Homework 01 — Solution

CS 624, 2022 Fall

Read the updated course homework policies before you start!

1. Consider the following algorithm for calculating the *cumulative sums* of an array. The input is an array of numbers, `A`. The output is a new array of number of the same length, `R`. (Array indexes start at 1.)

```
CumulativeSums(A) :=

R ← new array with length[A] elements
if length[A] > 0
  R[1] ← A[1]
end if
for j ← 2 to length[A]
  R[j] ← R[j-1] + A[j]
end for
return R
```

The correctness property for this algorithm is the following:

$$\texttt{R}[n] = \sum_{i=1}^{n} \texttt{A}[i] \qquad \text{for all } 1 \leq n \leq \texttt{length[A]}$$

(a) Prove that this algorithm terminates.

The only loop is a `for` loop, which terminates because it just iterates from 2 to `length[A]`. All of the other statements are simple statements, so they terminate.

(b) State the loop invariant for the `for` loop.

The loop invariant for iteration j is the following:
For each index $n \in \{1..j-1\}$,

$$\texttt{R}[n] = \sum_{i=1}^{n} \texttt{A}[i]$$

(c) Prove the correctness of the algorithm using the loop invariant.

Initialization: We must show that the loop invariant holds at the beginning of the first iteration ($j = 2$). The range for n in the loop invariant is $\{1..2-1\} = \{1\}$, and $\texttt{R[1]} = \sum_{i=1}^{1} \texttt{A}[i] = \texttt{A[1]}$ because of the assignment on the third line of the function body.

Maintenance: Assuming the loop invariant holds for j at the beginning of an iteration, we must show the invariant holds for $j+1$ at the end of the iteration. Based on the loop body's assignment:

$$\begin{aligned}
\texttt{R[}j\texttt{]} &= \texttt{R[}j-1\texttt{]} + \texttt{A[}j\texttt{]} \\
&= \sum_{i=1}^{j-1} \texttt{A[}i\texttt{]} + \texttt{A[}j\texttt{]} && \text{(by loop inv)} \\
&= \sum_{i=1}^{j} \texttt{A[}i\texttt{]} && \text{(absorb term into summation)}
\end{aligned}$$

No other array slots are assigned, so the loop invariant equation still holds for $n \in \{1..j-1\}$, and the assignment extends it to $\{1..j\}$.

Termination: When the loop exits, the loop invariant holds for $j = \texttt{length[A]} + 1$, which simplifies to the desired correctness property.

(d) What is the running time of this algorithm? Justify your answer.

The running time is $\Theta(n)$ where $n = \texttt{length[A]}$.

Assume $n \geq 1$. Let a be the cost of all operations outside of the `for` loop (assuming the `if` branch is taken); let b be the cost of each loop test; and let c be the cost of each loop iteration. The total cost is $T(n) = a + bn + c(n-1)$. Choose $c_1 = b$ and $c_2 = a + b + c$; then when $n \geq 1$, we have $c_1 n \leq T(n) \leq c_2 n$.

Or, less formally, the running time is $\Omega(n)$ because it must examine every element of the input array, and it is $O(n)$ because it performs a bounded amount of pre-loop work plus a loop of fewer than n iterations, each iteration performing a constant amount of work — that is, $O(1) + O(n)O(1) = O(n)$. Combine $O(n)$ and $\Omega(n)$ to get $\Theta(n)$.

2. Prove that if $f = O(g)$ and $g = O(h)$, then $f = O(h)$.

Since $f = O(g)$, there must be c_1 and n_1 such that for all $n \geq n_1$, $f(n) \leq c_1 g(n)$.
Since $g = O(h)$, there must be c_2 and n_2 such that for all $n \geq n_2$, $g(n) \leq c_2 h(n)$.
Choose $n_0 = \max(n_1, n_2)$ and $c = c_1 c_2$. Then for all $n \geq n_0$,

$$f(n) \leq c_1 g(n) \leq c_1 c_2 h(n) = ch(n)$$

and so $f = O(h)$.

3. Problem 3-4 (a, b, c, d) in the textbook (page 62).

Let f and g be asymptotically positive functions. Prove or disprove each of the following conjectures:

(a) $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

False. Here is a counter-example: Let $f(n) = n$ and $g(n) = n^2$. We know that $f = O(g)$ but $g \neq O(f)$.

(b) $f(n) + g(n) = \Theta(\min(f(n), g(n)))$.

False. Here is a counter-example: Let $f(n) = 1$ and $g(n) = n$. Then for $n \geq 1$, $\min(f(n), g(n)) = f(n) = 1$, which cannot bound $n+1$ above.

(c) $f(n) = O(g(n))$ implies $\lg(f(n)) = O(\lg(g(n)))$, where $\lg(g(n)) \geq 1$ and $f(n) \geq 1$ for all sufficiently large n.

Since $f(n) = O(g(n))$, there are c_1 and n_1 such that for all $n \geq n_1$, $f(n) \leq c_1 g(n)$.

Since the lg function is monotone:

$$\lg(f(n)) \leq \lg(c_1 g(n)) = \lg(c_1) + \lg(g(n))$$

Assume that $\lg(c_1) \geq 1$; for a big-O bound, we can always choose a *larger* constant c_1. We're guaranteed that for "sufficiently large n" (call it $n \geq n_2$), $\lg(g(n)) \geq 1$, so

$$\lg(f(n)) \leq \lg(c_1) + \lg(g(n)) \leq \lg(c_1)\lg(g(n))$$

So choose $n_0 = \max(n_1, n_2)$ and $c = \lg(c_1)$.

(d) $f(n) = O(g(n))$ implies $2^{f(n)} \in O(2^{g(n)})$.

False. Here is a counter-example: Let $f(n) = 2n$ and $g(n) = n$.

But $2^{2n} \neq O(2^n)$. Suppose it were; then there would be c such that

$$\begin{aligned} 2^{2n} &\leq c2^n \\ 2^{2n}/2^n &\leq c \\ 2^n &\leq c \end{aligned}$$

That is, the "constant" c would have to be larger than 2^n for all sufficiently large n, which is impossible.

4. Problem 4-1 (a, b, f, g) in the textbook (page 107).

Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leq 2$. Make your bounds as tight as possible, and justify your answers.

(a) $T(n) = 2T(n/2) + n^4$.

Apply the master theorem, with $p = \log_2 2 = 1$. Then $n^4 = \Omega(n^{p+\epsilon})$ with $\epsilon = 3$. So the theorem tells us that $T(n) = \Theta(n^4)$.

(b) $T(n) = T(7n/10) + n$.

Apply the master theorem, with $p = \log_{10/7} 1 = 0$. Then $n = \Omega(n^{p+\epsilon})$ with $\epsilon = 1$. So the theorem tells us that $T(n) = \Theta(n)$.

(f) $T(n) = 2T(n/4) + \sqrt{n}$.

Apply the master theorem, with $p = \log_4 2 = \frac{1}{2}$. Then $\sqrt{n} = \Theta(n^{\frac{1}{2}})$. So the theorem tells us that $T(n) = \Theta(n^{\frac{1}{2}} \lg n)$.

(g) $T(n) = T(n-2) + n^2$.

$T(n) = \Theta(n^3)$. One way to show it is to guess the solution and use induction to show the bounds. (For the upper bound, use $T(n) \le cn^3$, and for the lower bound use $T(n) \le c_1 n^3 - c_2 n^2$.

Here's an easier solution:
For simplicity, let's assume n is even. Then

$$T(n) = \sum_{k=1}^{n/2} (2k)^2 = 4 \sum_{k=1}^{n/2} k^2$$

By equation A.3 (p1147) in the textbook,

$$T(n) = \cdots = 4 \cdot \frac{1}{6} \left(\frac{n}{2}\right) \left(\frac{n}{2} + 1\right) (n+1)$$

Multiplying it out, we get a cubic polynomial whose n^3 coefficient is positive, so $T(n) = \Theta(n^3)$.

5. Problem 4.2 in Lecture notes 1 (`[aux01]`), page 7.

If there are positive constants a and c such that

$$T(n) = \sum_{j=2}^{n} (a + (j-1)c)$$

then there are constants A, B, and C such that

$$T(n) = An^2 + Bn + C$$

Of course A, B, and C depend on a and c, but to not depend on n. You should also show that $A > 0$. That's an important fact.

$$\begin{aligned}
T(n) &= \sum_{j=2}^{n} (a + (j-1)c) \\
&= \sum_{j=2}^{n} a + c \sum_{j=2}^{n} (j-1) && \text{(linearity)} \\
&= a(n-1) + c \sum_{j=1}^{n-1} j \\
&= a(n-1) + c \frac{(n-1)n}{2} && \text{(Equation A.1, p1146)} \\
&= \left(\frac{c}{2}\right) n^2 + \left(a - \frac{c}{2}\right) n - a
\end{aligned}$$

So $A = \frac{c}{2}$, $B = a - \frac{c}{2}$, and $C = -a$. A is positive since c is positive.

6. Let a binary tree be either NIL or a node with left and right attributes whose values are also binary trees. Define the mindepth function as follows:

$$\text{mindepth}(t) = \begin{cases} 0 & \text{if } t = \text{NIL} \\ 1 + \min(\text{mindepth}(\text{left}(t)), \text{mindepth}(\text{right}(t))) & \text{otherwise} \end{cases}$$

and define the countnil function as follows:

$$\text{countnil}(t) = \begin{cases} 1 & \text{if } t = \textsc{nil} \\ \text{countnil}(\text{left}(t)) + \text{countnil}(\text{right}(t)) & \text{otherwise} \end{cases}$$

Prove the following: If $\text{mindepth}(t) \geq n$, then $\text{countnil}(t) \geq 2^n$.

Hint: Use induction on n.

If $\text{mindepth}(t) \geq n$, then $\text{countnil}(t) \geq 2^n$.

Proof: by induction on n.

Case $n = 0$:
Goal: (for all t) if $\text{mindepth}(t) \geq 0$, then $\text{countnil}(t) \geq 2^0 = 1$.

Well, $\text{mindepth}(t) \geq 0$ always. Easy to show that $\text{countnil}(t) \geq 1$ for any tree. Done.

Case $n = k + 1$:
The inductive hypothesis is:

(for all t) if $\text{mindepth}(t) \geq k$, then $\text{countnil}(t) \geq 2^k$

Goal: (for all t) if $\text{mindepth}(t) \geq k + 1$, then $\text{countnil}(t) \geq 2^{k+1}$.

Consider an aribtrary tree t.
If $\text{mindepth}(t) \geq k+1$, then t can't be NIL. So we will use the non-NIL cases of the mindepth and countnil functions.

By case 2 of mindepth:

$$1 + \min(\text{mindepth}(\text{left}(t)), \text{mindepth}(\text{right}(t))) \geq k + 1$$

Cancel out the (1+):

$$\min(\text{mindepth}(\text{left}(t)), \text{mindepth}(\text{right}(t))) \geq k$$

Facts about $\min(a, b)$: $a \geq \min(a, b)$ and $b \geq \min(a, b)$. So:

$$\text{mindepth}(\text{left}(t)) \geq \min(..) \geq k$$
$$\text{mindepth}(\text{right}(t)) \geq \min(..) \geq k$$

Now we can apply the IH to the left and right children of t and get

$$\text{countnil}(\text{left}(t)) \geq \text{countnil}(\text{left}(t)) \geq 2^k$$
$$\text{countnil}(\text{right}(t)) \geq \text{countnil}(\text{right}(t)) \geq 2^k$$

Now calculate

$$\begin{aligned} \text{countnil}(t) &= \text{countnil}(\text{left}(t)) + \text{countnil}(\text{right}(t)) \\ &\geq 2^k + 2^k = 2^{k+1} \end{aligned}$$

Done.