Master Theorem

Claire To

University of California, Irvine

CompSci 161: Discussion 4

Outline

Review

- Case analysis
- Asymptotic complexity

Master Method

- Recurrence relations
- Recursion tree
- Master theorem
- Examples

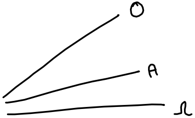
Case analysis

- Case analysis is NOT the same as asymptotic growth!
- **Worst case:** Always assume unless otherwise specified
- **Expected case:** Randomized algorithms, probability distributions
- **Best case:** Practically useless in theory

Example

Best case of searching for an element is the first element we check but that doesn't tell us anything insightful.
We want stronger performance guarantees!

Asymptotic complexity



1. Analyzing **algorithm** A
   - $\mathcal{O}$: A's growth never exceeds this upper bound (ceiling)
   - Θ: A's growth is tightly bounded above and below ($\mathcal{O} = \Omega$)
   - Ω: A's growth is always at least this lower bound (floor)

2. Analyzing **problem** P
   - $\mathcal{O}$: An algorithm exists to solve P
   - Θ: An optimal algorithm exists to solve P
   - Ω: Impossible to do better, fundamental theoretical limit

Example

- Sorting *problem*: $\Omega(n \log n)$ by decision tree argument
- Mergesort *algorithm*: $\Theta(n \log n)$ by master theorem!
  - Matches sorting lower bound - optimal :)
- Standard Quicksort *algorithm*:
  - Worst case: $\Theta(n^2)$ - not optimal
  - Expected case: $\Theta(n \log n)$ - optimal

medians-of-five subroutine achieves $\Theta(n \log n)$ worst case (+ high constant factors)

Recurrence relations

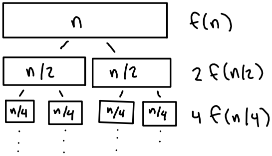
- Runtime of divide-and-conquer algorithms?
- Use master method to solve recurrence relations

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

1. $a = \#$ of recursive calls (subproblems)
2. $b =$ problem size reduction factor
3. $f(n) =$ work outside of recursive calls

Example

Mergesort: $T(n) = 2T(n/2) + n$

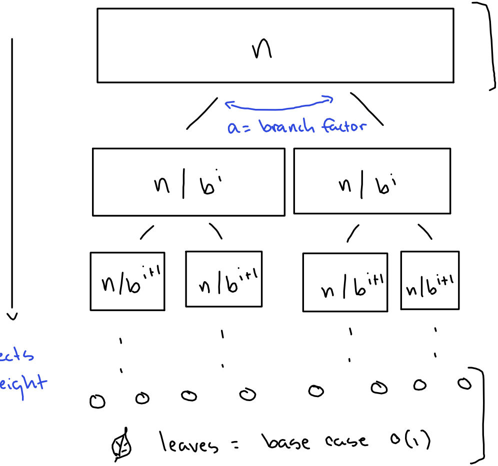


Recursion tree

i = recursion tree level

* tree models merge sort $2T(n/2)$

tree height $= \log_b n$



case 3

$f(n)$ work

a affects tree width

total work:

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i)$$

case 2

b affects tree height

case 1

$a^{\log_b n}$ leaves

$= n^{\log_b a}$ work

which part of the tree is larger? (dominating in work)

Master theorem

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

- Compare $f(n)$ against recursive work (function of a and b)
- 3 cases:
  1. If $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$ leaf work dominates
  2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$ summation
  3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$ top level work dominates

Examples

n

2n - size doubled!

bigger?

1. $T(n) = 4T(n/2) + n$

case 1! $f(n) = n \stackrel{?}{=} O(n^{[\log_4 2 = 2] - \varepsilon})$ yes for $\varepsilon = 1$ $\Rightarrow \boxed{T(n) = \Theta(n^2)}$

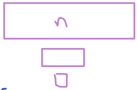
≈ equal

2. $T(n) = 2T(n/2) + n \log n$ merge sort tree -> each level has n elem + height = $\log_2 n$

case 2! $f(n) = n \log n \stackrel{?}{=} \Theta(n^{[\log_2 2 = 1]} \log^k n)$ yes for $k = 1$ $\Rightarrow \boxed{T(n) = \Theta(n \log^2 n)}$

bigger?

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



case 3! $f(n) = n \stackrel{?}{=} \Omega(n^{[\log_3 1 = 0] + \varepsilon})$ yes for $\varepsilon = 1$ $\Rightarrow \boxed{T(n) = \Theta(n)}$

4. $T(n) = 9T(n/3) + n^{2.5}$

case 3! $f(n) = n^{2.5} \stackrel{?}{=} \Omega(n^{[\log_3 9 = 2] + \varepsilon})$ yes for $\varepsilon = 0.5$ $\Rightarrow \boxed{T(n) = \Theta(n^{2.5})}$

5. $T(n) = 2T(n/2) + 1$
   Case 1: $f(n) = O(n^{1-\epsilon}) \rightarrow T(n) = \Theta(n)$
6. $T(n) = 2T(n/2) + n$
   Case 2: $f(n) = \Theta(n^1 \log^{k=0} n) \rightarrow T(n) = \Theta(n \log n)$
7. $T(n) = 2T(n/2) + n^2$
   Case 3: $f(n) = \Omega(n^{1+\epsilon}) \rightarrow T(n) = \Theta(n^2)$
8. $T(n) = 2T(n/4) + 1$
   Case 1: $f(n) = O(n^{1/2-\epsilon}) \rightarrow T(n) = \Theta(n^{1/2})$
9. $T(n) = 2T(n/4) + \sqrt{n}$
   Case 2: $f(n) = \Theta(n^{1/2} \log^0 n) \rightarrow T(n) = \Theta(n^{1/2} \log n)$
10. $T(n) = 2T(n/4) + n$
    Case 3: $f(n) = \Omega(n^{1/2+\epsilon}) \rightarrow T(n) = \Theta(n)$
11. $T(n) = 9T(n/3) + n$
    Case 1: $f(n) = O(n^{2-\epsilon}) \rightarrow T(n) = \Theta(n^2)$

12. $T(n) = T(2n/3) + 1$
Case 2: $f(n) = \Theta(n^0 \log^0 n) \to T(n) = \Theta(\log n)$
13. $T(n) = 3T(n/4) + n \log n$
Case 3: $f(n) = \Omega(n^{log_4 3+\epsilon}) \to T(n) = \Theta(n \log n)$
14. $T(n) = 2T(n/4) + n^2$
Case 3: $f(n) = \Omega(n^{1/2+\epsilon}) \to T(n) = \Theta(n^2)$
15. $T(n) = 2T(n/4) + n^4$
Case 3: $f(n) = \Omega(n^{1/2+\epsilon}) \to T(n) = \Theta(n^4)$
16. $T(n) = T(7n/10) + n$
Case 3: $f(n) = \Omega(n^{0+\epsilon}) \to T(n) = \Theta(n)$
17. $T(n) = 16T(n/4) + n^2$
Case 2: $f(n) = \Theta(n^2 \log^0 n) \to T(n) = \Theta(n^2 \log n)$
18. $T(n) = 7T(n/3) + n^2$
Case 3: $f(n) = \Omega(n^{log_3 7 \approx 1.8+\epsilon}) \to T(n) = \Theta(n^2)$
19. $T(n) = 7T(n/2) + n^2$
Case 1: $f(n) = O(n^{log_2 7 \approx 2.8-\epsilon}) \to T(n) = \Theta(n^{log_2 7})$