CS 201, Spring 2022
Homework Assignment 2
Due: April 6, 2022, 23:59

Note: Before implementing this assignment please make sure that you understand the concept of time complexity analysis. In this homework you will perform the time complexity analysis of several discrete logarithm algorithms. That is, given three integers *a*, *n* and *p*, mentioned algorithms will calculate and return $a^n \ (mod\ p)$.

Note: You do not have to run this code on the server and you can use your own computer. This homework is about complexity analysis and the report is an important component.

Consider the following three algorithms. Each algorithm has a different time complexity. The inputs to all algorithms will be the same.

Algorithm 1 (Naive algorithm): This function multiplies the *a* with itself in a for loop and takes modulus *p* in each step. After the loop ends, it returns the result.

Algorithm 2 (Naive algorithm with Cycle shortcut) : This function has the same behavior as previous one until the step *i* which satisfies $a^i \equiv 1 (mod\ p)$. After that step, it calculates $a^{\ n\ mod(i)} \ (mod\ p)$ and returns the result. Note that if *n* is less than *i* , the algorithm is equivalent to the previous one.

Algorithm 3 (Recursive Algorithm): This function is recursively calculates $a\ *\ (a^{\ (n-1)/2}\ (mod\ p))\ *\ (a^{\ (n-1)/2}\ (mod\ p))\ (mod\ p)$ if *n* is odd; and calculates $(a^{\ n/2}\ (mod\ p))\ *\ (a^{\ n/2}\ (mod\ p))\ (mod\ p)$ if n is even. Note that *n* = 1 is the base case for the function.

ASSIGNMENT:

Answer each of these questions:

1. Study algorithms and understand how the upper bounds are found for each. Describe how the complexity of each algorithm is calculated in your own words.
2. Report specifications of the computer that you used. Report RAM and Processor specifications in particular.

Answer each of these questions:

3. Create an implementation of all of these functions as global functions in your main file. Then, in your main function, call these functions to measure their running time. For the sake of simplicity, you should pick the number *a* constant greater than 1. You have to try different combinations of *n* and *p*; and observe how each one of these parameters (if at all) affects the execution time of the algorithms. Record each observation, you will need to report and comment on it. Create a table containing all the values that you have recorded and include it in your report. The layout of the table is shown below. You cannot completely change the column format of the table but you can add different *p*'s into each algorithm and also include as many rows as you like as long as the total number of rows (different values of n that you used) is more than or equal to 10. This means that you have to try at least 10 different n values. Your choice of n values should be adjusted so that you can observe the expected time complexities in the actual runs.

n	Algorithm 1			Algorithm 2			Algorithm 3		
	p=101	p=1009	p=10007	p=101	p=1009	p=10007	p=101	p=1009	p=10007
10^3									
$2*10^3$									
$3*10^3$									
$4*10^3$									
$5*10^3$									

Observe the organization of the table. The rows of the table represent all the different numbers of n values that we will try. In the example table above, we used values 10^3, $2*10^3$, … which have a difference of 10^3 between them. <u>You do not have to</u> use the same exact values or the same interval. Determine values and intervals that give you the best plot on your computer. However, make sure that values are evenly spread from

each other. Also for demonstration purposes, we used only 5 different n values in the table, while you have to use at least 10. As for the range of the values, make sure that it is sufficiently large so that your function call with the largest n value takes at least 1 minute.

4. Create a plot for each column of the table (total of 9 in example), where n values are on the x-axis and elapsed times are on the y axis. You have to make sure that the plots are high quality. Sloppy and poorly scaled plots will lead to a reduction of a significant amount of points even if you conducted all the experiments perfectly. Consider using Python or MATLAB for plotting, as they contain built-in libraries for plotting. If the plot you generated makes it hard to draw conclusions about the behavior of the algorithm, then you should probably consider another way of plotting it. Name your plots properly. So that we can differentiate which one is which. None that here you are simulating the asymptotic behavior of the function, and the plots you generate are not expected to look like perfect logarithmic or linear plots. There will be some fluctuations, especially with smaller n values.

You can use the following code segment to compute the execution time of a code block. For these operations, you must include the ctime header file.

```
double duration;
clock_t startTime = clock();
//Code block
//...
//Compute the number of seconds that passed since the starting time
duration = 1000 * double( clock() - startTime ) / CLOCKS_PER_SEC;
cout << "Execution took " << duration << " milliseconds." << endl;
```

An alternative code segment to compute the execution time is as follows. For these operations, you must include the chrono header file.

```
//Declare necessary variables
std::chrono::time_point< std::chrono::system_clock > startTime;
std::chrono::duration< double, milli > elapsedTime;
//Store the starting time
startTime = std::chrono::system_clock::now();
//Code block
...
//Compute the number of seconds that passed since the starting time
elapsedTime = std::chrono::system_clock::now() - startTime;
```

```
cout << "Execution took " << elapsedTime.count() << " milliseconds."
<< endl;
```

NOTES ABOUT SUBMISSION:

1. **This assignment will be graded by your TA Mahmud Sami Aydın (sami.aydin@bilkent.edu.tr). Thus, you should ask your homework-related questions directly to him.**
2. The assignment is due by April 6, 2022, 23:59 . You should upload your homework to the upload link on Moodle before the deadline. No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course web page for further discussion of the late homework policy as well as academic integrity.
3. You must submit a report (as a PDF file) that contains all information requested above (plots, tables, computer specification, discussion) and a cpp file that contains the main function that you used as the driver in your simulations as well as the solution functions in a single archive file.
4. You should prepare your report (plots, tables, computer specification, discussion) using a word processor and convert it into a PDF file. In other words, do not submit images of handwritten answers.. Handwritten answers and drawings will not be accepted. In addition, DO NOT submit your report in any other format than .pdf.
5. The code segments given above may display 0 milliseconds when the value of n is small. Of course, the running time cannot be 0 but it seems to be 0 because of the precision of the used clock. However, we will not accept 0 as an answer. Thus, to obtain a running time greater than 0, please consider running algorithm k times using a loop, and then divide the running time (which will be greater than 0) by k.
6. You can reuse the codes on the lecture slides if you need, but other than that, the code must be your own implementation.