Course: Data Structures and Algorithms in C#, Spring 2025

Homework #1 – Eudaly

1.) **Insert method:**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DSAHomework1
{
    class Homework1
    {
        // returns a new array populated with contents of original array
        // with the given value inserted at the given index

        //** Order of Growth Analysis **
        // The Order of growth for this algorithm is O(n) because the growth depends
on the size of the input array
        // There are two loops that iterate over the array, both of which depend on
the size of the array
        // The loops are not nested so the time complexity would be O(n) + O(n) =
O(2n), after discarding the constant
        // the time complexity is O(n)
        public static int[] Insert(int[] array, int index, int value)
        {
            //begin edge cases

            // array is null
            if (array == null ) // O(1)
            {
                throw new ArgumentException("input array cannot be null"); // O(1)
            }

            // index is negative
            if (index < 0) // O(1)
            {
                throw new ArgumentException("index cannot be negative"); // O(1)
            }

            // if index is greater that the last index of the current array, place
at end
            if (index > array.Length - 1) // O(1)
            {
                index = array.Length; // O(1)
            }

            // end edge cases

            // create new array one larger than original
            int[] newArray = new int[array.Length + 1]; // O(1)
```

```
            // copy elements up to insert point from original array to new array
            // This is O(n) because in the worst case the index will be the size of
the array, therefore it
            // depends on the size of the array
            for (int i = 0; i < index; i++) // O(n)
            {
                newArray[i] = array[i]; // O(1)
            }

            // insert value at index
            newArray[index] = value;

            // copy remaining elements from original array to new array
            for (int i = index; i < array.Length; i++) // O(n)
            {
                newArray[i + 1] = array[i]; // O(1)
            }

            return newArray; // O(1)

        }
    }
}
```

2.) Main method

```
using OfficeOpenXml;
using System.Diagnostics;
namespace DSAHomework1
{
    internal class Program
    {
        //Implement a main method that profiles the performance of Insert and
Outputs a table
        //showing the average time perinsert as the length of the array increases
        static void Main(string[] args)
        {

            // Set the EPPlus license before using it
            ExcelPackage.License.SetNonCommercialPersonal("Jeremy");

            // setting to allow fine tuning of the granularirty of the reading
            int NUM_READINGS = 60;
            int INSERTS_PER_READING = 1000;

            // start with an array containing one element haviong the value of 0
            int[] array = new int[0];

            //initialize random number generators
            System.Random randomIndex = new System.Random();
            System.Random randomValue = new System.Random();

            //set output directory
            string outputDirectory = @"C:\DSA - UC San
Diego\DSAHomework\DSAHomework1";

            // output to text file for raw data
```

```
            string textFilePath = Path.Combine(outputDirectory,
"performance_results.txt");

            //output to excel to generate scatter plot
            string excelFilePath = Path.Combine(outputDirectory,
"performance_results.xlsx");



            //take NUM_READINGS readings
            //Loop NUM_READINGS times
            // Each reading will be taken after INSERTS_PER_READING inserts

            //wrap logic in using so streamwriter and excel package are disposed of
properly
            using (StreamWriter writer = new StreamWriter(textFilePath))
            using (ExcelPackage excelPackage = new ExcelPackage())
            {
                ExcelWorksheet worksheet =
excelPackage.Workbook.Worksheets.Add("Homework 1 Performance Results");

                // Write headers to console, text file
                string header = $"{"Array Length",-15} {"Seconds per Insert",-20}";
                Console.WriteLine(header);
                writer.WriteLine(header);

                string separator = new string('-', 35);
                Console.WriteLine(separator);
                writer.WriteLine(separator);

                // Write headers in Excel
                worksheet.Cells[1, 1].Value = "Array Length";
                worksheet.Cells[1, 2].Value = "Seconds per Insert";

                //starting row for excel
                int excelRow = 2;

                for (int i = 0; i < NUM_READINGS; i++)
                {

                    Stopwatch sw = new Stopwatch();
                    sw.Start();

                    // insert INSERTS_PER_READING values into the array
                    for (int j = 0; j < INSERTS_PER_READING; j++)
                    {
                        int index = randomIndex.Next(0, array.Length);
                        int value = randomValue.Next();

                        try
                        {
                            array = Homework1.Insert(array, index, value);
                        }
                        catch (Exception e)
                        {
                            Console.WriteLine($"Unexpected Error: {e.Message}");
                        }
```

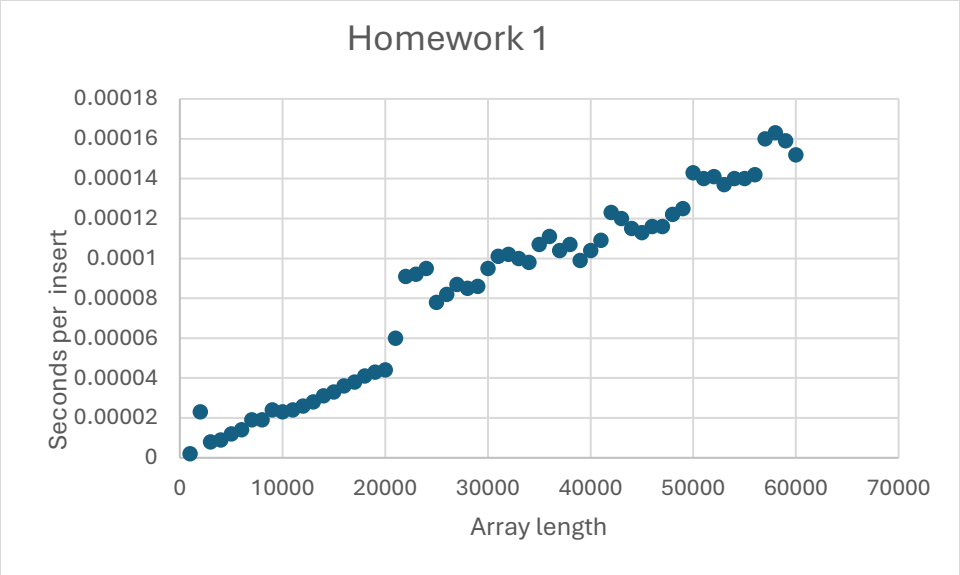
```
                    }

                    sw.Stop();

                    double secondsPerInsert = (sw.ElapsedMilliseconds / 1000.0) /
INSERTS_PER_READING;

                    string output = $"{array.Length,-15} {secondsPerInsert,-20:F6}";

                    // Write to console and file
                    Console.WriteLine(output);
                    writer.WriteLine(output);

                    // Write data to Excel
                    worksheet.Cells[excelRow, 1].Value = array.Length;
                    worksheet.Cells[excelRow, 2].Value = secondsPerInsert;
                    excelRow++;
                }

                Console.WriteLine("Writing to Excel file...");
                File.WriteAllBytes(excelFilePath, excelPackage.GetAsByteArray());
                Console.WriteLine("Excel file written successfully.");
            }

            Console.WriteLine($"\nResults saved to {Path.GetFullPath(textFilePath)}
and {Path.GetFullPath(excelFilePath)}");

        }
    }
}
```

Sample output:

```
Array Length    Seconds per Insert
-----------------------------------
1000            0.000003
2000            0.000029
3000            0.000013
4000            0.000015
5000            0.000018
6000            0.000020
7000            0.000022
8000            0.000025
9000            0.000026
10000           0.000026
11000           0.000031
12000           0.000035
13000           0.000039
14000           0.000044
15000           0.000055
16000           0.000051
17000           0.000055
18000           0.000050
19000           0.000060
20000           0.000065
21000           0.000059
22000           0.000086
23000           0.000093
24000           0.000096
```

25000	0.000131
26000	0.000109
27000	0.000102
28000	0.000099
29000	0.000114
30000	0.000097
31000	0.000095
32000	0.000120
33000	0.000116
34000	0.000107
35000	0.000106
36000	0.000114
37000	0.000112
38000	0.000111
39000	0.000125
40000	0.000116
41000	0.000115
42000	0.000122
43000	0.000122
44000	0.000114
45000	0.000124
46000	0.000125
47000	0.000138
48000	0.000138
49000	0.000135
50000	0.000143
51000	0.000138
52000	0.000136
53000	0.000158
54000	0.000167
55000	0.000160
56000	0.000163
57000	0.000156
58000	0.000167
59000	0.000154
60000	0.000156

3.) Scatter Plot:



4.) Big O analysis - Line by line analysis is provided in the provided Insert method code.

The Order of growth for this algorithm is O(n) because areas of the logic depends on the size of the input array. Specifically, there are two loops that are dependent on the size of the depend on the size of the array. The first loop copies elements up to the insert point from the original array to the new array. This is O(n) because, in the worst case, the index will be the size of the array, therefore it depends on the size of the array. The second loop depends on array.Length directly. The loops are not nested, so the time complexity would be O(n) + O(n) = O(2n). After discarding the constant, the time complexity is O(n).

5.) Comparison of graph and analysis

The graph shows a consistent degradation of performance as the size of the array increases. The point of the graph steadily trend in an upward direction visually which is consistent with what I would expect with O(n).

6.) Code comments, style, etc

See provided code