CS 344 Homework #1

Austin Bennett, Mark Barkalow, Jeremy Barkalow

September 29, 2019

1 Question #1

$\exists$ (c, N) s.t. $\forall$ n > N, f(n) $\leq$ c $\times$ g(n)

(a) 7n + log(n) = O(n)
Where c = 42 and N = 1
$\forall$ n > 1:
7n + log(n) $\leq$ 42n
n + log(n) $\leq$ n
n $\leq$ n ✓

(b) n^2 + 4n + 7 = $O(n^2)$
Where c = 42 and N = 1
$\forall$ n > 1:
n^2 + 4n + 7 $\leq$ 42n^2
n^2 + n $\leq$ n^2
$n^2 \leq n^2$ ✓

(c) n! = $O(n^n)$
Where c = 42 and N = 1
$\forall$ n > 1:
n $\times$ n-1 $\times$ n-2 $\times \cdots \times$ 1 $\leq$ 42n^n
n $\times$ n $\times$ n $\times \cdots \times$ 1 $\leq n^n$
$\approx n^n \leq n^n$ ✓

(d) $2^n = O(2^{2n})$
Where c = 42 and N = 1
$\forall$ n > 1:
$2^n \leq 42 \times 2^{2n}$
$2^n \leq 2^{2n}$
$2^n \leq 2^n \times 2^n$
$0 \leq 2^n$ ✓

2 Question #2

$\exists$ (c, n) s.t. $\forall$ n > N, f(n) $\leq$ c $\times$ g(n)

(a) f(n) = n^2, g(n) = n^3

Where c = 42 and N = 1

$\forall$ n > 1:

$n^2 = O(n^3)$

$n^2 \leq 42n^3$

$0 \leq 42n$ ✓

$n^3 = O(n^2)$

$n^3 \leq 42n^2$

$n \leq 42$ ✗ Not true $\forall$ n > 1

$\therefore$ f(n) = O(g(n))

(b) f(n) = $\log_2 n$, g(n) = $\log_3 n$

Where c = 42 and N = 1

$\forall$ n > 1:

$log_2 n \leq 42 log_3 n$

$\frac{\log_2 n}{\log_3 n} \leq 42$

$\frac{\log 3}{\log 2} \leq 42$ ✓

$\log_3 n \leq 42 \log_2 n$

$\frac{\log_3 n}{\log_2 n} \leq 42$

$\frac{\log 2}{\log 3} \leq 42$ ✓

$\therefore$ f(n) = Θ(g(n))

(c) f(n) = 2^n, g(n) = 3^n

Where c = 42 and N = 1

$\forall$ n > 1:

$2^n \leq 42 \times 3^n$

$\sqrt[n]{2^n \leq 42 \times 3^n}$

$2 \leq \sqrt[n]{42} \times 3$

$2 \leq \approx 1 \times 3$ ✓

$3^n \leq 42 \times 2^n$

$\sqrt[n]{3^n \leq 42 \times 2^n}$

$3 \leq \sqrt[n]{42} \times 2$

$3 \leq \approx 1 \times 2$ ✗

$\therefore$ f(n) = O(g(n))

(d) f(n) = 2^n, g(n) = 2^{n+1}

Where c = 42 and N = 1

$\forall$ n > 1:

$2^n \leq 42 \times 2^{n+1}$

$2^n \leq 42 \times 2 \times 2^n$

$2^n \leq 2^n$ ✓

$2^{n+1} \leq 42 \times 2^n$

$2^n \times 2 \leq 42 \times 2^n$

$2^n \leq 2^n$ ✓

$\therefore$ f(n) = Θ(g(n))

3 Question #3

$$T(n) = \begin{cases} 2 & \text{if n = 2} \\ 2\mathrm{T}(\frac{n}{2}) + \mathrm{n}) & \text{if n} = 2^k\text{, for k > 1} \end{cases} \tag{1}$$

Base step: let k = 1 so n = 2^1 = 2

Thus we have, T(2) = 2 ✓

Hypothesis step: Assume T(n) = n(log n) is true when k > 1 and where n = 2^k

So we assume that: T(2^k) = $2^k \times log(2^k)$

Inductive step: Given the previous assumption we get that T(2^{k+1}) = $2^{k+1} \times log(2^{k+1})$

From the recurrence relation we have T(n) = 2T($\frac{n}{2}$) + n

Letting n = 2^{k+1} we get that T(2^{k+1}) = 2T($\frac{2^{k+1}}{2}$) + 2^{k+1}

T(2^{k+1}) = 2T(2^k) + 2×2^k

T(2^{k+1}) = $2(2^k \times log(2^k)) + 2 \times 2^k$

T(2^{k+1}) = $2(2^k \times log(2^k) + 2^k)$

T(2^{k+1}) = $2^{k+1}(log(2^k) + 1)$

T(2^{k+1}) = $2^{k+1}(log(2^{k+1}))$ ✓

$\therefore$ T(n) = n lg n

4 Question #4

(a)

$$T(n) = \begin{cases} \Theta(1) & \text{if n = 1,} \\ \text{T(n-1)} + \Theta(\text{n}) & \text{if n > 1} \end{cases} \tag{2}$$

(b)

$$\begin{array}{c} \text{n} \\ | \\ \text{(n-1)} \\ | \\ \text{(n-2)} \\ | \\ \vdots \\ | \\ \text{(n-n+1)} \end{array}$$

Each level has an amount of work, c, = n
The height of the tree = n-1

$$T(n) = \sum_{k=0}^{n-1} ck = c(n-1)$$

$$= n(n-1)$$

$$= n^2 - n$$

Let us assume that, T(n) = $O(n^2)$

T(n+1) = T(n) + Θ(n+1) , from our equation in (a)
T(n+1) = n^2 + n + 1, remove constants (1) and dominated terms (n)
T(n+1) = $O(n^2)$
$\therefore$ Since T(n+1) = $O(n^2)$, $\Rightarrow$ T(n) = $O(n^2)$

5 Question #5

(a) and (b)
The original merge sort recurrence relation:

$$T(n) = \begin{cases} \Theta(1) & \text{if n = 1,} \\ 2T(\frac{n}{2}) + n & \text{if n > 1} \end{cases} \tag{3}$$

Updated recurrence relation with 4 array merge sorting:

$$T(n) = \begin{cases} \Theta(1) & \text{if n = 1,} \\ 4T(\frac{n}{4}) + n & \text{if n > 1} \end{cases} \tag{4}$$

(a) From the updated recurrence relation we can extrapolate two pieces of information
(1): 4T($\frac{n}{4}$) + n, and
(2): n
Expression (1): is the recursive call to quad merge sort each array and expression (2): the running time of actually merging the (already recursively called and sorted) arrays.
$\therefore$ The running time of merging the 4 arrays is O(n)

Now to solve for the running time of 4 array merge sort, we have:
T(n) = $4^k T(\frac{n}{4^k})$ + kn
Letting n = 4^k:
T(n) = nT($\frac{n}{n}$) + kn
= nT(1) + kn
= n + kn
From n = 4^k, we can infer that k = log n
Thus, we have T(n) = n + n(log n)
$\therefore$ T(n) = O(n log n) and the running time of merge sort has not improved upon the traditional algorithm.