University of Waterloo
School of Computer Science
CS 341 Algorithms, Winter, 2017
2 Hour Sample Midterm Exam
March 2nd, 2017

Instructors: Semih Salihoglu, Doug Stinson, Yaoliang Yu

Name:..

Student ID:...

Question	Marks	Total
1		10
2		15
3		15
4		18
5		22
5		20
Total		100
Verified		100

Instructions

- NO CALCULATORS OR OTHER AIDS ARE ALLOWED.
- You should have 10 pages in total.
- Make sure your name and student ID is recorded on the first page.
- Solutions will be marked for clarity, conciseness and correctness.
- If you need more space to complete an answer, you may continue on the two blank pages at the end.
- The backs of pages can be used for rough work, and will not be marked unless you specifically indicate you wish them considered.

Useful Facts and Formulas

1. Master Theorem (simplified version)
   Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

   $$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^y)$$

   in sloppy or exact form. Denote $x = \log_b a$. Then

   $$T(n) \in \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^x \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x. \end{cases}$$

2. Master Theorem (general version)
   Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

   $$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

   in sloppy or exact form. Denote $x = \log_b a$. Then

   $$T(n) \in \begin{cases} \Theta(n^x) & \text{if } f(n) \in O(n^{x-\epsilon}) \text{ for some } \epsilon > 0 \\ \Theta(n^x \log n) & \text{if } f(n) \in \Theta(n^x) \\ \Theta(f(n)) & \text{if } f(n)/n^{x+\epsilon} \text{ is an increasing function of } n \\ & \text{for some } \epsilon > 0. \end{cases}$$

3. $a^{\log_b c} = c^{\log_b a}$
4. $\sqrt{5} \approx 2.23$, $\quad \log_2 3 \approx 1.58$, $\quad \pi \approx 3.14$

1. *[10 marks total]* For each question below, give your answer together with a brief explanation. Show computations if it is appropriate to do so.

   (a) *[4 marks]* True or false: the closest pair problem *in one dimension* (i.e., for a set of points all on a line) can be solved in $O(n \log n)$ time without using the divide-and-conquer method from class. Briefly explain your answer.

   Answer: True. The points can be sorted with respect to their x-coordinates (or y-coordinates, if the line is vertical). This takes time $O(n \log n)$. Then scan through the points in order, determining the maximum distnace between consecutive points. This takes time $O(n)$. The total time of the algorithm is $O(n \log n)$.

   (b) *[3 marks]* Give a simplified Θ-bound for the expression $57n^{\sqrt{5}} + 39\sqrt{n}\, 3^{\log_2 n}$.

   Answer: $57n^{\sqrt{5}} \in \Theta(n^{\sqrt{5}}) \approx \Theta(n^{2.23})$.
   $39\sqrt{n}\, 3^{\log_2 n} = 39n^{0.5} n^{\log_2 3} \in \Theta(n^{0.5+\log_2 3}) \approx \Theta(n^{0.5+1.58}) = \Theta(n^{2.08})$.
   Because $2.08 < 2.23$, it follows that $57n^{\sqrt{5}} + 39\sqrt{n}\, 3^{\log_2 n} \in \Theta(n^{\sqrt{5}})$.

   (c) *[3 marks]* Which of the following two functions has the higher growth rate: $2^{\pi \log_2 n}$ or $n^3 (\log_2 n)^{20}$?

   Answer: $2^{\pi \log_2 n} = (2^{\log_2 n})^{\pi} = n^{\pi} \approx n^{3.14}$.
   $n^3 (\log_2 n)^{20} \in O(n^{3+\epsilon})$ for any $\epsilon > 0$.
   Therefore $2^{\pi \log_2 n}$ has the higher growth rate.

2. *[15 marks] Recurrences.*
Solve the following recurrence by using the recursion-tree method (you may assume that n is a power of 8):

$$T(n) = \begin{cases} 4T(n/8) + n^2 & \text{if } n > 1 \\ 2 & \text{if } n = 1. \end{cases}$$

Express the answer exactly as a sum, and then determine the growth rate of $T(n)$.

Answer:

Let $n = 8^j$. The costs of the nodes in the recursion tree are summarized as follows:

size of subproblem	# nodes	cost/node	total cost
$n = 8^j$	1	n^2	n^2
$n/8 = 8^{j-1}$	4	$(n/8)^2$	$4\,(n/8)^2 = n^2/16$
$n/8^2 = 8^{j-2}$	4^2	$(n/8^2)^2$	$4^2\,(n/8^2)^2 = n^2/16^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n/8^{j-1} = 8$	4^{j-1}	$(n/8^{j-1})^2$	$4^{j-1}\,(n/8^{j-1})^2 = n^2/16^{j-1}$
$n/8^j = 1$	4^j	2	2×4^j.

Summing the costs of all levels of the recursion tree, we have that

$$T(n) = 2 \times 4^j + n^2 \sum_{i=0}^{j-1} \left(\frac{1}{16}\right)^i .$$

$j = \log_8 n$, so $4^j = 4^{\log_8 n} = n^{\log_8 4} = n^{2/3}$.
The geometric sequence has ratio $1/16 < 1$, so $\sum_{i=0}^{j-1} (1/16)^i \in \Theta(1)$.
Therefore, $T(n) \in \Theta(n^{2/3} + n^2) = \Theta(n^2)$.

3. *[15 marks] Pseudocode analysis.*
Give a detailed analysis of the complexity of the procedure $f(n)$ in terms of the input parameter n. You can assume that n is a power of 2 in order to simplify the analysis.

```
     Procedure f(n)
1.   i ← 1
2.   S ← 0
3.   for j ← 1 to n do
4.       S ← S + j^3
5.   m ← n
6.   while m ≥ 1 do
7.       for j ← 1 to m do
8.           S ← S + (i − j)^2
9.       m ← ⌊m/2⌋
10.      i ← i + 1
11.  print(S)
```

Answer: Step 1 takes time $\Theta(1)$.
Step 2 takes time $\Theta(1)$.
Steps 3–4 take time $\Theta(n)$.
Step 5 takes time $\Theta(1)$.
Step 11 takes time $\Theta(1)$.

Steps 7–8 take time $\Theta(m)$ (for a given value of m).
Steps 9–10 take time $\Theta(1)$.
Steps 7–10 take time $\Theta(m)$ (for a given value of m).

In steps 6–10, m takes on the values $n, n/2, n/4, \ldots, 1$ (assuming n is a power of 2).
Therefore, steps 6–10 take time $\Theta(n + n/2 + n/4 + \cdots + 1) = \Theta(2n - 1) = \Theta(n)$.

Finally, steps 1–11 take time $\Theta(1 + 1 + n + 1 + n + 1) = \Theta(n)$.

4. *[22 marks total] Greedy algorithms.*

In the *Interval covering* problem, we are given a set X of n distinct real numbers $X = \{x_1, \ldots, x_n\}$, and an *interval length* L. We are required to find the minimum possible number of closed intervals, each of length L, such that every x_i is contained in at least one of the intervals. That is, we wish to find m intervals, say $I_1 = [a_1, a_1 + L], \ldots, I_m = [a_m, a_m + L]$, whose union contains all the x_i's, with m as small as possible.

In this question, we consider two possible greedy strategies for this problem.

> Strategy 1: Choose an interval that covers the maximum number of elements in X; remove all elements covered by this interval; and repeat.
>
> Strategy 2: Let x be the smallest element in X; choose the interval $[x, x + L]$; remove all elements covered by this interval; and repeat.

(a) *[10 marks]* By considering the problem instance $n = 6$, $X = \{2, 9, 12, 14, 17, 24\}$, and $L = 10$, prove that one of the two given strategies does not always find the optimal solution to the *Interval covering* problem.

Answer: We carry out the two strategies on the given problem instance.

Strategy 1: Choose the interval $[9, 19]$, which covers $9, 12, 14, 17$.
Then choose the interval $[2, 12]$, which covers 2.
Finally, choose the interval $[24, 34]$, which covers 24.
This strategy requires three intervals.

Strategy 2: Choose the interval $[2, 12]$, which covers $2, 9, 12$.
Then choose the interval $[14, 24]$, which covers $14, 17, 24$.
This strategy requires two intervals.

Since Strategy 2 uses fewer intervals than Strategy 1 on the given problem instance, we conclude that Strategy 1 is not always an optimal strategy.

(b) *[12 marks]* Give a complete proof that the other strategy always finds an optimal solution to the *Interval covering* problem.

Answer: We prove that the second strategy always yields an optimal solution. Assume that the elements in X are $x_1, \ldots, x_n$ in increasing order. Let $\mathcal{O}$ be any optimal solution for the set X. There must be an interval $[a, a+L] \in \mathcal{O}$ that covers x_1; hence, $a \leq x_1$.

Suppose that $a < x_1$. Consider the modified solution $\mathcal{O}' = \mathcal{O} \cup \{[x_1, x_1 + L]\} \backslash \{[a, a+L]\}$ (i.e., replace the interval $[a, a+L]$ by $[x_1, x_1 + L]$). It is easy to see that $\mathcal{O}'$ is also an optimal solution. Therefore we have shown that the interval $[x_1, x_1 + L]$ is contained in an optimal solution.

Let X' be the elements of X that are not covered by $[x_1, x_1 + L]$. By what we have shown above, it follows that an optimal solution for X consists of the interval $[x_1, x_1 + L]$ together with an optimal solution for the set X'.

By similar reasoning, an optimal solution for X' contains the interval $[x_i, x_i + L]$, where x_i is the smallest element in X'. Repeating this argument until no elements of the set X remain, we see that Strategy 2 is optimal.

Remark: This proof can be expressed more formally as a proof by induction on $|X|$, but the informal argument we have given is sufficient to receive full marks.

5. *[22 marks] Divide-and-conquer.*
Define the following sequence of numbers: $F_0 = 0$, $F_1 = 1$, and

$$\begin{aligned} F_{2n} &= (F_n + F_{n-1})^2 - {F_{n-1}}^2 \\ F_{2n+1} &= (F_n + F_{n-1})^2 + {F_n}^2 \end{aligned}$$

(This is in fact the Fibonacci number sequence.)

(a) *[10 marks]* Give a pseudocode description of an efficient recursive algorithm to compute F_n for a given integer $n \geq 0$, based on the above definition.
Answer:

Algorithm 0.1: F(n)

local g, n_1, n_2, g_1, g_2
if $n = 0$
  then $g \leftarrow 0$
  else if $n = 1$
  then $g \leftarrow 1$
  else $\begin{cases} n_1 \leftarrow \left\lfloor \frac{n}{2} \right\rfloor \\ n_2 \leftarrow n_1 - 1 \\ g_1 \leftarrow \mathsf{F}(n_1) \\ g_2 \leftarrow \mathsf{F}(n_2) \\ \textbf{if } (n \bmod 2) = 0 \\ \quad \textbf{then } g \leftarrow (g_1 + g_2)^2 - {g_2}^2 \\ \quad \textbf{else } g \leftarrow (g_1 + g_2)^2 + {g_1}^2 \end{cases}$
return (g)

(b) *[12 marks]* Determine (using O notation) the running time of your algorithm by writing a recurrence and solving it with the master method. Here, runnning time is measured in terms of the number of bit operations. Assume that the multiplication of two k-bit numbers requires $O(k^{1.59})$ time by Karatsuba and Ofman's algorithm. You may use the fact that $F_n \leq 2^n$, so the number of bits in F_n is at most n (but mention where this is used in your analysis).

Answer:

Let $T(n)$ denote the amount of time required to compute $F(n)$ using the algorithm above.

The operations performed on n (chopping off the last bit, subtracting 1 and extracting the last bit) take time $O(1)$, $O(\log n)$ and $O(1)$ respectively, because n is an integer having $O(\log n)$ bits.

When we compute g after the two recursive calls, we are performing arithmetic operations (additions, subtractions and multiplications) on the integers g_1 and g_2. Because

$$F_{\lfloor \frac{n}{2} \rfloor - 1} < F_{\lfloor \frac{n}{2} \rfloor} \leq 2^{\lfloor \frac{n}{2} \rfloor} < 2^{n/2},$$

these integers consist of at most $n/2$ bits. The time to compute g, given g_1 and g_2, is therefore $O(n/2 + (n/2)^{1.59}) = O(n^{1.59})$, because additions and subtractions take time $O(n/2)$ and multiplications take time $O((n/2)^{1.59})$.

Therefore, the recurrence relation for $T(n)$ is as follows:

$$T(n) = \begin{cases} O(1) & \text{if } n \in \{0, 1\} \\ T\left(\lfloor \frac{n}{2} \rfloor\right) + T\left(\lfloor \frac{n}{2} \rfloor - 1\right) + O(n^{1.59}) & \text{if } n \geq 2. \end{cases}$$

Ignoring floors, and noting that $T(n)$ is an increasing function, we obtain the following simplified recurrence:

$$T(n) \leq \begin{cases} O(1) & \text{if } n \in \{0, 1\} \\ 2\,T\left(\frac{n}{2}\right) + O(n^{1.59}) & \text{if } n \geq 2. \end{cases}$$

We can determine a O-bound on the growth rate of $T(n)$ using either version of the Master Theorem. Using the simplified version, we have $a = 2$, $b = 2$, $x = \log_2 2 = 1$ and $y = 1.59$. We are in case 3 and therefore $T(n) \in O(n^{1.59})$.