

CSC 301: Analysis of Algorithms

Lecture 8: The Master Theorem and Designing Divide and Conquer Algorithms

William Rebelsky
[rebelskyw]

Grinnell College

Wednesday 7th February, 2024

Reminders

- Homework 0 grades have been released. A 4 is required for an S. If you did not get a 5, please review which problems you got wrong. (Even if you did get a 5, please review any comments given). Regrade requests are open for a week.
- Homework 1 is due Tonight
- Homework 2 has been released on gradescope, it is due next Wednesday. Starting with this homework I expect typed submissions (L^AT_EX)
- Quiz 1 grades have been released, a 7 is required for an S. If you did not get all 9 correct, please look at which problems you got wrong. If you did not score a 7 or higher, I will be releasing a make up quiz on Friday that you may take for a token.

Table of Contents

- 1 Master Theorem
 - Master Theorem Definitions
 - Master Theorem Practice
 - When It Doesn't Apply

- 2 Designing Divide and Conquer Algorithms

Master Theorem

Only applies when the following are true:

- The recurrence is of the form $T(n) = aT(n/b) + f(n)$
- $a > 0$
- $b > 1$

Terms:

- $f(n)$ is a driving function
- $T(n)$ in this form is a master recurrence

Master Theorem

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

- a subproblems
- n/b size
- $f(n)$ time to divide and recombine

Master Theorem

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

- If there exists a constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b(a-\epsilon)})$ then $T(n) = \Theta(n^{\log_b(a)})$
- if there exists a constant $k \geq 0$ such that $f(n) = n^{\log_b(a)} \lg^k n$ then $T(n) = \Theta(n^{\log_b(a)} \lg^{k+1} n)$
- if there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b(a+\epsilon)})$ and if $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some $c < 1$, then $T(n) = \Theta(f(n))$

Master Theorem

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

- Case 1: $f(n) = O(n^{\log_b(a)})$ then $T(n) = \Theta(n^{\log_b(a)})$
(subproblems cost more than splitting and recombining)
- Case 2: if there exists a constant $k \geq 0$ such that
 $f(n) = n^{\log_b(a)} \lg^k n$ then $T(n) = \Theta(n^{\log_b(a)} \lg^{k+1} n)$
(subproblems and recombining similar cost)
- Case 3: $f(n) = \Omega(n^{\log_b(a)})$ then $T(n) = \Theta(f(n))$
(subproblems cost less than splitting and recombining)

Master Theorem Practice

- $T(n) = 9T(n/3) + n$
- $T(n) = T(2n/3) + 1$
- $T(n) = 3T(n/4) + n \lg n$
- $T(n) = 2T(n/2) + \Theta(n)$

Master Theorem Practice Answers

- $T(n) = 9T(n/3) + n$
 - Case 1: $\Theta(n^{\log_9(3)}) = \Theta(n^2)$
- $T(n) = T(2n/3) + 1$
 - Case 2: $\Theta(\lg n)$
- $T(n) = 3T(n/4) + n \lg n$
 - Case 3: $\Theta(n \lg n)$
- $T(n) = 2T(n/2) + \Theta(n)$
 - Case 2: $\Theta(n \lg n)$

When you can't use the Master Theorem

- Can't compare watershed and driving functions asymptotically
- There is a gap between cases 1 and 2 and cases 2 and 3
 - e.g. $T(n) = 2T(n/2) + \frac{n}{\lg n}$
- b is less than or equal to 1
- a is not greater than 0
- the function is in a different form

Table of Contents

- 1 Master Theorem
 - Master Theorem Definitions
 - Master Theorem Practice
 - When It Doesn't Apply

- 2 Designing Divide and Conquer Algorithms

Divide and Conquer

- We use Divide and Conquer to solve problems that can be simplified recursively
- It tends to let us reduce an n to a $\log n$ term. E.g.
 $\Theta(n^2) \rightarrow \Theta(n \log n)$
- Think of Merge Sort. Dividing the array into parts, sorting the parts, and re-merging is faster than sorting the entire array at once ($n \log n$ comparisons instead of n^2 comparisons)

Power of x

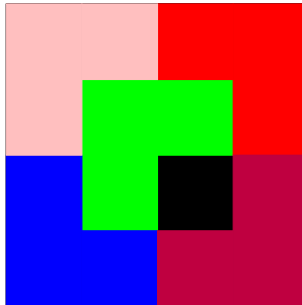
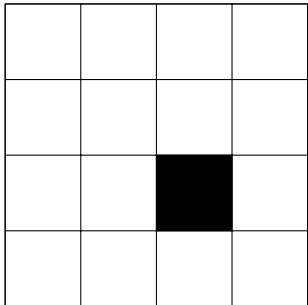
- Goal: given x and n , give the n th power of x , i.e. x^n
- Naive solution: for loop ($i=1:n$), answer times x . Runtime: $\Theta(n)$
- Divide and Conquer solution: Work in small groups, sketch algorithm, sketch runtime

Power of x Solution Sketch

- Inputs x, n
- Base case: $n=0$, return 1
- Recursive case: $\text{temp_power} = \text{power}(x, \lfloor n/2 \rfloor)$
- if n even, return temp_power^2
- if n odd, return $\text{temp_power}^2 * x$
- Runtime: $\log(n)$ (based on the number of recursive calls)

Tiling Problem

- Input: $n \times n$ grid (where n is a power of 2) with one missing cell
- Goal: Using an L shaped tile (2×2 with one of the corners missing), fill the entire grid.
- From Geeks for Geeks



Tiling Solution Sketch

- Base case: 2×2 grid with one missing tile. Can just fill it with the L
- Recursive case: Add a tile to the center, covering the 3 sub squares that aren't missing a tile