CSCI 2110 Data Structures and Algorithms
Fall 2024
Assignment No. 2

Date Given: Tuesday, September 24, 2024
Due (on Brightspace): Monday, October 7, 11.59 PM

Your second assignment is an "experimental" assignment on Algorithm Time Complexity, similar to Lab No. 3. In the lab, you developed and experimented with the complexities of prime number generation, matrix multiplication, and binary exponentiation.

You can use IntelliJ, Eclipse, or any other IDE of your choice to develop the program.

Commenting your code: Your code should be clear and easy to understand. While extensive comments are not required, you are expected to include brief, meaningful comments to explain complex logic or any non-obvious sections of your code. Use inline comments to clarify key parts of the code. For methods and classes, use Javadoc to briefly describe their purpose, parameters, and return values. This ensures others can understand your code's intent quickly.

Submission: One zip file containing the source codes (.java files) and sample outputs and graphs. Please see instructions at the end of the document.

Submission Deadline: Monday, October 7, 2024, 11.59 PM

Grace Time: Submissions will be accepted until 4.59 AM on Tuesday, October 8, 2024 without late penalty.

Late Penalty: Submissions received after the grace time will be subject to a 10% per day late penalty, for up to 5 days. For example, if you submit the assignment on Tuesday, October 8 at 12 noon and your score is 8/10, it will be reduced to 7.2/10. Submissions past five days after the grace submission time will not be accepted. The submission portal will close on Sunday, October 13, 4.59 AM.

Dropping of Assignments: One out of six assignments can be dropped during the semester. No SDA required.

In this assignment, you will develop and experiment the complexities of two nice and nifty math problems, namely, the ***Collatz sequence and its variant***. Don't worry if are not aware of this concept – it is easily explained below!

Note: The actual coding part of the assignment is relatively easy. It is an assignment to help you understand how algorithm complexity works for large values of n.

As with Lab No. 3, you can use the following code template to obtain the execution time of your code.

```
long startTime, endTime, executionTime;
startTime = System.currentTimeMillis();

//code snippet (or call to the method) here

endTime = System.currentTimeMillis();
executionTime = endTime - startTime;
```

The above code will give the time for executing the code snippet in milliseconds. You can display the executionTime using System.out.println and/or save it for drawing the result graphs.

Program Exercise No. 1: Collatz Sequence

The ***Collatz sequence*** of a positive integer n is defined as follows:

a) Start with the integer n (the starting number).
b) If the integer is even, divide it by 2 (integer division) (that is, n←n/2)
c) If the integer is odd, multiply it by 3 and add 1. (that is, n←3n + 1)
d) Repeat the process until n becomes 1.

For example, let n = 5. Then its Collatz sequence is:
5→16→8→4→2→1
Since it takes six iterations, the length of the Collatz sequence for starting number 5 is 6.

As another example, let n = 13. Then its Collatz sequence is:
13→40→20→10→5→16→8→4→2→1
The length of the Collatz sequence for starting number 13 is 10.

Although it is not proven yet, it is thought that all Collatz sequences end in 1.

NOTE: Collatz sequences can grow big pretty quickly. Use long data type or the BigInteger class that is part of the java.math package.

Step 1: Write a program to generate the Collatz sequence for a given positive integer n. Here is a sample screen dialog for the program:

```
Enter a positive integer: 5
The Collatz sequence for n = 5 is 5, 16, 8, 4, 2, 1
```

As you may see, this is a simple program! Test your program for different values of n.

Step 2: Now modify your program to calculate which starting number less than or equal to an input value of n produces the longest Collatz sequence. For example, if you were to find which starting number less than or equal to 5 produces the longest sequence, the answer would be 3 and the length of the sequence is 8. See below:

1: 1
2: 2→1
3: 3→10→5→16→8→4→2→1
4: 4→2→1
5: 5→16→8→4→2→1
As you can see, if n=5, the starting number with the longest sequence is 3 and the length of the longest sequence is 8.

You can also check for yourself that if n = 6, the starting number with the longest sequence is 6, and the length of the longest sequence is 9.

Here's a sample screen dialog for this modified program.

```
Enter a positive integer: 5
For n=5, the starting number of the longest Collatz sequence is 3
and the length of the longest sequence is 8.
```

Note that in the modified program, you are not displaying the Collatz sequences. Rather, you just display the starting number of the longest sequence and its length.

Test your program for different values of n.

Step 3: Next modify the program from Step 2 to calculate the time for execution of the core part of your code – code to determine the Collatz sequences up to n and determining the longest sequence and its starting number.

Here's a sample screen dialog for this step:

```
1000                                ← this is the input to the program
1000 871 179 3                      ← this is the output of the program
```

The value of n is 1000, the starting number of the longest Collatz sequence is 871, the length of the longest sequence is 179, and it took 3 milliseconds (on my machine).

Test your modified program for different values of n.

Step 4 (Final modified program – this is what you will submit as part of the assignment):

Your program should accept input in the following format:

a) Positive integers separated by whitespace will indicate the values of n.
b) Your program should exit when passed 0 as an input.

Your program should provide output in the following format:

a) There should be one line for each value of n.
b) Each line should contain four numerical values separated by whitespace, representing the starting value of n, the number producing the longest sequence, the length of that sequence, the execution time in milliseconds.

Test that your program works for n values of 5, 100, 1000, 10000, 100000, 1000000, and 10000000. Once you are sure that it is correctly identifying the longest sequences, set it up to

time your code's execution in milliseconds. Save the output in a text file called Exercise1.out. Plot a simple graph of n vs. execution time.

Here's a sample input and output of the program:

Exercise1.in

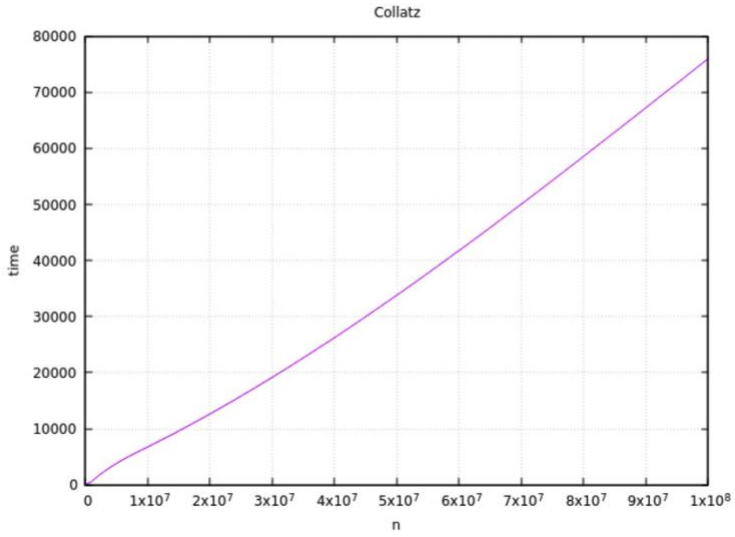
```
5 100 1000 10000 100000 1000000 10000000 0
```

Exercise1.out

```
5 3 8 0
100 97 119 1
1000 871 179 3
10000 6171 262 12
100000 77031 351 54
1000000 837799 525 411
10000000 8400511 686 4685
```

In the above, for example, for n = 5, the longest Collatz sequence starts at 3, the length of the longest sequence is 8, and it takes 0 milliseconds to compute.
As another example, for n = 10000000, the longest Collatz sequence starts at 8400511, the length of the longest sequence is 686, and it takes 4685 milliseconds to compute.

Sample graph:



Note: You may need to test additional cases in order to collect sufficient data to create smooth or representative plot.

In your submission folder, save the source code (.java file) for the last step, that is, Step 4, the sample input and output, and the graph.

Program Exercise No. 2: Cousin of Collatz Sequence

For this exercise, you will be writing a program in steps **very similar** to Exercise No. 1, except that you will be testing a sequence called the "Cousin of Collatz". It is also called the $7x \pm 1$ problem and is defined as follows:
Start at a positive integer n.
If n is even, divide it by 2 (integer division) (that is n ← n/2)
If n is odd and n % 4 is 1, then multiply it by 7 and add 1 (that is, n ← 7n+1)
If n is odd and n % 4 is 3, then multiply it by 7 and subtract 1 (that is, n ← 7n -1)
Repeat until n becomes 1.

(Note: As you know from your first year, % represents the remainder of an integer division. 15%4 is 3 whereas 17%4 is 1).

For example, if n = 5, then the $7x \pm 1$ sequence is:

5 → 36 → 18 → 9 → 64 → 32 → 16 → 8 → 4 → 2 → 1

The sequence can grow pretty large. For example, if n = 235, the sequence is the following (the odd integers are shown in bold):

235, 1644, 822, **411**, 2876, 1438, **719**, 5032, 2516, 1258, **629**, 4404, 2202, **1101**, 7708, 3854, **1927**, 13488, 6744, 3372, 1686, **843**, 5900, 2950, **1475**, 10324, 5162, **2581**, 18068, 9034, **4517**, 31620, 15810, **7905**, 55336, 27668, 13834, **6917**, 48420, 24210, **12105**, 84736, 42368, 21184, 10592, 5296, 2648, 1324, 662, **331**, 2316, 1158, **579**, 4052, 2026, **1013**, 7092, 3546, **1773**, 12412, 6206, **3103**, 21720, 10860, 5430, **2715**, 19004, 9502, **4751**, 33256, 16628, 8314, **4157**, 29100, 14550, **7275**, 50924, 25462, **12731**, 89116, 44558, **22279**, 155952, 77976, 38988, 19494, **9747**, 68228, 34114, **17057**, 119400, 59700, 29850, **14925**, 104476, 52238, **26119**, 182832, 91416, 45708, 22854, **11427**, 79988, 39994, **19997**, 139980, 69990, **34995**, 244964, 122482, **61241**, 428688, 214344, 107172, 53586, **26793**, 187552, 93776, 46888, 23444, 11722, **5861**, 41028, 20514, **10257**, 71800, 35900, 17950, **8975**, 62824, 31412, 15706, **7853**, 54972, 27486, **13743**, 96200, 48100, 24050, **12025**, 84176, 42088, 21044, 10522, **5261**, 36828, 18414, **9207**, 64448, 32224, 16112, 8056, 4028, 2014, **1007**, 7048, 3524, 1762, **881**, 6168, 3084, 1542, **771**, 5396, 2698, **1349**, 9444, 4722, **2361**, 16528, 8264, 4132, 2066, **1033**, 7232, 3616, 1808, 904, 452, 226, **113**, 792, 396, 198, **99**, 692, 346, **173**, 1212, 606, **303**, 2120, 1060, 530, **265**, 1856, 928, 464, 232, 116, 58, **29**, 204, 102, **51**, 356, 178, **89**, 624, 312, 156, 78, **39**, 272, 136, 68, 34, **17**, 120, 60, 30, **15**, 104, 52, 26, **13**, 92, 46, **23**, 160, 80, 40, 20, 10, **5**, 36, 18, **9**, 64, 32, 16, 8, 4, 2, **1**

As you can see, the Java code for generating the above sequence is simple!

Once you get your basic program for generating the above sequence working, **modify the program as you did in Step 4 of Exercise 1**, except that this time your code is for the "Cousin of Collatz" sequence.

As mentioned before, use long data type or the BigInteger class that is part of the java.math package.

Test your code for the following input:

```
5 100 1000 10000 100000 150000 180000 190000 0
```

Save the output in a text file called Exercise2.out. Plot a simple graph of n vs. execution time.

What to submit:
Submit one ZIP file containing all source code (files with .java suffixes), inputs and outputs of Exercise1 and Exercise2, and the graphs.

You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers.