Sample Solution for CS3323 Fall 2006 Assignment 1 (41 marks)

Due Friday Sept 29, by 5pm.

1. What does the following algorithm do? Analyze its worst-case running time, figure out its running time function, and express it using "Big-Oh" notation.

Algorithm Foo (a, n):
Input: two integers, a and n
Output: ?

$k \leftarrow 0$
$b \leftarrow 1$
while $k < n$ **do**
  $k \leftarrow k + 1$
  $b \leftarrow b * a$
return b

Solution: (5 marks) This algorithm computes a^n. The running time of this algorithm is $O(n)$ because

- the initial assignments take constant time
- each iteration of the **while** loop takes constant time
- there are exactly n iterations

2. What does the following algorithm do? Analyze its worst-case running time, figure out its running time function, and express it using "Big-Oh" notation.

Algorithm Bar (a, n):
Input: two integers, a and n
Output: ?

$k \leftarrow n$
$b \leftarrow 1$
$c \leftarrow a$
while $k > 0$ **do**
  if $k \bmod 2 = 0$ **then**
    $k \leftarrow k/2$
    $c \leftarrow c * c$
  else
    $k \leftarrow k - 1$
    $b \leftarrow b * c$
return b

Solution: (8 marks) This algorithm also computes a^n. Its running time is $O(\log n)$ for the following reasons:

The initialization and the **if** statement and its contents take constant time, so we need to figure out how many times the **while** loop gets called. Since k goes down (either gets halved or decremented by one) at each step, and it is equal to n initially, at worst the loop gets executed n times. But we can (and should) do better in our analysis.

Note that if k is even, it gets halved, and if it is odd, it gets decremented, and halved in the next iteration. So at least every second iteration of the **while** loop halves k. One can halve a number n at most $\lceil \log n \rceil$ times before it becomes ≤ 1 (each time we halve a number we shift it right by one bit, and a number has $\lceil \log n \rceil$ bits). If we decrement the number in between halving it, we still get to halve no more then $\lceil \log n \rceil$ times. Since we can only decrement k in between two halving iterations (unless n is odd or it is the last iteration), we get to do a decrementing iteration at most $\lceil \log n \rceil + 2$ times. So we can have at most $2\lceil \log n \rceil + 2$ iterations. This is obviously $O(\log n)$.

3. Algorithm A executes $10n \log n$ operations, while algorithm B executes n^2 operations. Determine the minimum integer value n_0 such that A executes fewer operations than B for $n \geq n_0$.

   Solution: (5 marks) Assume that the base of the log is 2. We must find the minimum integer n_0 such that $10n \log n < n^2$. Since n describes the size of the input data set that the algorithms operate upon, it will always be positive. Since n is positive, we may factor an n out of both sides of the inequality, giving us $10logn < n$. Let us consider the left and right hand side of this inequality. These two functions have one intersection point for $n > 1$, and it is located between $n = 58$ and $n = 59$. Indeed, $10 \log 58 \approx 58.57981 > 58$ and $10 \log 59 = 58.82643 < 59$. So for $1 \leq n \leq 58$, $10n \log n \geq n^2$, and for $n \geq 59$, $10n \log n < n^2$. So n_0 we are looking for is 59.

4. Prove or disprove each of the following statements:

   (a) $10n^2 + 8n + 2$ is $O(n^2)$.
   Proof: (3 marks)

$$\begin{aligned} 10n^2 + 8n + 2 &\leq 10n^2 + 8n^2 + 2n^2 \\ &= 20n^2 \end{aligned} \tag{1}$$

   Let $C = 20$ and $n_0 = 1$. We have $10n^2 + 8n + 2 \leq Cn^2$ for all $n \geq n_0$.

(b) $3(n+1)^7 + 2n\log n$ is $O(n^7)$.
Proof: (3 marks)

$$\begin{aligned} 3(n+1)^7 + 2n\log n &\leq 3(n+n)^7 + 2n\log n \\ &= (3\times 2^7)n^7 + 2n\log n \\ &\leq (3\times 2^7)n^7 + 2n^7 \\ &= (3\times 2^7+2)n^7 \end{aligned} \tag{2}$$

Let $C = 3\times 2^7 + 2$ and $n_0 = 1$. We have $3(n+1)^7 + 2n\log n \leq Cn^7$ for all $n \geq n_0$.

(c) $3n^5 + 10n^4\log_2 n - 10n^3 - 15n^2$ is $O(n^5)$
Proof: (3 marks)

$$\begin{aligned} 3n^5 + 10n^4\log_2 n - 10n^3 - 15n^2 &\leq 3n^5 + 10n^4\log_2 n \\ &= 3n^5 + 10n^5 \\ &= 13n^5 \end{aligned} \tag{3}$$

Let $C = 13$ and $n_0 = 1$. We have $3n^5 + 10n^4\log_2 n - 10n^3 - 15n^2 \leq Cn^5$ for all $n \geq n_0$.

(d) $10n^4$ is $O(10000n^3\log_2 n)$
Disproof: (4 marks) To make this true, we should have constants C and n_0, such that for all $n \geq n_0$,

$$\begin{aligned} 10n^4 &\leq C10000n^3\log_2 n \\ 10n &\leq C10000\log_2 n \\ \frac{n}{1000\log_2 n} &\leq C \end{aligned} \tag{4}$$

Since the growth rate of n is greater than $\log_2 n$ and C is a constant, $\frac{n}{1000\log_2 n} \geq C$ when n is large. Therefore, it is not possible to find an n_0 to make $\frac{n}{1000\log_2 n} \leq C$ for all $n \geq n_0$ given any constant C.

5. Order the following functions by the big-O notation, starting from the smallest one.

$$3^{\log_9 n} \quad \log_8 n^3 \quad \log_{10}\log_{10} n^{100} \quad \sqrt{n} \quad n^{0.001} \quad \log_2 n \quad (\log_2 n)^2$$

Solution: (5 marks)

The increasing order in growth rate is:

$$\log_{10}\log_{10} n^{100} \quad (\log_8 n^3, \log_2 n) \quad (\log_2 n)^2 \quad n^{0.001} \quad (3^{\log_9 n}, \sqrt{n})$$

6. Prove that if $f(n)$ is $O(g(n))$ and $d(n)$ is $O(h(n))$, then $f(n)+d(n)$ is $O(g(n)+h(n))$.

   Proof: (5 marks) Recall the definition of big-Oh notation: we need constants $c > 0$ and $n_0 \geq 1$ such that $f(n) + d(n) \leq c(g(n) + h(n))$ for every integer $n \geq n_0$.

   $f(n)$ is $O(g(n))$ means that there exists $c_f > 0$ and an integer $n_{0f} \geq 1$ such that $f(n) \leq c_f g(n)$ for every $n \geq n_{0f}$. Similarly, $d(n)$ is $O(h(n))$ means that there exists $c_d > 0$ and an integer $n_{0d} \geq 1$ such that $d(n) \leq c_d h(n)$ for every $n \geq n_{0d}$.

   Let $n_0 = \max(n_{0f}, n_{0d})$, and $c = \max(c_f, c_d)$. So $f(n) + d(n) \leq c_f g(n) + c_d h(n) \leq c(g(n) + h(n))$ for $n \geq n_0$. Therefore $f(n) + d(n)$ is $O(g(n) + h(n))$.