Assignment 01

Paul Jones and Matthew Klein
Professor Professor Kostas Bekris
Design and Analysis of Computer Algorithms (01.198.344)

February 16, 2014

Part A

Problem 1

In each of the following situations indicate whether $f = O(g)$ or $f = \Omega(g)$ or $f = \Theta(g)$:

1. $f(n) = \sqrt{2^{7x}}, g(n) = \lg(7^{2x})$

$$f(n) = \sqrt{2^{7x}} = \sqrt{128^x}$$
$$g(n) = \lg(7^{2x}) = \lg(49^x)$$
$$lg(49^1) \approx 5.6$$
$$\sqrt{128^1} \approx 11.3$$

   Notice that both of these functions only grow relative to x.

$$f = \Omega(g)$$

2. $f(n) = 2^{nln(n)}, g(n) = n!$

   The factorial, that is $n!$, function grows much, much faster than 2^n.

$$f = \Omega(g)$$

3. $f(n) = \lg(\lg^*(n)), g(n) = \lg^*(\lg(n))$

$$f = \Theta(g)$$

4. $f(n) = \frac{lg(n^2)}{n}, g(n) = lg^*(n)$

$$f(n) = \frac{lg(n^2)}{n} = \frac{2lg(n)}{n}$$

$$f = \Theta(g)$$

5. $f(n) = 2^n, g(n) = n^{\lg(n)}$

   This is comparing the exponential function to a function that is less than n^2.

$$f = \Omega(g)$$

6. $f(n) = 2^{\sqrt{\ln(n)}}, g(n) = n(\lg(n)^3)$

$$f(n) = 2^{\sqrt{n}}, g(n) = (2^n)(n^3)$$

$$f = \Omega g$$

7. $f(n) = e^{\cos(x)}, g(n) = \lg(x)$

$$f = \Omega(g)$$

8. $f(n) = \lg(n^2), g(n) = (\lg(n))^2$

$$f = \Theta(g)$$

9. $f(n) = \sqrt{4n^2 - 12n + 9}, g(n) = n^{\frac{3}{2}}$

$$f = \Theta(g)$$

10. $f(n) = \sum_{k=1}^{n} k, g(n) = (n+2)^2$

$$f = \Omega(g)$$

Problem 2

Algorithm 1: `Number_Theoretic_Algorithm` (integer n)

$N \leftarrow \mathit{Random\_Sample}(0, 2^n - 1)$;
if N *is even* **then**
    $N \leftarrow N + 1$ `/* Worse case, N is odd, 2 ** N - 1. */` ;
$m \leftarrow N \mod n$ `/* worse case same as n */` ;
for $j \leftarrow 0$ *to* m **do**
    if `Greatest_Common_Divisor`$(j, N) \neq 1$ **then**
        return `FALSE`; `/* GCD is O(n) */`
    Compute x, z so that $N - 1 = 2^z \cdot x$ and x is odd;
    $y_0 \leftarrow (N - 1 - j)^x \mod N$;
    for $i \leftarrow 1$ *to* m **do**
        $y_i \leftarrow y_{i-1}^2 \mod N$;
        $y_i \leftarrow y_i + y_{i-1} \mod N$;
    if `Low_Error_Primality_Test`(y_m) == `FALSE` **then**
        return `FALSE` `/* Naive primality test is O(sqrt(n)) */` ;
return `TRUE`;

Compute the asymptotic running time of the above algorithm as a function of its input parameter, given:

- The running times of integer arithmetic operations (e.g., multiplication of two large n-bit numbers is $O(n^2)$).
- Assume that sampling a number N is an operation linear to the number of bits needed to represent this number.

Do not just present the final result. For each line of pseudo-code indicate the best running time for the corresponding operation given current knowledge from lectures and recitations and then show how the overall running time emerges.

Worse case running n operations with times $O(n)$, $O(n)$, and $O(\sqrt{n})$.
That's a run time of $O(2n^2 + n^{\frac{3}{2}})$, resulting in big-O of $O(n^2)$.

Part B

Problem 3

- A tree with m children is $\log_m^(N + 1) - 1$.
- A perfect tree will only be changing based on the m, m' values. Whichever value is larger will run faster.
-

Problem 4

- I found out how to do this using a website, since I didn't understand how to from lecture ? 2^{902} mod 7 We can find the original, 2 mod 7 = 2 because 7 doesn't go into 2 at all. We can next square, finding 4 mod 7 = 4. Divide exponent in half, 2^{451} mod 7. Next we can do 4 mod 7 = 4 again, and square. 16 mod 7 = 2. Once again we cut our exponent, 2^{225} mod 7. Now we have $4 \cdot 2$ mod 7 $\rightarrow$ 8 mod 7 = 1. Next we square our other value, 4 mod 7 = 4. We divide exponent again, 2^{112} mod 7, and we do 16 mod 7 = 2. Another cut, 2^{56} mod 7. We can check 2^2 mod 7 = 4. Another time we cut, 2^{28} mod 7. We need to use previous value again, 16 mod 7 = 2. 2^{14} mod 7 from another cut, and we use 4 mod 7 = 4. We can cut again, 2^7 mod 7 and we use 4 mod 7 = 4. We are almost done and use 2^3 mod 7. We must check 8 mod 7 = 1, and now we are on the final step. 2^1 mod 7 = 4
- $11 \bmod 120 = 121, 13 \bmod 45 = 91, 9 \bmod 11 = 45$. For the last one and third one I used Extended Euclidean Algorithm discussed in class. I also used $p_i = p_{i-2} - p_{i-1}q_{i-2} \bmod n$.

  Third one: 35 mod 77 $\rightarrow$ $77 = 2(35) + 7$ and $p_0 = 0$. Next, $35 = 5(7) + 0$ and $p_1 = 1$. However, this can't be solved.

  Last one: 11 mod 1111 $\rightarrow$ $1111 = 101(11) + 0$. This one can't be solved either because we were unable to get past the step, like the third one.
- $\forall y \in [1, x-1] : \gcd(x, y) = 1$. If we want to find all of the modulo x^m between $0, 1, ..., x^m - 1$ then we can assume there are m total modulo inverses to compute.

Problem 5

Part C

Problem 6

Problem 7

Part D

Problem 8

Problem 9