Assignment - 1

CS 583 - Analysis of Algorithm

Mithilaesh Jayakumar
G01206238

1) Let $f(n)$ and $g(n)$ be asymptotically positive functions. Briefly prove or disprove each of the conjectures.

1.1) $\boxed{f(n) = O(g(n)) \text{ implies } g(n) = O(f(n))}$

$O(g(n))$ means that $g(n)$ is the upper bound (worst case) of $f(n)$ therefore for a given function $f(n)$, $g(n)$ can satisfy upper bound but $f(n)$ cannot satisfy as a upper bound for $g(n)$ (ie) $O(f(n))$.

Let us consider $f(n) = n$ and $g(n) = n^2$ then

$n = O(n^2)$ holds true for all $c > 0$, $n_0 \geq 1$ where $n \geq n_0$

for example when $c = 1$ and $n_0 = 1$ $\quad f(n) \leq c\,g(n)$

$c = 1$ and $n_0 = 2$ $\quad f(n) \leq c\,g(n)$

but $n^2 = O(n)$ does not hold true for all $c > 0$, $n_0 \geq 1$ where $n \geq n_0$

for example when $c = 1$ and $n_0 = 1$ $\quad g(n) \geq c\,f(n)$

$c = 2$ and $n_0 = 5$ $\quad g(n) \geq c\,f(n)$ which violates the definition of $O()$. Therefore the conjecture is <u>false</u>

1.2) $\boxed{f(n) + g(n) = \Theta(\min(f(n), g(n)))}$

By definition $\Theta()$ refers to the tight bounds (worst & best case) of an algorithm.

Therefore $f(n) = \Theta(g(n))$ means

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $c_1, c_2 > 0$, $n_0 \geq 1$ and $n \geq n_0$

Let us consider $f(n) = n$ and $g(n) = n^2$. We know that $\min(f(n), g(n))$ is always $f(n)$.

We need to prove that $f(n)+g(n) = O(f(n))$.

Let us assume $c_1 = 1$ and $c_2 = 1$, $n_0 = 2$

then

$$c_1 f(n) \leq f(n)+g(n)$$

but

$$c_2 f(n) \leq f(n)+g(n)$$ which violates the definition of $O()$.

Therefore the conjecture <u>does not hold true</u>.

1·3) $f(n) = O(g(n))$ implies $g(n) = O(f(n))$. By definition

$O()$ means the upper bound (worst case) of an algorithm.

$f(n) = O g(n)$ means

$f(n) \leq c\, g(n)$ where $n \geq n_0$, $c > 0$ and $n_0 \geq 1$

$O()$ means both the upper and lower bound of an algorithm. By definition

$f(n) = O(g(n))$ means

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $c_1, c_2 > 0$, $n_0 \geq 1$ and $n \geq n_0$

Let us consider $f(n) = n$ and $g(n) = n^2$ then to show that

$f(n) = O(g(n))$ we need to prove

$$f(n) \leq c g(n)$$

Assume $C = 1$ and $n_0 = 2$

then $f(n) \leq c g(n)$ holds true.

Now we need to show that $g(n) = O(f(n))$. In order to prove this

assume $c_1 = 1$, $c_2 = 2$ and $n_0 = 2$

then $c_1 f(n) \leq g(n)$

but $g(n) \geq c_2 f(n)$ which violates the definition of $O()$.

Therefore the conjecture is <u>false</u>

1.4) $\boxed{f(n) = \Theta(f(n/2))}$.

By definition of $\Theta(n)$, $f(n) = \Theta(g(n))$ means

$c_1 g(n) \leq f(n) \leq c_2 g(n)$ where $c_1, c_2 > 0$, $n \geq n_0$ and $n_0 \geq 1$

Let us consider $f(n) = n$ then to show that $f(n) = \Theta(f(n/2))$

we need to prove $c_1 f(n/2) \leq f(n) \leq c_2 f(n/2)$

Assume $n_0 = 4$, $c_1 = 1$ and $c_2 = 1$

then $c_1 f(n/2) \leq f(n)$ is true

but $c_2 f(n/2) \leq f(n)$ which violates the definition. Hence the conjecture is <u>false</u>

1.5) $\boxed{\lg(n) = \Omega(n^e)}$ where e is a small positive number

$\Omega()$ means the lower bound (best case) of an algorithm. By definition

$f(n) = \Omega(g(n))$ means

$f(n) \geq c\, g(n)$ where $c > 0$, $n_0 \geq 1$ and $n \geq n_0$

Let us consider $c = 2$, $n_0 = 4$ and $e = 1$ then

$\lg(n) < c\, n^e$ which violates the definition. Hence the conjecture is <u>false</u>

2) Solve the recurrence $T(n) = 2T(n/2) + 1$. You can assume $T(1)$ is a constant.

We can solve this by using the substitution method,

Let us consider iterations in terms of k

$T(n) = 2T(n/2) + 1$ which is for $k=1$

now for $k=2$, $T(n/2) = 2T((n/2)/2) + 1$

$= 2T(n/4) + 1$

substituting $T(n/2)$ in $T(n)$,

$T(n) = 2[2T(n/4) + 1] + 1$

$= 4T(n/4) + 2 + 1$

now for $k = 3$,

$T(n/4) = 2T((n/4)/2) + 1$

$= 2T(n/8) + 1$

substituting $T(n/4)$ in $T(n)$,

$T(n) = 4\cancel{2}[2T(n/8) + 1] + 2 + 1$

$= 8T(n/8) + 4 + 2 + 1$

Writing $T(n)$ as a general form in terms of k,

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

In order for $T(n)$ to stop, $T(n/2^k)$ must be equal to a constant C,

$T(n/2^k) = C$

Let us assume $C=1$ which implies $(n/2^k) = 1$

$n = 2^k$

$\Rightarrow \log_2 n = k$

substituting this in $T(n)$,

$$T(n) = 2^{\log_2 n} T\left(n / 2^{\log_2 n}\right) + 2^{\log_2 n - 1}$$

we know that,

$n = 2^{\log_2 n}$. So using this in $T(n)$

$$T(n) = nT(n/n) + n - 1$$

$$= n\,T(1) + n - 1$$

we know that $T(1) = 1$

$$= n + n - 1$$

$$= 2n - 1$$

Ignoring the constants $T(n)$ is $O(n)$.

3) Solve the recurrence $T(n) = 49T(n/25) + n^{3/2} \log n$. You can assume $T(1)$ is a constant.

We can solve this problem by using Master's theorem,

comparing $T(n) = 49T(n/25) + n^{3/2} \log n$ with the general form $T(n) = aT(n/b) + f(n)$ where $f(n) = \theta(n^k \log^p n)$

we get

$a = 49$

$b = 25$

$k = 3/2$

$p = 1$

and these values satisfy initial conditions for masters theorem,

$a = 49$ $(a \geq 1)$

$b = 25$ $(b > 1)$

$k = 3/2$ $(k \geq 0)$

$p = 1$ (p is a real number)

As $49 < (25)^{3/2}$ this comes under case 3 and as $p = 1$ $(p \geq 0)$

$T(n) = O(n^k \log^p n)$

substituting the values of k and p we get,

$T(n) = 49T(n/25) + n^{3/2} \log n$ is $\underline{O(n^{3/2} \log n)}$