Recurrence Relations

CS 4102: Algorithms

Fall 2021

Mark Floryan and Tom Horton

Recurrence Relations

Solving Recurrence Relations

- Several (four) methods for solving:
  - Directly Solve
  - Substitution method
    - In short, guess the runtime and solve by induction
  - Recurrence trees
    - We won't see this in great detail, but a graphical view of the recurrence
    - Sometimes a picture is worth 2^{10} words!
  - "Master" theorem
    - Easy to find Order-Class for a number of common cases
    - Different variations are called different things, depending on the source

Directly Solving (or Iteration Method)

Directly Solve (unrolling the recurrence)

- For Mergesort:
  - $T(n) = 2*T(n/2) + n$
  - Do it on board →

Another Example!!

- Consider:
  - $T(n) = 3*T(n/4) + n$

Unroll the recurrence

- $T(n) = 3*T(n/4) + n$
- $T(n) = 3*[3*T(n/16)+n/4] + n$
- $= 9T(n/16) + (7/4)n$

- $T(n) = 9T(n/16) + (7/4)n$
- $T(n) = 9[3T(n/64) + n/16] + (7/4)n$
- $T(n) = 27*T(n/64) + 9n/16 + 7n/4$
- $T(n) = 27*T(n/64) + 37n/16$ //Pattern??

- $T(n) = 3^d * T(n/4^d) + n * \sum(3/4)^{d-1}$ $\leftarrow$ *sum from 0 to d*

Unroll the recurrence

- $T(n) = 3^d * T(n/4^d) + n * \sum(3/4)^{d-1}$

- We hit base case when:
  - $n/(4^d) = 1$
  - $n = 4^d$
  - $d = \log_4(n)$ //seem familiar??

Unroll the recurrence

- $T(n) = \mathbf{3^d * T(n/4^d)} + n * \sum(3/4)^d$

- Let's do one term at a time.
- $3^d * T(n/4^d)$
- $3^{\log 4(n)} * T(1)$
- $3^{\log 4(n)} = n^{\log 4(3)}$ //huh? this is a log rule

Unroll the recurrence

- $T(n) = 3^d * T(n/(4^d)) + \mathbf{n * \sum(3/4)^{d-1}}$

- Let's do one term at a time.
  - $n * \sum(3/4)^{d-1}$ //note summation part approaches 4 as d grows
  - $n * \sum(3/4)^{d-1} <= 4*n = \Theta\mathbf{(n)}$

Unroll the recurrence

- $T(n) = 3^d * T(n/4^d) + n * \sum(3/4)^d$

- $T(n) = 3^{\log4(n)} + \Theta(n)$
- $T(n) = n^{\log4(3)} + \Theta(n)$ //log rules

- $T(n) = o(n) + \Theta(n)$

- ***$\underline{T(n) = \Theta(n)}$***

Substitution Method

Iteration or Substitution Method

- Strategy
  - 1. Consider Mergesort Recurrence
    - $T(n) = 2*T(n/2) + n$
  - 2. Guess the solution
    - Let's go with n*log(n) **Remember logs are all base 2 (usually)
  - 3. Inductively Prove that recurrence is in proper order class
    - For n*log(n), we need to prove that $T(n) <= c*n*\log(n)$
    - For some 'c' constant and for all n >= n0
    - Remember, we get to choose the 'c' and 'n0' values
  - Do it on board →

Substitution Method: Subtleties

- Consider:
  - $T(n) = 2*T(n/2) + 1$ $T(1)=1$
- Let's make our guess:
  - We are thinking $O(n)$
- Try to prove:
  - $T(n) <= c*n$
- What happens? How do we fix this issue?
- On board →

Substitution Method: Subtleties

- Consider:
  - $T(n) = 2*T(n/2) + 1$

Substitution Method: Subtleties

- Summary of the problem / issue:
  - $T(n) = 2*T(n/2) + 1$
  - $T(n) <= 2(c*(n/2)) + 1$
  - $T(n) <= c*n + 1$

- What is the issue here?
- $c*n + 1$ is TOO LARGE.
- Need to prove exact form of inductive hypothesis

Substitution Method: Subtleties

- Here is how we fix the issue. Subtract lower order term.
- Inductive Hypothesis:
  - T(n) <= c*n – d //d is a constant term. Note c*n-d <= c*n

- Fix:
  - T(n) = 2*T(n/2) + 1
  - T(n) <= 2(c*(n/2) - d) + 1
  - T(n) <= c*n -2d + 1 <= c*n //as long as d >= 1/2

Substitution Method: Another Pitfall

- Consider Mergesort recurrence again:
  - T(n) = 2*T(n/2) + n
- Let's make our guess:
  - We are thinking O(n) ← Note that this is INCORRECT!
- Try to prove:
  - T(n) <= c*n

- What happens?
- On board →

Substitution Method: Another Pitfall

- Consider Mergesort recurrence again:
  - $T(n) = 2*T(n/2) + n$

Substitution Method: Pitfall Example

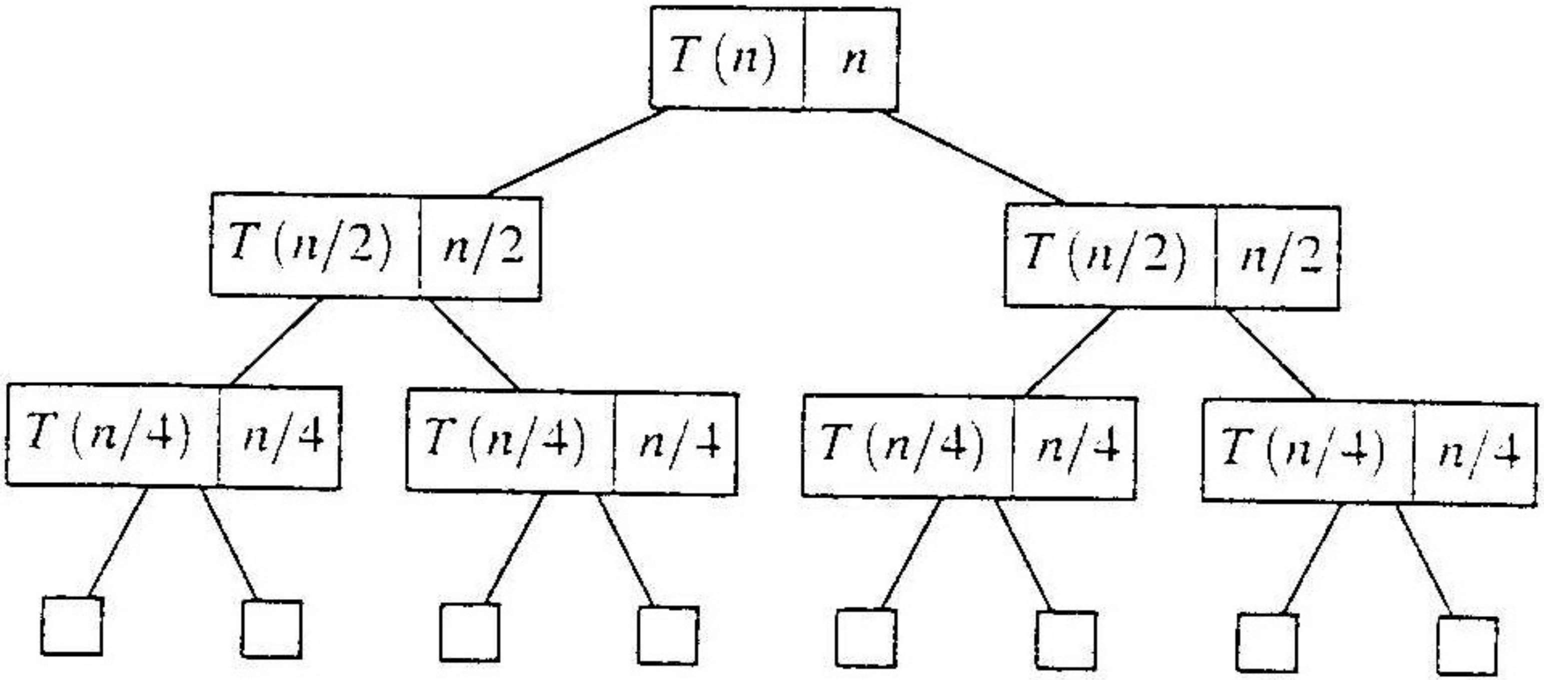
- Attempt to prove:
  - $T(n) = 2*T(n/2) + n$
  - $T(n) <= 2*(c*n/2) + n$
  - $T(n) <= c*n + n$

- Again, need to prove EXACT form of inductive hypothesis.
- Subtracting off a lower order term won't help.
  - Why?

Recursion Tree Method

Recursion Tree Method

- Evaluate: T(n) = 2*T(n/2) + n
  - Work copy: T(k) = T(k/2) + T(k/2) + k
  - For k=n/2, T(n/2) = T(n/4) + T(n/4) + (n/2)
- [size| non-recursive cost]



Recursion Tree: Total Cost

- To evaluate the total cost of the recursion tree
  - sum all the non-recursive costs of all nodes
  - = Sum (rowSum(cost of all nodes at the same depth))
- Determine the maximum depth of the recursion tree:
  - For our example, at tree depth d the size parameter is $n/(2^d)$
  - the size parameter converging to base case, i.e. case 1
  - such that, $n/(2^d) = 1$,
  - $d = \lg(n)$
  - The rowSum for each row is n
- Therefore, the total cost, $T(n) = n \lg(n)$

The Master Theorem

The Master Theorem

- Given: a *divide and conquer* algorithm
  - An algorithm that divides the problem of size *n* into *a* subproblems, each of size *n*/b
  - Let the cost of each stage (i.e., the work to divide the problem + combine solved subproblems) be described by the function *f(n)*
- Then, the Master Theorem gives us a cookbook for the algorithm's running time
  - Some textbooks has a simpler version they call the "Main Recurrence Theorem"
  - We'll splits it into individual parts

The Master Theorem (from Cormen)

- If $T(n) = a\,T(n/b) + f(n)$
  - then let $k = \lg a / \lg b = \log_b(a)$ (critical exponent)
- Then three common cases:
  - If $f(n) \in O(n^{k-\varepsilon})$ for some positive ε, then $T(n) \in \Theta(n^k)$
  - If $f(n) \in \Theta(n^k)$ then $T(n) \in \Theta(\, f(n) \log(n)\,) = \Theta(n^k \log(n))$
  - If $f(n) \in \Omega(n^{k+\varepsilon})$ for some positive ε, and $a\, f(n/b) \leq c\, f(n)$ for some $c < 1$ and sufficiently large n, then $T(n) \in \Theta(f(n))$
- Note: none of these cases may apply

Using the Master Theorem

- $T(n) = 9T(n/3) + n$
  - $A = 9$, $b = 3$, $f(n) = n$

- Master Theorem
  - $k = \lg 9 / \lg 3 = \log_3 9 = 2$
  - Since $f(n) = O(n^{\log_3 9 - \varepsilon})$, where $\varepsilon=1$, case 1 applies:

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

  - Thus the solution is $T(n) = \Theta(n^2)$ since $k=2$

Problems to Try

- Can you use a theorem on these?
- Assume $T(1) = 1$

- $T(n) = T(n/2) + \lg n$
- $T(n) = T(n/2) + n$
- $T(n) = 2T(n/2) + n$ (like Mergesort)
- $T(n) = 2T(n/2) + n \lg n$

More Master Theorem Examples

Problems to Try

- Let's try these?

- $T(n) = 7T(n/3) + n^2$
- $T(n) = 3T(n/3) + n/2$
- $T(n) = 4T(n/2) + n / \log(n)$
- $T(n) = 3T(n/3) + n / \log(n)$

Problems to Try: Solutions

- $T(n) = 7T(n/3) + n^2$
  - $k = \log_3(7) = 1.77$
  - $n^k = n^{1.77}$ n^2
  - Case 3: n^2

regularity: $7*f(n/3) <= c*f(n)$

$7*n^2/9 <= c*n^2$

$(7/9)n^2 <= cn^2$ //YES

Problems to Try: Solutions

- $T(n) = 3T(n/3) + n/2$
  - $k = \log3(3) = 1$
  - $n^k = n$ $n/2$
  - Case 2: nlogn

Problems to Try: Solutions

- T(n) = 4T(n/2) + n / log(n)
  - k = log2(4) = 2
  - n^2 n / log(n)
  - Case 1: n^2

Problems to Try: Solutions

- T(n) = 3T(n/3) + n / log(n)
  - k = log3(3) = 1
  - n n / log(n)
  - Case 1 doesn't apply because f(n) not polynomially smaller
  - e.g., n / log(n) !<= n^0.99 for large n