Algorithms and Data Structures

Lec04

Solving recurrences

Dr. Mohammad Ahmad

Divide and Conquer

- Recursive in structure
  - *Divide* the problem into sub-problems that are similar to the original but smaller in size
  - *Conquer* the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
  - *Combine* the solutions to create a solution to the original problem

An Example: Merge Sort

***Sorting Problem*:** Sort a sequence of n elements into non-decreasing order.

- ***Divide*:** Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each
- ***Conquer:*** Sort the two subsequences recursively using merge sort.
- ***Combine*:** Merge the two sorted subsequences to produce the sorted answer.

Merge-Sort (A, p, r)

INPUT: a sequence of n numbers stored in array A

OUTPUT: an ordered sequence of n numbers

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
if p < r
    then q ← ⌊(p+r)/2⌋
        MergeSort (A, p, q)
        MergeSort (A, q+1, r)
        Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)

Analysis of Merge Sort

- Running time $T(n)$ of Merge Sort:
- Divide: computing the middle takes $\Theta(1)$
- Conquer: solving 2 sub-problems takes $2T(n/2)$
- Combine: merging n elements takes $\Theta(n)$
- Total:

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

$$\Rightarrow T(n) = \Theta(n \lg n)$$

Recursion-tree Method

- Recursion Trees
  - Show successive expansions of recurrences using trees.
  - Keep track of the time spent on the sub problems of a divide and conquer algorithm.

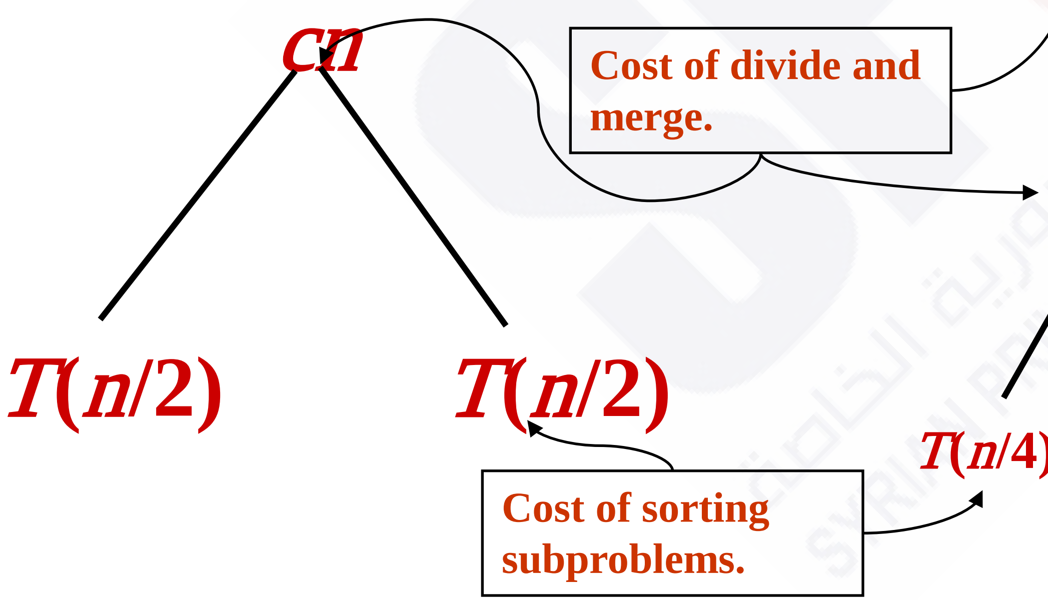
Recursion-tree method

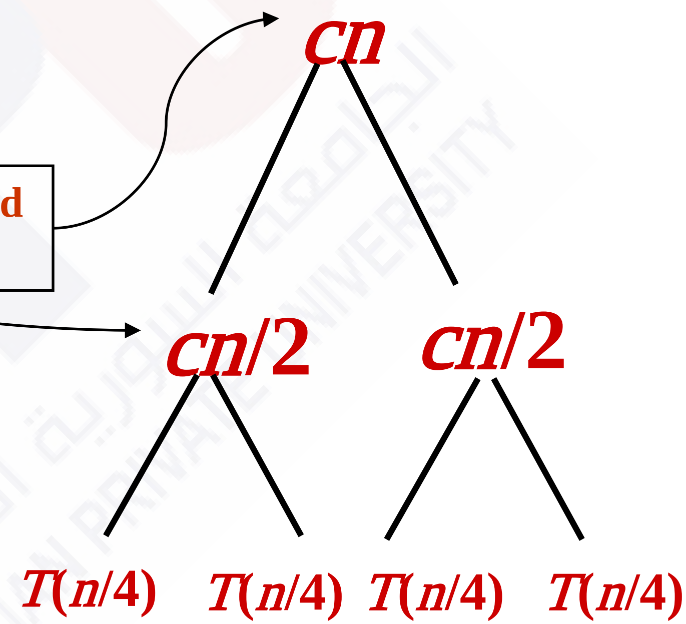
- A recursion tree models the costs (time) of a recursive execution of an algorithm.
- The recursion tree method is good for generating guesses for the substitution method.
- The recursion-tree method can be unreliable.
- The recursion-tree method promotes intuition, however.

Recursion Tree for Merge Sort

For the original problem, we have a cost of cn, plus two sub-problems each of size $(n/2)$ and running time $T(n/2)$.

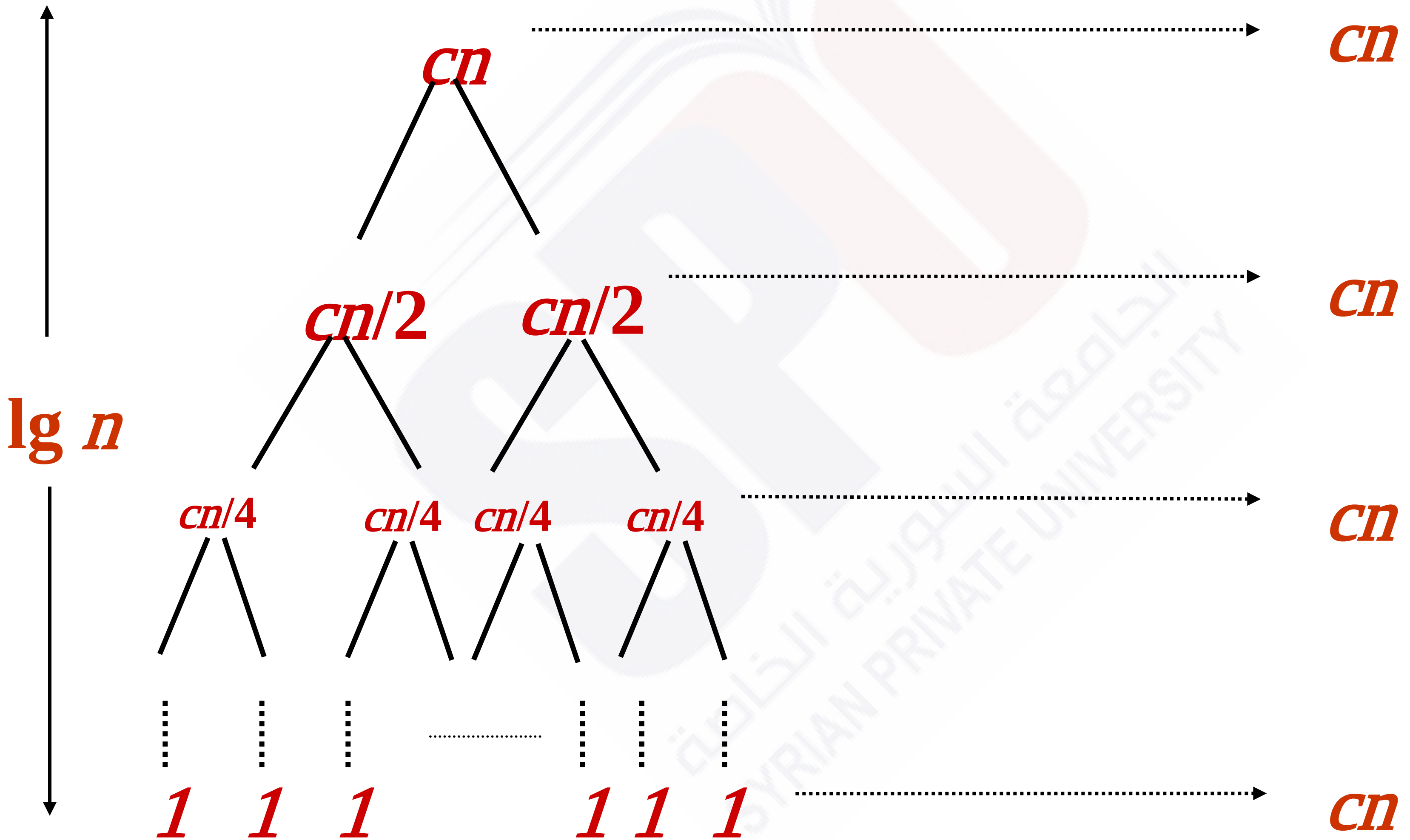
Each of the size $n/2$ problems has a cost of $cn/2$ plus two sub-problems, each costing $T(n/4)$.





Recursion Tree for Merge Sort

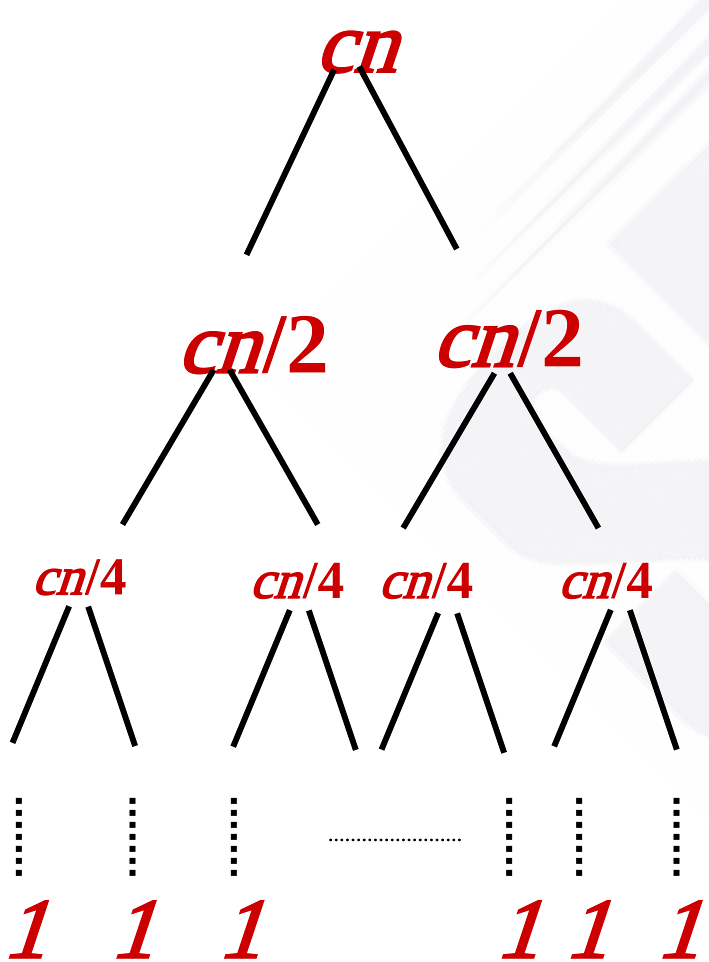
Continue expanding until the problem size reduces to 1.



Total : $cn \lg n + cn$

Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



- **Each level has total cost *cn*.**
- **Each time we go down one level, the number of sub-problems doubles, but the cost per sub-problem halves $\Rightarrow$ *cost per level remains the same*.**
- **There are $\lg n + 1$ levels, height is $\lg n$.**
- **Total cost = sum of costs at each level = $(\lg n + 1)cn = cn\lg n + cn = \Theta(n \lg n)$.**

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$T(n)$$

Example of recursion tree

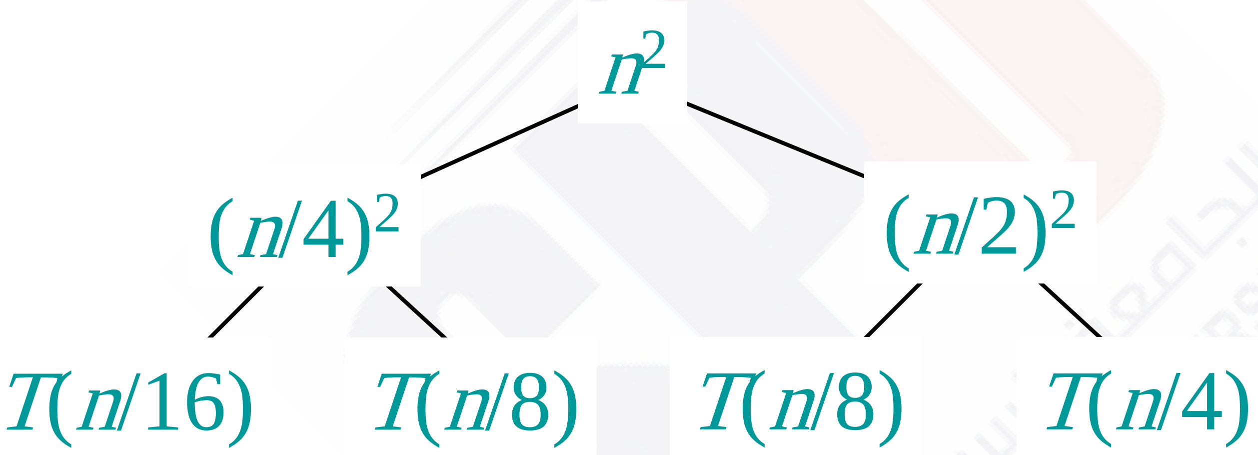
Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2$$

$T(n/4)$ $\qquad\qquad$ $T(n/2)$

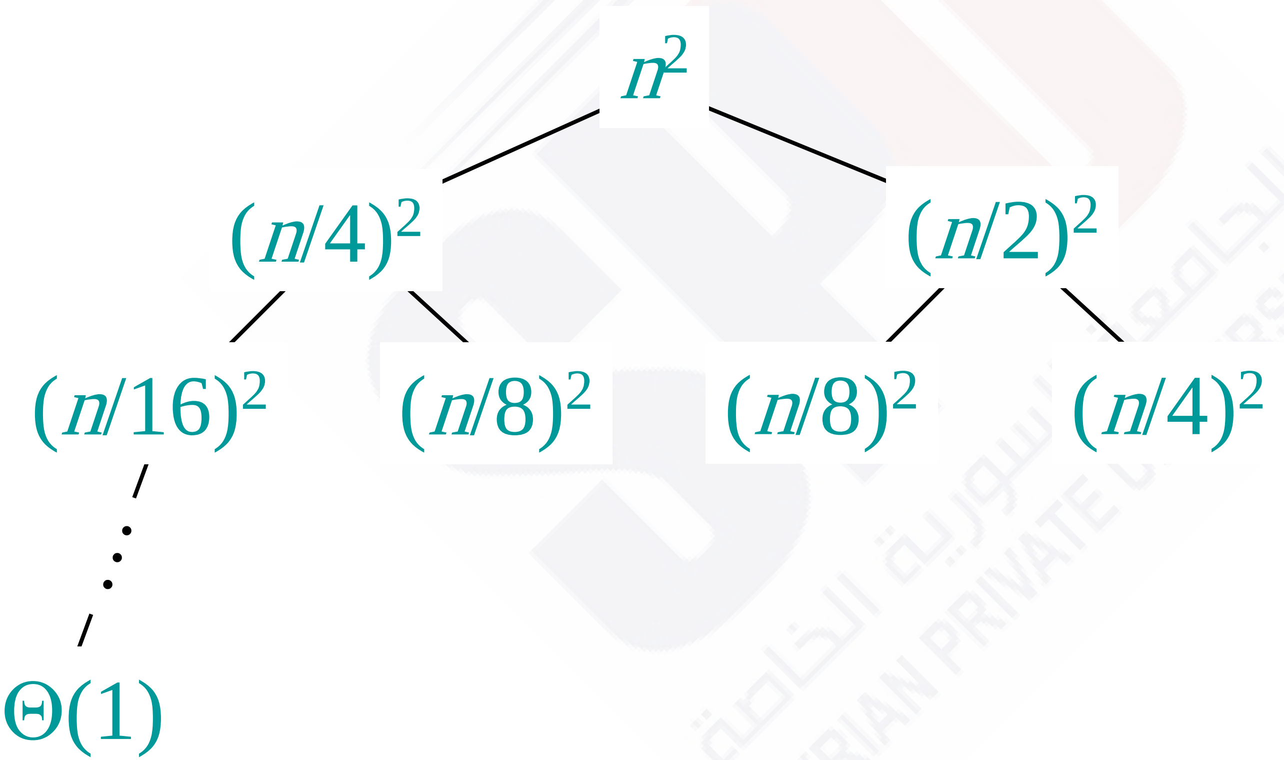
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$$n^2 \quad \cdots\cdots \quad n^2$$

$$(n/4)^2 \qquad (n/2)^2$$

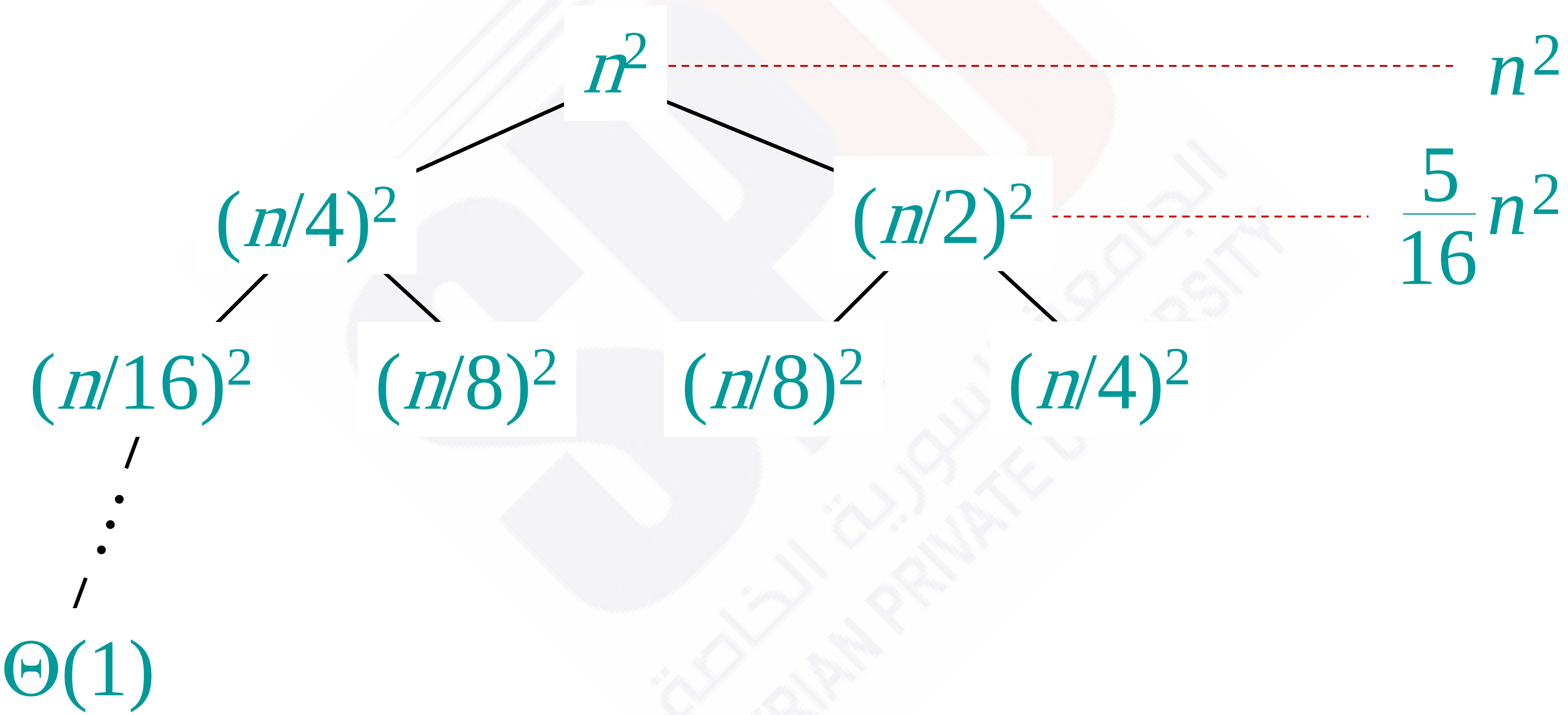
$$(n/16)^2 \quad (n/8)^2 \quad (n/8)^2 \quad (n/4)^2$$

$$\vdots$$

$$\Theta(1)$$

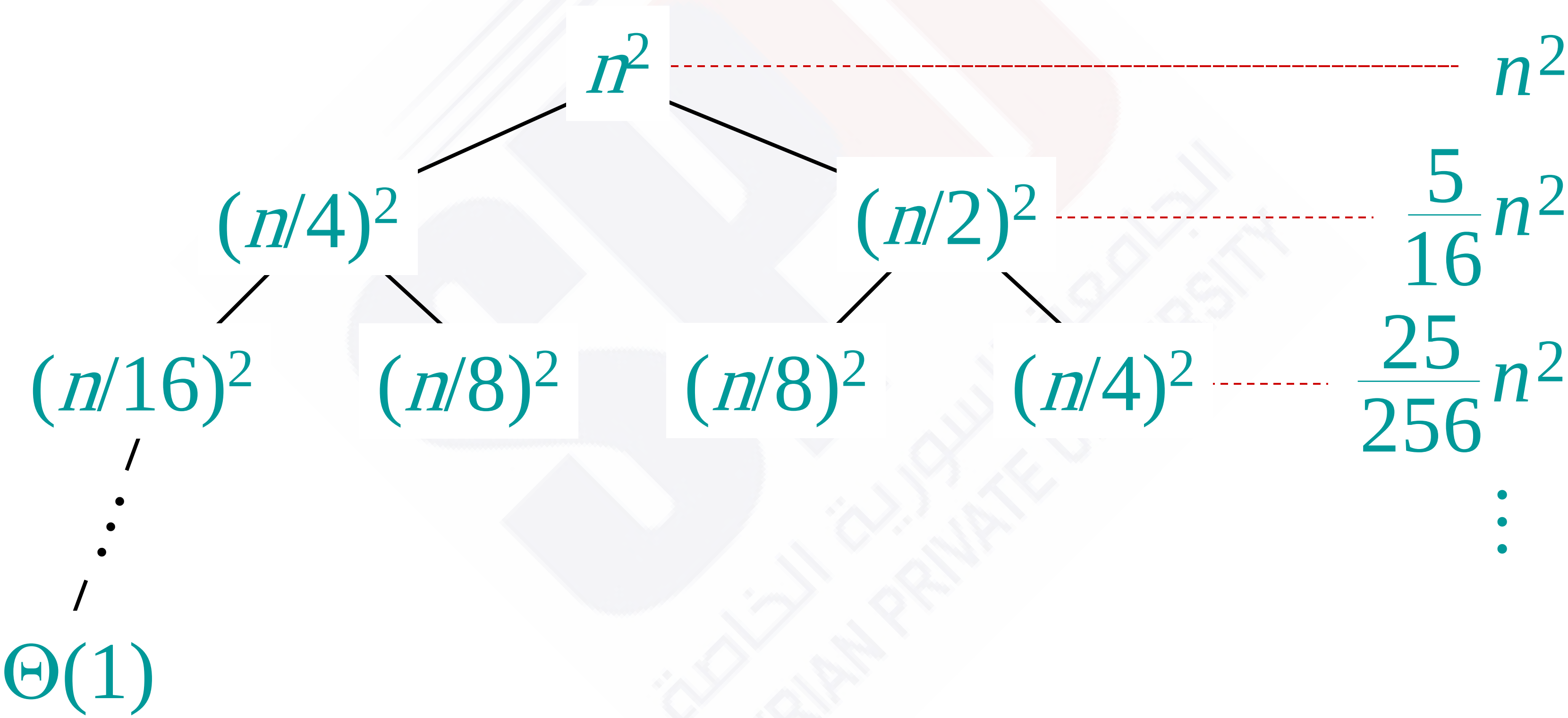
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



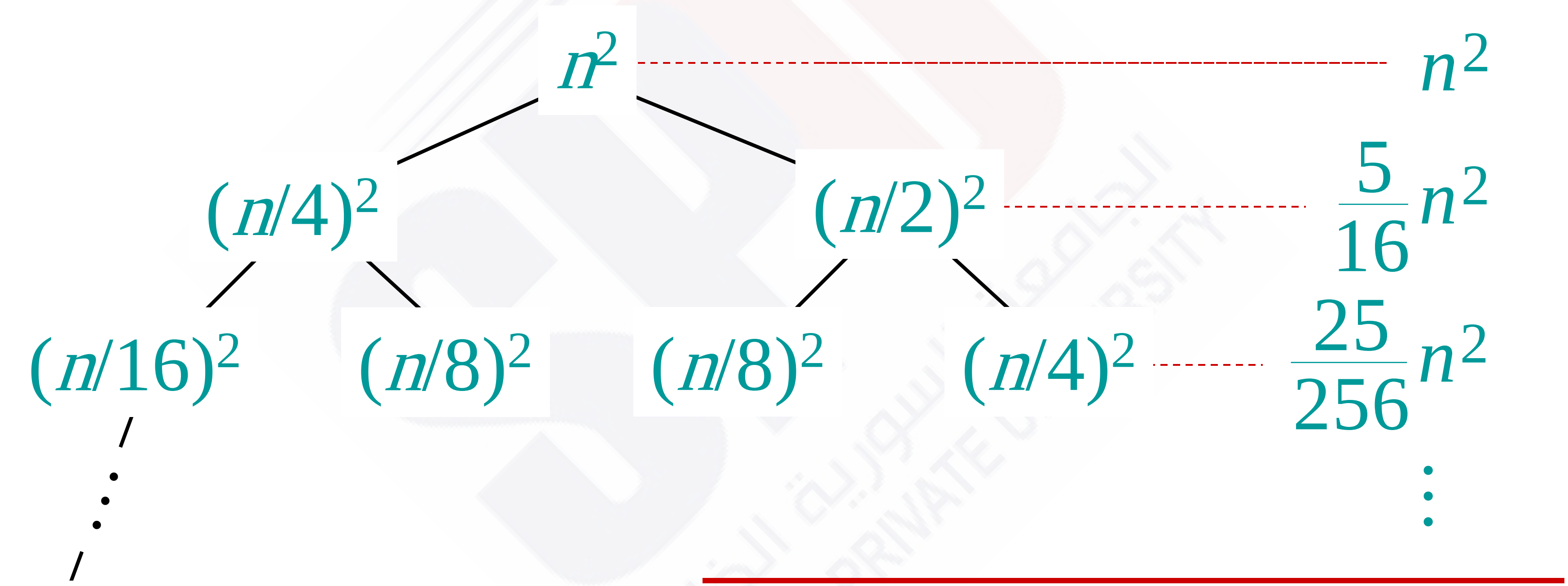
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



$\Theta(1)$

$$\text{Total} = n^2\left(1 + \frac{5}{16} + \left(\frac{5}{16}\right)^2 + \left(\frac{5}{16}\right)^3 + \cdots\right)$$

$$= \Theta(n^2)$$ *geometric series*

Geometric series

$$1 + x + x^2 + \cdots + x^n = \frac{1 - x^{n+1}}{1 - x} \quad \text{for } x \neq 1$$

$$1 + x + x^2 + \cdots = \frac{1}{1 - x} \quad \text{for } |x| < 1$$

The master method

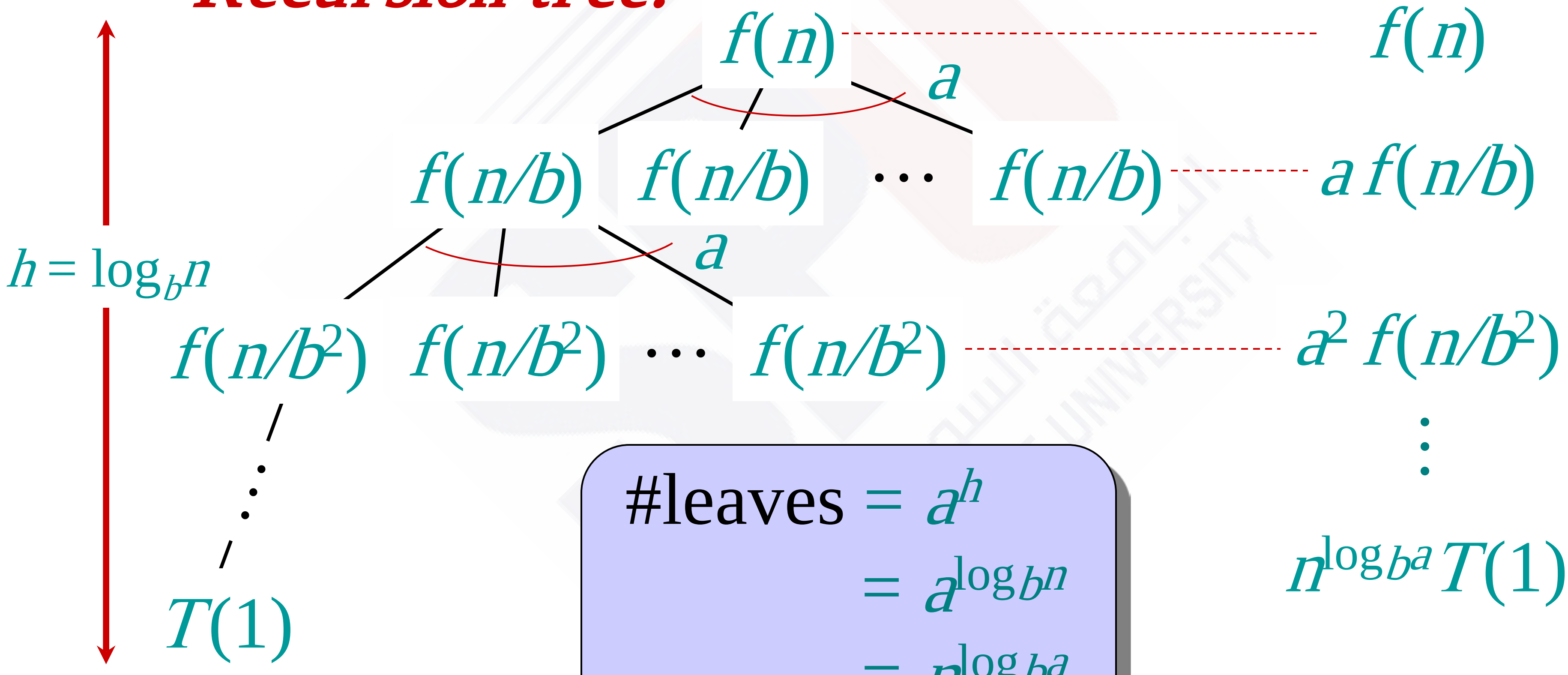
The master method applies to recurrences of the form

$$T(n) = a\,T(n/b) + f(n)\ ,$$

where $a \geq 1$, $b > 1$, and f is asymptotically positive.

Idea of master theorem

Recursion tree:



$$\#\text{leaves} = a^h = a^{\log_b n} = n^{\log_b a}$$

Three common cases

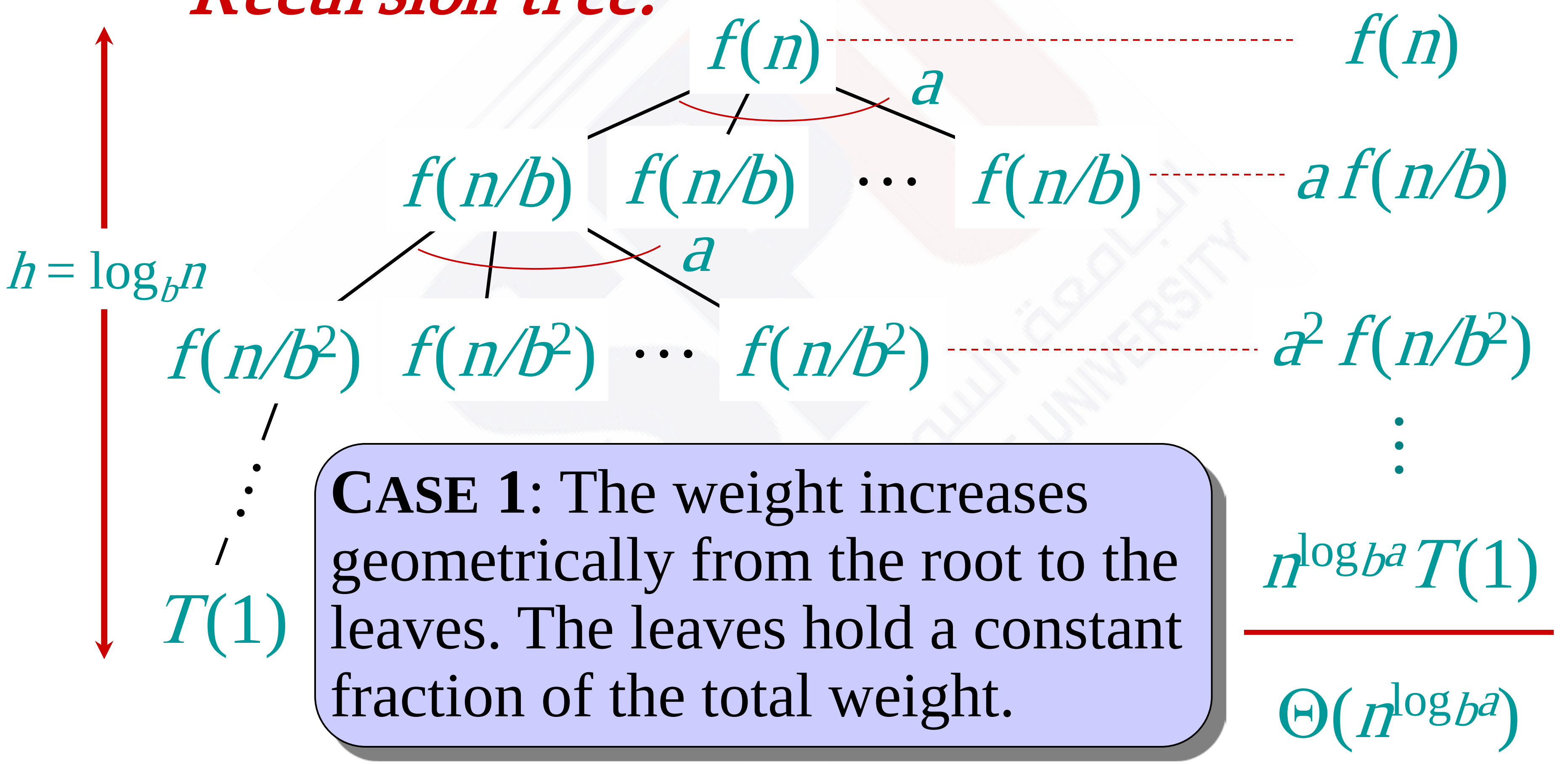
Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an n^{ε} factor).

   Solution: $T(n) = \Theta(n^{\log_b a})$.

Idea of master theorem

Recursion tree:



CASE 1: The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.

Three common cases

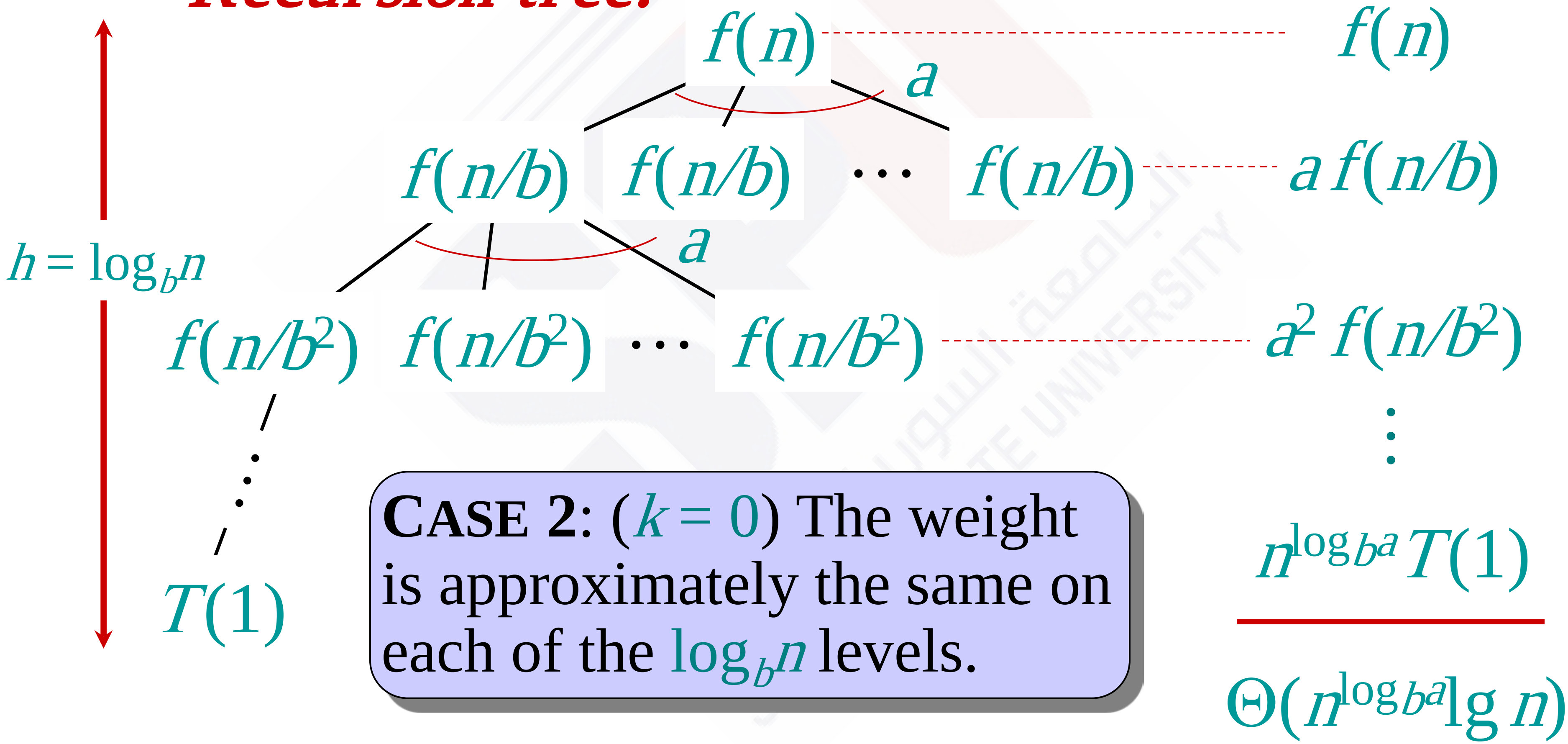
Compare $f(n)$ with $n^{\log_b a}$:

2. $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for some constant $k \geq 0$.
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.

Idea of master theorem

Recursion tree:



CASE 2: ($k = 0$) The weight is approximately the same on each of the $\log_b n$ levels.

Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

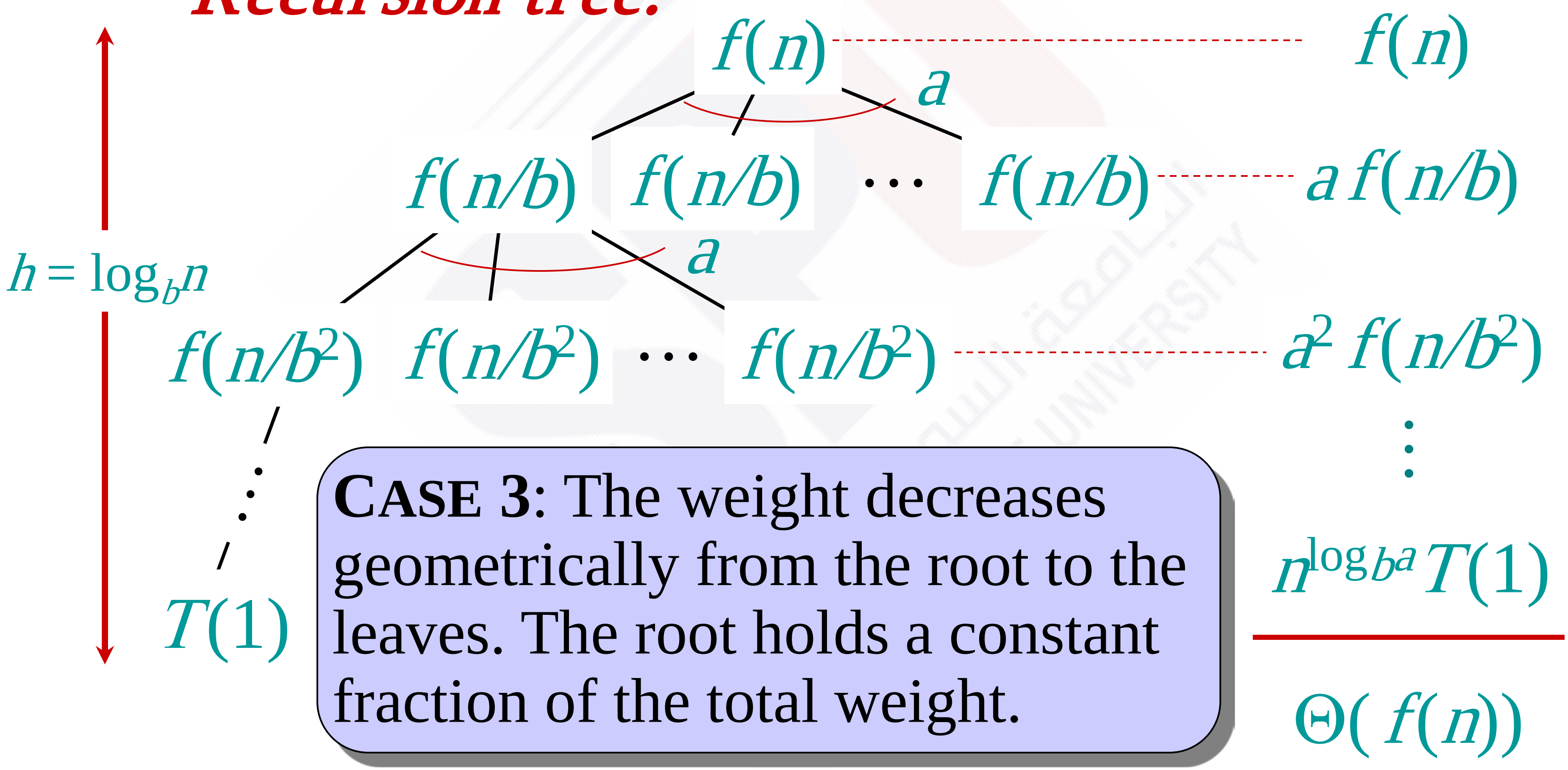
3. $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an n^{ε} factor),

and $f(n)$ satisfies the ***regularity condition*** that $a\, f(n/b) \le c\, f(n)$ for some constant $c < 1$.

Solution: $T(n) = \Theta(\, f(n))$.

Idea of master theorem

Recursion tree:



CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.

Examples

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

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n.$

CASE 1: $f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$.

$\therefore\ T(n) = \Theta(n^2)$.

Ex. $T(n) = 4T(n/2) + n^2$

$a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^2;\ f(n) = n^2.$

CASE 2: $f(n) = \Theta(n^2 \lg^0 n)$, that is, $k = 0$.

$\therefore\ T(n) = \Theta(n^2 \lg n)$.

Examples

Ex. $T(n) = 4T(n/2) + n^3$

$a = 4$, $b = 2 \Rightarrow n^{\log_b a} = n^2$; $f(n) = n^3$.

CASE 3: $f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4(cn/2)^3 \le cn^3$ (reg. cond.) for $c = 1/2$.

$\therefore\ T(n) = \Theta(n^3)$.