CS 624: Notes 04

Ryan Culpepper

September 19, 2022

These notes contain an outline of what I said in lecture (but only an outline), and they also contain interactive questions and exercises. The corresponding slides are in `slides03.pdf`.

1 Administrative

- Homework questions on Piazza

2 From last time

We introduced *asymptotic efficiency*:

- $O(f)$ means eventually bounded above by cf
- $\Omega(f)$ means eventually bounded below by cf
- $\Theta(f)$ means eventually bounded above and below by $c_1 f$ and $c_2 f$

The run time of an algorithm can often be specified by a *recurrence* (a recursive equation). We generally want to solve such recurrences, or at least find asymptotic bounds on the solution.

One method is to guess a bound and then use the recurrence equation (and induction) to prove it. Now we'll discuss other techniques.

3 Calculating asymptotic bounds (continued)

3.1 Slide 21: Recursion Trees

3.2 Slide 24: The Master Method

Given a recurrence of the form

$$T(n) = aT(n/b) + f(n) \quad \text{where } a \geq 1,\ b > 1,\ f \text{ ultimately positive}$$

Let $p = \log_b a$.

1. If $f(n) = O(n^{p-\epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^p)$.
2. If $f(n) = \Theta(n^p)$, then $T(n) = \Theta(n^p \log n)$.
3. If $f(n) = \Omega(n^{p+\epsilon})$ and if f "is not too wiggly", then $T(n) = \Theta(f(n))$.

Case 2 has two more specific sub-cases:

2a. If $f(n) = O(n^p)$ then $T(n) = O(n^p \log n)$.

2b. If $f(n) = \Omega(n^p)$ then $T(n) = \Omega(n^p \log n)$.

Slide 30: Why doesn't case 2 imply these two sub-cases?

> Case 2 requires a tight bound on f. Cases 2a and 2b give us *some* information for (loose) upper or lower bounds.

3.2.1 examples

Example 1: $T(n) = 4 * T(n/2) + n$

> $p = \log_2 4 = 2$
> Which case? Case 1: because $n = O(n^{2-\epsilon}), \epsilon = 1$
> $T(n) = \Theta(n^2)$

Example 2: $T(n) = 4 * T(n/2) + n^2$

> $p = 2$
> Which case? Case 2, because $n^2 = \Theta(n^2)$
> $T(n) = \Theta(n^2 \log n)$

Example 3: $T(n) = 4 * T(n/2) + n^3$

> $p = 2$
> Which case? Case 3, because $n^3 = \Omega(n^{2+1})$
> ... and n^3 is not "too wiggly"
> $T(n) = \Theta(n^3)$

Example 4: $T(n) = 4 * T(n/2) + n^2/(\log n)$

> $p = 2$
> Which case?
>
> > Does case 1 apply? Is $n^2/(\log n) = O(n^{p-\epsilon}) = O(n^p/n^\epsilon)$?
> > No, because $\log n$ grows more slowly than n^ϵ for any $\epsilon > 0$.
> > None of the main three cases applies.
> > Case 2a, because $n^2/(\log n) = O(n^2)$.
>
> $T(n) = O(n^2 \log n)$
>
> What about a lower bound?
> $T(n) \geq 4 * T(n/2) + n$
> so $T(n) = \Omega(n^2)$
> But we have no Θ bound.

4 Generating functions (briefly)

Every sequence a_n
$$\{a_n\} = \langle a_0, a_1, a_2, ...\rangle$$
has an associated "generating function" defined by

$$F(x) = \sum_{n=1}^{\infty} a_n x^n$$

In some cases, the series corresponds to a well-known function (because it is the Maclaurin series for that function, for example).

For example, the generating function for the sequence defined by

$$a_n = \frac{1}{n!}$$

is the exponential function
$$F(x) = e^x$$

Now we have (potentially) three different *views* of the same information:

- the sequence
- the series (an infinite polynomial)
- (maybe) the function

By switching back and forth between these views, we can apply the tools associated with all three.

(slides 36-47)

5 Introduction to Heaps

(switch to slides03)

5.1 Slide 4: Pre-Heap

5.2 Slide 12: Heap

A *heap* is a pre-heap with the additional property:

- the key at each node is greater than or equal to the keys of that node's descendents

(stopped slide 14)