Recitation 2 — Recurrences

Parallel and Sequential Data Structures and Algorithms, 15-210 (Spring 2013)

January 23, 2012

1 Announcements

- HW1 is due on Monday January 28. Hopefully you have all started by now; if not, now would be a good time.
- If you are not able/want to use Piazza to contact the course staff, you may send email to

  `15210-staff@lists.andrew.cmu.edu.`

- Questions from lecture or homework?

2 Recurrences

Today we will be talking about how to solve recurrences. This will be helpful for you when doing your next homework assignment.

Let's start by solving a recurrence which should be familiar to all of you as a warmup:

$$W(n) = 2W(n/2) + O(n)$$

Suppose $W(1) = O(1)$. We claim that $W(n) = O(n)$. Is this true? Let's try to prove it by induction.

Base case: Given.

Inductive hypothesis: For all i<n, $W(i) = O(i)$.

Inductive case:

$$\begin{aligned} W(n) &= 2W(n/2) + O(n) \\ &= 2\left[O(n/2)\right] + O(n) \\ &\le 2O(n) + O(n) \\ &= O(n) \end{aligned}$$

So, we proved that $W(n) = O(n)$. Or did we?

2.1 A Closer Look

What went wrong? Let's take a closer look at the definition of Big-O.

Definition 2.1. $f = O(n)$ if there exists $c > 0$ and $n_0 > 0$ such that $f(n) \leq cn$ for all $n > n_0$.

Using Definition 2.1 we can prove the following lemma:

Lemma 2.2. *If $f = O(n)$, there exist constants k_1, k_2 so that $f(n) \leq k_1 n + k_2, n \geq 0$*

Proof. By the definition of Big-O, $f = O(n)$, so there exists constants c and n_0 such that $f(n) \leq cn$ for $n > n_0$. Then $k_1 = c, k_2 = \max(f(i) : 0 \leq i < n_0)$ works. □

So, when we say $W(n) = O(n)$, we mean that there exists some n_0, c such that for all $n > n_0$, $W(n) \leq cn$, and want to show that there exists constants k_1 and k_2 such that $W(n) \leq k_1 n + k_2$ for all $n \geq 0$. This isn't the case in our proof of the inductive case:

$$\begin{aligned} W(n) &\leq 2W(n/2) + cn \\ &\leq 2\left[k_1 n/2 + k_2\right] + cn \\ &= (k_1 + c)n + 2k_2 \\ &\not\leq k_1 n + k_2 \end{aligned}$$

Do you see what went wrong?

Since $c > 0$, there is no choice of c that makes this proof go through.

2.2 Doing It Correctly

Now let's try correctly proving $W(n) = O(n \log n)$. We assume there are constants n_0 and c such that for all $n > n_0$, $W(n) \leq cn \log n$. So we want to show that there are constants k_1 and k_2 such that $W(n) \leq k_1 n \log n + k_2$. To make the proof go through we let $k_1 = 2c$ and $k_2 = c$. The base case holds because $W(1) = k_2 = O(1)$. Here is the proof of the inductive case:

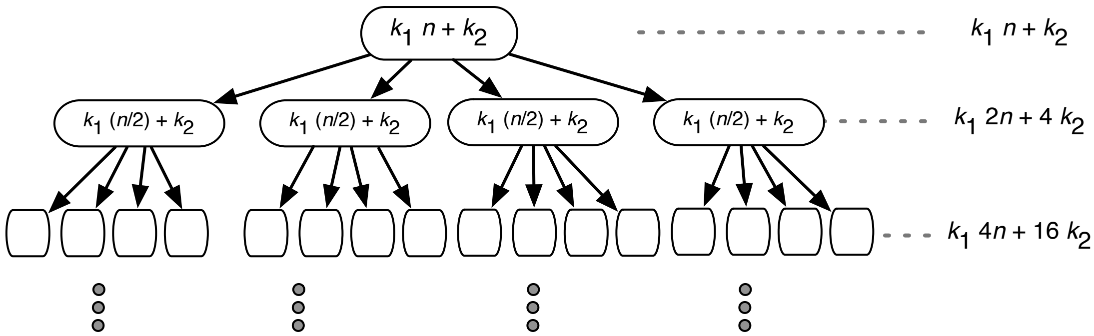
$$\begin{aligned} W(n) &\leq 2W(n/2) + cn \\ &\leq 2(k_1 \tfrac{n}{2} \log(\tfrac{n}{2}) + k_2) + cn \\ &= k_1 n(\log n - 1) + 2k_2 + cn \\ &= k_1 n \log n + k_2 + (cn + k_2 - k_1 n) \\ &\leq k_1 n \log n + k_2, \end{aligned}$$

where the final step follows because $cn + k_2 - k_1 n \leq 0$ as long as $n > 1$.

2.3 Brick Method

Yesterday in lecture we went over the brick method for determining if a recurrence is root-dominated, leaf-dominated, or balanced. It's a good way to get started when solving a recurrence.

- For $W(n) = 4W(n/2) + O(n)$, the recursion tree is:



  That is, we have at level i:

Problem Size	$n/2^i$
Node Cost	$\leq k_1(n/2^i) + k_2$
Number of Nodes	4^i

  So the cost at each level is bounded by

$$4^i \cdot \left(k_1(n/2^i) + k_2\right) = k_1 \cdot 2^i \cdot n + 4^i \cdot k_2$$

  This gives us a stack of bricks which is dominated at the leaves because the cost at level i geometrically *increases* by more than a constant factor of 2. So $W(n) = O(\text{number of leaves}) = O(n^2)$, since the leaves are at level $\log_2 n$ and there are $4^{\log_2 n} = n^2$ of them.

- For $W(n) = W(3n/4) + O(n)$, we have at level i:

Problem Size	$(3/4)^i n$
Node Cost	$\leq k_1(3/4)^i n + k_2$
Number of Nodes	1

  The cost at each level is bounded by

$$1 \cdot \left(k_1(3/4)^i + k_2\right) = k_1 \cdot (3/4)^i \cdot n + k_2$$

  This gives us a stack of bricks which is dominated at the root node because the cost at level i geometrically *decreases* by a constant factor of 3/4. So $W(n) = O(\text{cost at root}) = O(n)$.

- For $W(n) = 2W(n/2) + O(n)$, we have at level i:

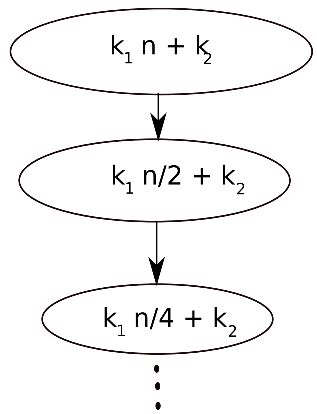
Problem Size	$n/2^i$
Node Cost	$\leq k_1(n/2^i) + k_2$
Number of Nodes	2^i

The cost at each level is bounded by

$$2^i \cdot \Big(k_1(n/2^i) + k_2\Big) = k_1 \cdot n + 2^i \cdot k_2$$

This gives us a stack of bricks which is balanced throughout because the cost at every level is the same, within a constant factor. So $W(n) = O(\text{height of tree} * \text{work at each level}) = O(n \log n)$.

- For $W(n) = W(n/2) + O(n)$, we have at level i:



Problem Size	$(1/2)^i n$
Node Cost	$\leq k_1(1/2)^i n + k_2$
Number of Nodes	1

The cost at each level is bounded by

$$1 \cdot \Big(k_1(1/2)^i + k_2\Big) = k_1 \cdot (1/2)^i \cdot n + k_2$$

This gives us a stack of bricks which is dominated at the root node because the cost at level i geometrically *decreases* by a constant factor of $1/2$. So $W(n) = O(\text{cost at root}) = O(n)$.