Homework 1 Solutions
Fundamental Algorithms, Spring 2003, Professor Yap

Due: Mon Feb 10, in class
SOLUTION PREPARED BY T.A. Igor Chikanian

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.
- There are links from the homework page to the old homeworks from two previous classes, including solutions. Feel free to study these.

---

1. (10 Points) Let $p(x) = 3x^3 - 1000x^2 + 1$. Prove the following statement: $p(x) = \Theta(x^3)$. NOTE: You should normally break up a Θ-statement into an upper bound statement and a lower bound statement.

   Solution: We need to show that (1) $p(x) = \mathcal{O}(n^3)$, and (2) $p(x) = \Omega(n^3)$.

   Showing (1) amounts to showing that there exists $c > 0$ such that $p(x) \leq cx^3$ (eventually), i.e., there is some x_0 such that for all $x \geq x_0$, $p(x) \leq cx^3$. The following sequence of assertions are seen to follow:

   For $x \geq 1$, we have:

$$\begin{aligned} 1000x^2 &\geq 1, \\ -1000x^2 + 1 &\leq 0, \\ 3x^3 - 1000x^2 + 1 &\leq 3x^3. \qquad \text{(Adding } 3x^3 \text{ to both sides)} \end{aligned}$$

   Thus, we can choose $c = 3$ and $x_0 = 1$. Q.E.D.

   Showing (2) amounts to showing that there exists $c > 0 : p(x) \geq cx^3$ (eventually), i.e., there is some x_0 such that for all $x \geq x_0$, $p(x) \geq cx^3$. For $x \geq 1000$, we have:

$$\begin{aligned} x^3 - 1000x^2 &\geq 0, \\ x^3 - 1000x^2 + 1 &\geq 0, \\ 3x^3 - 1000x^2 + 1 &\geq 2x^3, \qquad \text{(Adding } 2x^3 \text{ to both sides)} \end{aligned}$$

   Thus, we can choose $c = 2$ and $x_0 = 1000$. Q.E.D.

2. (15 Points)
   (a) What is the relation between these two statements: $f(n) \neq O(n^2)$ and $f(n) = \Omega(n^2)$.
   (b) Construct an example of $f(n)$ for which these two statements are not equivalent.

   Solution: Let us expand both statements:

   statement 1: $f(n) \neq O(n^2)$
   The following lines are each equivalent to statement 1:
   $\nexists c > 0 : f(n) \leq cn^2$ (eventually)
   $\nexists c > 0 : \exists N > 0 : \forall n > N : f(n) \leq cn^2$
   $\forall c > 0 : \nexists N > 0 : \forall n > N : f(n) \leq cn^2$
   $\forall c > 0 : \forall N > 0 : \not\forall n > N : f(n) \leq cn^2$
   $\forall c > 0 : \forall N > 0 : \exists n > N : f(n) > cn^2$

   statement 2: $f(n) = \Omega(n^2)$
   The following lines are each equivalent to statement 2:
   $\exists c > 0 : f(n) \geq cn^2$ (eventually)
   $\exists c > 0 : \exists N > 0 : \forall n > N : f(n) \geq cn^2$

   For part (b) consider $f(n) = \begin{bmatrix} n^3, & if\ n\ is\ odd \\ 0, & if\ n\ is\ even \end{bmatrix}$

   Given $c > 0$ and $N > 0$ let n be: $n > N, n > c$ and n is odd.

Then $f(n) = n^3 > cn^2$, so statement 1 is satisfied.
On the other hand, $\forall c > 0 : \forall N > 0$, take $n = 2N$,
then $f(n) = 0 < cn^2$, so statement 2 is not satisfied.

For part (a) the following truth table of examples shows that there is no relationship between the two statements:

Example	$f(n) \neq \mathcal{O}(n^2)$	$f(n) = \Omega(n^2)$
$f(n) = n^3$	*True*	*True*
$(part(b)above)$	*True*	*False*
$f(n) = n^2$	*False*	*True*
$f(n) = n$	*False*	*False*

3. (20 Points) This question should be done without resort to Calculus.
(a) Show that $H_n \to \infty$ as $n \to \infty$.
(b) Show that $H_n = O(n^{1/k})$ for all positive integer $k \geq 2$. HINT: Break the summation into k parts – this is similar to an argument in Section 6 of Lecture 2.
(c) Conclude from (b) that $\log n = O(n^\alpha)$ for all real $\alpha > 0$.

Solution:

(a) Break up the sum H_n into infinite number of parts B_i:
$B_0 = 1$
$B_1 = 1/2$
$B_2 = 1/3 + 1/4$
$B_3 = 1/5 + 1/6 + 1/7 + 1/8$
...
$B_i = \frac{1}{2^{i-1}+1} + ... + \frac{1}{2^i}$, a series of 2^{i-1} terms, each no less then $\frac{1}{2^i}$. Thus we see each $B_i \geq 1/2$.
So $H_{2^n} = B_0 + B_1 + B_2 + ... + B_n > n/2 \to \infty$ as $n \to \infty$.

(b) Write

$$\begin{aligned} H_n &= 1 + \frac{1}{2} + \cdots + \frac{1}{\lceil n^{1/k} \rceil} \\ &\quad + \frac{1}{\lceil n^{1/k} \rceil + 1} + \cdots + \frac{1}{\lceil n^{2/k} \rceil} \\ &\quad + \frac{1}{\lceil n^{2/k} \rceil + 1} + \cdots + \frac{1}{\lceil n^{3/k} \rceil} \\ &\quad + \cdots \\ &\quad + \frac{1}{\lceil n^{(k-1)/k} \rceil + 1} + \cdots + \frac{1}{n} \\ &= 1 + A_1 + A_2 + \cdots + A_k, \end{aligned}$$

where

$$A_i = \sum_{j=\lceil n^{(i-1)/k} \rceil + 1}^{\lceil n^{i/k} \rceil} \frac{1}{j}.$$

The largest term in each A_i is $\frac{1}{\lceil n^{(i-1)/k} \rceil + 1} < \frac{1}{n^{(i-1)/k}}$ The number of terms in A_i is $\lceil n^{i/k} \rceil - \lceil n^{(i-1)/k} \rceil \leq n^{i/k} - n^{(i-1)/k} + 1$. Therefore, each $A_i \leq \frac{n^{i/k} - n^{(i-1)/k} + 1}{n^{(i-1)/k}} = n^{1/k} - 1 + \frac{1}{n^{(i-1)/k}} \leq n^{1/k}$.

Rewriting $H_n = 1 + A_1 + A_2 + \ldots + A_k \leq 1 + kn^{1/k} = \mathcal{O}(n^{1/k})$ for any constant natural k.

(c) Since for all $\alpha > 0$ there exists $k > 0$ such that $1/k < \alpha$, we can conclude that $n^{1/k} < n^\alpha$ for such k eventually, and so $\forall \alpha > 0 : log(n) = \Theta(H_n) = \mathcal{O}(n^{1/k}) = O(n^\alpha)$.

4. (10 Points) You have been asked to update Java's standard class "BigInteger" that perform arithmetic and other operations on arbitrarily large integers. You first determined that the old implementation of the multiplication algorithm takes time $T_0(n) = 2n^2 + 20n + 10$ for all $n \geq 1$ (this is, of course, an implementation of the High School Algorithm). Since you learned about Karatsuba, you decided to implement it, and found that its running time is $T_1(n) = 10n^\alpha + 20n + 60$ for all $n \geq 1$, where $1.584 < \alpha = \lg 3 < 1.585$. How can you take advantage of these two multiplication algorithms for your next release of Java's "BigInteger" class? NOTE: You should do some numerical calculations with $T_0(n)$ and $T_1(n)$ using perhaps a pocket calculator.

   Solution: Consider the difference $\delta(n) = T_0(n) - T_1(n)$:

$$\begin{aligned}\delta(n) &= (2n^2 + 20n + 10) - (10n^\alpha + 20n + 60) \\ &= 2n^2 - 10n^\alpha - 50 \\ &= 2n^\alpha(n^{2-\alpha} - 5) - 50\end{aligned}$$

   We want to find a value n_0 such that $\delta(n) \geq 0$ for all $n \geq n_0$. We can use a calculator to check that $\delta(50) > 20$. NOTE: we should always remember that exponentiation, n^c, is only defined when $n > 0$.

   Now we want to conclude that for all $n \geq 50$, $\delta(n) \geq 0$. To see that, let $\delta_1(n) = 2n^\alpha$ and $\delta_2(n) = n^{2-\alpha} - 5$. Now $\delta_1(n)$ is positive and increasing for all $n > 1$. Now $\delta_2(n)$ is increasing for all $n > 1$, but not necessarily positive. Therefore, the moment it becomes positive, it will remain positive. At $n = 50$, $\delta_2(n)$ is positive: e.g., your pocket calculator can show you that $\delta_2(50) > 0.07$.

   Furthermore, $\delta_1(n)\delta_2(n) > 50$ when $n = 50$ (this was the first calculation we mentioned, i.e., $\delta(50) > 20$). We conclude that $\delta_1(n)\delta_2(n) - 50$ is positive for all $n \geq 50$. This is exactly what we wanted to show.

   [NOTE: we deliberately avoided calculus in our argument. If you like, you may use calculus to compute derivatives of $\delta(n)$, etc, to get the same conclusion.]

   The new implementation of Java's multiplication may look as follows:

   int Multiply(int n, int m) {
   if (($bitSize(n) > 50$) or ($bitSize(m) > 50$)) return Karatsuba(n,m);
   else return HighSchool(n,m);
   }

   The running time of this algorithm is bounded by $T_1(\max\{bitSize(n), bitSize(m)\})$
   when $\max\{bitSize(n), bigSize(m)\} \geq 50$.

5. (20 Points) Use the Rote Method to solve the following recurrence: $T(n) = 8T(n/2) + n$. The method involves 4 steps (Expand, Guess, Verify, Stop). Make sure that each step is clearly marked and explained. Be sure to tell us what initial condition you choose.

   Solution:

   Expand:

$$\begin{aligned}T(n) &= 8T(n/2) + n \\ &= 8(8T(n/2^2) + n/2) + n \\ &= 8^2T(n/2^2) + 4n + n \\ &= 8^2(8T(n/2^3) + n/2^2) + 4n + n \\ &= 8^3T(n/2^3) + 16n + 4n + 1 \\ &= 8^3(8T(n/2^4) + n/2^3) + 16n + 4n + n \\ &= 8^4T(n/2^4) + 64n + 16n + 4n + n.\end{aligned}$$

   Guess: $T(n) = 8^kT(n/2^k) + \sum_{i=0}^{k-1} 4^i n$.

Verify:

$$\begin{aligned} T(n+1) &= 8^k(8T(n/2^{k+1}) + n/2^k) + \sum_{i=0}^{k-1} 4^i n \\ &= 8^{k+1}T(n/2^{k+1}) + 4^k n + \sum_{i=0}^{k-1} 4^i n \\ &= 8^{k+1}T(n/2^{k+1}) + \sum_{i=0}^{k} 4^i n \end{aligned}$$

Stop: Let us set the initial condition as $T(n) = 0$ when $n \le 1$, and let $k = \lfloor \log_2 n \rfloor + 1$. We get:

$$\begin{aligned} T(n) &= n(1 + 4 + 4^2 + 4^3 + ... + 4^{\lfloor \log_2 n \rfloor}) \\ &= n(4^{\lfloor \log_2 n \rfloor + 1} - 1)/3 \\ &\le (4/3)n2^{\log_2(n^2)} \\ &= \Theta(n^3). \end{aligned}$$

(We used geometric series summation here.)

6. (20 Points) Let $T(n) = 10T(n/3) + n$.
(a) Use Real Induction to show that $T(n) = O(n^\alpha)$ when $\alpha = 3$.
(b) By examining your proof in (a), find the best possible value for α such that your proof will still work.

Solution: (both (b) and (a)) Let us guess that $T(n) \le cn^\alpha - bn$ for some $c > 0, b > 0$. Using this assumption for n/3 we get: $T(n) \le 10(c(n/3)^\alpha - bn/3) + n = (10/3^\alpha)cn^\alpha - ((10/3)b - 1)n$.

Let us choose $\alpha \ge \log_3(10)$, c=100, and b=3. Then $T(n) \le cn^\alpha - 9n < cn^\alpha - bn$. Also, the base case holds: If we use the initial condition $T(n) = 1$ for $n < 1$ then:

$T(1) = 10 + 1 = 11 < 100 - 3 = c1^\alpha - b$.

Thus by induction, $T(n) \le cn^\alpha - bn = \mathcal{O}(cn^\alpha)$. This proof works for $\alpha \ge \log_3(10)$, so it works for $\alpha = 3$ as in part (a).