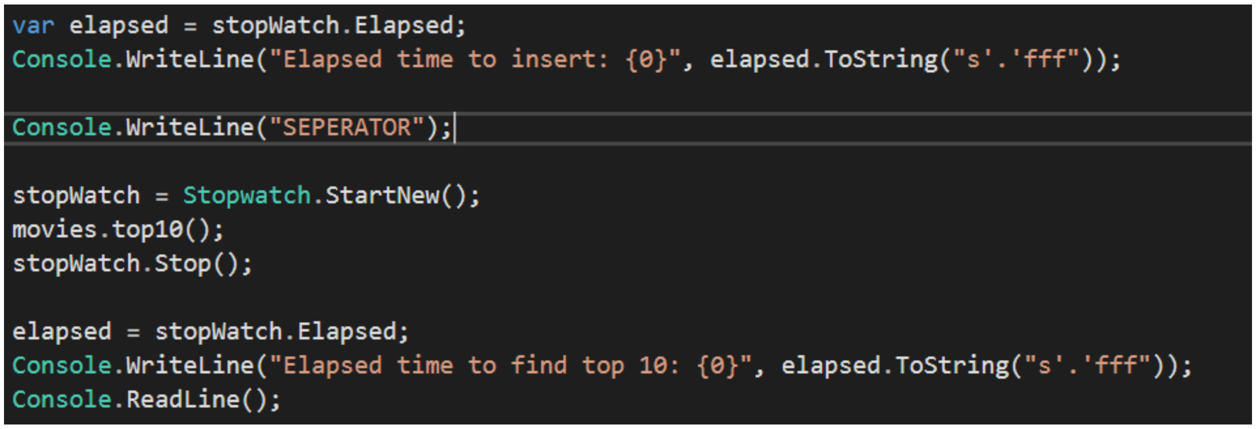
```
public class RandomContent
{
    private static readonly Random random = new Random();

    public static string String() => Guid.NewGuid().ToString();

    public static int Int() => random.Next(5000000);

    public static T Enum<T>()
    {
        var values = System.Enum.GetValues(typeof(T));
        return (T)values.GetValue(random.Next(values.Length));
    }

    public static DateTime DateTime() => System.DateTime.Now.AddDays(random.Next(1000));
}
```

The main method to test “top10” function:



```
public static void Main(String[] args)
{
    MovieCollection movies = new MovieCollection();
    Stopwatch stopWatch = new Stopwatch();
    stopWatch.Start();
    for (int i = 0; i < 100000; i++)
    {
        Movie temp = new Movie(RandomContent.String(),
        RandomContent.String(),
        RandomContent.String(),
        RandomContent.Int(),
        RandomContent.DateTime(),
        RandomContent.Int(),
        RandomContent.Int(),
        RandomContent.Enum<Genre>(),
        RandomContent.Enum<Classification>());
        movies.Insert(temp);
    }
    stopWatch.Stop();
```

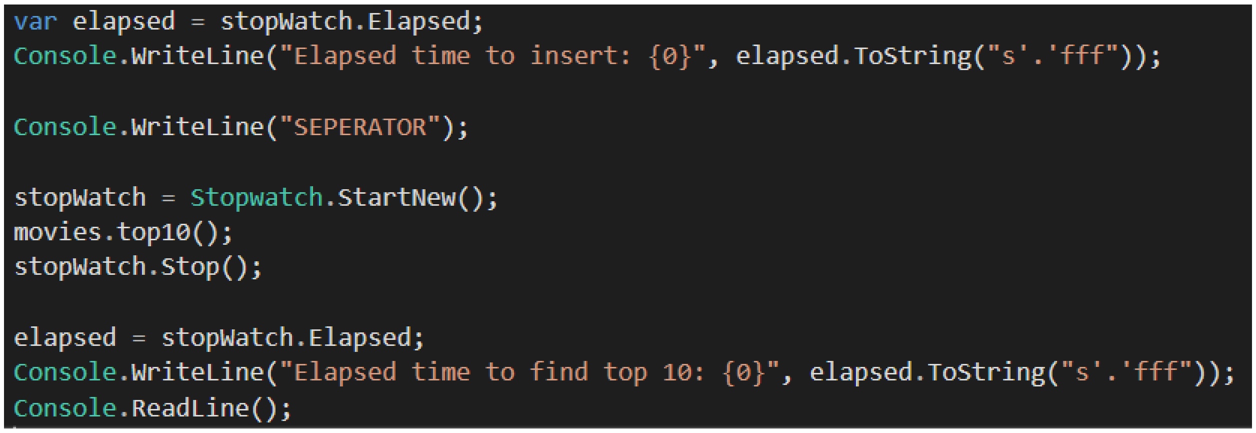
Create a movie collection, then try to add 100000 movies into the collection, with the stopWatch to check how long does it take to insert all the movies.



```
var elapsed = stopWatch.Elapsed;
Console.WriteLine("Elapsed time to insert: {0}", elapsed.ToString("s'.'fff"));

Console.WriteLine("SEPERATOR");

stopWatch = Stopwatch.StartNew();
movies.top10();
stopWatch.Stop();

elapsed = stopWatch.Elapsed;
Console.WriteLine("Elapsed time to find top 10: {0}", elapsed.ToString("s'.'fff"));
Console.ReadLine();
```

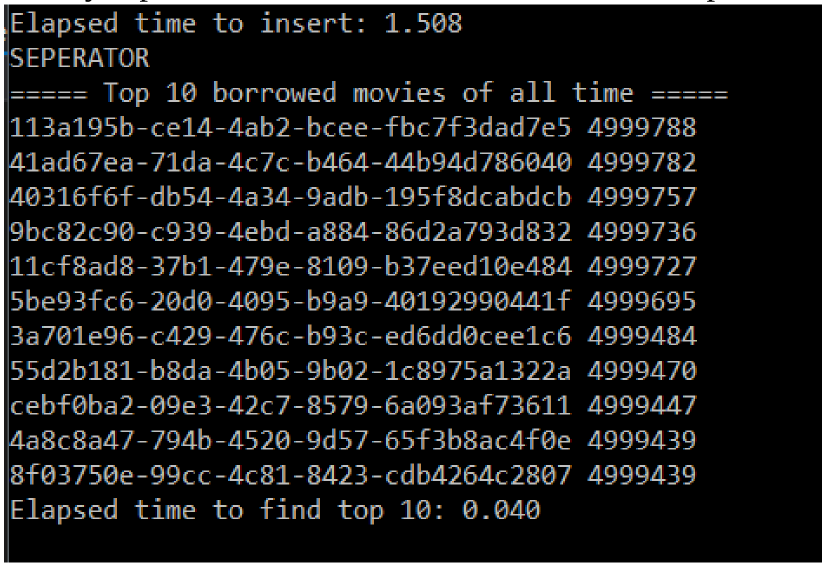
Printed out the time it takes to insert all movies then a separator.

After that I reset the stopWatch and call out the top10 methods.



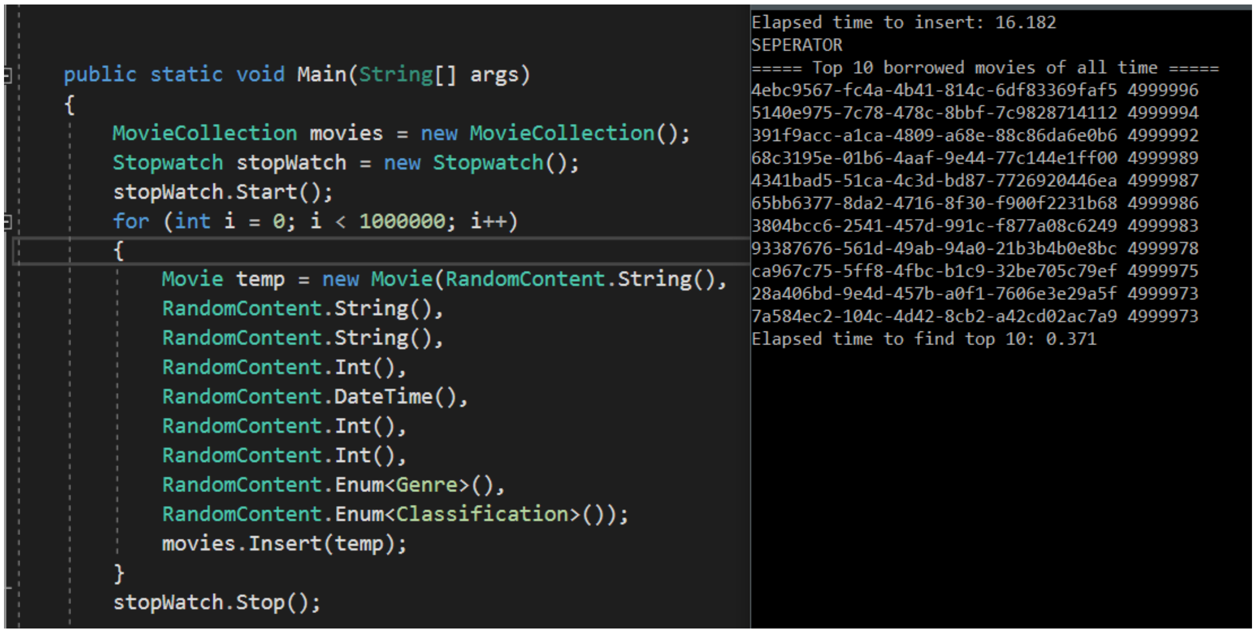
```
var elapsed = stopWatch.Elapsed;
Console.WriteLine("Elapsed time to insert: {0}", elapsed.ToString("s'.'fff"));

Console.WriteLine("SEPERATOR");

stopWatch = Stopwatch.StartNew();
movies.top10();
stopWatch.Stop();

elapsed = stopWatch.Elapsed;
Console.WriteLine("Elapsed time to find top 10: {0}", elapsed.ToString("s'.'fff"));
Console.ReadLine();
```

Finally, I printed out the time it takes to find the top 10



```
Elapsed time to insert: 1.508
SEPERATOR
===== Top 10 borrowed movies of all time =====
113a195b-ce14-4ab2-bcee-fbc7f3dad7e5 4999788
41ad67ea-71da-4c7c-b464-44b94d786040 4999782
40316f6f-db54-4a34-9adb-195f8dcabdcb 4999757
9bc82c90-c939-4ebd-a884-86d2a793d832 4999736
11cf8ad8-37b1-479e-8109-b37eed10e484 4999727
5be93fc6-20d0-4095-b9a9-40192990441f 4999695
3a701e96-c429-476c-b93c-ed6dd0cee1c6 4999484
55d2b181-b8da-4b05-9b02-1c8975a1322a 4999470
cebf0ba2-09e3-42c7-8579-6a093af73611 4999447
4a8c8a47-794b-4520-9d57-65f3b8ac4f0e 4999439
8f03750e-99cc-4c81-8423-cdb4264c2807 4999439
Elapsed time to find top 10: 0.040
```

Another case for finding top10 in 1 million movies



```
public static void Main(String[] args)
{
    MovieCollection movies = new MovieCollection();
    Stopwatch stopWatch = new Stopwatch();
    stopWatch.Start();
    for (int i = 0; i < 1000000; i++)
    {
        Movie temp = new Movie(RandomContent.String(),
        RandomContent.String(),
        RandomContent.String(),
        RandomContent.Int(),
        RandomContent.DateTime(),
        RandomContent.Int(),
        RandomContent.Int(),
        RandomContent.Enum<Genre>(),
        RandomContent.Enum<Classification>());
        movies.Insert(temp);
    }
    stopWatch.Stop();
```

```
Elapsed time to insert: 16.182
SEPERATOR
===== Top 10 borrowed movies of all time =====
4ebc9567-fc4a-4b41-814c-6df83369faf5 4999996
5140e975-7c78-478c-8bbf-7c9828714112 4999994
391f9acc-a1ca-4809-a68e-88c86da6e0b6 4999992
68c3195e-01b6-4aaf-9e44-77c144e1ff00 4999989
4341bad5-51ca-4c3d-bd87-7726920446ea 4999987
65bb6377-8da2-4716-8f30-f900f2231b68 4999986
3804bcc6-2541-457d-991c-f877a08c6249 4999983
93387676-561d-49ab-94a0-21b3b4b0e8bc 4999978
ca967c75-5ff8-4fbc-b1c9-32be705c79ef 4999975
28a406bd-9e4d-457b-a0f1-7606e3e29a5f 4999973
7a584ec2-104c-4d42-8cb2-a42cd02ac7a9 4999973
Elapsed time to find top 10: 0.371
```

4) Submission:

In the zip file contains a report and a folder which contains a project’s code. Tester.cs is a file with the code I use to test the top 10 function.