CS5800 Algorithms *Out: September 13, 2019*

Problem Set 1

Ravi Sundaram *Due: September 20, 2019*

Problem 1 (Proof by Contradiction) 10 points

P is a given program which takes an input string and always returns 1. (That is, program P computes the function $f(n) = 1$, for every $n \in \mathbb{N} = \{1, 2, ...\}$.) Prove that there is no equality checking program which when given another program Q on input, can conclusively determine whether $P = Q$ (i.e. in terms of behavior Q computes the same function as P).

Problem 2 (Proof by Induction) 20 points

1. For all $n \geq 1$ prove that the following summation formula is correct:
$$\frac{1}{1 \times 3} + \frac{1}{3 \times 5} + \frac{1}{5 \times 7} + \cdots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$$
2. Prove that $2^n + 1$ is divisible by 3 for all odd natural numbers n.
3. Let n and k be non-negative integers with $n \geq k$. Prove $\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1}$.
4. Let $f_0 = 0, f_1 = f_2 = 1$ and define $f_n = f_{n-1} + f_{n-2}$ for all $n \geq 1$. These are called the Fibonacci numbers. Prove that for all $n \geq 1$, the following holds:
$$f_{n+1} < \left(\frac{7}{4}\right)^n$$

Problem 3 (Asymptotics) 20 points

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 4 (More Asymptotics) 20 points

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 5 (Programming: Modular exponentiation) 30 points

See Piazza for a post containing a link to the HackerRank page, where you will be submitting the assignment. In addition to the below description, it also contains more formal requirements for how your program should behave.
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.