Algorithms & Data Structures 2018/19 Coursework

clvp22

January 21, 2019

Question 4.

(a)

Firstly we can say $x^3 + 3x + 2$ is $O(x^3)$ because there exists some k and C such that,

$$x^3 + 3x + 2 \leq C \cdot x^3$$

for every $x \geq k$.
(example; $C = 6$, $k = 0$)

So, $x^3 + 3x + 2$ is $O(x^3)$. And $2x^4$ is $O(x^3 + 3x + 2)$ if and only if $2x^4$ is $O(x^3)$. We now can assume there are some constants k and C such that,

$$2x^4 \leq C \cdot x^3$$

for every $x \geq k$.

So,

$$C \geq 2x$$

however, $2x$ grows monotonically for $x > 0$. This is a contradiction because then no constant value for C can be found for $k > 0$. Therefore $2x^4$ is not $O(x^3)$ and consequently not $O(x^3 + 3x + 2)$.

(b)

For $x \geq 2$, $1 \leq \log x \leq x \leq x^2 \leq x^3$.

So,

$$4x^3 + 2x^2 \cdot \log x + 1 \leq 4x^3 + 2x^2 \cdot x + x^3 = 8x^3$$

for every $x \geq k$.

This inequality holds using $k = 2$ and $C = 8$ as a pair of witnesses,

$$4x^3 + 2x^2 \cdot \log x + 1 \leq C \cdot x^3$$

for every $x \geq k$.

Therefore $4x^3 + 2x^2 \cdot \log x + 1$ is $O(x^3)$.

(c)

$3x^2 + 7x + 1$ is $\omega(x \cdot \log x)$ if and only if $x \cdot \log x$ is $o(3x^2 + 7x + 1)$, which is,

$$\lim_{x \to \infty} \frac{C \cdot x \log x}{3x^2 + 7x + 1}$$

for every $x \geq k$.

For $x \geq 2$, $1 \leq \log x \leq x \leq x^2$.
So $3x^2 + 7x + 1$ grows faster than $x \cdot \log x$ so $\lim_{x \to \infty} \frac{C \cdot x \log x}{3x^2 + 7x + 1}$ holds for $k = 2$ and $C = 1$. Therefore, $x \cdot \log x$ is $o(3x^2 + 7x + 1)$, and consequently $3x^2 + 7x + 1$ is $\omega(x \cdot \log x)$.

(d)

For $x \geq 2$, $\log x \leq x \leq x^2$.

So,

$$x^2 + 4x \geq x^2 = x \cdot x \geq x \cdot \log x$$

for every $x \geq k$.

This inequality holds using $k = 2$ and $C = 1$ as a pair of witnesses,

$$x^2 + 4x \geq C \cdot x \log x$$

Therefore $x^2 + 4x$ is $\Omega(x \cdot \log x)$.

(e)

$f(x) + g(x)$ is $\Theta(f(x) \cdot g(x))$, if $f(x) + g(x)$ is $O(f(x) \cdot g(x))$ and if $f(x) + g(x)$ is $\Omega(f(x) \cdot g(x))$.
Suppose $f(x)$ is $O(g(x))$, or in other words,

$$f(x) \leq C \cdot g(x)$$

We can now ignore $f(x)$ from our previous statement as $g(x)$ dominates $f(x)$. Our question can be re-constructed as follows,

Is $g(x)$ $\Theta(f(x) \cdot g(x))$, for some $f(x)$, where $f(x)$ is $O(g(x))$.

Again, $g(x)$ is $\Theta(f(x) \cdot g(x))$, if $g(x)$ is $O(f(x) \cdot g(x))$ and if $g(x)$ is $\Omega(f(x) \cdot g(x))$.
Or in other words,

$$g(x) \leq C \cdot f(x) \cdot g(x)$$

and

$$g(x) \geq C \cdot f(x) \cdot g(x)$$

so,

$$C_1 \cdot f(x) \cdot g(x) \leq g(x) \leq C_2 \cdot f(x) \cdot g(x)$$

Clearly this inequality can only be satisified if and only if $f(x)$ is a non-zero constant function. If however we said $g(x)$ is $O(f(x))$ then the same principles can be applied and the original statement could only be satisified if and only if $g(x)$ is a non-zero constant funtcion instead. So $f(x) + g(x)$ is not $\Theta(f(x) \cdot g(x))$ for non-constant functions $f(x)$ and $g(x)$.

Question 5.

We can only use master theorm for recurrences in the form,

$$T(n) = aT(n/b) + f(n)$$

for $a \geq 1$ and $b > 1$.

We have the 3 cases:

1. If $f(n) = O(n^{\log_b(a)-\epsilon})$ for some $\epsilon > 0$ then $T(n) = \Theta(n^{\log_b(a)})$.
2. If $f(n) = \Theta(n^{\log_b(a)})$ then $T(n) = \Theta(n^{\log_b(a)} \log n)$.
3. If $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some $\epsilon > 0$ then and if $af(n/b) \leq cf(n)$ for some $c < 1$ and all n is large then $T(n) = \Theta(f(n))$

(a)

$T(n) = 9T(n/3) + n^2$, $a = 9$, $b = 3$, $f(n) = n^2$.

Trying case 2, $n^{\log_3(9)} = n^2$, $f(n)$ is $\Theta(n^2)$, so we have a case 2.

Therefore, $T(n)$ is $\Theta(n^2 \log n)$

(b)

$T(n) = 4T(n/2) + 100n$, $a = 4$, $b = 2$, $f(n) = 100n$.

Trying case 2, $n^{\log_2(4)} = n^2$, $f(n)$ is not $\Theta(n^2)$.

Trying case 1, $n^{\log_2(4)-\epsilon} = n^{2-\epsilon}$, for $\epsilon = 1$ we have n, $f(n)$ is $O(n)$, so we have a case 1.

Therefore, $T(n)$ is $\Theta(n^2)$

(c)

$T(n) = 2^n T(n/2) + n^3$, $a = 2^n$, $b = 2$, $f(n) = n^3$.

This recurrence can't be solved using master theorem because a is not constant.

(d)

$T(n) = 3T(n/3) + c \cdot n$, $a = 3$, $b = 3$, $f(n) = c \cdot n$.

Trying case 2, $n^{\log_3(3)} = n$, $f(n)$ is $\Theta(n)$, so we have a case 2.

Therefore $T(n) = \Theta(n \log n)$

(e)

$T(n) = 0.99T(n/7) + 1/(n^2)$, $a = 0.99$, $b = 7$, $f(n) = \frac{1}{n^2}$.

This recurrence can't be solved using master theorm because $a = 0.99$, so the condition $a \geq 1$ is not met.

Question 6.

(b)

The worst case input for an input size of 16 is: $1, 5, 9, 13, 3, 7, 11, 15, 2, 6, 10, 14, 4, 8, 12, 16$

Our *msort* recursively splices the list into 4 sublists,

$$1, 5, 9, 13|3, 7, 11, 15|2, 6, 10, 14|4, 8, 12, 16$$

Selection sort is not an adaptive sorting algorithm (every input of size n takes the same amount of time). So the sorted list 1,5,9,13 would take the same time as any other permutation of 1,5,9,13, using selection sort. Therefore it doesn't matter if our sublists are sorted or not we always take $O(n^2)$, or more specifically $n(n-1)/2$ comparisons.

Merging 2 lists of size n takes atleast n and at most $2n-1$ comparisons. It takes $2n-1$ comparisons if the left and right lists store alternating elements - an element is added to the new list from one list and then the next element is added from the other list and so on until one of the lists in empty. Clearly the 2 lists, 1,5,9,13 and 3,7,11,15 store alternating elements so $2n-1$ comparions will be needed to merge these 2 lists. Again, the 2 lists 2,6,10,14 and 4,8,12,16 store alternating elemenst so $2n-1$ comparisons will be needed to merge these 2 lists.

After we unwind one level of recursion we get the 2 sublists,

$$1, 3, 5, 7, 9, 11, 13, 15|2, 4, 6, 8, 10, 12, 1, 4, 16$$

These 2 sublists store alternating elements so $2n-1$ comparisons are needed again to merge these 2 lists. Because we always require the maximum amount of comparisons $(2n-1)$ when merging 2 lists together this is a worst case input for an input size of 16. We need 53 comparisons $= 4(4(4-1)/2) + 2(2(4)-1) + 2(8) - 1$