Divide and Conquer Algorithms

CPSC 320 2024W1

Definitions

A divide and conquer algorithm proceeds by…

- Dividing the input into two or more **smaller instances** of the same problems – we call these **subproblems**
- Solving the subproblems recursively
- Combining the subproblem solutions to obtain a solution to the original problem
- We'll also consider **prune and search** algorithms, which look for element(s) with a specific property and only recurse on one subproblem

Examples

Some divide and conquer algorithms you are already familiar with:

- QuickSort, MergeSort ("classic" divide and conquer)
- Binary search (prune and search)

Recurrence relations

- The running time **T(n)** of a recursive function can be described using a **recurrence relation:**
  - **T(n)** is defined in terms of one or more terms of the form **T(something smaller than n)**
  - Example:

One recursive call on n/2 items.

Two recursive calls on n/4 items.

n^2 work not done inside a recursive call

$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Clicker Question

Binary-Search(v, A[1...n])
if n ≥ 1
let v' = A[n/2]
if v < v' then Binary-Search(v, A[1..n/2])
elseif v > v' " " " (v, A[n/2+1,..n])
else done
else [base case]

We define a recurrence by:

T(n) = T(size of recursive call 1) + T(size of recursive call 2) + … + [work done outside of recursive calls

Which of the following functions best describes the worst-case runtime of **binary search?**

$T(n) =$

A. $T(n/2) + \log n$

B. $T(n/2) + 1$ work not counting recursive call

C. $T(n/2) + T(\log n) + 1$

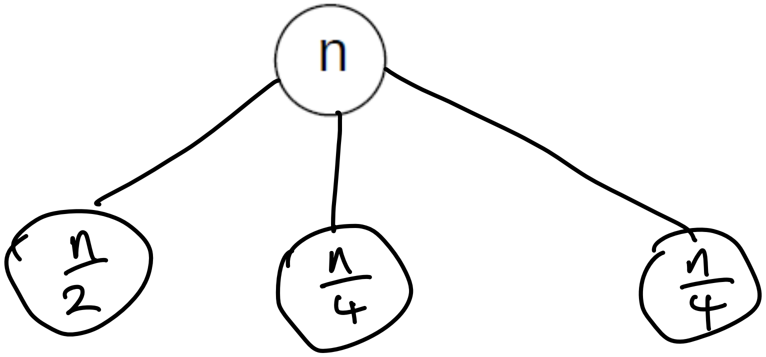
D. $\cancel{2T(n/2)} + 1$

Recursion Trees

- One way to solve a recurrence relation is to draw a **recursion tree**
  - Represent the recursion with a tree where each node represents a recursive subproblem
  - Inside each node, write the size of the subproblem this call to the function solves
  - Next to each node, write the amount of work done by the call to the function, **not including** any time spent in recursive calls
  - Compute the total amount of non-recursive work done on each row
  - Then add up the work done over all rows

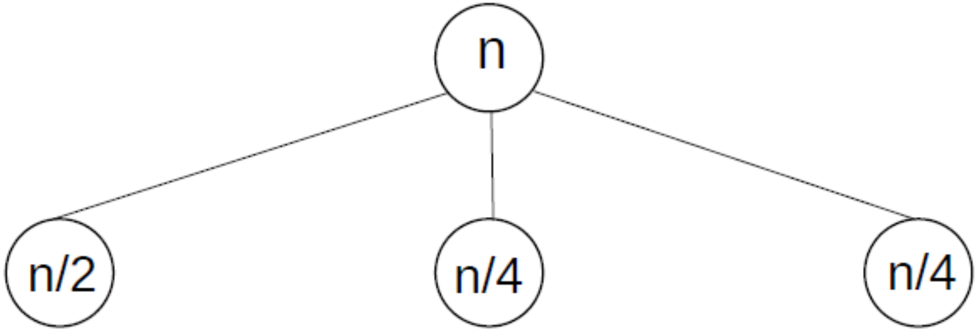
$$T(n)=\begin{cases} T(\underline{n/2})+\underline{2}\,T(\underline{n/4})+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: drawing the tree



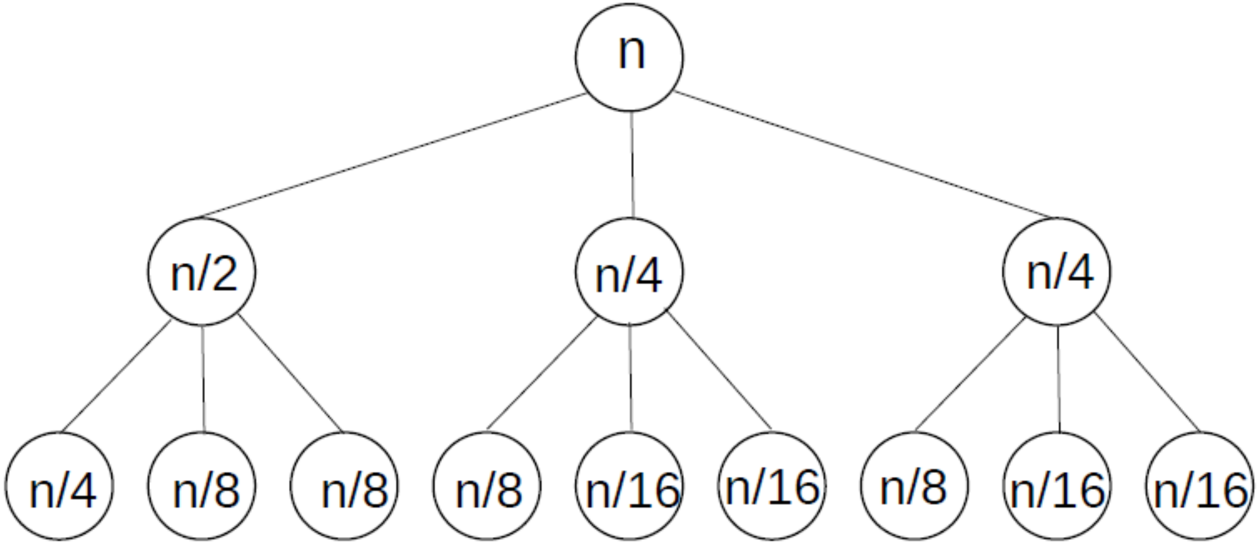
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: drawing the tree (continued)



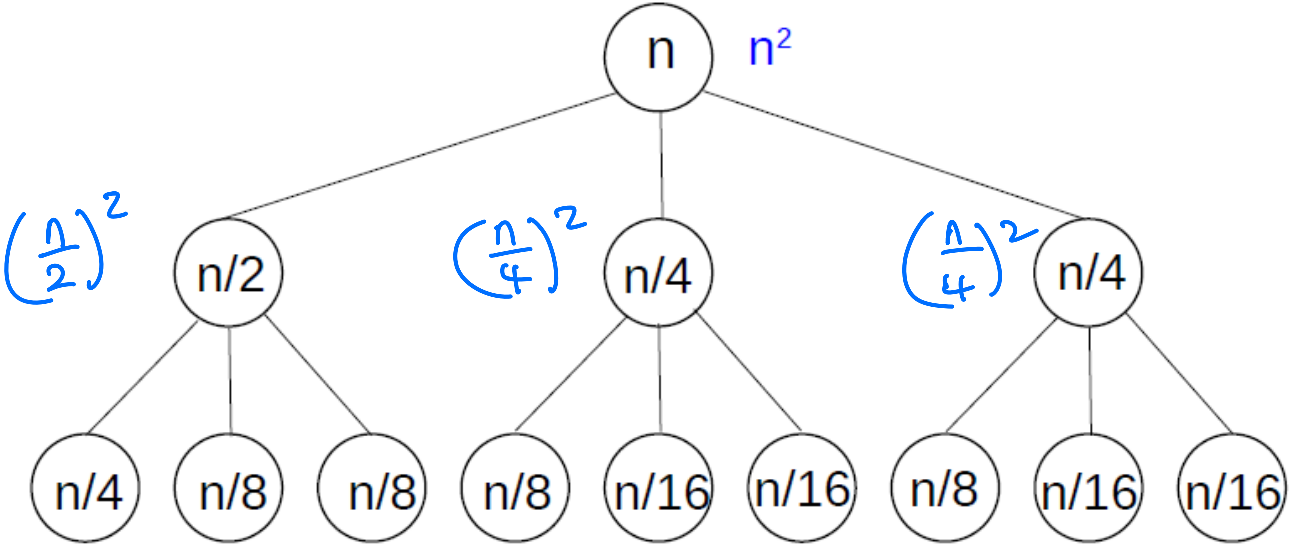
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & if\ n\geq 4 \\ \Theta(1) & if\ n\leq 3 \end{cases}$$

Example: drawing the tree (continued)



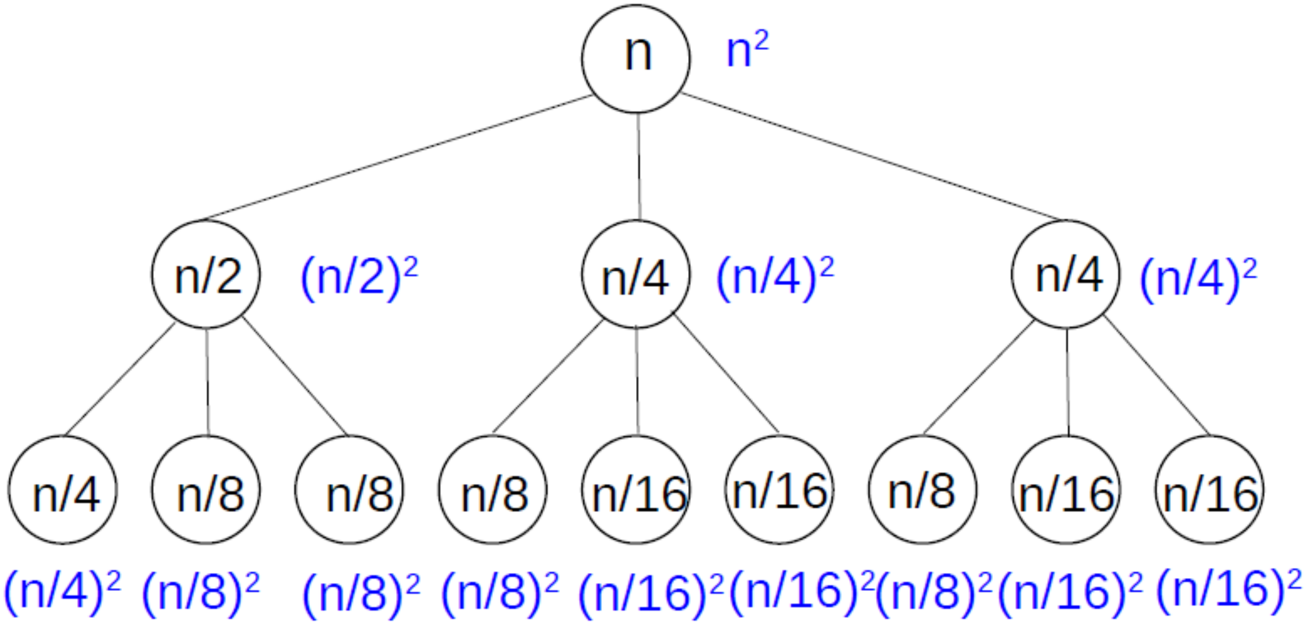
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & if\ n\geq 4 \\ \Theta(1) & if\ n\leq 3 \end{cases}$$

Example: work done at each node



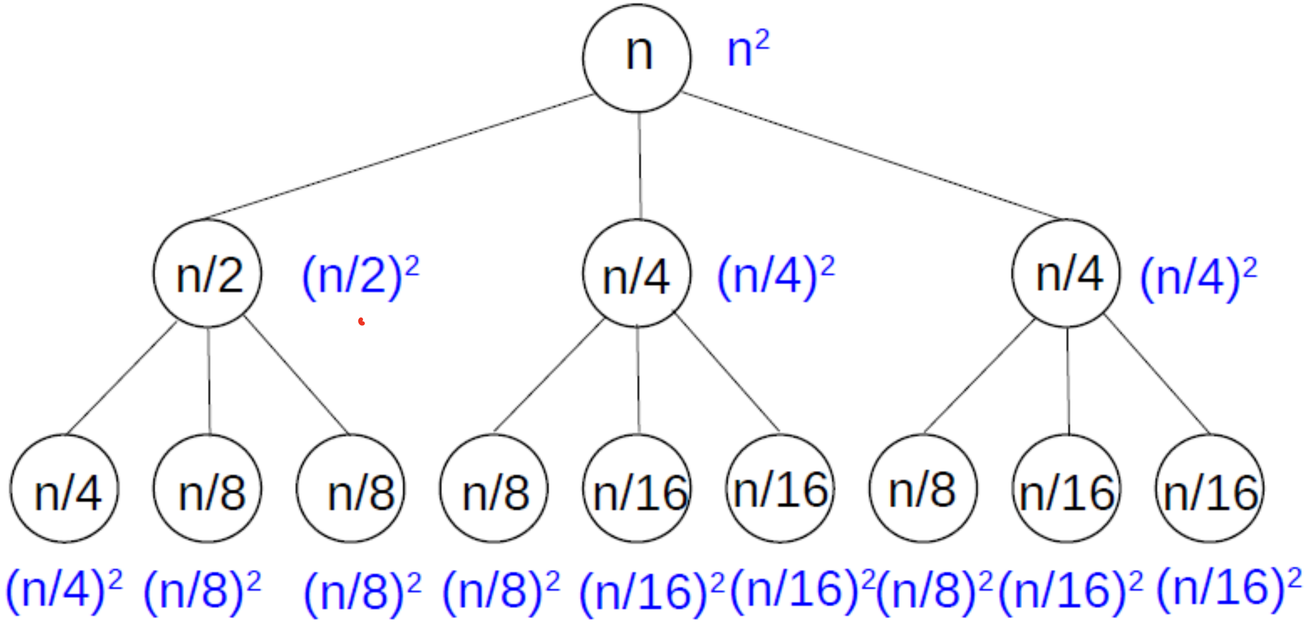
$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

Example: work done at each node (continued)



$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: work done on each row



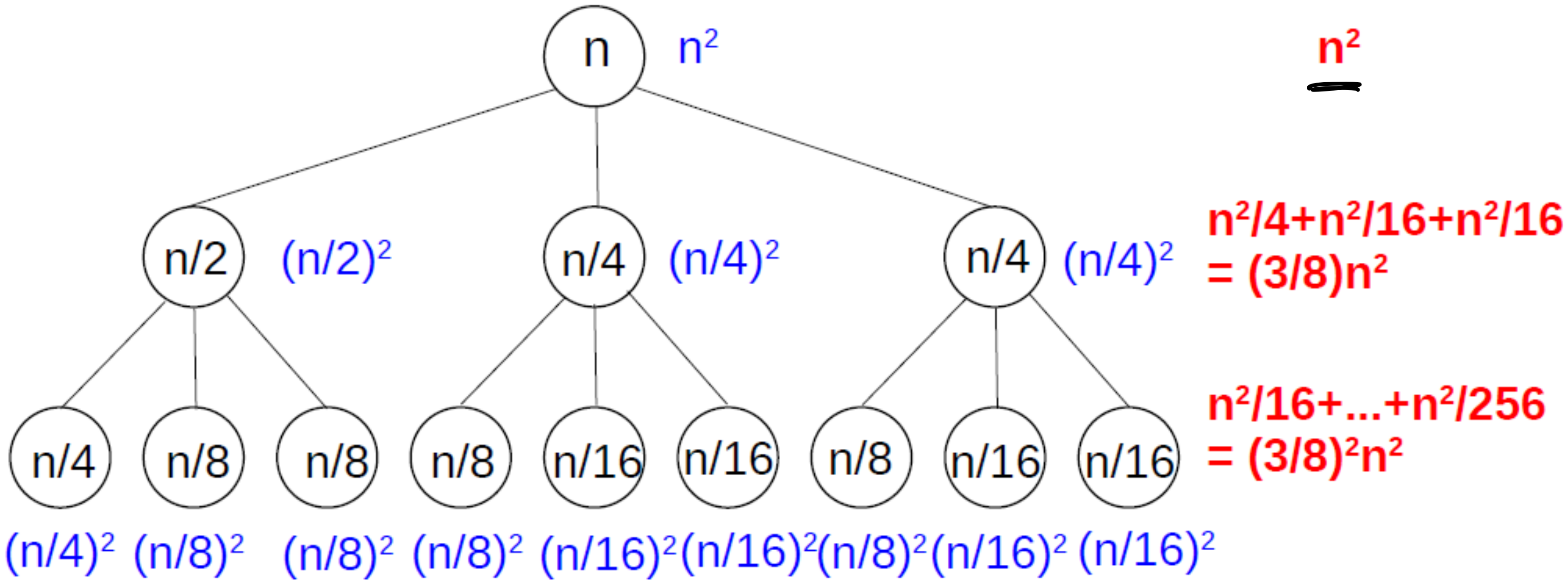
n^2

$n^2\left(\frac{1}{4}+\frac{1}{16}+\frac{1}{16}\right)$

$= n^2\left(3/8\right)$

$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: work done on each row



$$T(n) = \begin{cases} T(n/2) + 2T(n/4) + n^2 & \text{if } n \geq 4 \\ \Theta(1) & \text{if } n \leq 3 \end{cases}$$

- Example: summing up the work on all the rows
  - The total work is $n^2 + (3/8)n^2 + (3/8)^2 n^2 + \ldots$

$$= n^2 \left(1 + (3/8) + (3/8)^2 + \ldots \right)$$

$$= n^2 \cdot \frac{1}{1 - 3/8} = n^2 \cdot \frac{8}{5} = O(n^2)$$

implying deepest leaf has infinite depth

Lower bound is $\Omega(n^2)$, work at root.

Geometric Sums. Here are useful formulas, when summing up runtimes over levels of a recurrence tree. A *geometric sum* has the form $\sum_{i=0}^{n} x^i$, where $x > 0$. Note that when $x = 1$, $\sum_{i=0}^{n} x^i = n$. When $x \neq 1$ we have that:

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1} = \frac{1 - x^{n+1}}{1 - x},$$

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x}, \text{ if } 0 < x < 1.$$

$$T(n)=\begin{cases} T(n/2)+2T(n/4)+n^2 & \text{if } n\geq 4 \\ \Theta(1) & \text{if } n\leq 3 \end{cases}$$

- Example: summing up the work on all the rows
  - The total work is $n^2 + (3/8)n^2 + (3/8)^2n^2 + \ldots$
  - This is a geometric series, and $3/8 < 1$.
  - So the sum converges to $\frac{1}{1-3/8}n^2$
  - Hence $T(n)\in\Theta(n^2)$

The Master Theorem

- Most divide and conquer algorithms split the input into equal-size subproblems
- Most recursion trees fall into one of three categories:
  - The work per level increases geometrically
  - The work per level is constant (e.g., MergeSort)
  - The work per level decreases geometrically (e.g., the previous example)

The Master Theorem Corollary

Theorem: Let $T : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be defined by

$$T(n) = \begin{cases} aT(\underline{n/b}) + cn^k, & \text{for } n \geq n_0, \\ c, & \text{for } n < n_0, \end{cases}$$

where $a > 0$, $b > 1$, $c > 0$, and $k \geq 0$ are constants.

1 • If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$.

2 • If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.

3 • If $a < b^k$, then $T(n) = \Theta(n^k)$.

This version will be sufficient for assignment problems and exams in this course.

Example: $T(n) = \underline{3}T(n/\underline{2}) + n$

(runtime of integer multiplication, see text)

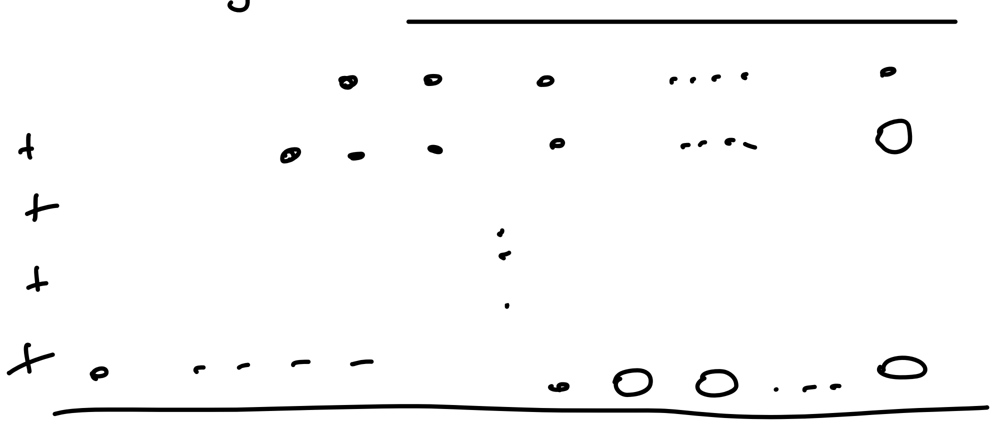
$a = 3$, $b = 2$, $k = 1$. $b^k = 2$

Case 1. $T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.5\ldots})$

Old-fashioned multiplication: $x \times y$

$$x = x_n \, x_{n-1} \, \cdots \, x_1$$

$$\times \; y = y_n \, y_{n-1} \, \cdots \, y_1$$

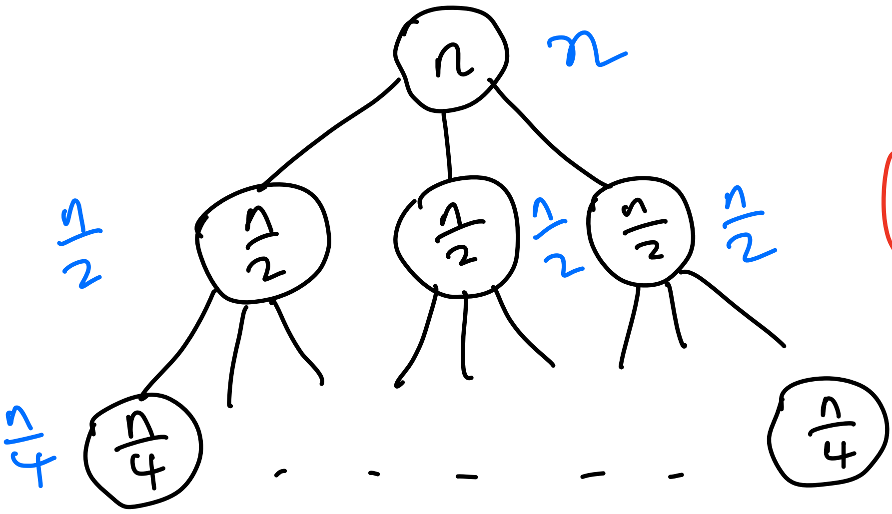


$[x_n x_{n-1} \ldots x_1 \times y_1]$

$[x_n x_{n-1} \ldots x_n \times y_2]$

$\Theta(n^2)$ algorithm

Recursion tree for $T(n) = 3T\left(\frac{n}{2}\right) + n$



$$n$$
$$+$$
$$\left(\frac{a}{b^k}\right)n = \frac{3}{2}n$$
$$+$$
$$\left(\frac{a}{b^k}\right)^2 n = \frac{9}{4}n = \left(\frac{3}{2}\right)^2 n$$

$$n\left[\underbrace{1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \cdots \left(\frac{3}{2}\right)^{\log_2 n}}_{}\right]$$

$$[\text{check}] = \Theta(n^{\log_2 3})$$

What about

$$T(n) = 3T(n/2) + \underline{n^2} \quad ?$$

$$T(n) = \begin{cases} aT(n/b) + cn^k, & \text{for } n \geq n_0, \\ c, & \text{for } n < n_0, \end{cases}$$

where $a > 0$, $b > 1$, $c > 0$, and $k \geq 0$ are constants.

1 • If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$.
2 • If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.
3 • If $a < b^k$, then $T(n) = \Theta(n^k)$.

$a = 3$, $b = 2$, $k = 2$ $\quad b^k = 4$

We're in case 3!

So $T(n) = \Theta(n^2)$

Another example of case 2: Binary Search:

$T(n) = T\left(\frac{n}{2}\right) + 1$

$a = 1$, $b = 2$, $k = 0$. So $T(n) = \Theta(n^0 \log n)$
$= \Theta(\log n)$

The Master Theorem Corollary

Theorem: Let $T : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be defined by

$$T(n) = \begin{cases} aT(n/b) + cn^k, & \text{for } n \geq n_0, \\ c, & \text{for } n < n_0, \end{cases}$$

where $a > 0$, $b > 1$, $c > 0$, and $k \geq 0$ are constants.

- If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$.
- If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.
- If $a < b^k$, then $T(n) = \Theta(n^k)$.

This version will be sufficient for assignment problems and exams in this course.

Mergesort: $T(n) = 2T(\frac{n}{2}) + c \cdot n$

$a = 2$, $b = 2$, $k = 1$. $a = b^k$.

$T(n) = \Theta(n \log n)$

The Master Theorem [Bentley, Haken, Saxe]

- Theorem: Let $a \geq 1$, $b > 1$ be real constants, let $f: N \to R^+$, and let $T(n)$ be defined by:

nonrecursive part.

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n \geq n_0 \\ \Theta(1) & \text{if } n < n_0 \end{cases}$$

where n/b might be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $af(n/b) < \delta f(n)$ for some $0 < \delta < 1$ and all n large enough, then $T(n) \in \Theta(f(n))$.

regularity condition

First five worksheet recurrences: for which ones are we **not able** to use the Master Theorem? Select all that apply.

A. $T(n) = 5T(\sqrt{n}) + n$ — $a = 5$, $f(n) = n$ No constant b ?!

✓B. $T(n) = T(n/2) + 1$ — binary search!

✓C. $T(n) = T(n/4) + n$ — $a = 1$, $b = 4$ $f(n) = n$

✓D. $T(n) = 3T(n/9) + \sqrt{n}\log_2 n$

E. $T(n) = \sqrt{n}T(n/3) + n^2$ — no constant a

Z. $T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + cn$ — Problems have different size.

The Master Theorem [Bentley, Haken, Saxe]

- Theorem: Let $a \geq 1$, $b > 1$ be real constants, let $f: N \rightarrow R^+$, and let $T(n)$ be defined by:

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n \geq n_0 \\ \Theta(1) & \text{if } n < n_0 \end{cases}$$

where n/b might be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then

1. If $f(n) \in O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If $f(n) \in \Theta(n^{\log_b a} \log^k n)$ for some $k \geq 0$ then $T(n) \in \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and $af(n/b) < \delta f(n)$ for some $0 < \delta < 1$ and all n large enough, then $T(n) \in \Theta(f(n))$.

regularity condition

The Master Theorem [Bentley, Haken, and Saxe]

How to apply the theorem:

- Compute $\log_b a$
- Compare it to the exponent of **n** in **f(n)**
  - If $\log_b a$ is larger: case 1.
  - If they are equal: maybe case 2.
  - If $\log_b a$ is smaller: check regularity condition, and if it holds then case 3.

First five worksheet recurrences: for which ones are we **not able** to use the Master Theorem? Select all that apply.

A. $T(n) = 5T(\sqrt{n}) + n$
B. $T(n) = T(n/2) + 1$
C. $T(n) = T(n/4) + n$
D. $T(n) = 3T(n/9) + \sqrt{n}\log_2 n$
E. $T(n) = \sqrt{n}T(n/3) + n^2$