COMP 355
Advanced Algorithms

Recurrences & Master Theorem

Rhodes College

Solving Recurrences

When analyzing algorithms, recall that we only care about the *asymptotic behavior*.

Recursive algorithms are no different. Rather than solve exactly the recurrence relation associated with the cost of the algorithm, it is enough to give an asymptotic characterization.

The main tool for doing this is the *master theorem*.

Master Theorem

Theorem 4.1 (CLRS)

Let a ≥ 1 and b > 1 be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Where we interpret n/b to mean either the floor (n/b) or ceil(n/b). Then $T(n)$has the following asymptotic bounds:

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

Simplified Master Theorem

Let $T(n)$ be a monotonically increasing function that satisfies

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$T(1) = c$$

Where a ≥ 1, b > 1, c > 0. If $f(n) = \Theta\left(n^d\right)$ where d ≥ 0, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Pitfalls of Master Theorem

You *cannot* use the Master Theorem if

- $T(n)$ is not monotone, ex: $T(n) = \sin n$
- $f(n)$ is not a polynomial, ex: $T(n) = 2T(\frac{n}{2}) + 2^n$
- b cannot be expressed as a constant, ex: $T(n) = T(\sqrt{n})$

Note here, that the Master Theorem does *not* solve a recurrence relation.

Does the base case remain a concern?

Rhodes College

Master Theorem: Example 1

- Let $T(n) = T(n/2) + ½ n^2 + n$. What are the parameters?

  a = 1

  b = 2

  k = 2

  Therefore, which condition applies?

  $1 < 2^2$, case 3 applies

- We conclude that

$$T(n) \in \Theta(n^k) = \Theta(n^2)$$

Rhodes College

Master Theorem: Example 2

- Let T(n)= 2 T(n/4) + √n + 42. What are the parameters?

  a = 2
  b = 4
  k = 1/2

  Therefore, which condition applies?

  $2 = 4^{1/2}$, case 2 applies

- We conclude that

$$T(n) \in \Theta(n^k \log n) = \Theta(\log n\sqrt{n})$$

Master Theorem: Example 3

- Let T(n)= 3 T(n/2) + 3/4n + 1. What are the parameters?

  a = 3
  b = 2
  k = 1

  Therefore, which condition applies?

  $3 > 2^1$, case 1 applies

- We conclude that $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$

- Note that $\log_2 3 \approx 1.584$…, can we say that $T(n) \in \Theta(n^{1.584})$

  No, because $\log_2 3 \approx 1.5849$… and $n^{1.584} \notin \Theta(n^{1.5849})$

'Fourth Condition'

- Recall that we cannot use the Master Theorem if f(n), the non-recursive cost, is not a polynomial
- There is a limited 4th condition of the Master Theorem that allows us to consider poly-logarithmic functions
- **Corollary**: If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some k≥0 then

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$$

- This final condition is fairly limited and it is presented merely for sake of completeness..

Rhodes College

'Fourth' Condition: Example

- Say we have the following recurrence relation

T(n)= 2 T(n/2) + n log n

- Clearly, a=2, b=2, but f(n) is not a polynomial. However, we have f(n)∈Θ(n log n), k=1
- Therefore by the 4th condition of the Master Theorem we can say that

$$T(n) \in \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n^{\log_2 2} \log^2 n) = \Theta(n \log^2 n)$$

Rhodes College

Other Ways To Solve Recurrences

The CLRS book refers to both the Substitution Method and Recurrence Trees.

Substitution Method (in book)
1. Guess the form of the solution. (this can be difficult).
2. Use mathematical induction to find the constants and show that the solution works.

In order to guess a solution, you may need to build a recurrence tree first.

I present here a way to do backward substitution instead, rather than start with a guess.

Rhodes College

Backward Substitution: Example (1)

Example:

$$T(n) = \begin{cases} 5, \ n = 1 \\ T(n-1) + 2n, \ n > 1 \end{cases}$$

- We begin by unfolding the recursion by a simple substitution of the function values
- We observe that T(n-1) = T((n-1) -1) + 2(n-1) = T(n-2) + 2 (n-1)
- Substituting into the original equation T(n)=T(n-2)+2(n-1)+2n

Rhodes College

Backward Substitution: Example (2)

- If we continue to do that we get
  $T(n) = T(n-2) + 2(n-1) + 2n$
  $T(n) = T(n-3) + 2(n-2) + 2(n-1) + 2n$
  $T(n) = T(n-4) + 2(n-3) + 2(n-2) + 2(n-1) + 2n$
  …..
  $T(n) = T(n-i) + \Sigma_{j=0}^{i-1} 2(n - j)$ *function's value at the i^{th} iteration*
- Solving the sum (use the summation rules on the ref. sheet) we get:
  $T(n) = T(n-i) + 2ni – 2(i-1)(i-1+1)/2 + 2n$
  $T(n) = T(n-i) + 2ni – i^2 + i$

Rhodes College

Backward Substitution: Example (3)

- We want to get rid of the recursive term
  $T(n) = T(n-i) + 2ni – i^2 + i$
- To do that, we need to know at what iteration we reach our base case, i.e. for what value of i can we use the initial condition T(1)=5?
- We get the base case when n-i=1 or i=n-1
- Substituting in the equation above we get
  $T(n) = 5 + 2n(n-1) – (n-1)^2 + (n-1)$
  $T(n) = 5 + 2n^2-2n – (n^2-2n+1) + (n-1) = n^2 + n + 3$

Rhodes College

Recurrence Trees

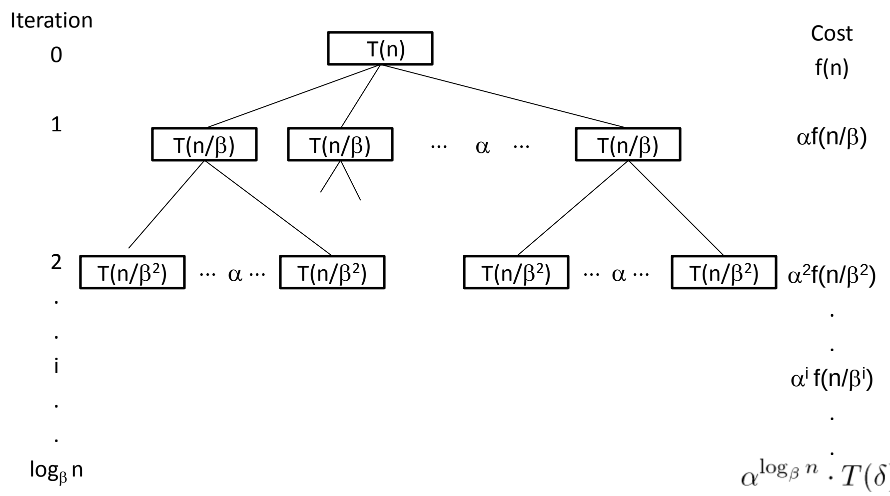
- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion-tree method can be unreliable, just like any method that uses ellipses (…).
- The recursion-tree method promotes intuition, however.
- The recursion tree method is good for generating guesses for the substitution method

Recurrence Trees (1)

- When using recurrence trees, we graphically represent the recursion
- Each node in the tree is an instance of the function. As we progress downward, the size of the input decreases
- The contribution of each level to the function is equivalent to the number of nodes at that level times the non-recursive cost on the size of the input at that level
- The tree ends at the depth at which we reach the base case
- As an example, we consider a recursive function of the form

$$T(n) = \alpha T(n/\beta) + f(n), \quad T(\delta) = c$$

Recurrence Trees (2)



Recurrence Trees (3)

- The total value of the function is the summation over all levels of the tree

$$T(n) = \Sigma_{i=0}^{\log_\beta n} \alpha^i f(n/\beta^i)$$

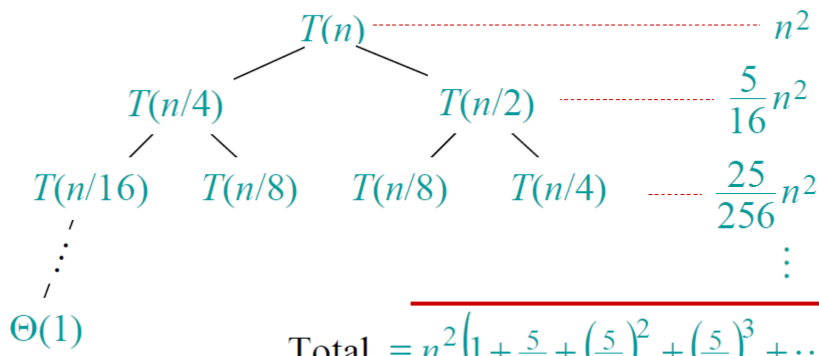
$$T(n) = \alpha^{\log_\beta n} T(\delta) + \sum_{i=0}^{\log_\beta n - 1} \alpha^i f(\frac{n}{\beta^i})$$

- Consider the following concrete example

$$T(n) = T(n/4) + T(n/2) + n^2$$

Recurrence Tree: Example (2)

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



Total $= n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$
$= \Theta(n^2)$ *geometric series*

Practice

Use the Master Theorem to solve the following recurrences, or if the Master Theorem cannot be applied, say so.

a) T(n) = 9T(n/3) + 1
b) T(n) = T(2n/3) + 1
c) T(n) = 3T(n/4) + n lg n
d) T(n) = 2T(n/4) + n

Rhodes College