CS325 Winter 2017: practice question set 1

This set of practice questions help you review the following concepts:

- Asymptotic growth of functions.
- Proof by induction.
- Characterize run time of recursive algorithms with recurrence relations.
- Solving recurrence relations using recursion tree and telescoping.

DPV: Algorithms by Dasgupta, Papadimitriou & Vazirani

1. DPV 0.1

   The last column provides an intuitive characterize of the relationship between f and g in terms of the asymptotic runtime.

	$f(n)$	$g(n)$	$f = O(g)$?	$f = \Omega(g)$?	$f = \theta(g)$?	
a	$n - 100$	$n - 200$	*yes*	*yes*	*yes*	$=$
b	$n^{1/2}$	$n^{2/3}$	*yes*	*no*	*no*	$<$
c	$100n + \log n$	$n + (\log n)^2$	*yes*	*yes*	*yes*	$=$
d	$n \log n$	$10n \log 10n$	*yes*	*yes*	*yes*	$=$
e	$\log 2n$	$\log 3n$	*yes*	*yes*	*yes*	$=$
f	$10 \log n$	$\log(n^2)$	*yes*	*yes*	*yes*	$=$
g	$n^{1.01}$	$n(\log n)^2$	*no*	*yes*	*no*	$>$
h	$n^2 / \log n$	$n(\log n)^2$	*no*	*yes*	*no*	$>$
i	$n^{0.1}$	$(\log n)^{10}$	*no*	*yes*	*no*	$>$
j	$(\log n)^{\log n}$	$n / \log n$	*no*	*yes*	*no*	$>$
k	$\sqrt{n}$	$(\log n)^3$	*no*	*yes*	*no*	$>$
l	$n^{1/2}$	$5^{\log_2 n}$	*yes*	*no*	*no*	$<$
m	$n2^n$	3^n	*yes*	*no*	*no*	$<$
n	2^n	2^{n+1}	*yes*	*yes*	*yes*	$=$
o	$n!$	2^n	*no*	*yes*	*no*	$>$
p	$(\log n)^{\log n}$	$2^{(\log_2 n)^2}$	*yes*	*no*	*no*	$<$
q	$\sum_{i=1}^{n} i^k$	n^{k+1}	*yes*	*yes*	*yes*	$=$

   The last column shows the intuitive relationship. It may help you reason about the relationship but it should not be used in the formal setting.

2. DPV 0.2 **Solution:**

   First, we note that if $c = 1$, $g(n) = n + 1 = \theta(n)$. For the other two cases, we will first derive a close-form representation of the sum. Note that we have

$$g(n) = 1 + c + c^2 + ... + c^n$$

   thus:

$$c \times g(n) = c + c^2 + ... + c^{n+1}$$

   Substracting the first equation from the second on both sides, we have:

$$(c - 1)g(n) = c^{n+1} - 1$$

Thus we arrive at a close-form representation:

$$g(n) = \frac{c^{n+1} - 1}{c - 1}$$

If $c < 1$, we have:

$$\lim_{n\to\infty} g(n) = \frac{0-1}{c-1} = \frac{1}{1-c} \quad (\textit{since } \lim_{n\to\infty} c^{n+1} = 0)$$

From this we conclude that $g(n) = \Theta(1)$ for $c < 1$.

If $c > 1$, we will use the limit trick:

$$\begin{aligned} \lim_{n\to\infty} \frac{g(n)}{c^n} &= \lim_{n\to\infty} \frac{c^{n+1}-1}{c^{n+1}-c^n} \\ &= \lim_{n\to\infty} \frac{c - \frac{1}{c^n}}{c-1} && (\textit{divide by } c^n \textit{ simultaneously top and bottom}) \\ &= \frac{c}{c-1} && (\lim_{n\to\infty} \tfrac{1}{c^n} = 0 \textit{ for } c > 1) \end{aligned}$$

Because the limit goes to a nonzero finite constant, this allows us to conclude that $g(n) = \Theta(c^n)$ for $c > 1$.

For the purpose of this class, you are not required to learn to derive the close-form solution for a geometric serie. But it is required that you can recognize the geometric series and understand its asymptotic complexity in the three cases because they are frequently encountered in run time analysis.

3. DPV 0.3(a)

   Solution: *(Base cases:)*

   if $n = 6$, we have $F_6 = 8 \geq 2^{6/2} = 8$.

   if $n = 7$, we have $F_7 = 13 \geq 2^{7/2} = 11.31$.

   Note that $F_{n+1} = F_n + F_{n-1}$. We need the inductive assumption to cover both F_n and $F_n n - 1$. That is why we need to base cases here.

   (Inductive assumption): Assume that the statement is true for $6, 7, ..., n$. Our goal is to show that the statement is true for $n+1$.

   (Inductive case:) if $n \geq 7$, the inductive assumption implies that $F_n \geq 2^{0.5(n)}$ and $F_{n-1} \geq 2^{0.5(n-1)}$.

   As such, we have: $F_{n+1} = F_n + F_{n-1} \geq 2^{n/2} + 2^{(n-1)/2} = 2^{(n-1)/2}(2^{1/2}+1) \geq 2^{(n-1)/2} \cdot 2 = 2^{(n+1)/2}$

   Thus, for all $n \geq 6$, we have $F_n \geq 2^{0.5n}$.

4. DPV 2.3 ***Solution:***

   (a) *From the recurrence relation, $T(n) \leq 3T(n/2) + cn$ for some constant c. So:*

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

   So the general pattern is:

   $$T(n) \leq 3^kT(n/2^k) + cn\sum_{i=0}^{k}(3/2)^i$$

   Since n is reduced by a factor of 2 in each step, the depth of the recursion tree is $\log_2 n$. Setting $k = \log_2 n$ and using the fact that $\sum_{i=0}^{k}(3/2)^i = O((3/2)^k)$ by the Geometric Series Lemma,

   $$T(n) \leq 3^{\log_2 n}T(1) + cn3^{\log_2 n}/2^{\log_2 n}$$

   Since $3^{\log_2 n} = n^{\log_2 3}$ and $T(1) \leq c$,

   $$T(n) \leq 2cn^{\log_2 3} = O(n^{\log_2 3})$$

(b) *This one is easier:*

$$\begin{aligned} T(n) &\leq T(n-1) + c \text{ for some constant } c \\ &\leq T(n-2) + c + c = T(n-2) + 2c \end{aligned}$$

And the general pattern is:

$$T(n) = T(n-k) + ck$$

Since n is reduced by 1 in each iteration, it will take $k = n$ recursions to reach the base case $T(0)$, setting $k = n$ results in:

$$T(n) = O(n)$$

5. DPV 2.4 *The recursive relations for each of the three algorithms are:*
   A: $T(n) = 5T(n/2) + O(n)$
   B: $T(n) = 2T(n-1) + c$
   C: $T(n) = 9T(n/3) + O(n^2)$

   You can use master theorem to solve for A and C, we have:
   A: $T(n) = O(n^{\log_2 5})$
   C: $T(n) = O(n^2 \log n)$

   As for case B, we will use the technique introduced in problem 3:
   $T(n) = 2T(n-1) + c = 2(2T(n-2) + c) + c = 4T(n-2) + (2+1)c = 4(2T(n-3) + c) + (2+1)c = 8T(n-3) + (4+2+1)c = 2^k T(n-k) + c\sum_{i=0}^{k-1} 2^i$
   The depth $k = n$, thus we have $T(n) = O(2^n)$

   Among the three, algorithm C has the lowest time complexity.