CMPSC 465 Data Structures & Algorithms
Spring 2021 Paul Medvedev and Chunhao Wang

HW 2

Due February 4, 10:00 pm

Instructions: You may work in groups of up to three people to solve the homework. You must write your own solutions and explicitly acknowledge up everyone whom you have worked with or who has given you any significant ideas about the HW solutions. You may also use books or online resources to help solve homework problems. All consulted references must be acknowledged.

You are encouraged to solve the problem sets on your own using only the textbook and lecture notes as a reference. This will give you the best chance of doing well on the exams. Relying too much on the help of group members or on online resources will hinder your performance on the exams.

Late HWs will be accepted until 11:59pm with a 20% penalty. HWs not submitted by 11:59pm will receive 0. There will be no exceptions to this policy, as we post the solutions soon after the deadline. However, you will be able to drop the three lowest HW grades.

For the full policy on HW assignments, please consult the syllabus.

1. **(0 pts.) Acknowledgements.** The assignment will receive a 0 if this question is not answered.

   (a) If you worked in a group, list the members of the group. Otherwise, write "I did not work in a group."

   (b) If you received significant ideas about the HW solutions from anyone not in your group, list their names here. Otherwise, write "I did not consult without anyone my group members".

   (c) List any resources besides the course material that you consulted in order to solve the material. If you did not consult anything, write "I did not consult any non-class materials."

2. **(30 pts.) Compare Growth Rates.** In each of the following situations, indicate whether $f = O(g)$, or $f = \Omega(g)$, or both (in which case $f = \Theta(g)$). Give a one sentence justification for each of your answers.

	$f(n)$	$g(n)$
(a)	$n^{1.5}$	$n^{1.3}$
(b)	2^{n-1}	2^n
(c)	$n^{1.3\log n}$	$n^{1.5}$
(d)	3^n	$n2^n$
(e)	$(\log n)^{100}$	$n^{0.1}$
(f)	n	$(\log n)^{\log\log n}$
(g)	2^n	$n!$
(h)	$\log(e^n)$	$n\log n$
(i)	$n+\log n$	$n+(\log n)^2$
(j)	$5n+\sqrt{n}$	$\log n+n$

Solution: There are many different ways to justify the solutions, and we provide just one.

(a) $f = \Omega(g)$; both are polynomials and $1.5 > 1.3$.

(b) $f = \Theta(g)$; $2^{n-1} = \frac{1}{2}2^n$, so f and g are within a factor 2 of each other.

(c) $f = \Omega(g)$; $n^{1.3\log n} > n^{\log n}$, and $n^{\log n}$ grows faster than any polynomial in n (e.g. for $n \geq 4$, $\log n \geq 2 > 1.5$).

(d) $f = \Omega(g)$; Because polynomials grow slower than exponentials, $n = O(1.1^n)$ and then by multiplication property, $n2^n = O(1.1^n 2^n) = O(2^{n\log 1.1}2^n) = O(2^{n(1+\log 1.1)}) = O((2^{1+\log 1.1})^n)$. Because exponentials grow according to their base, and $2^{1+\log 1.1} < 3$ the last term is $O(3^n)$. You could also argue using limits: $\lim_{n\to\infty} \frac{3^n}{n2^n} = \lim_{n\to\infty} \frac{\left(\frac{3}{2}\right)^n}{n} = \infty$, so f grows much faster than g.

(e) $f = O(g)$; $\log(n)$ to any constant power grows much more slowly than n^c for any constant $c > 0$.

(f) $f = \Omega(g)$; $n = 2^{\log n}$ and $(\log n)^{\log\log n} = 2^{(\log\log n)^2}$, so f grows faster than g since $\log n$ grows faster than $(\log\log n)^2$.

(g) $f = O(g)$; Set $c = 2$ and $n_0 = 1$ and observe that $\frac{2^n}{n!} = \prod_{i=1}^{n} \frac{2}{i} \leq 2$

(h) $f = O(g)$; $\log(e^n) = n\log(e) = O(n)$, so $f(n)$ grows more slowly than $g(n) = n\log n$ by multiplicative property.

(i) $f = \Theta(g)$; In both f and g the linear term n dominates the other term, so applying the additivity property, both f and g grow as $\Theta(n)$.

(j) $f = \Theta(g)$; The term $5n$ dominates $\sqrt{n}$ in f, and n dominates $\log n$ in g, so again applying additivity property, both grow as $\Theta(n)$.

3. (20 pts.) Polynomials and Horner's rule (HW) Suppose we want to evaluate the polynomial $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$ at some point x_0.

(a) Consider the brute force algorithm that computes first a_1x_0, $a_2x_0^2$, $a_3x_0^3, \ldots, a_nx_0^n$ (independently) and then adds all of these numbers to a_0 to obtain $p(x_0)$. How many sums and how many multiplications are involved? (You can give your answer in O-notation.)

(b) We show next how to do better using *Horner's rule*. Prove that the following algorithm outputs $p(x_0)$. Use the loop invariant proof structure we saw in class.

$z = a_n$;
for $i = n-1$ **to** 0 **do**
| $z = z \cdot x_0 + a_i$;
end
return z;

(c) How many additions and multiplications does this algorithm use as a function of n? (You can give your answer in O-notation.)

Solution

(a) $O(n^2)$ multiplications and $O(n)$ additions.

(b) Lets use z_k to denote the value of z at the start of the k^{th} iteration. Our loop invariant (LI) is that right prior to the start of the k^{th} iteration, $z_k = \sum_{j=0}^{k-1} a_{n-j}x_0^{k-j-1}$.

First, we prove the initialization, i.e. that the LI holds at the start of the loop, when $k = 1$. Then, the LI gives us that $z_1 = a_nx_0^{k-1} = a_n$, which is exactly the value of z according to the pseudocode.

Second, for the induction step, lets assume that the LI holds at the start of iteration $k-1$ and lets show that it holds at the start of iteration k. At the start of iteration k, the value of the loop counter is still

$i = n-k+1$. We can then restate the assignment of z in the pseudocode as: $z_k = z_{k-1}x_0 + a_{n-k+1}$ Plugging in the LI for $k-1$, we have

$$z_k = z_{k-1}x_0 + a_{n-k+1} = x_0 \sum_{j=0}^{k-2} a_{n-j}x_0^{k-j-2} + a_{n-k+1} = \sum_{j=0}^{k-2} a_{n-j}x_0^{k-j-1} + a_{n-k+1}$$
$$= \sum_{j=0}^{k-2} a_{n-j}x_0^{k-j-1} + a_{n-k+1}x_0^0 = \sum_{j=0}^{k-1} a_{n-j}x_0^{k-j-1}$$

Finally, for the termination step, we need to show that if the LI holds at the end of the loop execution, then the algorithm is correct. We can treat the end of the execution is being right prior to the $n+1$ iteration, i.e. $k = n+1$. Plugging in the LI, we get $z_{n+1} = \sum_{j=0}^{n} a_{n-j}x_0^{n-j} = \sum_{j=0}^{n} a_j x_0^j = p(x)$.
This completes the proof. But if you were wondering how one comes up with the loop invariant, its always good to try a few examples. For example,

$k=1$	$\rightarrow$	i not set	$\rightarrow$	$z_1 = a_n$
$k=2$	$\rightarrow$	$i = n-1$	$\rightarrow$	$z_2 = a_n x_0 + a_{n-1}$
$k=3$	$\rightarrow$	$i = n-2$	$\rightarrow$	$z_3 = a_n x_0^2 + a_{n-1}x_0 + a_{n-2}$
$k=4$	$\rightarrow$	$i = n-3$	$\rightarrow$	$z_4 = a_n x_0^3 + a_{n-1}x_0^2 + a_{n-2}x_0 + a_{n-3}$

Seeing this will let you build an intuition for what's happening and generalize to a loop invariant.

(c) Every iteration of the for loop uses one multiplication and one addition, so the routine uses n additions and n multiplications.

4. (30pt pts.) Solving recurrences Solve the following recurrence relations and give a Θ bound for each of them. Do not use the Master Theorem. You must use the recursion tree method. For each recurrence, make sure you state the branching factor, the height of the tree, the size of the subproblems at depth k, and the number of subproblems at depth k. It is not necessary to draw the tree, though you may do so anyway to help yourself visualize the problem.

(a) $T(n) = 2T(n/2) + \sqrt{n}$
(b) $T(n) = 2T(n/3) + 1$
(c) $T(n) = 5T(n/4) + n$
(d) $T(n) = 7T(n/7) + n$
(e) $T(n) = 9T(n/3) + n^2$

Solution

	branching factor	height	size	number	W_k	Tot. work
a)	2	$\log_2 n$	$n/2^k$	2^k	$\sqrt{2}^k\sqrt{n}$	$\sum_{k=0}^{\log_2 n}(\sqrt{2})^k\sqrt{n} = \Theta(n)$
b)	2	$\log_3 n$	$n/3^k$	2^k	2^k	$\sum_{k=0}^{\log_3 n} 2^k = \Theta(n^{\log_3 2}) \approx \Theta(n^{0.63})$
c)	5	$\log_4 n$	$n/4^k$	5^k	$(5/4)^k n$	$\sum_{k=0}^{\log_4 n}(5/4)^k n = \Theta(n^{1+\log_4(5/4)}) = \Theta(n^{\log_4 5}) \approx \Theta(n^{1.16})$
d)	7	$\log_7 n$	$n/7^k$	7^k	n	$\sum_{k=0}^{\log_7 n} n = \Theta(n\log n)$
e)	9	$\log_3 n$	$n/3^k$	9^k	n^2	$\sum_{k=0}^{\log_3 n} n^2 = \Theta(n^2\log n)$