University of Waterloo
CS 341 Algorithms Fall 2013
Midterm

Friday, November 1, 2013, 4:30PM-6:00PM
No aids allowed.

Name	
Signature	
Student Number	
Section Number	

Question	*Mark*	*Question*	*Mark*
1a		*5a*	
1b		*5b*	
1c		*5c*	
1d		*6a*	
2a		*6b*	
2b		*6c*	
3		*7a*	
4a		*7b*	
4b		*7c*	
TOTALs			

1. (a) Prove that the relation "$\in O$" is transitive, i.e. $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ implies $f(n) \in O(h(n))$ [5 points].

Solution

Since $f(n) \in O(g(n))$, there exists positive constants c_f and n_f such that $0 \leq f(n) \leq c_f g(n)$ for all $n \geq n_f$.

Since $g(n) \in O(h(n))$, there exists positive constants c_g and n_g such that $0 \leq g(n) \leq c_g h(n)$ for all $n \geq n_h$.

Let $n_0 = \max(n_f, n_g)$ and $c = c_f c_g$. Since c is the product of two positive numbers, it is also positive. We have $0 \leq f(n) \leq c_f g(n) \leq c_f c_g h(n) = ch(n)$ for all $n \geq n_0$.

Hence $f(n) \in O(g(n))$.

Marking Scheme for Typical Errors

-1 for not mentioning that c_f is positive

-1 for not mentioning the condition for all $n > n_f$

-1 for using the sane values c_f, n_f for $f(n)$ and $g(n)$

-1 for not mentioning that $c = c_f c_g$

-1 for not mentioning that $n_0 = \max(n_f, n_g)$

(b) Show $\log_a n \in \Theta(\log_b n)$ for all $a, b > 1$ [5 points].

Solution

For logarithms, $\log_a n = \log_b n / \log_a b$ for all $n > 0$.

Setting c_1 and c_2 equal to $1/\log_b n$ which is positive we have $c_1 \log_b n \leq \log_a n \leq c_2 \log n$ for all $n > 0$ hence by definition of Θ we have $\log_a n \in \Theta(\log_b n)$.

Marking Scheme for Typical Errors

-2 for not using some sort of log formula

-2 for making a one sided argument (i.e. just bounded from above)

-2 for not saying explicitly what c_1 and c_2 are

-3 if c_1 and c_2 were not constants but instead were functions of n

-1 when using an argument based on the limit as $n \to \infty$ that $\log_a n$ and $\log_a n$ are positive for $n \geq 1$

(c) If $f(n) = 2^{n+1}$, then $f(n) \in O(2^n)$. Prove true or false [5 points].

Solution

$f(n) \in O(2^n)$ means there exists a C such that $f(n) \leq C2^n$ or
$2^{n+1} \leq C2^n$ By picking a constant $C \geq 2$ the inequality holds. Thus, the claim is True

Marking Scheme for Typical Errors -2 for saying it is of $\Theta(2^n)$ and not concluding $O(2^n)$ from it.
-2 for not explicitly specifying a C
-1 for not obtaining boundary of C.

Common Mistake:

You should show the existence of a C, which can be any constant number. Couple of students made wring cocnlusion by mentioning that it should work for every arbitrary constant.

(d) If $g(n) = 2^{2n}$, then $g(n) \in O(2^n)$. Prove true or false [5 points].

Solution

Assume the claim is true, thus there exists a constant C such that $2^{2n} \leq C2^n$
$2^n 2^n \leq C2^n$
$C \geq 2^n$ It is not possible to find such constant since it is dependent on variable n and hence is a contradiction and the claim is false.

Marking Scheme for Typical Errors

-4 for not proving the claim
-5 for wrong proofs

Common Mistake

Note $2^{2n} \neq 2^2 2^n$

2. (a) Prove by induction that $T(n) \in O(n)$ [5 points].

$$T(n) = \begin{cases} T\left(\frac{2n}{3}\right) + T\left(\frac{2n}{9}\right) + T\left(\frac{2n}{27}\right) + \Theta(1) & \text{if } n > 1 \\ 1 & \text{if } n = 1. \end{cases}$$

Solution:
Note, to prove this statement by induction, we need to show that $\exists c > 0$ such that $T(n) \leq cn$ for all $n > n_0$. Note that this c is fixed for all $n > n_0$ and not a function of n. For this proof, we will assume that the constant $\Theta(1) = 1$.
Base Case ($n = 1$): We have that $T(1) = 1$, so $T(1) \leq 1 \times n = 1$, so we have that $T(1) \in O(n)$.
Induction Hypothesis: $T(n) \leq n$, for all $1 \leq n \leq k$.
Induction Step: Consider $n = k + 1$. We have,

$$T(k+1) = T\left(\frac{2(k+1)}{3}\right) + T\left(\frac{2(k+1)}{9}\right) + T\left(\frac{2(k+1)}{27}\right) + \Theta(1)$$
$$T(k+1) \leq \frac{2(k+1)}{3} + \frac{2(k+1)}{9} + \frac{2(k+1)}{27} + 1$$
$$T(k+1) \leq \frac{26}{27}(k) + 1$$
$$T(k+1) \leq k + 1$$

as desired. Specifically, we have shown that for $n \in \mathbb{N}$, $T(n) \leq n$ so $T(n) \in O(n)$. We have proven this recurrence for the special case of our constant being 1, but we know that every $T(n)$ in this family only differs by a constant which does not impact the complexity. So indeed, we have shown the entire family of $T(n) \in O(n)$.

Common Mistakes
Many students directly substituted $O(n)$ for each $T(n)$ in the recurrence. That is, they said by induction that, $T(n) = O(n) + O(n) + O(n) + \Theta(1)$. Any easy way to see that this is incorrect is to try this same proof with showing $T(n) \in O(1)$. We would have $T(n) = O(1) + O(1) + O(1) + \Theta(1) \in O(1)$ by the same logic. Obviously this is not true. When doing induction, remember we are showing that our function $T(n)$ is asymptotically bounded by a fixed constant factor of n. When we substitue the order notation in, we are saying that the constants don't matter when in reality, these constants be growing as a function of n, or cascade up the tree. For the future, recall our induction is proving there is some cn such that $T(n) \leq cn$ for all $n > n_0$.

(b) Using whichever method you prefer, find the Θ bound for $T(n)$ [5 points].

$$T(n) = \begin{cases} T(n-1) + \lg n & \text{if } n > 2 \\ 1 & \text{if } n \leq 2. \end{cases}$$

Solution:
We will solve this recurrence directly. For $n > 2$, we have,

$$T(n) = T(n-1) + \lg n$$
$$T(n) = 1 + \sum_{i=1}^{n-2} \lg(n-i) + \lg n$$
$$T(n) = 1 + \sum_{i=1}^{n-1} \lg(i+1)$$
$$T(n) = 1 + \lg(\prod_{i=2}^{n} i)$$
$$T(n) = 1 + \lg(n!)$$

From the course notes (Stirling's approximation), we have that $T(n) \in \Theta(n \lg n)$.

3. Use the Master Theorem to obtain an $O()$ expression for $T(n)$ [5 points].

$$T(n) = \begin{cases} 4T\left(\frac{n}{2}\right) + n & \text{if } n \geq 2 \\ 1 & \text{if } n < 2. \end{cases}$$

Solution:

$a = 4, b = 2 \implies x = \log_2 4 = 2$

$y = 1$

$x > y \implies$ By Master Theorem, $T(n) = \Theta(n^2)$

i.e. $T(n) = O(n^2)$

Marking Scheme for Typical Errors

- Giving $\Theta(n^2)$ instead of $O(n^2)$ as final answer. -1 mark if everything else is correct.

- Deducting 2 or 3 marks if order is incorrect (depending upon the analysis).

- Awarded 1 mark if attempted but no significant progress.

- Deducted 2 marks if O() expression obtained correctly without use of Master Theorem but with correct analysis.

4. Analyze the following pieces of pseudocode and for each of them give a tight ($\Theta(n)$) bound on the running time [5 points each].

(a)
```
function f(n)
  r = 0
  for i = 1 to n - 1
    for j = i + 1 to n
      for k = 1 to j
        r = r + 1
  return(r)
```

Solution:

Method 1: (Finding formula for total running time and evaluating it)

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}\Theta(1) \\
&= \Theta(1).\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}1 \\
&= \Theta(1).\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} j \\
&= \Theta(1).\sum_{i=1}^{n-1}\Big(\sum_{j=0}^{n} j - \sum_{j=0}^{i} j\Big) \\
&= \Theta(1).\sum_{i=1}^{n-1}\Big(\frac{n(n+1)}{2} - \frac{i(i+1)}{2}\Big) \\
&= \Theta(1).\Big(\frac{(n-1)n(n+1)}{2} - \frac{(n-1)n(n+1)}{6}\Big) \\
&= \Theta(1).\Theta(n^3) = \Theta(n^3)
\end{aligned}$$

In the second last step above, we have used the formula $\sum_{i=1}^{n} i(i+1) = n(n+1)(n+2)/3$. Note that $i(i+1) = i(i+1)(i+2)/3 - (i-1)i(i+1)/3$, from which we get the formula using telescoping sums.

Method 2: (Showing upper and lower bounds separately)

The innermost loop runs j times, middle loop runs $(n-i)$ times and the outermost loop runs $n-1$ times. Also, $j \leq n$ and $i \geq 1$. Therefore, all three loops run at most n times. Hence, $T(n) = O(n^3)$.

Furthermore, for the lower bound we have:

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}\Theta(1) \\
&\geq \sum_{1}^{\lfloor\frac{n}{2}\rfloor}\sum_{j=\lfloor\frac{n}{2}\rfloor+1}^{n}\sum_{k=1}^{\lfloor\frac{n}{2}\rfloor}\Theta(1) \qquad \text{(summing positive terms over a smaller domain)} \\
&= \Omega(n^3)
\end{aligned}$$

Marking scheme:
For method 1, 1 mark for correct formula representing running time, 3 marks for correct evaluation of summation, 1 for correct final answer. Upto 1 mark taken away for each minor mistake and upto 2 marks taken away for a major mistake.

For method 2, 1 mark for correct answer and two marks each for correct proof of upper bound and lower bound.

(b)
```
function g(n)
  r = 0
  for i = 1 to n
    for j = 1 to i
      for k = j to i + j
        r = r + 1
return(r)
```

Solution:
Method 1: (Finding formula for total running time and evaluating it)

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}\Theta(1) \\
&= \Theta(1).\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}1 \\
&= \Theta(1).\sum_{i=1}^{n}\sum_{j=1}^{i}(i+1) \\
&= \Theta(1).\sum_{i=1}^{n} i(i+1) && \Big(\text{Summing over j, so i is a constant}\Big) \\
&= \Theta(1).\frac{n(n+1)(n+2)}{3} && \Big(\text{Using formula for } \textstyle\sum i(i+1)\Big) \\
&= \Theta(1).\Theta(n^3) = \Theta(n^3)
\end{aligned}$$

Method 2: (Showing upper and lower bounds separately)
The innermost loop runs $(i+1)$ times, the middle loop runs i times and the outermost loop runs n times (and $i \leq n$). The running time would only increase if we were to let the innermost loop and middle loop run $n+1$ and n times respectively. Therefore, $T(n) \leq (n+1)n^2\Theta(1) = O(n^3)$.

Also, note that

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j}^{i+j}\Theta(1) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=0}^{i}\Theta(1) \\
&\geq \sum_{i=\lfloor\frac{n}{2}\rfloor}^{n}\sum_{j=1}^{\lfloor\frac{n}{2}\rfloor}\sum_{k=0}^{\lfloor\frac{n}{2}\rfloor}\Theta(1) \qquad \text{(Summing positive terms over a smaller domain)} \\
&= \Omega(n^3)
\end{aligned}$$

Same marking scheme as in part (a).

5. Let A be an array of n integers containing the temperatures for Halloween, recorded every millisecond at the UWaterloo weather station. Ashish wrote a a straightforward program to find the maximum temperature during the day in $O(n)$ time. However Ashish's grad supervisor is unhappy since n is rather large (86.4 million data points per day). Luckily for him, his roommate Samantha is a CS major taking CS341. She knows that the temperature steadily increased starting from 4C at 0:00am, reaching a peak at some point during the day and then slowly decreasing to 13C just before midnight at 11:59pm, 24 hours later. She then used this property to give a divide-and-conquer solution that runs in $O(\log n)$ time.

   (a) Give pseudocode describing Samantha's solution. [5 points].

Algorithm 1 Samantha's Solution

```
1: function MAX(A, l, r)
2:     mid ← ⌊(l+r)/2⌋
3:     if mid == l then
4:         scan_for_max(A[l...r])
5:     else if A[mid] > A[mid − 1] then
6:         max(A, m, r)
7:     else
8:         max(A, l, m)
9:     end if
10: end function
```

Algorithm 2 Alternative Solution

```
1: function MAX(A, l, r)
2:     a ← ⌊(r−l)/3⌋ + l
3:     b ← ⌊2(r−l)/3⌋ + l
4:     if A[a] > A[b] then
5:         max(A, l, b)
6:     else
7:         max(A, a, r)
8:     end if
9: end function
```

(b) Write the recurrence for the pseudocode above. [5 points].

Solution:

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

or for alternative solution,

$T(n) = T(\frac{2n}{3}) + \Theta(1)$

(c) Show formally using whichever technique you would like that the solution to the recurrence is $O(\log n)$. [5 points].

Solution:

Using Master Theorem,

$x = \log_2 1 = 0,$
$y = 0$
$x = y \implies T(n) = \Theta(\log n)$ i.e. $T(n) = O(\log n)$

6. (a) Given a set of floating point numbers F and a positive number d create an efficient algorithm to find the largest subset $F' \subseteq F$ where the distance between any two members of F' is at least d. [5 points].

Solution

Sort the set of floats F in $\Theta(n \log n)$ time, where $n = |F|$.

Take the first number, say $x_0 \in F$, and place it in F'. Now find the next smallest number x_1 satisfying $x_1 \geq x_0 + d$.

Start at x_1 and repeat.

Note: Since greedy is the optimal approach, if you used another approach, like divide and conquer, you were deducted marks for this part and for part c) because you would have been proving something not true.

(b) Analyze and state the asymptotic worse-case running time of your algorithm, O in terms of n where n is the number of floats in F [5 points].

Sorting is $\Theta(n \log n)$. Then we go through F in $O(n)$ time. So in total we have $O(n \log n)$.

(c) Prove that your algorithm is optimal [5 points].

Solution

Let $g = \{g_0, \ldots, g_k\}$ be our greedy solution $O = \{o_0, \ldots, o_t\}$ be our optimal solution. Suppose their first point of difference is at some position α. Since greedy got g_α by picking the next smallest number d or more away, if optimal did not pick this we can replace o_α with g_α and the rest of optimal's solution will still be at least d apart from all others in its list. Therefore, we have constructed another optimal solution that differs now at position $\alpha + 1$. This is a contraction that they have first difference at position α.

Note: You can also use induction to argue the above replacement can be repeated so now everything in optimal is made the same as greedy.

7. (a) Draw a directed graph G with less than eight vertices and a particular starting vertex s such that G has a back edge, a forward edge and a cross edge assuming you perform a depth first search at s. Indicate these edges with the letters *b, f, c* respectively [5 points].

(b) If you perform a depth first search and a breadth first search on the same graph starting at the same vertex, will the edges be labelled the same? Give a brief argument if it is true or a counter example if it is false [5 points].

(c) Create a graph with n vertices and a particular starting vertex s such that $\Theta(n)$ nodes are simultaneously in the discovered or visited state (i.e. coloured grey) during a breadth-first search starting from s [5 points].