CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani

HW 1

Due January 25, 11:59pm

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc).

For questions asking you to give an algorithm, you must respond in what we will refer to as the *four-part format* for algorithms: high-level description, pseudocode, running time analysis, and proof of correctness.

Read the **Homework FAQ** post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

You risk receiving no credit for any homework that does not adhere to these guidelines.

No late homeworks will be accepted. **No exceptions.** Do not ask for extensions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 13 homework assignments, the lowest two scores will be dropped.

This homework is due Monday, January 25, at 11:59pm via Gradescope. Please submit via PDF.

1. (20 pts.) Compare Growth Rates

In each of the following, indicate whether $f = O(g)$, $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). Briefly justify each of your answers.

	$f(n)$	$g(n)$
(a)	$n^{1/2}$	$n^{2/3}$
(b)	$\log_3 n$	$\log_4 n$
(c)	$2^n + n^5$	$2^n + \log n$
(d)	$n \log(n^4)$	$n^2 \log(n^3)$
(e)	$n^{1.01}$	$n \log^2 n$
(f)	$n \log n$	$(\log n)^{\log n}$
(g)	$\sqrt{n}$	$(\log n)^3$
(h)	2^n	2^{n+1}
(i)	4^n	$n!$

Solution:

(a) $f = O(g)$; g has the higher exponent.

(b) $f = \Theta(g)$; using the log change of base formula, $\frac{\log n}{\log 3}$ and $\frac{\log n}{\log 4}$ differ only by a constant factor.

(c) $f = \Theta(g)$; the identical exponential term dominates.

(d) $f = O(g)$; $f(n) = 4n \log(n)$ and $g(n) = 3n^2 \log(n)$, and the polynomial in g has the higher degree.

(e) $f = \Omega(g)$; the polynomial in f has the higher degree, dominating the log factors.

(f) $f = O(g)$; $g(n) = 2^{\log((\log n)^{\log n})} = 2^{(\log n) \log \log n} = n^{\log \log n}$, which dominates any polynomial.

(g) $f = \Omega((\log n)^3)$; any polynomial dominates a product of logs.

(h) $f = \Theta(g)$; $g(n) = 2f(n)$ so they are constant factors of each other.

(i) $f = O(g)$; $\frac{f(n)}{g(n)} = \prod_{i=1}^{n} \frac{4}{i} < \frac{4^4}{4!}(\frac{4}{5})^{n-4}$ which approaches zero as n grows, so g grows faster than f.

2. (15 pts.) Geometric Growth

Prove that the following formula holds, given a positive real number c.

$$\sum_{i=0}^{k} c^i = \begin{cases} \Theta(c^k) & \text{if } c > 1 \\ \Theta(k) & \text{if } c = 1 \\ \Theta(1) & \text{if } c < 1 \end{cases}$$

The moral: in asymptotics, the sum of a geometric series is simply the first term if the series is strictly decreasing, the last term if the series is strictly increasing, or the number of terms if the series is unchanging. This idea underlies the Master theorem for solving recurrences.

Solution:

Note that c is considered a constant: we're finding asymptotic growth as a function of k. By the formula for the sum of a partial geometric series, for $c \neq 1$: $S(k) := \sum_{i=0}^{k} c^i = \frac{1-c^{k+1}}{1-c}$. Thus,

- If $c > 1$, for sufficiently large k, $c^{k+1} > c^{k+1} - 1 > c^k$. Dividing this inequality by $c-1$ yields: $\frac{c}{c-1}c^k > S(k) > \frac{1}{c-1}c^k$. Thus, $S(k) = \Theta(c^k)$, since it is bounded above and below by constant factors, $\frac{c}{c-1}$ and $\frac{1}{c-1}$ respectively.
- If $c = 1$, then every term in the sum is 1. Thus, $S(k) = k+1 = \Theta(k)$.
- If $c < 1$, then $c^{k+1} < 1$, so $S(k) = \frac{1-c^{k+1}}{1-c} < \frac{1}{1-c}$. Since $S(k)$ is upper-bounded by a constant, $S(k) = O(1)$. Since $S(k) > 1$, it's also in $\Omega(1)$, so $S(k) = \Theta(1)$.

3. (25 pts.) Recurrence Relations

Solve the following recurrence relations and give a Θ bound for each of them.

(a) (i) $T(n) = 3T(n/4) + 4n$

(ii) $T(n) = 45T(n/3) + .1n^3$

(iii) $T(n) = T(n-1) + c^n$, where c is a constant.

(iv) $T(n) = 2T(\sqrt{n}) + 3$, and $T(2) = 3$. (Hint: this means the recursion tree stops when the problem size is 2)

(b) (i) Consider the recurrence relation $T(n) = 2T(n/2) + n\log n$. We can't plug it directly into the Master theorem, so solve it by adding the size of each layer.
Hint: split up the $\log(n/(2^i))$ *terms into* $\log n - \log(2^i)$*, and use the formula for arithmetic series.*

(ii) A more general version of Master theorem, like the one on Wikipedia, incorporates this result. The case of the master theorem which applies to this problem is:
If $T(n) = aT(n/b) + f(n)$ *where* $a \geq 1$*,* $b > 1$*, and* $f(n) = \Theta(n^c \log^k n)$ *where* $c = \log_b a$*, then* $T(n) = \Theta(n^c \log^{k+1} n)$.
Use the general Master theorem to solve the following recurrence relation:
$T(n) = 9T(n/3) + n^2 \log^3 n$.

Solution:

(a) (i) Since $\log_4 3 < 1$, by the Master Theorem, $T(n) = \Theta(n)$.

(ii) Since $\log_3 45 > 3$, by the Master Theorem, $T(n) = \Theta(n^{\log_3 45})$.

(iii) Expanding out the recurrence, we have $T(n) = \sum_{i=0}^{n} c^i$. From the previous problem, we know that this is $\Theta(1)$, $\Theta(n)$, or $\Theta(c^n)$, depending on if $c < 1$, $c = 1$, or $c > 1$.

(iv) Let $k = \log n$. Let $S(k)$ be the running time as a function of k, meaning $S(k) = T(n)$. Then, we can write the recurrence relation

$$\begin{aligned} S(k) &= T(n) \\ &= T(2^k) \\ &= 2T(\sqrt{2^k}) + 3 \\ &= 2T(2^{k/2}) + 3 \\ &= 2S(k/2) + 3 \end{aligned}$$

Using the Master Theorem, this solves to

$$S(k) = O(k)$$

Then, we use this result to get

$$\begin{aligned} T(n) &= S(k) \\ &= O(k) \\ &= O(\log n) \end{aligned}$$

(b) (i) The amount of work done at the ith layer is $2^i(n/2^i \log(n/2^i))$. Since there are a total of $\log_2 n$

layers, we can find the total work done as follows:

$$\begin{aligned}
\sum_{i=1}^{\log n} 2^i (n/2^i \log(n/2^i)) &= \sum_{i=1}^{\log n} n \log(n/2^i) \\
&= n \sum_{i=1}^{\log n} \log n - \log 2^i = n \log^2 n - n \sum_{i=1}^{\log n} i \log 2 \\
&= n \log n - n \log(2) \times \frac{(\log n)(\log n + 1)}{2} \\
&= n \log^2 n \left(1 - \frac{\log 2}{2}\right) - n \log n \times \frac{\log 2}{2} \\
&= \Theta(n \log^2 n)
\end{aligned}$$

where step (4) uses the formula for arithmetic series, and step (6) discards the constant factors, and then discards the second term because its log factor, $(\log n)$, is dominated by $(\log^2 n)$ in the first term.

(ii) Since $\log_3 9 = 2$ (the exponent on n), by Case 2 of Wikipedia's general Master theorem, $T(n) = \Theta(n^2 \log^4 n)$.

4. (25 pts.) Integer Multiplication In class, we covered the integer multiplication technique used by Al Khwarizmi and some European countries. With this method, we multiply and divide numbers by 2 at each iteration. However, there's nothing special about the number 2 here – we could perform this technique by multiplying and dividing another number instead. In particular, let's modify this technique to multiply and divide the numbers by 3 at each iteration. Following the format described in the book, we can create two columns, one for each number being multiplied. For each row, we divide the number in the left column by 3, ignoring the remainder, and multiply the number in the right column by 3. To get the final result, we need to add rows of the right column up. Which rows should be added together to get the final result?

Hint: you may have to multiply some rows by a constant.

(a) Show how to multiply the numbers 11 and 13 using your modified multiplication scheme.
Solution:

11	13 $(\times 2)$
3	39 $(\times 0)$
1	117 $(\times 1)$
	143

(b) Formally describe and prove the correctness of your modified algorithm. Use the four-part format for algorithms (High-level description, pseudocode, running time analysis, and proof of correctness). Your proof of correctness may be brief.
Solution:
Main Idea: To modify the original algorithm, we simply multiply and divide by 3 at each step. When adding up the right-hand column at the end, we multiply each number in the right column by the remainder of the left-hand column when divided by 3. This extends the idea of taking the binary representation in the original algorithm to using a ternary representation, since the possible remainders when dividing by 3 are 0, 1, and 2.
Pseudocode:

```
procedure MULTIPLY((x, y))
    if y == 0 then
        return 0
    z = multiply(x, ⌊y/3⌋)
    if y%3 == 0 then
        return 3z
    else if y%3 == 1 then
        return x + 3z
    else
        return 2x + 3z
```

Proof of correctness: This algorithm directly implements the rule

$$x \cdot y = \begin{cases} 3(x \cdot \lfloor y/3 \rfloor) & \text{if } y\%3 == 0 \\ x + 3(x \cdot \lfloor y/3 \rfloor) & \text{if } y\%3 == 1 \\ 2x + 3(x \cdot \lfloor y/3 \rfloor) & \text{if } y\%3 == 2 \end{cases}$$

which can be easily verified as mathematically correct. Furthermore, it correctly implements the base case for $y = 0$. Thus, the algorithm is correct

Running time analysis: Let t be the number of ternary digits required to represent x and y. Each iteration divides y by 3, decreasing the number of ternary bits required to represent it by 1. Furthermore, each iteration performs at most a multiplication by 1 or 2 and an addition, each of which takes $O(t)$ time. Thus, the running time of this algorithm is $O(t^2)$. Furthermore, since the number of ternary digits required to represent a number differs by a constant factor from n, the number of bits required to represent it, the running time is $O(n^2)$.

5. (20 pts.) More Integer Multiplication The following algebraic identities can be used to design a divide-and-conquer algorithm for multiplying n-bit numbers. Assume we are multiplying two numbers a and b. Let a_2, a_1, and a_0 be the top $n/3$, middle $n/3$, and bottom $n/3$ bits of a respectively. Define b_2, b_1, and b_0 in the same way. We can write the product $a \cdot b$ as

$$\begin{aligned} a \cdot b &= \left(2^{2n/3}a_2 + 2^{n/3}a_1 + a_0\right)\left(2^{2n/3}b_2 + 2^{n/3}b_1 + b_0\right) \\ &= 2^{4n/3} \cdot a_2b_2 + 2^n \cdot (a_1b_2 + a_2b_1) + 2^{2n/3} \cdot ((a_2b_0 + a_0b_2) + a_1b_1) + 2^{n/3} \cdot (a_1b_0 + a_0b_1) + a_0b_0 \end{aligned}$$

Furthermore, we know the terms in the above expression can be written as follows:

$$\begin{aligned} (a_1b_2 + a_2b_1) &= (a_1 + a_2) \cdot (b_1 + b_2) - a_1b_1 - a_2b_2 \\ (a_0b_2 + a_2b_0) &= (a_0 + a_2) \cdot (b_0 + b_2) - a_0b_0 - a_2b_2 \\ (a_0b_1 + a_1b_0) &= (a_0 + a_1) \cdot (b_0 + b_1) - a_0b_0 - a_1b_1 \end{aligned}$$

Using this, we can derive a recursive multiplication algorithm.

(a) Rewrite the product $a \cdot b$ in terms of as few distinct recursive multiplications as possible (i.e. one such recursive multiplication might be the product $a_0 \cdot b_0$.

Solution:

$$\begin{aligned} a \cdot b = {} & 2^{4n/3} \cdot a_2b_2 \\ & + 2^n \cdot ((a_1 + a_2) \cdot (b_1 + b_2) - a_1b_1 - a_2b_2) \\ & + 2^{2n/3} \cdot ((a_0 + a_2) \cdot (b_0 + b_2) - a_0b_0 - a_2b_2 + a_1b_1) \\ & + 2^{n/3} \cdot ((a_0 + a_1) \cdot (b_0 + b_1) - a_0b_0 - a_1b_1) \\ & + a_0b_0 \end{aligned}$$

Note that this uses 6 distinct multiplications: $(a_1 + a_2) \cdot (b_1 + b_2), (a_0 + a_2) \cdot (b_0 + b_2), (a_0 + a_1) \cdot (b_0 + b_1), a_0b_0, a_1b_1, a_2b_2$.

(b) Write the recurrence relation for running time $T(n)$ if we perform multiplications in this way.

Solution:

There are 6 distinct multiplications, each of which operates on numbers with $n/3$ bits. Furthermore, the amount of work done to combine these results is $O(n)$ because there are a constant number of additions and bit-shifts being done, each of which takes $O(n)$

$$T(n) = 6T(n/3) + O(n)$$

(c) What is the running time of the algorithm?

Solution:

Using the Master Theorem, we have

$$T(n) \in O\left(n^{\log_3(6)}\right)$$