CS5800 Algorithms *Out: 9 September 2022*

Problem Set 1

Ravi Sundaram *Due: 20 September 2022*

Problem 1 (Asymptotics) 27

Arrange the following functions in ascending order of growth rate. Also give reasoning for the order.

$g_{01}(n) = n^{\frac{101}{100}}$
$g_{02}(n) = n2^{n+1}$
$g_{03}(n) = n(\log n)^3$
$g_{04}(n) = n^{\log\log n}$
$g_{05}(n) = \log(n^{2n})$
$g_{06}(n) = n!$
$g_{07}(n) = 2^{\sqrt{\log n}}$
$g_{08}(n) = 2^{2^{n+1}}$
$g_{09}(n) = \log(n!)$
$g_{10}(n) = \lceil\log(n)\rceil!$
$g_{11}(n) = 2^{\log\sqrt{n}}$
$g_{12}(n) = \sqrt{2}^{\log n}$

Problem 2 (More Asymptotics) 25

For a given two functions f, g and h. Decide whether each of the following statements are correct and give a proof for each part.

1. If $f(n) = \Omega(g(n))$ and $g(n) = \Omega(f(n))$, then $f(n) = \Theta(g(n))$
2. If $f(n) = o(g(n))$, then $g(n) \notin O(f(n))$
3. If $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then $f(n) + g(n) = O(h(n))$
4. If $f(n) = O(h(n))$ and $g(n) = O(h(n))$, then $f(n) \cdot g(n) = O(h(n))$
5. If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$

Problem 3 (Modular Arithmetic Computations) 18

Use properties of modular arithmetic and fast modular exponentiation to compute these.

1. Compute $3^{1500} \mod 11$.
2. Compute $5^{4358} \mod 10$.
3. Compute $6^{22345} \mod 7$.
4. Compute $\gcd(648, 124)$.
5. Compute $\gcd(123456789, 123456788)$.
6. Compute $\gcd(10^{117}, 2^{200})$.

Problem 4 (Programming: modular-exponentiation) 30

Submit your code at the following Hackerrank link:
https://www.hackerrank.com/cs5800-fall22-pset1.

Description: Implement modular exponentiation in a way that outputs the intermediary steps of the algorithm. **Problem Statement:** **IMPORTANT** You are NOT allowed to use built in language functions which trivialize the task of computing exponents. Any submissions which use this and avoid the task at hand will be given a 0.

In this problem, you will have to efficiently implement modular exponentiation. Recall that the problem of modular-exponentiation is, given positive integers a and n, and a non-negative integer x, calculate $a^x \mod n$.

One way of doing this is exponentiation by squaring. It involves repeatedly squaring the base a and reducing it $\mod n$. Doing so yields the values a, $a^2 \mod n$, $a^4 \mod n$, $a^8 \mod n$, ... etc. By combining these in the correct way, and using the fact that every number has a binary representation, we can compute $a^x \mod n$ in time $O(\log x)$.

Implement modular-exponentiation by squaring, and output the intermediary values of $a^{2^i} \mod n$, as well as the final value $a^x \mod n$.

Note that in addition to the above, the challenge page also describes the input/output format, and gives a few examples.

Input Format: The input will be exactly one line, with three space delimited integers a, x, and n.

Constraints: The integers will satisfy $2 \leq a, x, n \leq 2^{64}$.

Output Format: On the first line, output the d-bit binary representation of x.

On the next d lines, output the values:

$a^1 \mod n$,

$a^2 \mod n$,

$a^4 \mod n$,

$\vdots$

$a^{2^d} \mod n$,

On the last line, output $a^x \mod n$

Sample Input: 3 7 10

Sample Output: 111

3

9

1

7

Explanation: 7 in binary is written as 111.

$3^1 = 3 \mod 10$

$3^2 = 9 \mod 10$

$3^4 = 81 = 1 \mod 10$

$3^7 = 2187 = 7 \mod 10$