Advanced Programming Topics

Solving recurrences and the Master Theorem
Lecture 4
James Marshall

Q: In lecture 3, how did we know that

Q: is there a general method for analysing divide-and-conquer algorithms?

Some tricks for solving recurrences

In counting the steps involved in executing an algorithm, generally we end up with open-form solutions involving summations, *e.g.*

With some know-how, these can typically be reduced to closed-form solutions suitable for asymptotic analysis. For instance, the example above can be shown to be in

But how? Gauss worked this one out when he was a schoolboy…

Solving arithmetic series (in general)

Solving geometric series (e.g. lecture 1)

Other tricks available, see e.g. CLRS appendix A

The Master Theorem (e.g. Theorem 4.1 in CLRS)

For constants $a \geq 1$, $b > 1$ and for $n \geq 0$, let the running time $T(n)$ of an algorithm be defined by the recurrence

$$T(n) = a\,T(n/b) + f(n)$$

Then we can work out an asymptotic bound for $T(n)$ if one of the following three conditions holds:

1. If $f(n) \in O\left(n^{\log_b a - \varepsilon}\right)$ for some constant $\varepsilon > 0$, then $T(n) \in \Theta\left(n^{\log_b a}\right)$
2. If $f(n) \in \Theta\left(n^{\log_b a}\right)$ then $T(n) \in \Theta\left(n^{\log_b a} \log n\right)$
3. If $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ for some constant $\varepsilon > 0$, and if $a\,f(n / b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) \in \Theta\left(f(n)\right)$

Explaining the Master Theorem

What is $n^{\log_b a}$? It is the number of leaves in the recurrence tree. Why?

So the Master Theorem compares the work done in 'bookkeeping' (splitting into subproblems, and then combining solutions) against the amount of work done solving the simplest subproblems, the leaves:

1. 'Cheap' bookkeeping - dividing and combining is *polynomially cheaper* than the work done at the leaves, so the latter dominates the total computational complexity.
2. 'Marginal' bookkeeping - dividing and combining is asymptotically as expensive as the work done at the leaves, so neither dominates in the total computational complexity.
3. 'Expensive' bookkeeping - dividing and combining is *polynomially more expensive* than the work done at the leaves, so the former dominates the total computational complexity

Caveats of the Master Theorem

N.B. *polynomially cheaper* (for example) means just that, n^{ε} cheaper; logarithmically cheaper is not sufficient.

N.B. condition 3 requires that bookkeeping associated breaking down a problem and combining solutions is cheaper than the bookkeeping associated with the original problem; in other words the costs of bookkeeping do not increase as we go down the recurrence tree.

Applying the Master Theorem - mergesort