EE 602 – Algorithm I	**(UH Manoa, Fall 2020)**

Algorithms Assignment 4

September 23, 2020

Instructor: Yingfei Dong *Name: Frendy Lio Can*

Problem 1

To prove this, we have to show that there exists constants $c_1, c_2, n_0 > 0$ such that
$0 \leq c_1 n^b \leq (n+a)^b \leq c_2 n^b$, $\forall n \geq n_0$.

We also can observe that $n + a \leq 2n$, when $|a| \leq n$; and $n + a \geq \frac{1}{2}n$, when $|a| \leq \frac{n}{2}$.

Thus, if $n \geq 2|a|$

$$\begin{aligned} & 0 \leq \frac{n}{2} \leq n + a \leq 2n \Leftrightarrow \\ \Leftrightarrow \quad & 0^b \leq \frac{n}{2}^b \leq (n+a)^b \leq (2n)^b, \quad b > 0 \\ \Leftrightarrow \quad & 0 \leq 2^{-b} n^b \leq (n+a)^n \leq 2^b n^b \end{aligned}$$

$\therefore \quad (n+a)^b = \Theta(n^b) \quad \because \quad \exists c_1, c_2, n_0 : c_1 = 2^{-b}, c_2 = 2^b, n_0 = 2|a| \quad$ when $b > 0$

Problem 2

Yes, $2^{n+1} = O(2^n)$ and $2^2n \neq O(2^n)$.

For $2^{n+1} = O(2^n)$

Assume that $2^{n+1} = O(2^n)$. Thus, we need to prove that there exists constants $c, n_0 > 0$ such that $0 \leq 2^{n+1} \leq c2^n \quad , \forall n \geq n_0$. We can observe that $0 \leq 2^{n+1} \leq c2^n \Leftrightarrow 0 \leq 2 \cdot 2^n \leq c2^n$.

Clearly, we can observe that there exists constants $c, n_0 > 0$ such that $c_0 \geq 2, n \geq 1$. Therefore $2^{n+1} = O(2^n)$.

For $2^2n \neq O(2^n)$

Assume that $2^2n = O(2^n)$. Thus, we need to prove that there exists constants $c, n_0 > 0$ such that $0 \leq 2^{2n} \leq c2^n \quad , \forall n \geq n_0$. We can observe that $0 \leq 2^{2n} \leq c2^n \Leftrightarrow 0 \leq 2^n \cdot 2^n \leq c2^n$.

Clearly, we can observe that there does not exists any constants c when $n \to \infty$. This contradicts the assumption $2^2n = O(2^n)$. Therefore, we can conclude that $2^2n \neq O(2^n)$

Problem 3

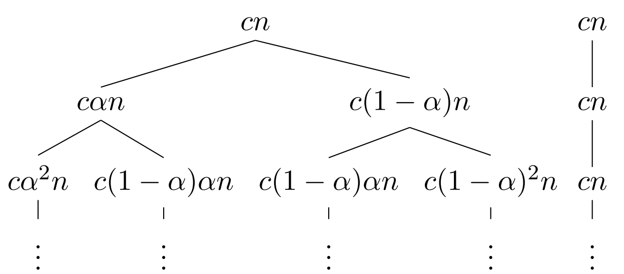
If $T(n) = O(n^2)$, there must exists constants $c, n_0 > 0$ such that $T(n) \leq cn^2, \quad \forall n \geq n_0$.

$$\begin{aligned}
T(n) = T(n-1) + n &\leq c(n-1)^2 + n \\
&\leq cn^2 - 2cn + 1 + n \\
&\leq cn^2 + (-2c+1)n + 1 \\
&\leq cn^2 + (-2c+1)n \\
&\leq cn^2 - 2cn \\
&\leq cn^2
\end{aligned}$$

We can observe that for $n \to \infty$ and $c \geq 1, \quad T(n) \leq cn^2$. Therefore, $T(n) = O(n^2)$.

Problem 4

Recursion tree:



We can observed that $f(n) = cn$, $a = c$ and $b = 1/(1-\alpha)$.

Therefore, in order to calculate the lower bound, we add the cost of each level of the tree by knowing that the tree is completed when the Recursion tree level is $log_b n$:

$$\begin{aligned}
T(n) &= \sum_{j=0}^{log_b \cdot n} f(n) \\
&= \sum_{j=0}^{log_b \cdot n} cn \\
&= \sum_{j=1}^{log_b \cdot n} cn + cn \\
&\geq n log_b n \\
&\geq n lgn \\
\therefore T(n) &= \Omega(n lgn)
\end{aligned}$$

In order to calculate the upper bound, we assume that there exists a constant b such that $T(n) \leq b \cdot n \cdot lgn$.

Thus, the upper bound is:

$$\begin{aligned}T(n) =&T(\alpha n) + T((1-\alpha)n) + cn\\ &\leq b(\alpha n)\cdot lg(\alpha n) + b[(1-\alpha)n]\cdot lg[(1-\alpha)n] + cn\\ &\leq [b\alpha nlg\alpha + b\alpha nlgn] + [b(1-\alpha)nlg(1-\alpha) + b(1-\alpha)nlgn] + cn\\ &\leq bnlgn + bn(\alpha lg\alpha + (1-\alpha)lg(1-\alpha)) + cn\\ &\leq bnlgn\end{aligned}$$

We can apply the same logic and find the lower bound which is:

$$T(n) \geq bnlgn$$

We can conclude that $T(n) = \Theta(nlgn)$, for any $n \geq n_0$ and $c \geq -b(\alpha lg\alpha + (1-\alpha)lg(1-\alpha))$.

Problem 5

a) $T(n) = 2T(\frac{n}{2}) + n^3$

We have $a = 2, b = 2, f(n) = n^3$ and $n^{log_b a} = n^{log_2 2} = n$. We can observe that case 3 of the master method applies since $f(n) = \Omega(n^{log_b a+\epsilon}), \epsilon = 3$.

Therefore, $T(n) = \Theta(n^3)$.

b) $T(n) = T(\frac{9n}{10}) + n$

We have $a = 1, b = 10/9, f(n) = n$ and $n^{log_{10/9} 1} = n^0 = 1$. We can observe that case 3 of the master method applies since $f(n) = \Omega(n^{log_b a+\epsilon}), \epsilon = 1$.

Therefore, $T(n) = \Theta(n)$.

c) $T(n) = 16T(\frac{n}{4}) + n^2$

We have $a = 16, b = 4, f(n) = n^2$ and $n^{log_4 16} = n^2$. We can observe that case 2 of the master method applies since $f(n) = \Theta(n^{log_b a})$.

Therefore, $T(n) = \Theta(n^{log_b a} lgn) = \Theta(n^2 lgn)$.

d) $T(n) = 7T(\frac{n}{3}) + n^2$

We have $a = 7, b = 3, f(n) = n^2$ and $n^{log_3 7} = n^{1.77}$. We can observe that case 3 of the master method applies since $f(n) = \Omega(n^{log_b a+\epsilon}), \epsilon \approx 0.23$.

Therefore, $T(n) = \Theta(n^2)$.

e) $T(n) = 7T(\frac{n}{2}) + n^2$

We have $a = 7, b = 2, f(n) = n^2$ and $n^{log_2 7} = n^{2.81}$. We can observe that case 1 of the master method applies since $f(n) = \Omega(n^{log_b a - \epsilon}), \epsilon \approx 0.81$.

Therefore, $T(n) = \Theta(n^{log_b a}) = \Theta(n^{log_2 7})$.

f) $T(n) = 2T(\frac{n}{4}) + n^{1/2}$

We have $a = 2, b = 4, f(n) = n^{1/2}$ and $n^{log_4 2} = n^{1/2}$. We can observe that case 2 of the master method applies since $f(n) = \Omega(n^{log_b a})$.

Therefore, $T(n) = \Theta(n^{log_4 2} lgn)$.

g) $T(n) = T(n-1) + n$

$$\begin{aligned} T(n) =& T(n-1) + n \\ T(n-1) =& T(n-2) + n - 1 \\ T(n-2) =& T(n-3) + n - 2 \end{aligned}$$

Therefore,

$$\begin{aligned} T(n) =& T(n-1) + n = T(n-2) + n - 1 + n = T(n-3) + n - 2 + n - 1 + n \\ T(n) =& T(n-k) + kn - k(k-1)/2 \end{aligned}$$

Thus, if our base case for $T(1)$ is $n - k = 1 \Rightarrow k = n - 1$. Thus,

$$T(n) = T(1) + (n-1)n - \frac{(n-1)(n-2)}{2}$$

We can clearly observed that it has complexity $\Theta(n^2)$.

h) $T(n) = T(n^{1/2}) + 1$

Let $n = 2^m$. Thus, $T(2^m) = T(2^{m \cdot 1/2}) + 1$. Assume tht $T(2^m) = S(m)$ where S is some function of m.

Thus, $S(m) = S(m/2) + 1$

We have $a = 1, b = 2, f(n) = 1$ and $n^{log_2 1} = 1$. We can observe that case 2 of the master method applies since $f(n) = \Omega(1)$.

Therefore, $S(m) = \Theta(lgm)$. Thus, since $n = 2^m \Rightarrow lgn = m$, $T(n) = \Theta(lg(lgn))$