CPSC 320: Intermediate Algorithm Design & Analysis

Divide & Conquer and Recurrences

Steve Wolfman

Problem-Solving Approaches

Many problems can be solved by the same broad style of approach. We'll run into several of these styles:

- Input consuming (like insertion sort)
- Output producing (like selection sort)
- Divide-and-Conquer (like merge sort)
- Greedy
- Dynamic Programming

Divide-and-Conquer Approach

When a larger problem can be divided into similar sub-problems, it's often possible to solve the larger problem by:

- dividing it into smaller pieces
- recursively solving the pieces
- assembling the larger solution from the smaller ones

Merge Sort Reminder

How can we sort a list by divide-and-conquer?

- Break the list into two (roughly) equal-sized pieces
- Sort the pieces (using Merge Sort)
- Merge the two sorted lists back together

We need a base case for the recursion to "bottom out"!

Fortunately, any list of length 1 is already sorted.

Skyline Problem

Analyzing Iterative Algorithms

We label lines that take constant time and build up summations to represent loops.

Together, these form an equation like T(n) that indicates the number of "simple computational steps" to solve a problem of size n.

Analyzing Recursive Algorithms

We label lines that take constant time and build up summations to represent loops.

Together, these form an equation like T(n) that indicates the number of "simple computational steps" to solve a problem of size n.

We *already* have a formula for how long a recursive call takes: T(·)

Just add in a T(·) term with an appropriate size argument based on how large a "piece" we pass to the recursive call.

Recurrence Example: FindMax

```
FindMax(arr)
  if arr.length == 1: return arr[0]
  else: return larger of arr[0] and
                  FindMax(arr[1..arr.length-1])
```

$T(1) \leq b$

$T(n) \leq c + T(n-1)$ *if* $n > 1$

Recurrence Example: FindMax

$T(1) \leq b$

$T(n) \leq c + T(n-1)$ *if* $n > 1$

Analysis:

$T(n) \leq c + c + T(n-2)$ (by substitution)

$T(n) \leq c + c + c + T(n-3)$ (by substitution, again)

$T(n) \leq kc + T(n-k)$ (extrapolating $0 < k \leq n$)

$T(n) \leq (n-1)c + T(1) = (n-1)c + b$ (for $k = n-1$)

Analyzing Recurrence Relations: Approaches

- Guess-and-Test
  - Guess the solution (magically), prove that it works by induction.
  - Requires a magical guess!
- Repeated Substitution/Recursion Tree
  - Either repeatedly substitute the recursive definition for recursive terms or draw out the tree of recursive calls generated, annotating each level by the time spent on that level. Devise an expression for the total time.
- Master Method
  - Essentially a summary of common recursion tree patterns.
  - Completely mechanical, but does not always apply!

Analysis of Skyline Problem

More Examples + Pitfalls

- Skyline, Guess-and-Test $T(n) \in O(n)$
- $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1$, $T(1) = 1$, Guess-and-Test $T(n) \in O(n)$
- $T(n) = 3T(n/4) + n^2$, $T(1) = 1$
- $T(n) = T(n/2) + 1$, $T(1) = 1$

Master Method: Context

Let $a \geq 1$ and $b > 1$ be constants and $f(n) : \mathbf{N} \rightarrow \mathbf{R}^+$ and $T(n)$ be defined by:

$T(n) = aT(\text{“}n/b\text{”}) + f(n)$

where $T(n) \in \Theta(1)$ for sufficiently small n, and “n/b” can be $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

Master Method: Cases

Assuming the previous slide’s context holds:

(1) If $f(n) \in O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$

Dominated by “leaf” cost.

(2) If $f(n) \in \Theta(n^{\log_b a})$ then, $T(n) \in \Theta(n^{\log_b a} \lg n)$

“Balanced” cost.

(3) If $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$ and if $af(\text{“}n/b\text{”}) \leq cf(n)$ for some constant $c < 1$ and sufficiently large n, then $T(n) \in \Theta(f(n))$

Dominated by “root” cost.

More Examples (if time available)

- Integer Multiplication
- Closest Pair of Points, 1-D
- Closest Pair of Points, 2-D

What’s Next?

- Selection (and a bit of Randomized Algorithms): CLRS Chapter 9

On Your Own

- PRACTICE various analysis methods, particularly use of recursion trees/repeated substitution and the Master Method.
- MANY practice problems available in CLRS
- Review recursion