Notes for Recitation 14

The Akra-Bazzi Theorem

Theorem 1 (Akra-Bazzi, strong form). *Suppose that:*

$$T(x) = \begin{cases} \text{is defined} & \text{for } 0 \le x \le x_0 \\ \sum_{i=1}^{k} a_i T(b_i x + h_i(x)) + g(x) & \text{for } x > x_0 \end{cases}$$

where:

- $a_1, \ldots, a_k$ *are positive constants*
- $b_1, \ldots, b_k$ *are constants between 0 and 1*
- x_0 *is "large enough" in a technical sense we leave unspecified*
- $|g'(x)| = O(x^c)$ *for some* $c \in \mathbb{N}$
- $|h_i(x)| = O(x/\log^2 x)$

Then:

$$T(x) = \Theta\left(x^p\left(1 + \int_1^x \frac{g(u)}{u^{p+1}}\, du\right)\right)$$

where p satisfies the equation $\sum_{i=1}^{k} a_i b_i^p = 1$.

The only difference between the strong and weak forms of Akra-Bazzi is the appearance of this $h_i(x)$ term in the recurrence, where $h_i(x)$ represents a small change in the size of the subproblems ($O(x/\log^2 x)$). Notice that, despite the change in the recurrence, the solution $T(x)$ remains the same in both the strong and weak forms, with no dependence on $h_i(x)$! In algorithmic terms, this means that *small* changes in the size of subproblems have no impact on the asymptotic running time.

Example: Let's compare the Θ bounds for the following divide-and-conquer recurrences.

$$T_a(n) = 3T\left(\frac{n}{3}\right) + n \qquad\qquad T_b(n) = 3T\left(\left\lfloor\frac{n}{3}\right\rfloor\right) + n$$

For $T_a(n)$ we have $a_1 = 3$, $b_1 = 1/3$, $g(n) = n$, $p = 1$.

For $T_b(n)$ we have the same parameters as for $T_a(n)$, plus $h_1(n) = \lfloor n/3 \rfloor - n/3$.

Using the strong Akra-Bazzi form, the $h_1(n)$ falls out of the equation:

$$T_a(n) = T_b(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du)) = \Theta(n \log n).$$

The addition of the ceiling operator changes the value of $n/3$ by at most 1, which is easily $O(n/\log^2 n)$. So floor and ceiling operators have no impact on the asymptotic solution to a recurrence.

1 TriMergeSort

We noted in lecture that reducing the size of subproblems is much more important to the speed of an algorithm than reducing the number of additional steps per call. Let's see if we can improve the $\Theta(n \log n)$ bound on `MergeSort` from lecture.

Let's consider a new version of `MergeSort` called `TriMergeSort`, where the size n list is now broken into *three* sublists of size $n/3$, which are sorted recursively and then merged. Since we know that floors and ceilings do not affect the asymptotic solution to a recurrence, let's assume that n is a power of 3.

1. How many comparisons are needed to merge three lists of 1 item each?

   Solution. 3. For example, a merge of lists {4},{5}, and {2} compares 4 with 5 and 4 with 2, adding 2 to the final list. Then it compares 4 with 5 and adds 4 to the final list. Finally, it appends the remaining 5. (This could be made more efficient, but let's not worry about that here.) ■

2. In the worst case, how many comparisons are needed to merge three lists of $n/3$ items, where n is a power of 3?

   Solution. $2(n-2)+1$. The worst case occurs if the first list empties when there is exactly 1 item in each of the other two. Prior to this, each of the other $n-2$ numbers requires 2 comparisons before going into the big list. After this, we only need 1 more comparison between the 2 leftover items. ■

3. Define a divide-and-conquer recurrence for this algorithm. Let $T(n)$ be the number of comparisons to sort a list of n items.

   Solution. $T(n) = 3T(n/3) + 2n - 3$. ■

4. We could analyze the running time of this using plug-and-chug, but let's try Akra-Bazzi. First, what is p?

Solution. $p = 1$. Using $\sum_{i=1}^{k} a_i b_i^p = 1$ with $a_1 = 3$ and $b_1 = 1/3$, we get the constraint that $3(1/3)^p = 1$. ■

5. Does the condition $|g'(x)| = O(x^c)$ hold for some $c \in N$?

   Solution. Yes. $g(n) = 2n - 3$, so $|g'(n)| = 2 = O(n^c)$ for $c = 0$. ■

6. Determine the theta bound on T(n) by integration.

   Solution.

$$
\begin{aligned}
T(n) =& \Theta\left(n^p\left(1+\int_1^n \frac{g(u)}{u^{p+1}}\,du\right)\right)\\
=& \Theta\left(n\left(1+\int_1^n \frac{2u-3}{u^2}\,du\right)\right)\\
=& \Theta\left(n\left(1+\int_1^n \frac{2}{u} - \int_1^n \frac{3}{u^2}\,du\right)\right)\\
=& \Theta\left(n\left(1+2\log u\Big|_1^n + \frac{3}{u}\Big|_1^n\right)\right)\\
=& \Theta\left(n\left(1+2\log n + \frac{3}{n} - 3\right)\right)\\
=& \Theta\left(2n\log n - 2n + 3\right)\\
=& \Theta\left(n\log n\right)
\end{aligned}
$$

■

7. Turns out that any equal partition of the list into a constant number of sublists $c > 1$ will yield the same theta bound. Can you see why?

   Solution. Given a constant size partition, the recurrence will always be in the form:

$$T(n) = cT(n/c) + \left[(c-1)(n-(c-1)) + \sum_{i=1}^{c-2} i\right]$$

The first term creates the constraint that $c(1/c)^p = 1$, which always gives $p = 1$. The second term will always be $\Theta(n)$, which dominates the final bound after integration. Therefore, no matter what $c > 1$ we choose, $T(n) = \Theta(n \log n)$. ■

2 OverSort

We have devised an error-tolerant version of `MergeSort`. We call our exciting new algorithm `OverSort`.

Here is how the new algorithm works. The input is a list of n distinct numbers. If the list contains a single number, then there is nothing to do. If the list contains two numbers, then we sort them with a single comparison. If the list contains more than two numbers, then we perform the following sequence of steps.

- We make a list containing the first $\frac{2}{3}n$ numbers and sort it recursively.
- We make a list containing the last $\frac{2}{3}n$ numbers and sort it recursively.
- We make a list containing the first $\frac{1}{3}n$ numbers and the last $\frac{1}{3}n$ numbers and sort it recursively.
- We merge the first and second lists, throwing out duplicates.
- We merge this combined list with the third list, again throwing out duplicates.

The final, merged list is the output. What's great is that even if the sorter occasionally forgets about a number, the `OverSort` algorithm still outputs a complete, sorted list!

1. Let $T(n)$ be the maximum number of comparisons that `OverSort` could use to sort a list of n distinct numbers, assuming the sorter never forgets a number and n is a power of 3. What is $T(3)$? Write a recurrence relation for $T(n)$. (*Hint:* Merging a list of j distinct numbers and a list of k distinct numbers, and throwing out duplicates of numbers that appear in both lists, requires at most $j + k - d$ comparisons, when $d > 0$ is the number of duplicates.)

   Solution. When $n = 3$, we begin with a list (a, b, c) of 3 distinct numbers. `OverSort` starts by forming the lists (a, b), (b, c), and sorts each of them. Sorting a list of length two takes 1 comparison, so the total number of comparisons required to sort both lists is 2. Next it merges these two lists. There must be exactly one duplicate in the two lists, so, using the Hint, we conclude that this takes $2 + 2 - 1 = 3$ more comparisons and yields a sorted length three list of all the elements.

   `OverSort` then forms the list (a, c), uses 1 comparison to sort it, and merges this sorted list of length two with the previous length three list. However, merging these two lists takes only $2 + 3 - 2 = 3$ comparisons, because there are two duplicates. So $T(3)$, the worst case number of comparisons, is $2 + 3 + 1 + 3 = 9$.

   Now for $n > 3$, `OverSort` will form and then sort three different lists of length $(2/3)n$. The first two overlap by $(1/3)n$, so merging them takes $(2/3)n + (2/3)n - (1/3)n = n$

comparisons and yields a list of length n. This list and the remaining list of length $(2/3)n$ overlap by $(2/3)n$, so merging them requires only another n comparisons. So

$$T(n) = 3T\left(\lceil \frac{2}{3}n \rceil\right) + 2n.$$

■

2. Now we're going to apply the Akra-Bazzi Theorem to find a Θ bound on $T(n)$. Begin by identifying the following constants and functions:

   - The constant k.

     Solution. $k = 1$ ■

   - The constants a_i.

     Solution. $a_1 = 3$ ■

   - The constants b_i.

     Solution. $b_1 = 2/3$ ■

   - The functions h_i.

     Solution. $h_1(x) = 0$ ■

   - The function g.

     Solution. $g(x) = 2x$ ■

   - The constant p. You can leave p in terms of logarithms, but you'll need a rough estimate of its value later on.

     Solution.

$$\begin{aligned} 3 \cdot \left(\frac{2}{3}\right)^p &= 1 \\ \ln 3 + p(\ln 2 - \ln 3) &= 0 \\ p &= \frac{\ln 3}{\ln 3 - \ln 2} = 2.7095\ldots \end{aligned}$$

■

3. Does the condition $|g'(x)| = O(x^c)$ for some $c \in \mathbb{N}$ hold?

   Solution. Yes. $g'(x) = 2 = O(x^0)$. ■

4. Does the condition $|h_i(x)| = O(x/\log^2 x)$ hold?

   Solution. Yes. $0 = O(x/\log^2 x)$. ■

5. Determine a Θ bound on $T(n)$ by integration.

Solution.

$$\begin{aligned}
T(n) &= \Theta\left(n^p \cdot \left(1 + \int_1^n \frac{2u}{u^{1+p}}\ du\right)\right) \\
&= \Theta\left(n^p \cdot \left(1 + 2\int_1^n \frac{1}{u^p}\ du\right)\right) \\
&= \Theta(n^p) \\
&= \Theta\left(n^{2.7095\ldots}\right)
\end{aligned}$$

Note that since $p > 1$, the integral nonnegative and bounded above by a constant, no matter how large n grows. Thus, everything except n^p is absorbed by the Θ. ■

3 Divide & Conquer

Find Θ bounds for the following divide-and-conquer recurrences. Assume $T(1) = 1$ in all cases.

1. $T(n) = 3T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + n$

   Solution. $a_1 = 3$, $b_1 = 1/3$, $h_1(n) = \lfloor n/3 \rfloor - n/3$, $g(n) = n$, $p = 1$,

   $$T(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du)) = \Theta(n \log n).$$

   ■

2. $T(n) = 4T\left(\left\lfloor \frac{n}{3} \right\rfloor\right) + n^2$

   Solution. $a_1 = 4$, $b_1 = 1/3$, $h_1(n) = \lfloor n/3 \rfloor - n/3$, $g(n) = n^2$, $p = \log_3 4$,

   $$T(n) = \Theta(n^{\log_3 4}(1 + \int_1^n \frac{u^2}{u^{\log_3 4+1}} du)) = \Theta(n^{\log_3 4}(1 + \int_1^n u^{1-\log_3 4} du)) = \Theta(n^2).$$

   ■

3. $T(n) = T\left(\left\lceil \frac{n}{4} \right\rceil\right) + T\left(\left\lfloor \frac{3n}{4} \right\rfloor\right) + n$

   Solution. $a_1 = 1$, $a_2 = 1$, $b_1 = 1/4$, $b_2 = 3/4$, $h_1(n) = \lceil n/4 \rceil - n/4$, $h_2(n) = \lfloor 3n/4 \rfloor - 3n/4$, $g(n) = n$, $p = 1$,

   $$T(n) = \Theta(n(1 + \int_1^n \frac{u}{u^2} du)) = \Theta(n \log n).$$

   ■