The Master method for solving recurrences

Instructor: Prof. Ashwin Ganesan

International School of Engineering (INSOFE)
Mumbai 400093
Maharashtra, India

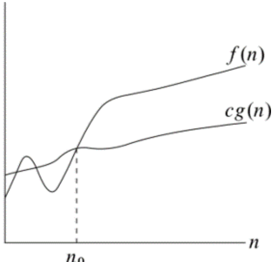
Outline of presentation

1. More on asymptotic notation
2. Divide and conquer algorithms
3. The Master theorem
4. Examples
5. The substitution method for solving recurrences

2 More on asymptotic notation

big-Omega notation

Defn: We write $f(n) = \Omega(g(n))$ (pronounced "f of n is big omega of g of n") if there exist positive constants c and n_0 such that $f(n) \geq cg(n)$, for all $n \geq n_0$.



$f(n) = \Omega(g(n))$

Similar to how $O(\ )$ gives an upper bound on f, $\Omega(\ )$ gives a lower bound on f, i.e. $f(n) = \Omega(g(n))$ means f is bounded from below by some constant multiple of g (for all sufficiently large n)

Example

Prove that $3n^2 + 4n + 6 = \Omega(n^2)$.

Solution: Want to show there exist positive constants c, n_0, such that $3n^2 + 4n + 6 \geq cn^2$, for all $n \geq n_0$.

Know $3n^2 + 4n + 6 \geq 3n^2$, for all $n \geq 1$

Take $c = 3, n_0 = 1$ $\blacksquare$

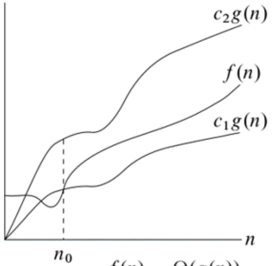
Give the tightest lower bound

- We showed $3n^2 + 4n + 6 = \Omega(n^2)$
- True or False? $3n^2 + 4n + 6 = \Omega(n)$
  - True. Proof: $3n^2 + 4n + 6 \geq 4n$ for all $n \geq 1$.
    So take $c = 4, n_0 = 1$
  - But it is a stronger (better) statement to say
    $3n^2 + 4n + 6 = \Omega(n^2)$
  - Similar to how saying # students in class is ≥ 10 is better than saying # students in class is ≥ 5.
  - Want lower bound to be as high as possible.

big-Theta notation

We write $f(n) = \Theta(g(n))$ (pronounced "f of n is big-theta of g of n") if there exist positive constants c_1 and c_2 such that $f(n)$ is sandwiched between $c_1 g(n)$ and $c_2 g(n)$ for all sufficiently large n, ie there exists c_1, c_2, n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0.$$



$f(n) = \Theta(g(n))$

Intuitively, Θ-notation simplifies the given function and expresses the exact asymptotic growth rate of the given function. For example, $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Review questions

True or False?

1. $3n^2 + 6 = O(n^3)$
2. $3n^2 + 6 = O(n^2)$
3. $3n^2 + 6 = O(n)$
4. $3n^2 + 6 = \Omega(n^2)$
5. $3n^2 + 6 = \Omega(n^3)$
6. $3n^2 + 6 = \Omega(n)$
7. $3n^2 + 6 = \Theta(n^2)$
8. $3n^2 + 6 = \Theta(n^3)$

Other definitions for asymptotic notation

- We defined $f(n) = O(g(n))$ to mean: $\exists c, n_0 > 0$ such that $f(n) \leq cg(n)$, for all $n \geq n_0$.
- Some of the literature defines $O(g(n))$ to be $O(g(n)) := \{f(n) : \exists c, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$
- We write $f(n) = O(g(n))$ if $f(n)$ is a member of the set $O(g(n))$
- Examples of functions in $O(n^2)$: $n^2, n^2 + n, n^2 + 1000n, 1000n^2 + 1000n, n, n/1000, n^{1.999}, n^2/\log\log n$

Other definitions for asymptotic notation

- Similarly, $\Omega(g(n)) := \{f(n) : \exists c, no > 0$ such that $0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$.
- Examples of functions in $\Omega(n^2)$: $n^2, n^2 + n, n^2 - n, n^3, n^{2.0001}$, $n^2 \log\log n, 2^{2^n}$
- $\Theta((g(n)) := \{f(n) : \exists c1, c2, n0 > 0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n0\}$

Running time (without modifier) of insertion sort

- Can talk about running times without modifier "best case" or "worst case"
- Insertion sort is a $O(n^2)$ algorithm because
  - Its worst case running time is a quadratic function of n, which is $O(n^2)$
  - i.e. there is a function $f(n)$ that is $O(n^2)$ such that for any input of size n, the running time is at most $f(n)$
- An algorithm is $\Omega(g(n))$ if for any input of size n, the running time is at least a constant times $g(n)$, for all sufficiently large n
- Insertion sort is a $\Omega(n)$ algorithm because
  - Its best case running time is a linear function of n
- The running time of insertion sort belongs to both $\Omega(n)$ and $O(n^2)$ since the running time falls anywhere between a linear function and a quadratic function of n

Running time (with or without modifiers)

- The running time of insertion sort is $\Omega(n)$ and $O(n^2)$
- The running time of insertion sort is not $\Omega(n^2)$
  - i.e. running time is not always lower-bounded by a quadratic
  - There exist a input (already sorted input) for which running time is linear i.e. $\Theta(n)$
- The worst-case running time of insertion sort is $\Omega(n^2)$
  - Because the worst-case running time of insertion sort is a quadratic function $f(n)$, and a quadratic function $f(n)$ is $\Omega(n^2)$
  - The worst-case running time of insertion sort is also $O(n^2)$, and $\Theta(n^2)$
- The best-case running time of insertion sort is $O(n)$, $\Theta(n)$ and also $\Omega(n)$

3 Divide and conquer algorithms

The Master method for solving recurrences

- The master method provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT(n/b) + f(n)$$

  where $a \geq 1$ and $b > 1$ are constants, $f(n)$ is an asymptotically positive function
- To use master method, need to memorize three cases
- Then can solve many recurrences quite easily, often without pencil and paper
- The recurrence above arises in algorithms that solve a problem of size n by
  - Dividing the problem into a subproblems, each of size n/b
  - The a subproblems are solved recursively, each in time $T(n/b)$
  - $f(n)$ is the work done outside the recursive calls, i.e. cost of "dividing" the problem and "combining" the results of subproblems

Example: Mergesort

```
Statement                                    cost
--------------------------------------------
Procedure MergeSort(A, p, r)
    if p < r
        q = floor((p+r)/2)                   divide; worst case c
        MergeSort(A, p, q)                   conquer; cost T(ceiling(n/2)
        MergeSort(A,q+1,r)                   conquer; cost T(floor(n/2))
        Merge(A, p, q, r)                    combine; cost c'n
```

The worst-case running time $T(n)$ of merge sort is then

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n), & n \geq 2 \end{cases}$$

Example: Mergesort

When n is even, can write

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T(n/2) + \Theta(n), & n \geq 2 \end{cases}$$

This is a special case of $T(n) = aT(n/b) + f(n)$, where $a = 2, b = 2,\ f(n) = \Theta(n)$.

Remark: This form of recurrence assumes all subproblems have the same size.

Divide and conquer algorithms

In general, a divide-and-conquer recursive algorithm that divides a problem of size n into a subproblems, each of size n/b, and combines the solutions in time $f(n)$ has time complexity $T(n)$ that satisfies

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ aT(n/b) + f(n), & n \geq 2 \end{cases}$$

```
Procedure foo(n):
    if n < 1 then exit
    solve first subproblem            \\cost is T(n/b)
    solve second subproblem           \\cost is T(n/b)
    ...
    solve last subproblem             \\cost is T(n/b)
    combine solutions and return      \\cost is f(n)
```

4 The Master theorem

Master theorem

A cookbook method for solving certain recurrences.

Theorem: Suppose $T(n) := aT(n/b) + f(n), n \geq 1$ for some $a \geq 1, b > 1$ and function f. Then,

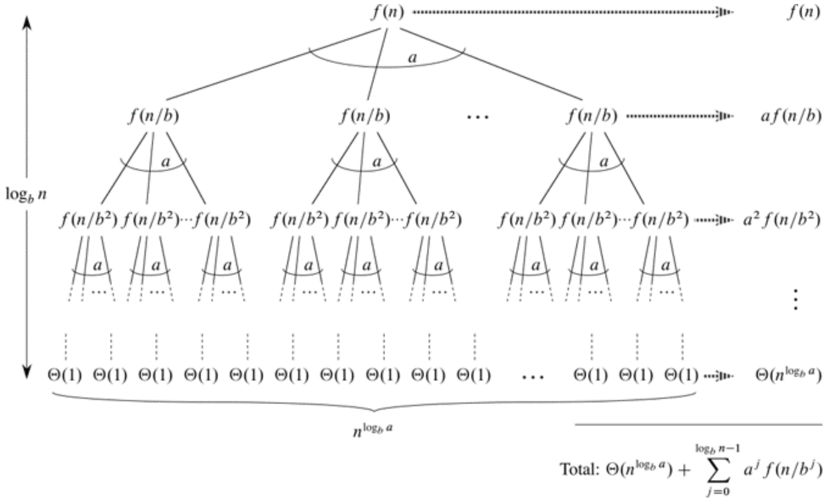
(1) If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

(2) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

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

Proof of theorem is in CLRS, Section 4.6.
Don't need to understand proof in order to apply the theorem.

The recursion tree generated by $T(n) = aT(n/b) + f(n)$



Proof for exact powers: the tree is a complete a-ary tree, has $n^{\log_b a}$ leaves, has height $\log_b n$, cost of nodes at each depth is shown to the right.

Intuition

- We compare the cost of the root with the cost in the leaves:
  - *Case 1 of Master theorem:*
    If cost of root $f(n)$ is polynomially smaller than cost in leaves, then total cost is dominated by cost in leaves.
  - *Case 2 of Master theorem:*
    If each level of tree has the same cost, then the total cost is (cost per level) x (height of tree).
  - *Case 3 of Master theorem:*
    If the cost of root is polynomially larger than cost of leaves, and a regularity condition is also satisfied, then the total cost of the tree is dominated by the cost of root.
    - Regularity condition is satisfied if $f(n)$ is a polynomial

Example (mergesort recurrence)

Suppose the running time $T(n)$ of an algorithm satisfies the recurrence

$$T(n) = \begin{cases} c', & \text{if } n = 1 \\ 2T(n/2) + cn, & \text{if } n > 1 \end{cases}$$

Find the asymptotic growth rate of T(n).

Solution:

- $a = 2, b = 2, f(n) = cn$
- cost of root $= cn$
- cost of leaves $=$ # of leaves $= n^{\log_b a} = n$
- cost of leaves $= \Theta$(cost of root).
- By case (2) of Master theorem, $T(n) = \Theta(n \log n)$

∎

5 Examples

Example

Give tight asymptotic bounds for the following recurrence, if possible, using the master theorem:

$$T(n) = 9T(n/3) + n$$

Solution:

- $a = 9, b = 3, f(n) = n$.
- cost of root $= n$
- cost of leaves $=$ #num leaves in recursion tree $=$ $n^{\log_b a} = n^{\log_3 9} = n^2$.
- cost of root is polynomially smaller than cost of leaves
- By case 1 of Master theorem, $T(n) = \Theta(n^2)$.

Example

Give tight asymptotic bounds for the following recurrence, if possible, using the master theorem:

$$T(n) = T(2n/3) + 1$$

Solution:

- $a = 1, b = 3/2, f(n) = 1.$
- cost of root $= 1$
- cost of leaves $= a^{\log_b n} = n^{\log_b a} = n^{\log_{3/2} 1} = 1.$
- (cost of root) and (cost of leaves) have same growth rate.
- By case 2 of Master theorem, $T(n) = \Theta(\log n)$.

Example

Give tight asymptotic bounds for the following recurrence, if possible, using the master theorem:

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

Solution:

- $a = 2, b = 2, f(n) = n \log n$
- cost of root $= n \log n$
- cost of leaves $= n^{\log_b a} = n^1 = n$.
- (cost of root) is larger than (cost of leaves), but not polynomially larger, ie $\frac{n \log n}{n} \not\geq n^{\epsilon}$ for any $\epsilon > 0$. So, the master theorem doesn't apply.

Example*

Give tight asymptotic bounds for the following recurrence, if possible, using the master theorem:

$$T(n) = 3T(n/4) + n \log n$$

Solution:

- $a = 3, b = 4, f(n) = n \log n$
- cost of root $= n \log n$
- cost of leaves $= n^{\log_b a} = n^{\log_4 3} = n^{1-\epsilon}$ for some $\epsilon > 0$.
- (cost of root) is polynomially larger than (cost of leaves) because $\frac{n \log n}{n^{1-\epsilon}} = n^{\epsilon} \log n \geq n^{\epsilon}$ for some $\epsilon > 0$. We are potentially in case 3.
- Check for regularity condition: Want to show $af(n/b) < cf(n)$ for some $c < 1$ and all sufficiently large n. $f(n) = n \log n$. So, want $a\frac{n}{b} \log \frac{n}{b} \leq cn \log n$, i.e. $\frac{3}{4} n \log \frac{n}{4} \leq cn \log n$ Take $c = 3/4$. By case 3 of Master theorem, $T(n) = \Theta(n \log n)$.

6 The substitution method for solving recurrences

The substitution method for solving recurrences

The substitution method comprises two steps:

1. Guess the solution
2. Use induction to find the constants and show the solution works.
   2.1 Use induction hypothesis to verify guess
   2.2 Check base case

Called "substitution method" because we substitute the guessed solution when applying the inductive hypothesis.

Example

Solve for $T(n)$ if it satisfies the recurrence

$$T(n) = \begin{cases} 1, & \text{if } n = 1 \\ 2T(n/2) + n, & \text{if } n > 1 \end{cases}$$

Solution:

1. *Guess:* $T(n) = n \log n + n$.
   Can obtain guess by drawing a recursion tree.
   Here, the given recurrence has an exact function, rather than asymptotic notation, and the solution is also exact.
2. *Proof by strong induction:*
   Basis step: $n = 1 \implies n \log n + n = 1 = T(n)$

Example

Inductive step:
Fix $n \geq 2$ and assume the assertion holds for all $k < n$. Then,

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\left(\frac{n}{2}\log\frac{n}{2} + \frac{n}{2}\right) + n \quad \text{(by inductive hypothesis)} \\ &= n\log n + n \end{aligned}$$

$\blacksquare$

In general, we'll use asymptotic notation, in both the recurrence relation and the solution.
For example: $T(n) = 2T(n/2) + \Theta(n)$ has the solution $T(n) = \Theta(n \log n)$.

Example: worst-case performance of quicksort

Solve the recurrence

$$T(n) = \begin{cases} 1, & \textit{if } n = 1 \\ T(n-1) + n, & \text{if } n \geq 2 \end{cases}$$

Solution:
Guess: Expand the recurrence to obtain

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= \cdots \\ &= T(1) + 2 + 3 + \cdots + n \\ &= n(n+1)/2 \end{aligned}$$

Inductive step: Can now prove that $T(n) = n(n+1)/2$ using induction (how?). Final answer: $T(n) = \Theta(n^2)$. ∎

Example: worst-case performance of merge sort*

Solve the recurrence

$$T(n) = \begin{cases} 1, & \textit{if } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n, & \text{if } n \geq 2 \end{cases}$$

Solution:
Guess: Can draw a recursion tree and obtain the guess $T(n) = O(n \log n)$.
Will prove the guess by induction.
Inductive step: Assume $T(k) \leq ck \log k, \forall k < n$
Then,

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor \log\lfloor n/2 \rfloor + n \quad \text{(by inductive hypothesis)} \\ &\leq 2c(n/2)\log(n/2) + n \\ &= cn \log n - cn + n \\ &\leq cn \log n, \text{ if c } \geq 1 \end{aligned}$$

Example: worst-case performance of merge sort*

Base case: (check boundary condition)

- Want $T(n) \leq cn \log n, \forall n \geq 2$,
- Know $T(2) = 4 \quad T(3) = 5$
- So want $4 = T(2) \leq c \cdot 2$ and $5 = T(3) \leq c \cdot 5 \log 5$
- Take $c = 2$.

We proved by induction that $T(n) \leq 2n \log n, \forall n \geq 2$.

Hence, $T(n) = O(n \log n)$. ∎

Acknowledgements and References

Many of the examples, images, and other content in the slides were taken (often verbatim) from the following sources:

- Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein, "Introduction to Algorithms, Third Edition", MIT Press, 2009