

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

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

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

$$f = \Omega(g)$$

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

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

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

$$f = \Theta(g)$$

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)

```

1  $N \leftarrow \text{Random\_Sample}(0, 2^n - 1)$ ;
2 if  $N$  is even then
3    $N \leftarrow N + 1$  /* Worse case, N is odd,  $2 \cdot N - 1$ . */ ;
4  $m \leftarrow N \bmod n$  /* worse case same as  $n$  */ ;
5 for  $j \leftarrow 0$  to  $m$  do
6   if Greatest_Common_Divisor( $j, N$ )  $\neq 1$  then
7     return FALSE; /* GCD is  $O(n)$  */
8   Compute  $x, z$  so that  $N - 1 = 2^z \cdot x$  and  $x$  is odd;
9    $y_0 \leftarrow (N - 1 - j)^x \bmod N$ ;
10  for  $i \leftarrow 1$  to  $m$  do
11     $y_i \leftarrow y_{i-1}^2 \bmod N$  ;
12     $y_i \leftarrow y_i + y_{i-1} \bmod N$ ;
13  if Low_Error_Primality_Test( $y_m$ ) == FALSE then
14    return FALSE /* Naive primality test is  $O(\sqrt{n})$  */ ;
15 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 ? $2^{902} \bmod 7$ We can find the original, $2 \bmod 7 = 2$ because 7 doesn't go into 2 at all. We can next square, finding $4 \bmod 7 = 4$. Divide exponent in half, $2^{451} \bmod 7$. Next we can do $4 \bmod 7 = 4$ again, and square. $16 \bmod 7 = 2$. Once again we cut our exponent, $2^{225} \bmod 7$. Now we have $4 \cdot 2 \bmod 7 \rightarrow 8 \bmod 7 = 1$. Next we square our other value, $4 \bmod 7 = 4$. We divide exponent again, $2^{112} \bmod 7$, and we do $16 \bmod 7 = 2$. Another cut, $2^{56} \bmod 7$. We can check $2^2 \bmod 7 = 4$. Another time we cut, $2^{28} \bmod 7$. We need to use previous value again, $16 \bmod 7 = 2$. $2^{14} \bmod 7$ from another cut, and we use $4 \bmod 7 = 4$. We can cut again, $2^7 \bmod 7$ and we use $4 \bmod 7 = 4$. We are almost done and use $2^3 \bmod 7$. We must check $8 \bmod 7 = 1$, and now we are on the final step. $2^1 \bmod 7 = 4$
- $11 \bmod 120 = 121, 13 \bmod 45 = 91, 9 \bmod 11 = 45$. 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: $35 \bmod 77 \rightarrow 77 = 2(35) + 7$ and $p_0 = 0$. Next, $35 = 5(7) + 0$ and $p_1 = 1$. However, this can't be solved.
Last one: $11 \bmod 1111 \rightarrow 1111 = 101(11) + 0$. 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 $0, 1, \dots, x^m - 1$ then we can assume there are m total modulo inverses to compute. An example is that there $x = 2, m = 2$ to keep it simple. This means that every number from $1 \rightarrow 1 : \gcd(1, 1) = 1$ which is correct. Now we need to find $0, \dots, 2^2 - 1$ which becomes $0, \dots, 3$. We have a total of 4 numbers to modulo inverse. The running time to find is the amount multiplied by the time it takes to run the euclidean algorithm. There's a total of x^m to find and the Extended Euclidean algorithm takes $\log(m^2)$. Our total runtime is $x^m \log(m^2)$.

Problem 5

Part C

Problem 6

- The hash function for the family is definitely consistent because each item is only either 0, 1, and we are modding by the total amount of choices, but I'm not sure how to prove this other than by what was stated in class: $Pr = h_\alpha(x) = h_\alpha(y) = Pr \sum_{i=1}^4 \alpha_i \cdot x_i = \sum_{i=1}^4 \alpha_i \cdot y_i \bmod N =$

$$Pr \sum_{i=1}^3 \alpha_i (x_i - y_i) = \alpha_4 (y_4 - x_4 \bmod N)$$

given x, y and randomly picked a1,a2,a3: c is constant

•

Problem 7

Part D

Problem 8

Problem 9