Dalhousie University
CSCI 3110 — Design and Analysis of Algorithms I
Fall 2016
Midterm Examination
October 26
8:37AM-9:52AM

Instructions (Read Carefully):

1. Aids allowed: one 8.5" by 11" piece of paper with anything written or printed on it (both sides). No textbooks, computers, calculators, or other aids.
2. This exam has 7 pages, including this page. Ensure that you have a complete paper.
3. Understanding the exam questions is part of the exam. Therefore, questions will **not** be interpreted. Proctors will confirm or deny any error or ambiguities only. If you are unsure of your own understanding, clearly state reasonable assumptions that will not trivialize the questions.

Problem	**Marks**	**Score**	**Marker**
1	20		
2	25		
3	25		
4	15		
5	15		
Total	100		

1. [20 marks] True-false: 5 marks each. No justification necessary.

   (a) If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n) + f_2(n) = O(g_1(n) + g_2(n))$.

   True

   (b) If $f(n) = O(g(n))$ then $f(n) \leq g(n)$ for all sufficiently large n.

   False

   (c) All the recurrences of the form T(n) = a T(n/b) + f(n), where $a \geq 1$ and $b > 1$ can be solved using the master theorem.

   False

   (d) $(\sqrt{n})! = o(n^{\sqrt{n/4}})$.

   True.

   By Stirling's approximation, $\frac{\sqrt{n}!}{n^{\sqrt{n/4}}} = \frac{\sqrt{2\pi\sqrt{n}}(\sqrt{n}/e)^{\sqrt{n}}(1+\Theta(1/\sqrt{n}))}{\sqrt{n}^{\sqrt{n}}} = \frac{\sqrt{2\pi\sqrt{n}}(1+\Theta(1/\sqrt{n}))}{e^{\sqrt{n}}}$

   You can then use the limit approach to show that $(\sqrt{n})! = o(n^{\sqrt{n/4}})$.

2. [25 marks] (Order of growth)

(a) [6 marks] Complete the formal definition of $\Omega(g(n))$ by filling in the blank below:

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that

$\underline{0 \le cg(n) \le f(n) \text{ for all } n \ge n_0}$

(b) [9 marks] Place the following functions in the three blanks below so that $f_1(n) = o(f_2(n))$ and $f_2(n) = o(f_3(n))$ (Recall that $\lg n = \log_2 n$):

$4^n \qquad n^{3/2} \qquad n \lg n$

$f_1(n) = \underline{n \lg n}$ $f_2(n) = \underline{n^{3/2}}$ $f_3(n) = \underline{4^n}$

(c) [10 marks] Prove that $f_1(n) = o(f_2(n))$ using the limit approach.

Hint: l'Hôpital's rule may be helpful.

$$\begin{aligned}
\lim_{n\to\infty} \frac{n \lg n}{n^{3/2}} &= \lim_{n\to\infty} \frac{\lg n}{n^{1/2}} \\
&= \lim_{n\to\infty} \frac{1/(n \ln 2)}{(1/2)n^{-1/2}} \quad \text{(l'Hôpital)} \\
&= \lim_{n\to\infty} \frac{2}{(\ln 2)n^{1/2}} \\
&= 0.
\end{aligned}$$

3. [25 marks] (Recurrence)

(a) [16 marks] For each of the following recurrences, use the "master theorem" and give the solution using big-Θ notation. Simply write down your answer and no justification is necessary.

If the "master theorem" does not apply to a recurrence, indicate this, but you need not show your reasoning or give a solution.

(a) $T(n) = 2T(n/2) + n$
$\Theta(n \lg n)$

(b) $T(n) = 4T(n/2) + \Theta(n)$
$\Theta(n^2)$

(c) $T(n) = 16T(n/4) + \Theta(n^2 \lg \lg n)$
Does not apply.

(d) $T(n) = 2T(\lceil n/3 \rceil) + n/\lg n$
$\Theta(n/\lg n)$

(b) [9 marks] In class we learned that the running time of Karatsuba's algorithm satisfies

$$T(n) = \begin{cases} 3T(n/2) + cn & \text{if } n > 1; \\ c & \text{if } n = 1. \end{cases} \tag{1}$$

I then wrote down

$$T(2^k) \leq c(3^{k+1} - 2^{k+1}), \tag{2}$$

and asked you to prove it for any integer $k \geq 0$ as an exercise after class.

Now prove, by induction, that equation (2) is indeed true for any integer $k \geq 0$.

Hint: Prove by induction on k. The base case is $k = 0$.

Use the space on the top half of the next page to write down your solution.

Your solution to Question 3(b) may be given here.

Proof: Clearly this holds for $k = 0$.
Assume that the claim holds for k, and we prove it for $k + 1$. We have

$$\begin{aligned} T(2^{k+1}) &= 3T(2^k) + c2^{k+1} \\ &\leq 3c(3^{k+1} - 2^{k+1}) + c2^{k+1} \\ &= c(3^{k+2} - 2^{k+2}) \end{aligned}$$

4. [15 marks] A company has n employees with salary (in dollars) stored in an integer array $S[1..n]$. Due to economic problems, the total salary budget for all employees suddenly becomes B dollars, where $B < \sum_{1 \leq i \leq n} S[i]$. To cope with this, management decides to cap all salaries at C dollars. That is, if $S[i] > C$, then $S[i]$ will be reduced to C. If $S[i] \leq C$, then $S[i]$ is unchanged.

   Note: For questions (a)-(c), to receive full marks, an $O(n \lg n)$-time algorithm is required. Any correct solution that requires more running time (even a brute-force solution) may still be awarded up to **half of the total marks** for these three sub-questions.

   For question (d), only a correct $O(n \lg n)$-time algorithm is worth any mark.

   (a) [6 marks] To make this problem easier, in questions (a)-(c), the value of C must be equal to an entry of S, i.e., one of the existing employees' salary.

   Describe, in English, an efficient algorithm to find the largest possible salary cap C, where C is equal to one of the existing employees' salary, so that the resulting capped salaries sum to $\leq B$.

   There's no need to give pseudocode, but you can if it makes your explanation clearer.

   Use the space on the top of the next page to write down your solution.

Your solution to Question 4(a) may be given here.

- Sort the salary array S in ascending order and store the sorted result in S'.
- For each i compute $T[i] = \sum_{j=1}^{i} S'[j] + (n-i)S'[i]$, which is the value of the sum of the capped salaries should we choose $S'[i]$ to be C. The array T can be computed using one loop in $O(n)$ time, by making use of the equation $T[i] = T[i-1] - (n-i+1)S'[i-1] + S'[i] + (n-i)S'[i]$.
- Use linear scan (or binary search) to find the index k such that $T'[k] \le B < T'[k+1]$ and return $S'[k]$ as the result.

(b) [3 marks] Justify the correctness of your algorithm briefly.

We claim that $T[i] = \sum_{j=1}^{i} S'[j] + (n-i)S'[i]$ is the value of the sum of the capped salaries when capped at $S'[i]$, because all salaries $S'[j] \le S'[i]$ if $j \le i$ (due to sorting) and all salaries $S'[j] \ge S'[i]$ if $j > i$ (again, due to sorting). Due to sorting, $T[i] \le T[j]$ if $i \le j$, so the linear scan will find the largest salary cap.

(c) [2 marks] Analyze the running time briefly and give the result using big-O notation, in terms of n.

Sorting requires $O(n \lg n)$ time and other steps require $O(n)$ time. Thus the total running time is $O(n \lg n)$.

(d) [4 marks] Describe, in English, an efficient algorithm to find the largest possible salary cap C (C is NOT necessarily equal to one of the existing employees' salary) so that the resulting capped salaries sum to $\le B$. There's no need to give pseudocode, but you can if it makes your explanation clearer.

Justification of correctness and running time analysis are not required.

Perform the steps for (a). After computing k, instead of returning it, solve the equation $\sum_{j=1}^{k} S'[j] + (n-k)C = B$ to compute the value of C.

5. [15 marks] You are given an array $A[1..n]$ of positive integers, where $n \geq 2$. Your goal is to find the largest ratio between two array elements where the numerator occurs after the denominator in the array. In other words, your goal is to compute

$$\max\{\frac{A[i]}{A[j]} : i, j \in \{1, 2, \ldots, n\} \text{ with } i > j\}$$

Note: To receive full marks, an $O(n \lg n)$-time algorithm is required. Any correct solution that requires more running time (even a brute-force solution) may still be awarded up to **half of the total marks** for this question.

(a) ([8 marks]) Describe, in English, an efficient algorithm that solves this problem. There's no need to give pseudocode, but you can if it makes your explanation clearer. (Hint: divide-and-conquer.)

We divide the array A into two halves, $A[1..\lfloor n/2 \rfloor]$ and $A[\lfloor n/2 \rfloor + 1..n]$. Then the answer is the maximum among the following three values:

- $\max\{\frac{A[i]}{A[j]} : i, j \in \{1, 2, \ldots, \lfloor n/2 \rfloor\} \text{ with } i > j\}$
- $\max\{\frac{A[i]}{A[j]} : i, j \in \{\lfloor n/2 \rfloor + 1, \ldots, n\} \text{ with } i > j\}$
- $\max\{A[\lfloor n/2 \rfloor + 1], \ldots, A[n]\} / \min\{A[1], \ldots, A[\lfloor n/2 \rfloor]\}$

We can compute the first two values above by recursing in each half of the array A. We can compute the third value by performing a linear scan in $A[\lfloor n/2 \rfloor + 1..n]$ to find the largest element in it, and the another scan in $A[1..\lfloor n/2 \rfloor]$ for the smallest element.

(b) ([4 marks]) Justify the correctness of your algorithm briefly.

Suppose that $A[i]/A[j]$ is the answer. Then either $A[i]$ and $A[j]$ are both in the first half of the array, or are both in the second half, or $A[i]$ is in the second half but $A[j]$ is in the first. The three values that we gave above cover the maximum ratio $A[i]/A[j]$ in each case.

(c) [3 marks] Analyze the running time briefly and give the result using big-O notation, in terms of n.

Since it requires $O(n)$ time to perform the linear scan for the largest element in $A[\lfloor n/2 \rfloor + 1..n]$ and the smallest element in $A[1..\lfloor n/2 \rfloor]$, the recurrence for the running time is $T(n) = 2T(n/2) + O(n)$. By the master's theorem, $T(n) = O(n \lg n)$.