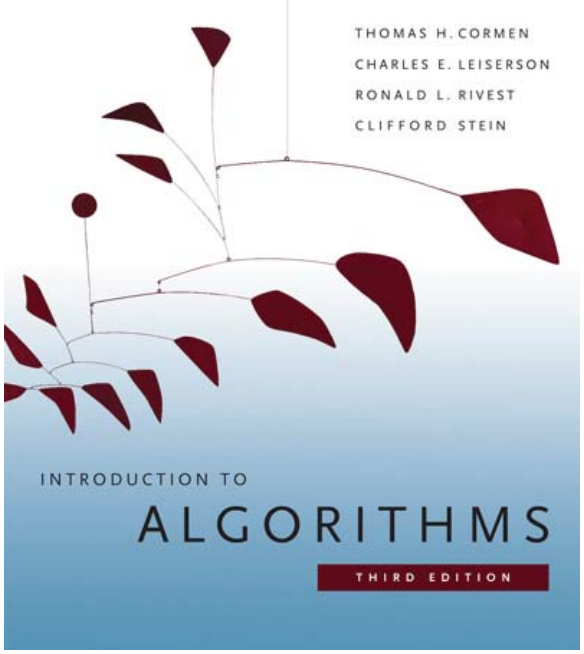
# 6.006- *Introduction to Algorithms*




Courtesy of MIT Press. Used with permission.

# *Lecture 3*

# Menu

- Sorting!
  - Insertion Sort
  - Merge Sort
- Solving Recurrences

# The problem of sorting

***Input:*** array $A[1\ldots n]$ of numbers.

***Output:*** permutation $B[1\ldots n]$ of $A$ such that $B[1] \le B[2] \le \cdots \le B[n]$.

e.g. $A = [7, 2, 5, 5, 9.6] \rightarrow B = [2, 5, 5, 7, 9.6]$

How can we do it efficiently ?

# Why Sorting?

- Obvious applications
  - Organize an MP3 library
  - Maintain a telephone directory
- Problems that become easy once items are in sorted order
  - Find a median, or find closest pairs
  - Binary search, identify statistical outliers
- Non-obvious applications
  - Data compression: sorting finds duplicates
  - Computer graphics: rendering scenes front to back

# Insertion sort

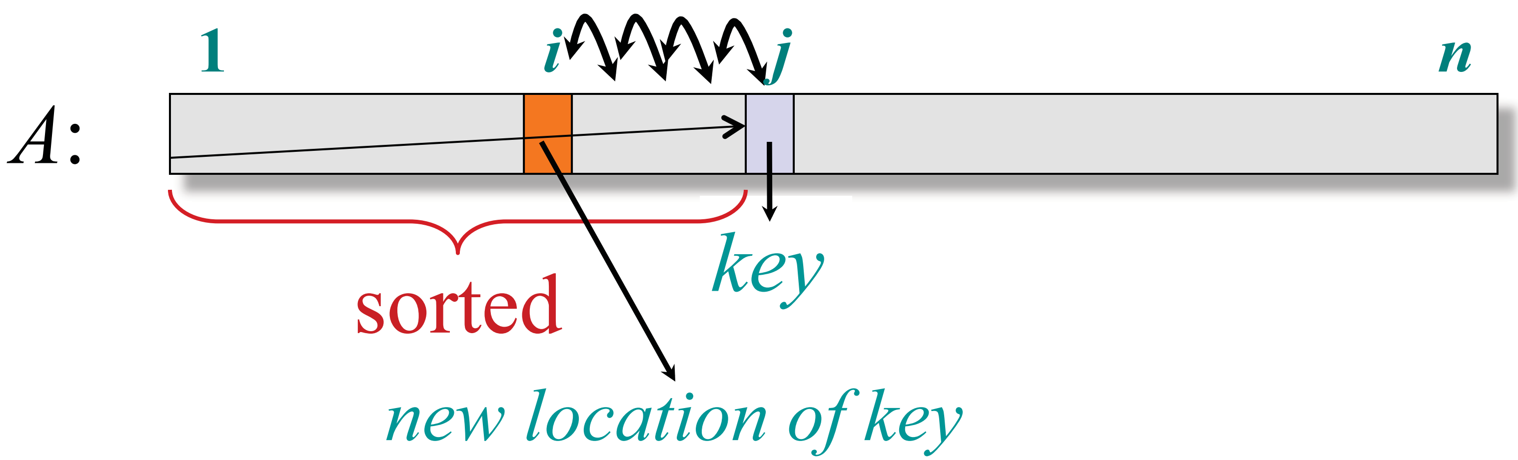
INSERTION-SORT $(A, n)$ $\triangleright$ $A[1 \,.\,.\, n]$

**for** $j \leftarrow 2$ **to** $n$

**insert key** $A[j]$ **into the (already sorted) sub-array** $A[1 \,..\, j\text{-}1]$**.**

**by pairwise key-swaps down to its right position**

**Illustration of iteration *j***



# Example of insertion sort

8 2 4 9 3 6

# Example of insertion sort

8 2 4 9 3 6

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |

# Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

# Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 3 | 4 | 8 | 9 | 6 |

# Example of insertion sort

| | | | | | |
|---|---|---|---|---|---|
| 8 | 2 | 4 | 9 | 3 | 6 |
| 2 | 8 | 4 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 4 | 8 | 9 | 3 | 6 |
| 2 | 3 | 4 | 8 | 9 | 6 |

# Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

Running time? $\Theta(n^2)$ because $\Theta(n^2)$ compares and $\Theta(n^2)$ swaps

e.g. when input is $A = [n, n-1, n-2, \ldots, 2, 1]$

# Binary Insertion sort

**BINARY-INSERTION-SORT** $(A, n)$ $\triangleright$ $A[1 \,.\,.\, n]$

**for** $j \leftarrow 2$ **to** $n$

**insert key** $A[j]$ **into the (already sorted) sub-array** $A[1 \,..\, j\text{-}1]$.

**Use binary search to find the right position**

Binary search with take $\Theta(\log n)$ time.
However, shifting the elements after insertion will still take $\Theta(n)$ time.

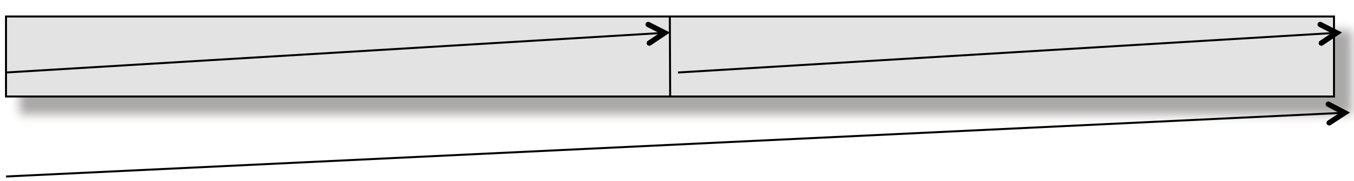
Complexity: $\Theta(n \log n)$ comparisons
$(n^2)$ swaps

# Meet Merge Sort

divide and conquer

**MERGE-SORT** $A[1 \,.\,.\, n]$

1. If $n = 1$, done (nothing to sort).
2. Otherwise, recursively sort $A[\, 1 \,.\,.\, n/2 \,]$ and $A[\, n/2+1 \,.\,.\, n \,]$ .
3. "***Merge***" the two sorted sub-arrays.



***Key subroutine:*** **MERGE**

# Merging two sorted arrays

| | |
|---|---|
| 20 | 12 |
| 13 | 11 |
| 7 | 9 |
| 2 | 1 |

# Merging two sorted arrays

| | |
|---|---|
| 20 | 12 |
| 13 | 11 |
| 7 | 9 |
| 2 | 1 |

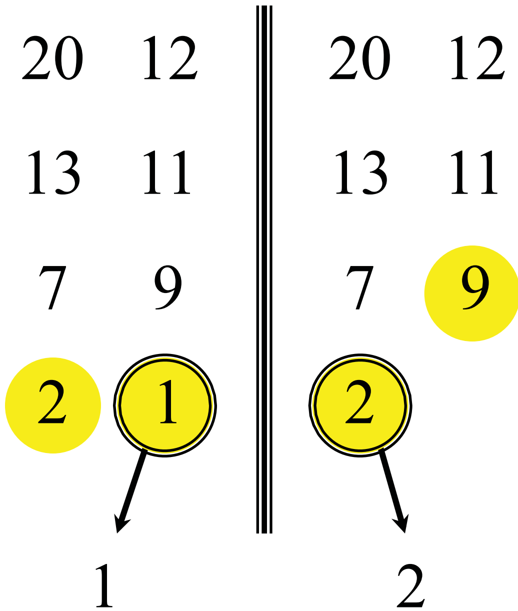
1

# Merging two sorted arrays

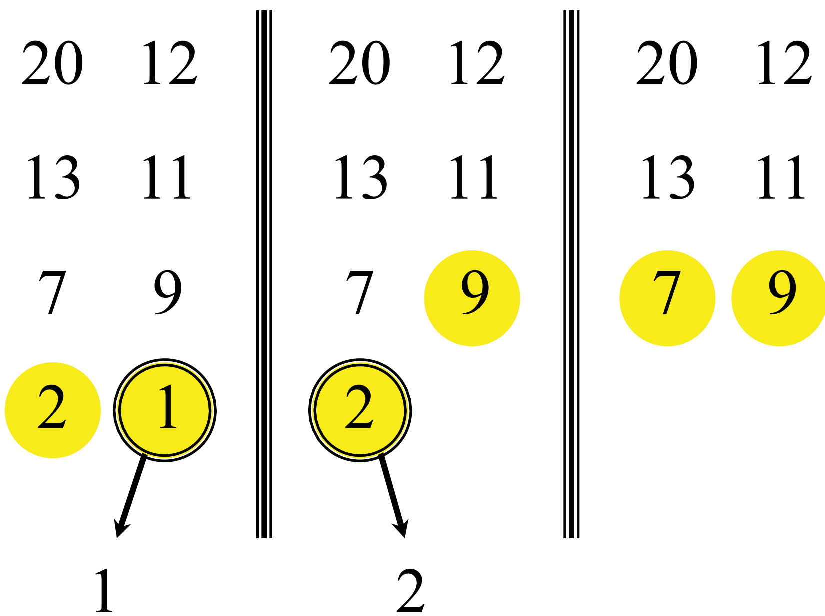
| | | | | |
|---|---|---|---|---|
| 20 | 12 | ‖ | 20 | 12 |
| 13 | 11 | ‖ | 13 | 11 |
| 7 | 9 | ‖ | 7 | 9 |
| 2 | 1 | ‖ | 2 | |

1

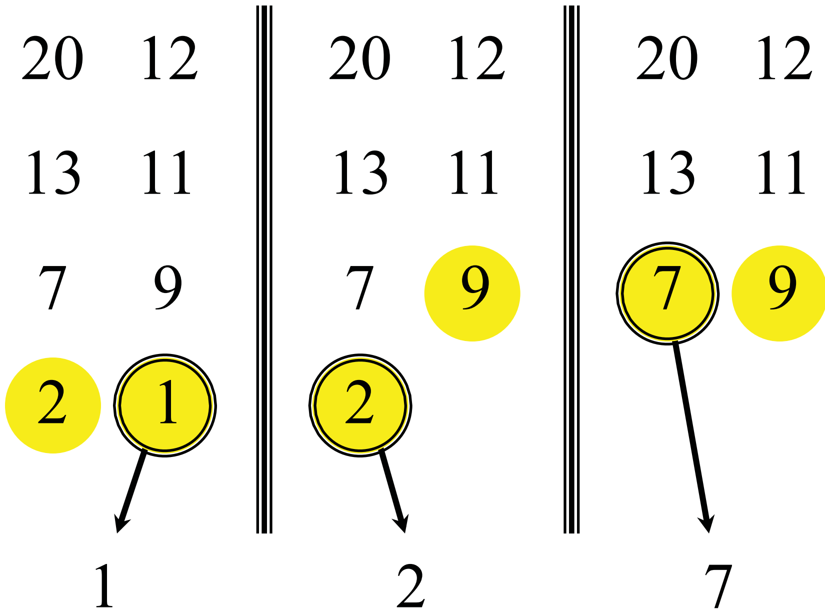
# Merging two sorted arrays



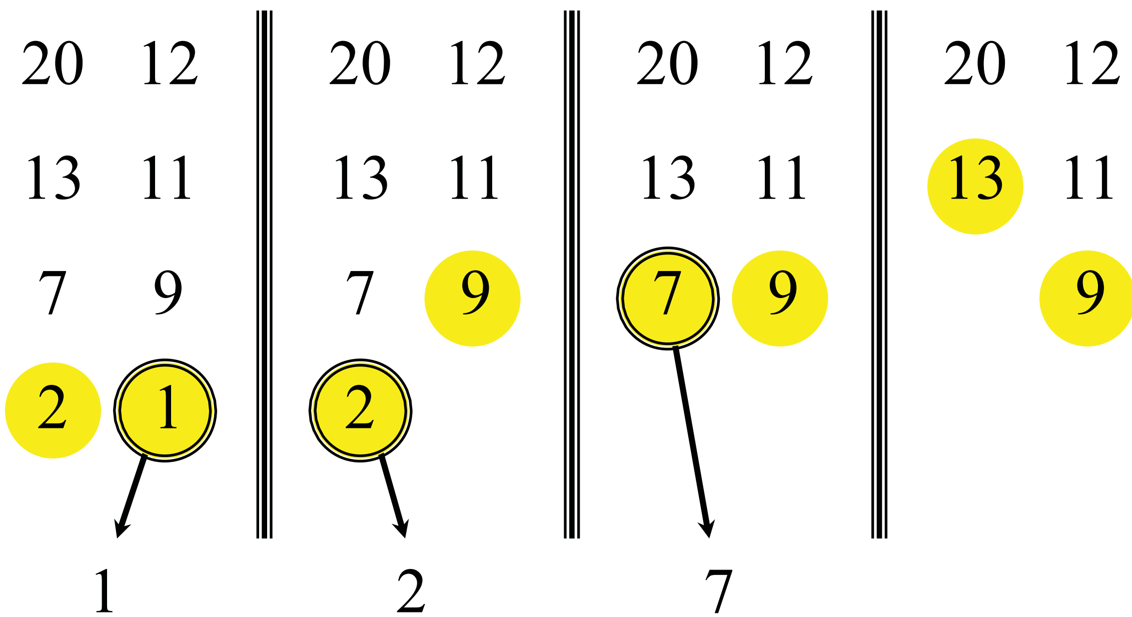
# Merging two sorted arrays



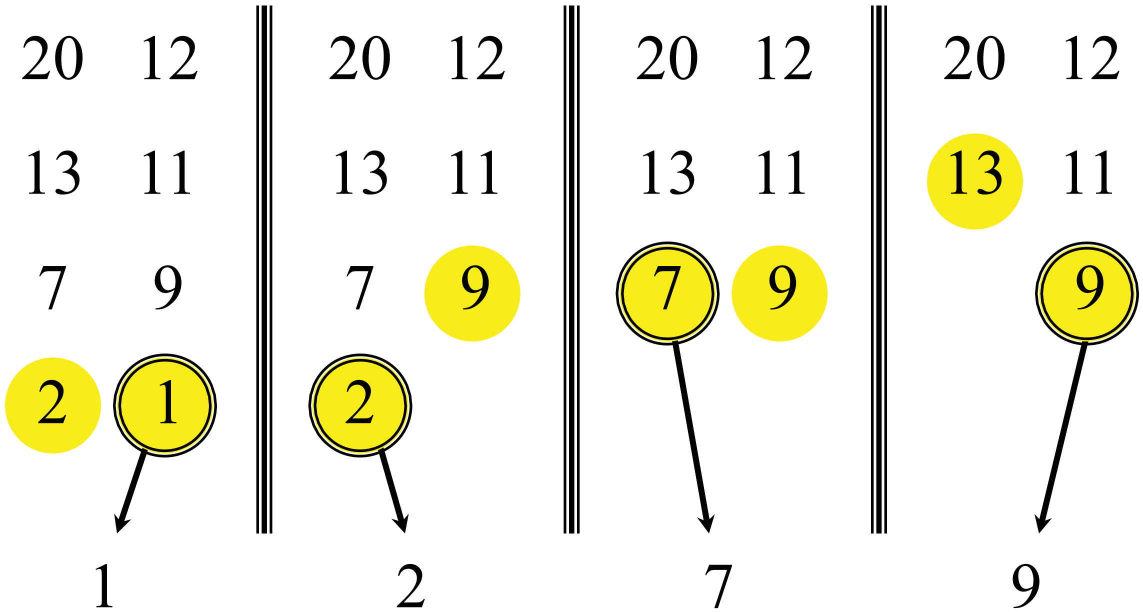
# Merging two sorted arrays



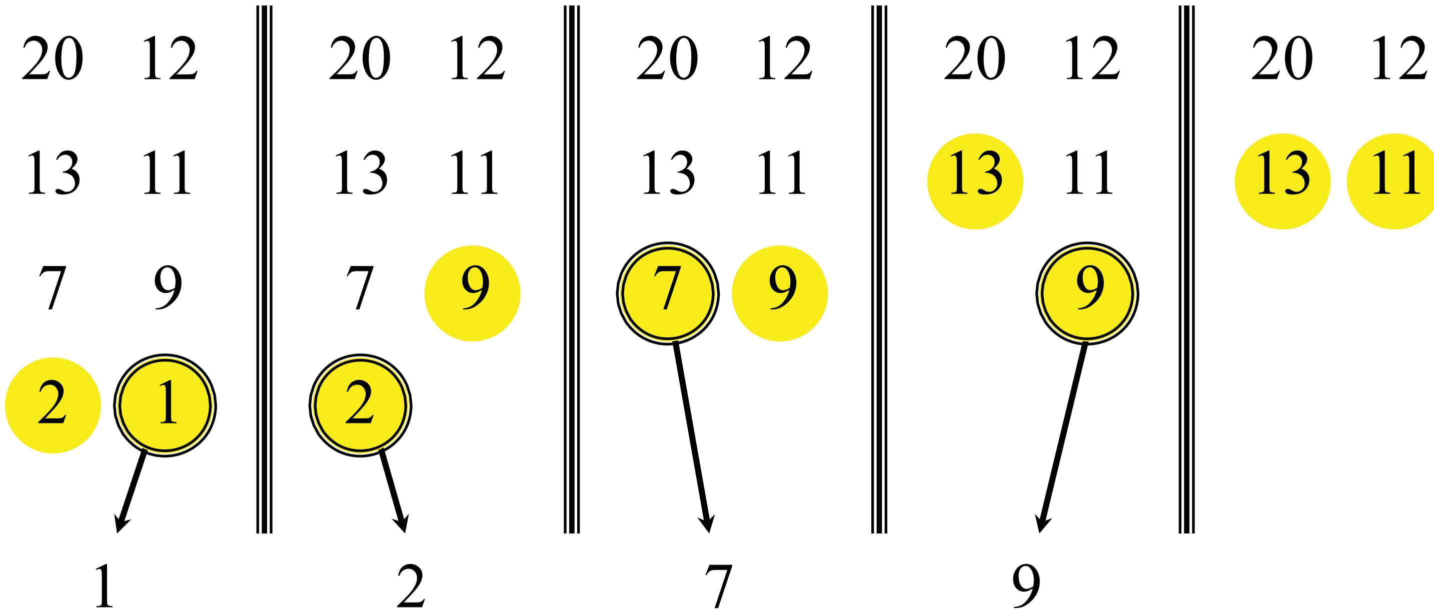
# Merging two sorted arrays



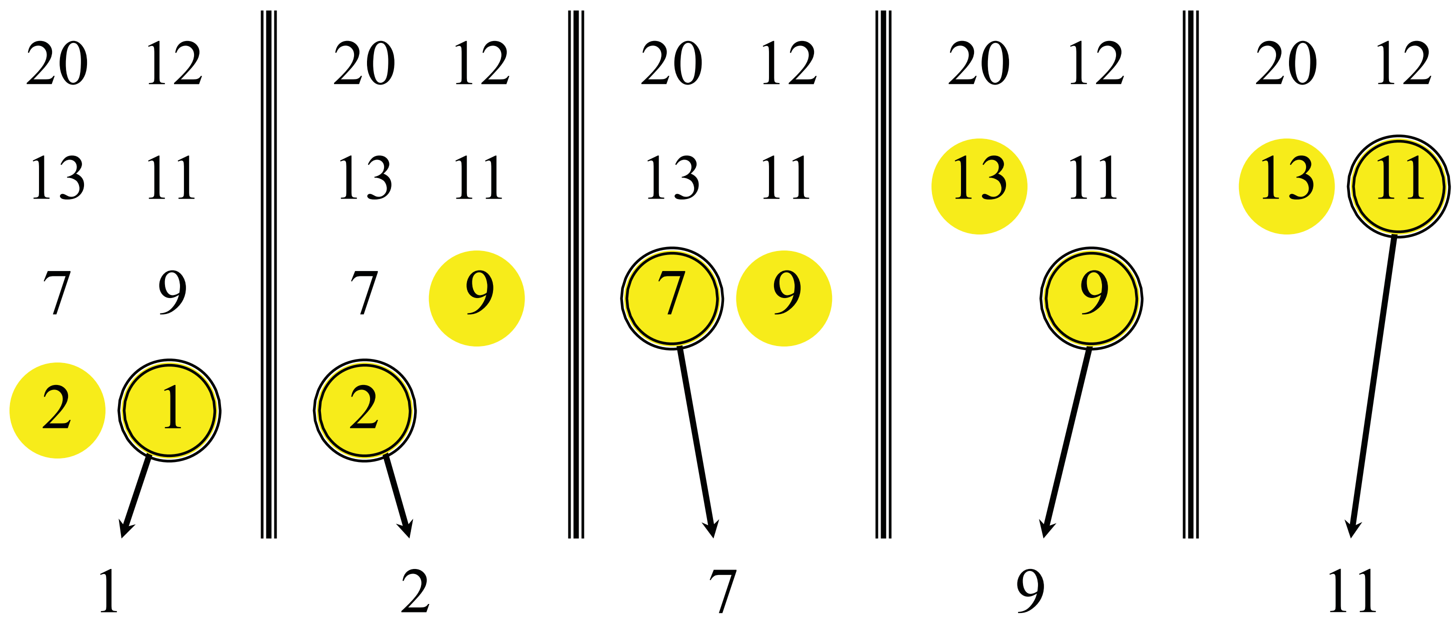
# Merging two sorted arrays



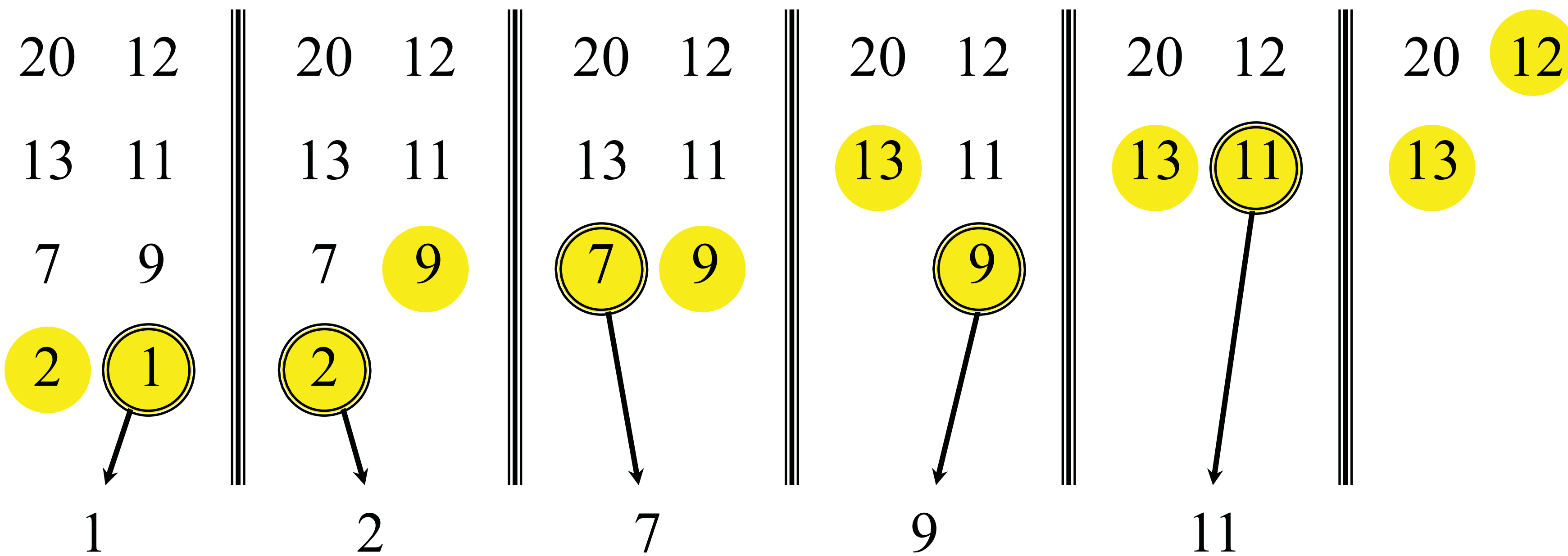
# Merging two sorted arrays



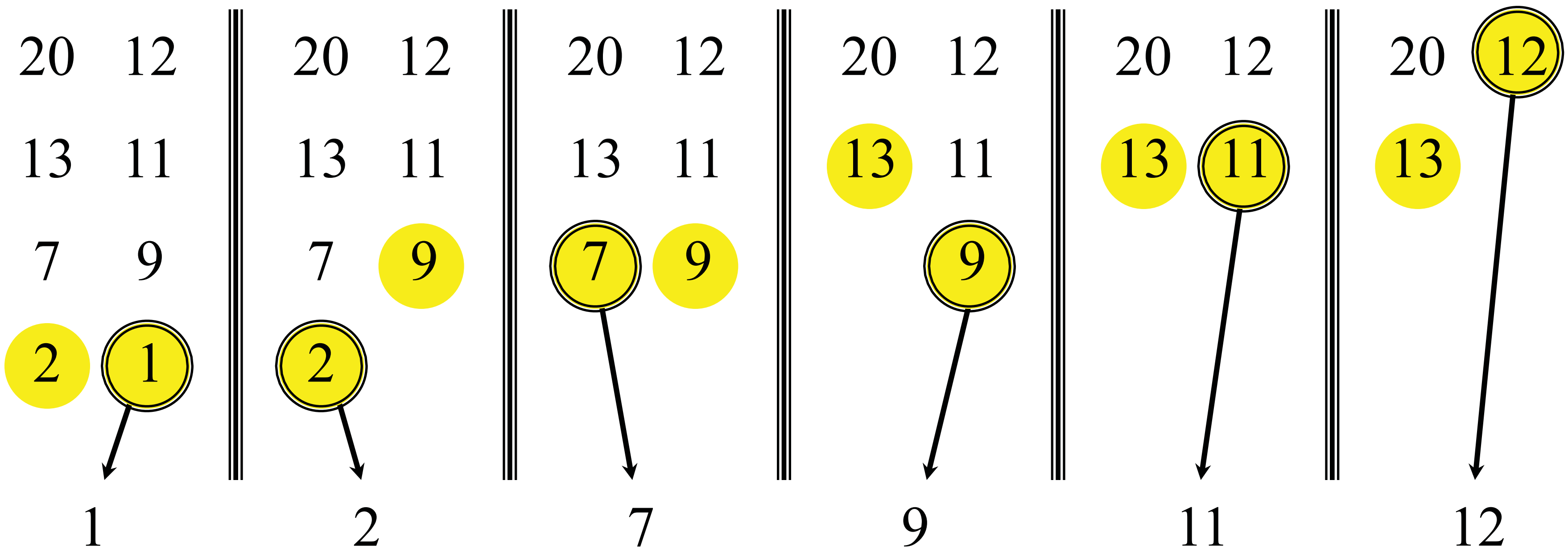
# Merging two sorted arrays



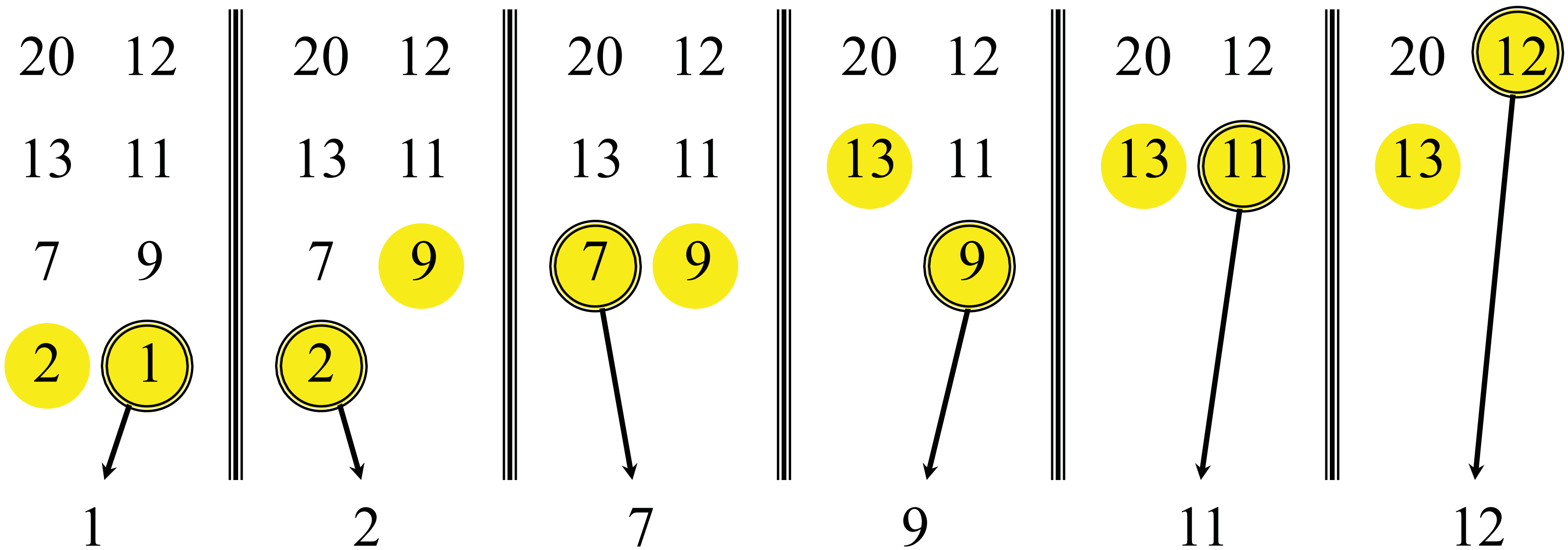
# Merging two sorted arrays



# Merging two sorted arrays



# Merging two sorted arrays




Time = $\Theta(n)$ to merge a total of $n$ elements (linear time).

# Analyzing merge sort

| | |
|---|---|
| MERGE-SORT $A[1 \,.\,.\, n]$ | $T(n)$ |
| 1. If $n = 1$, done. | $\Theta(1)$ |
| 2. Recursively sort $A[\, 1 \,.\,.\, \lceil n/2 \rceil \,]$ and $A[\, \lceil n/2 \rceil + 1 \,.\,.\, n \,]$ . | $2T(n/2)$ |
| ***3. "Merge"*** the two sorted lists | $\Theta(n)$ |

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

$$T(n) = ?$$

# Recurrence solving

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

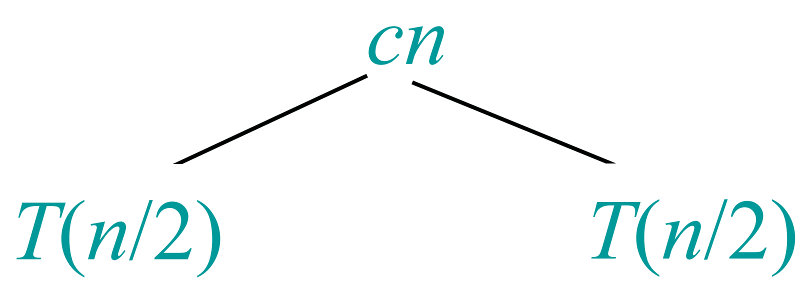
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$$T(n)$$

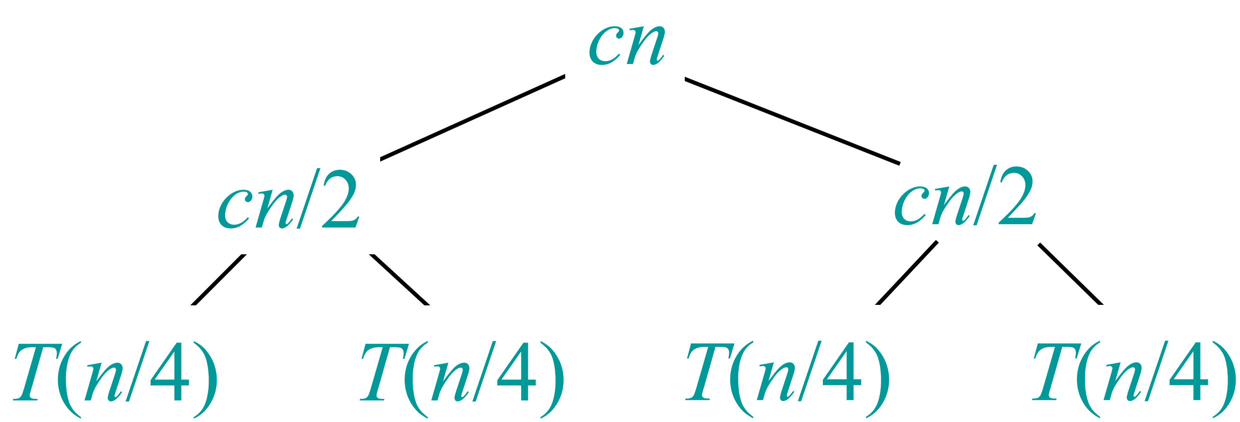
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



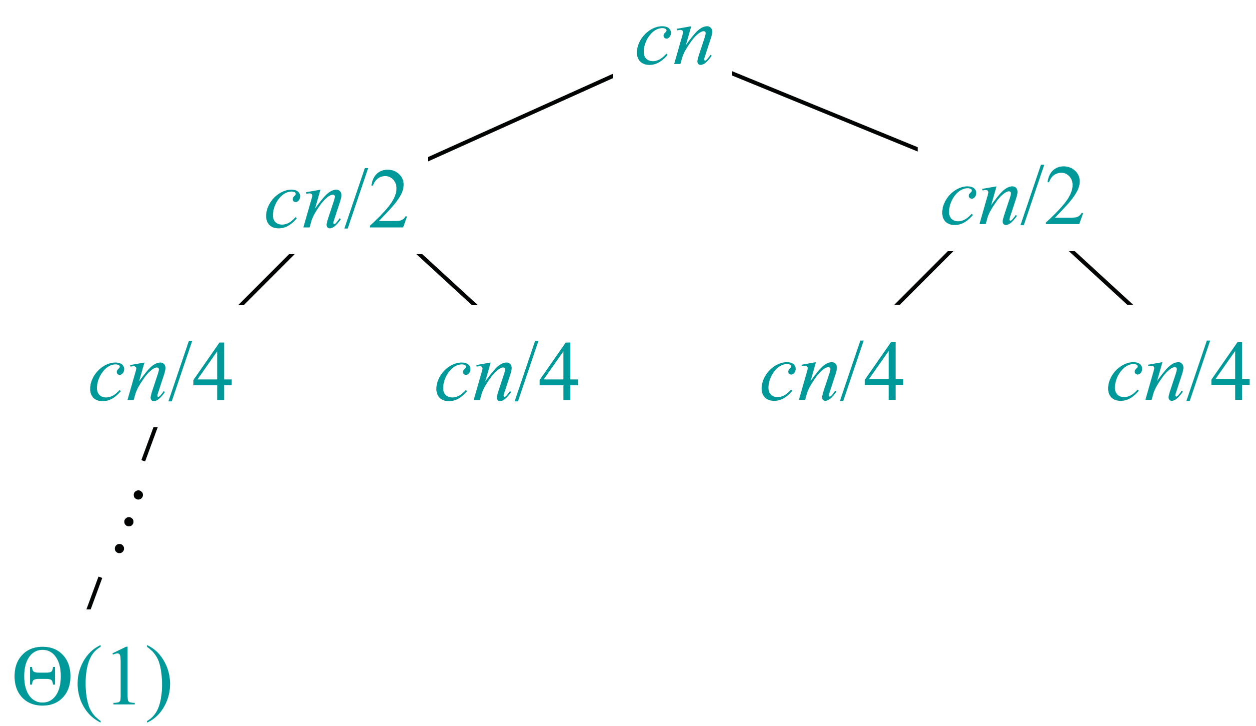
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



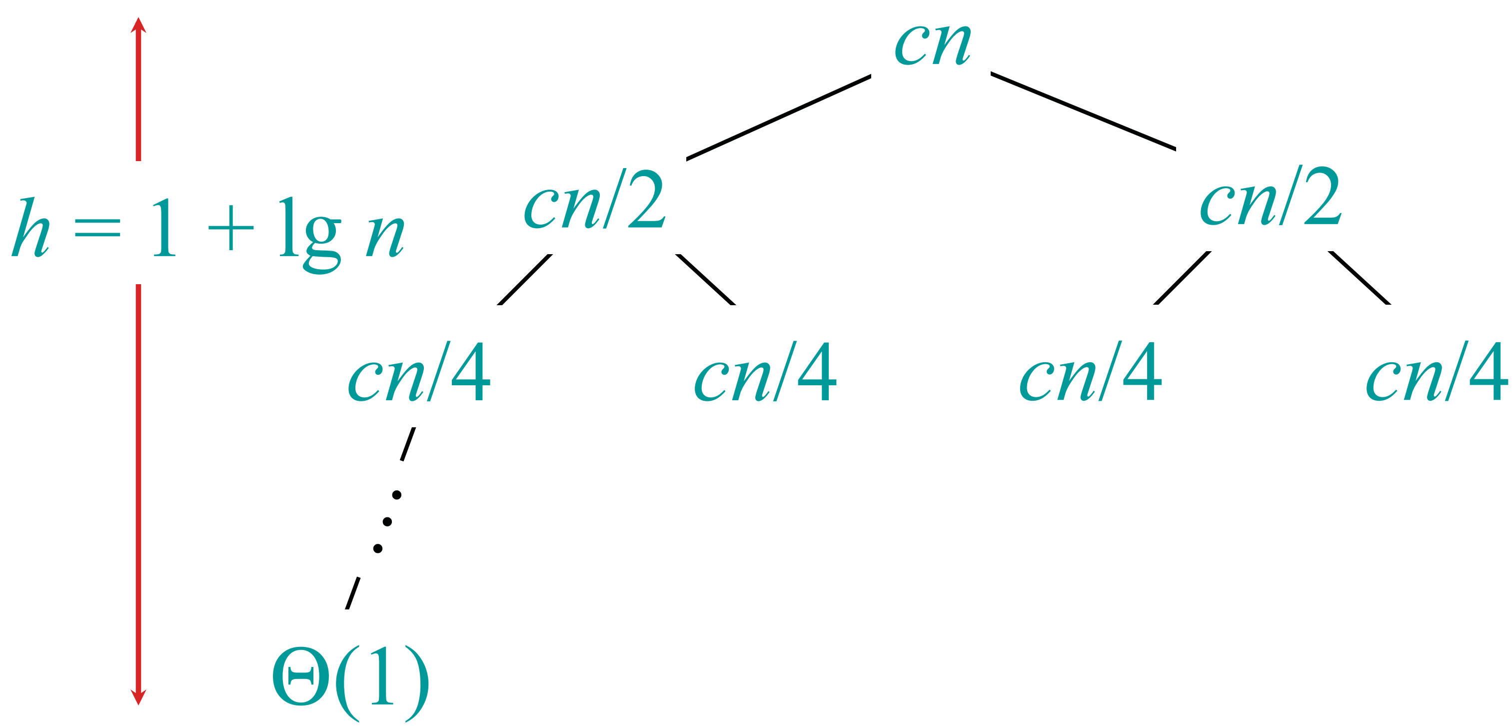
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



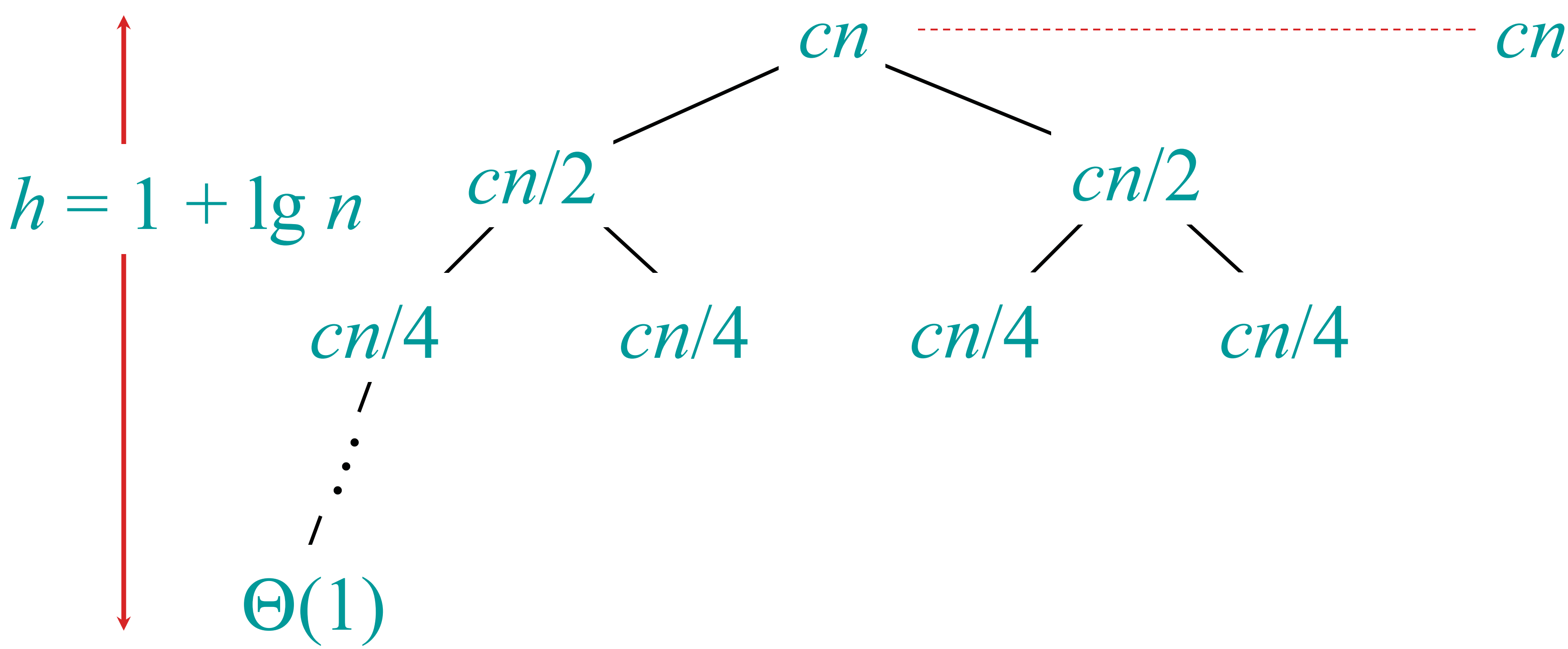
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



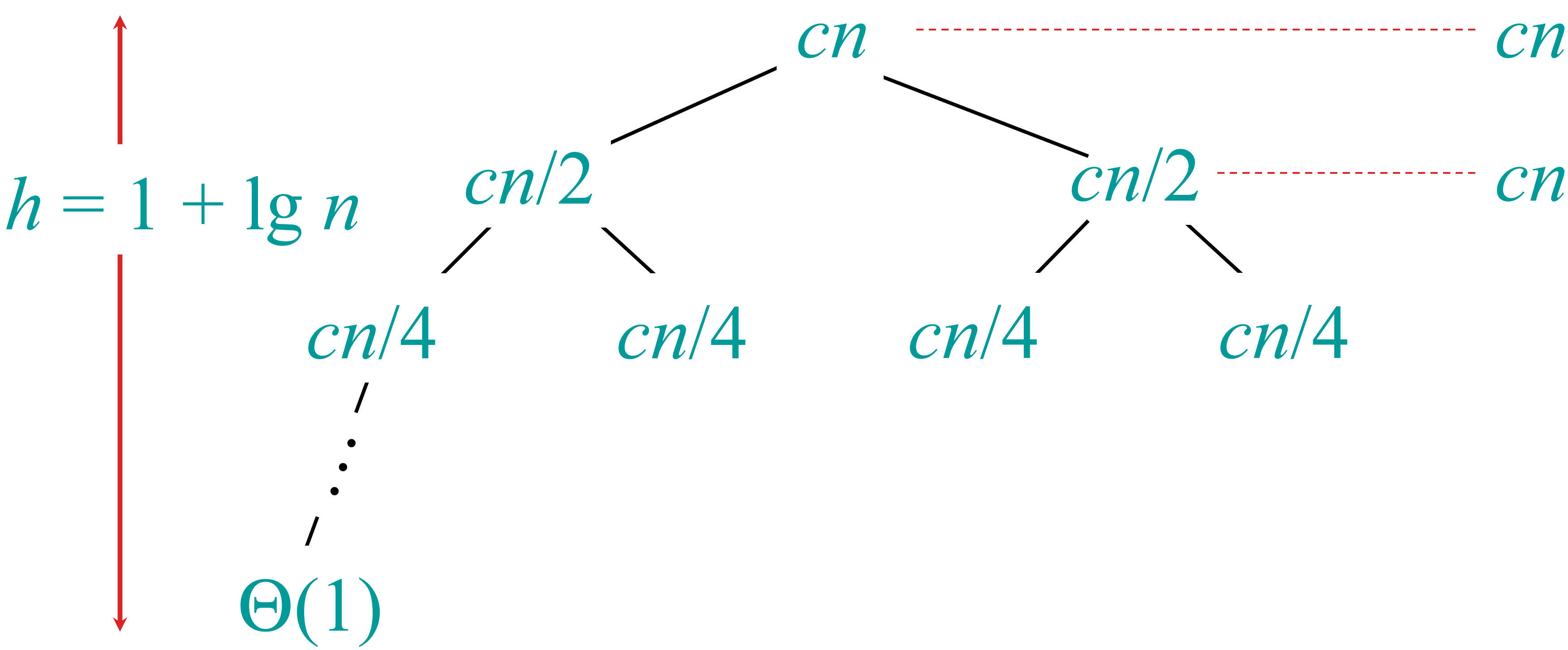
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



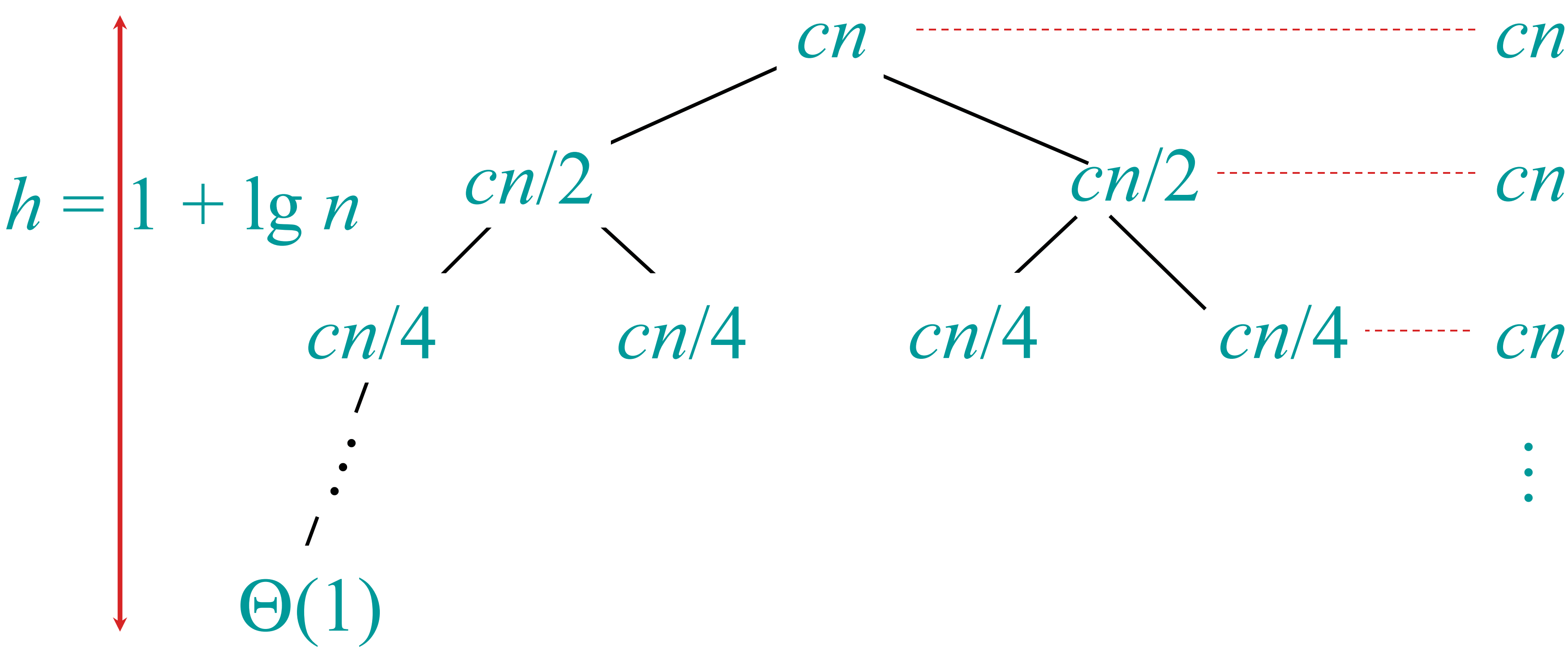
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



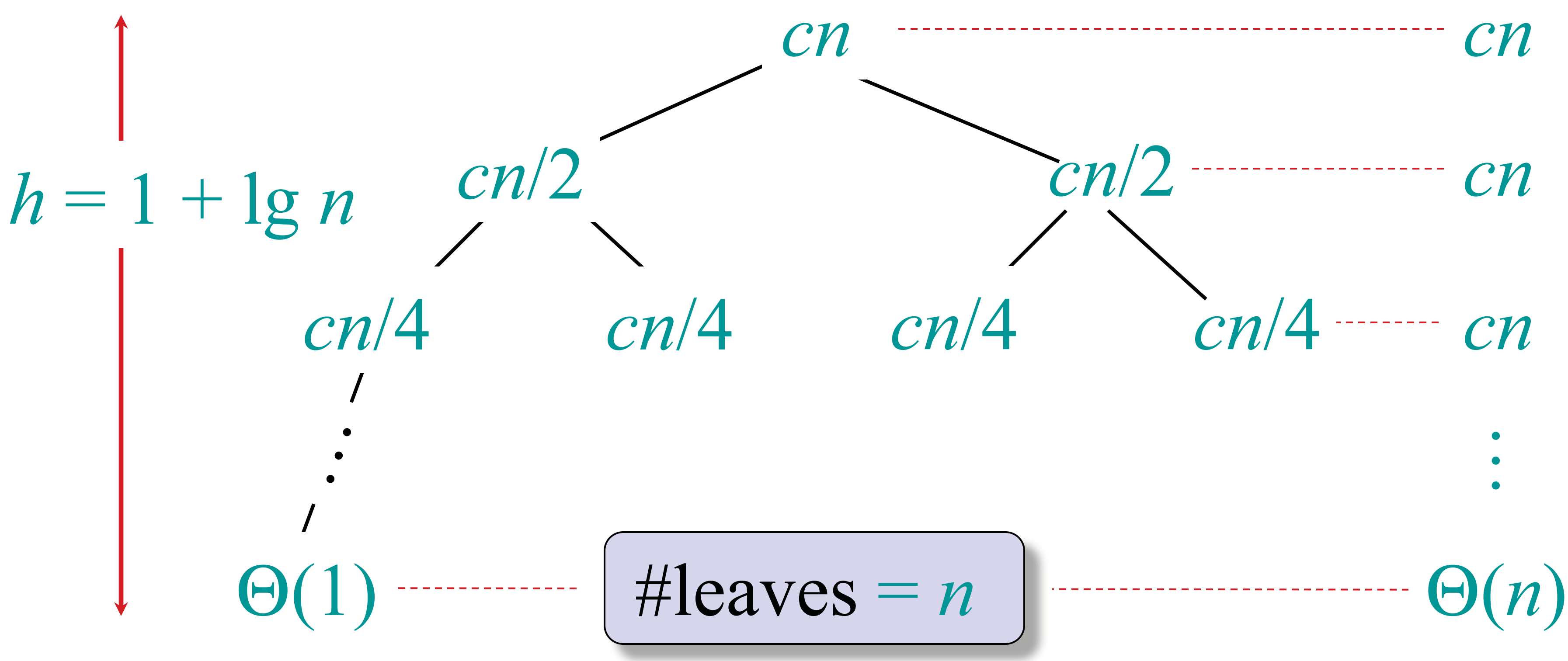
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



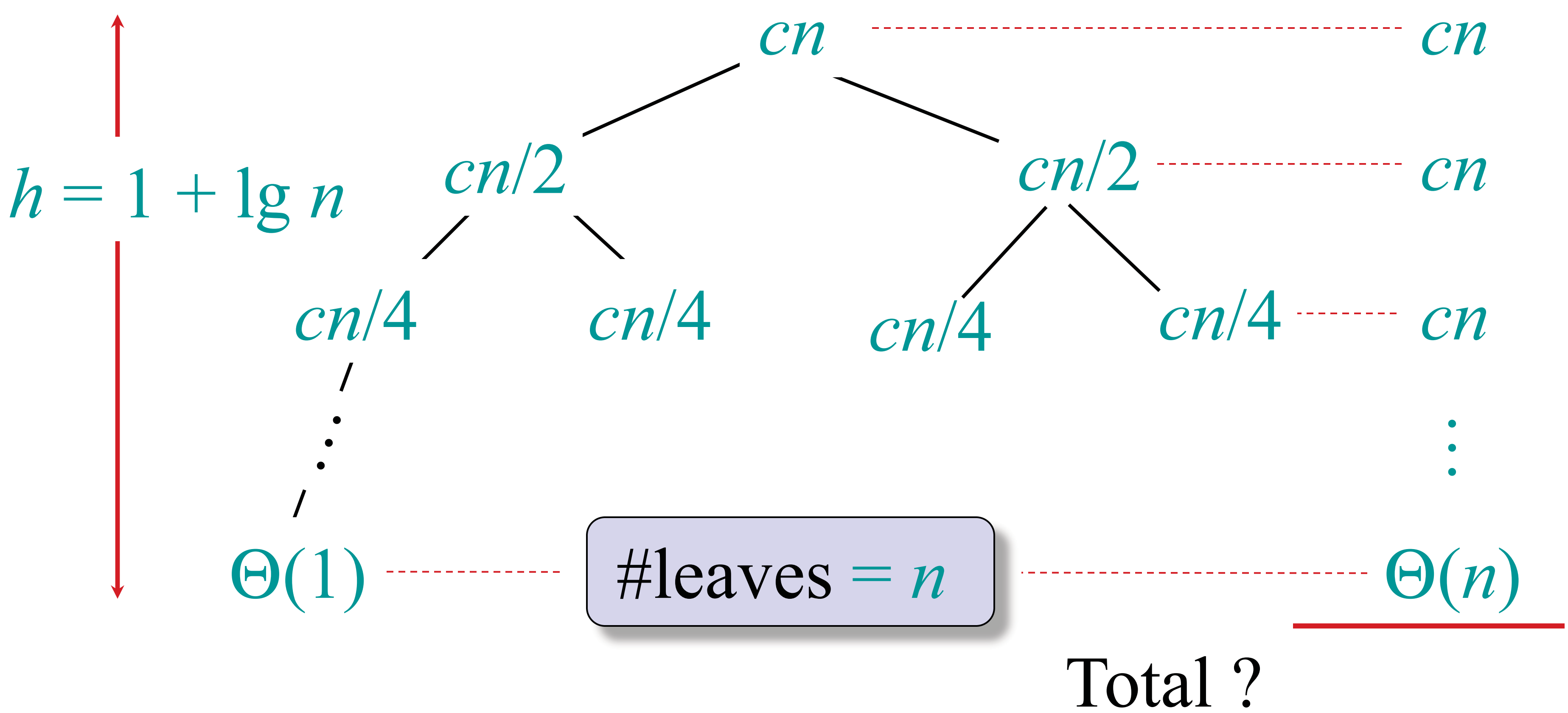
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



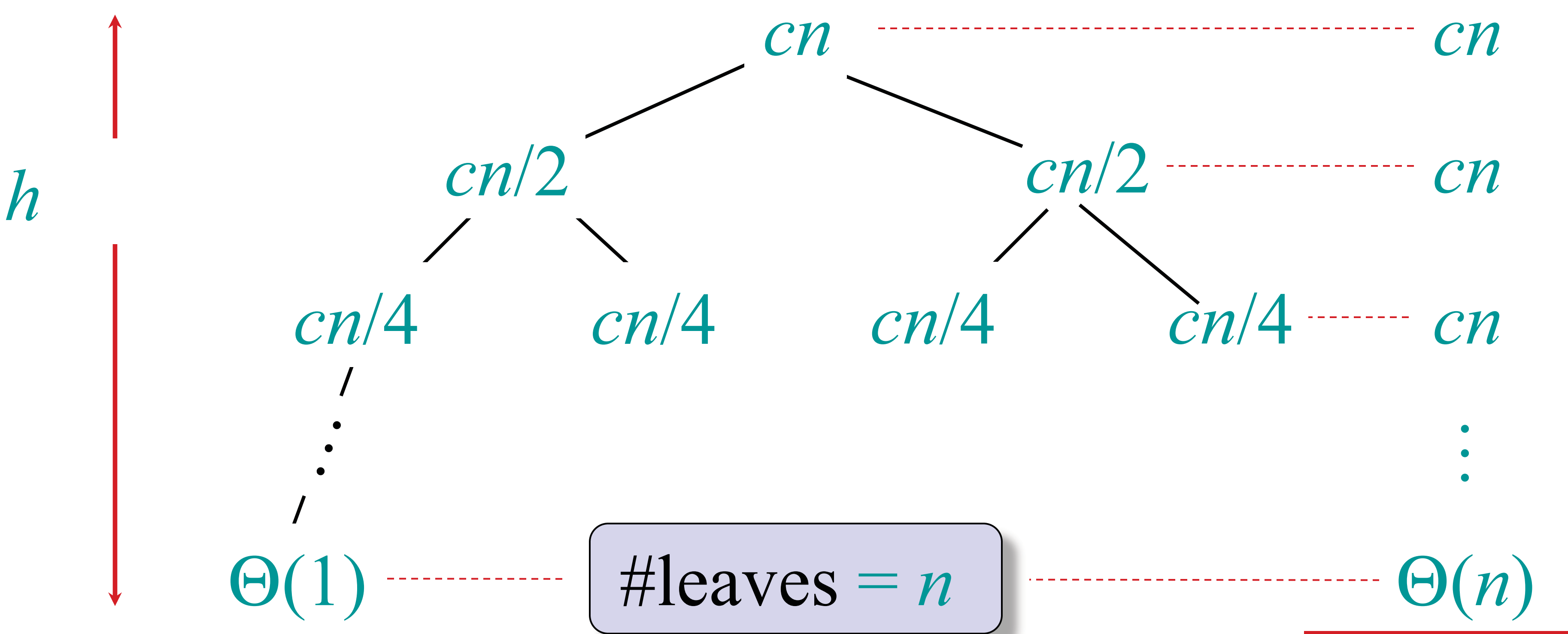
# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



# Recursion tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

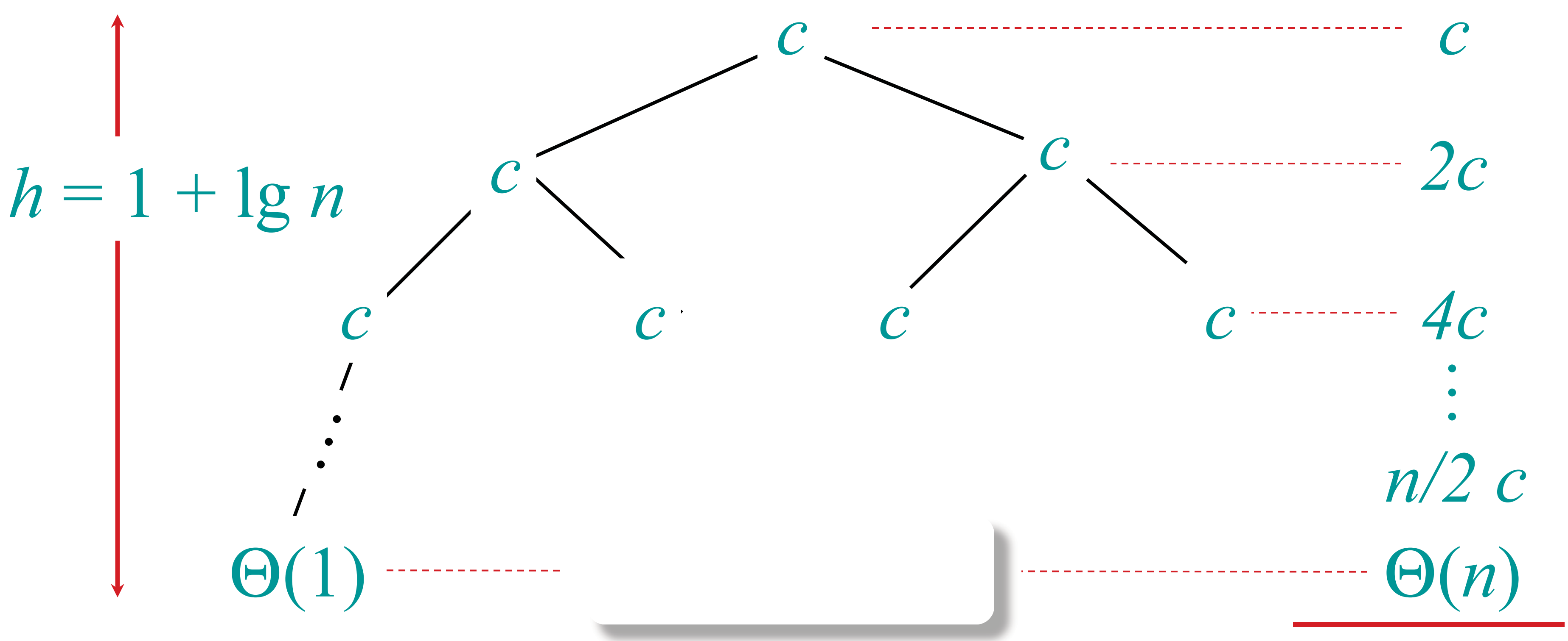



Total $= \Theta(n \lg n)$

Equal amount of work done at each level

# Tree for different recurrence

Solve $T(n) = 2T(n/2) + c$, where $c > 0$ is constant.



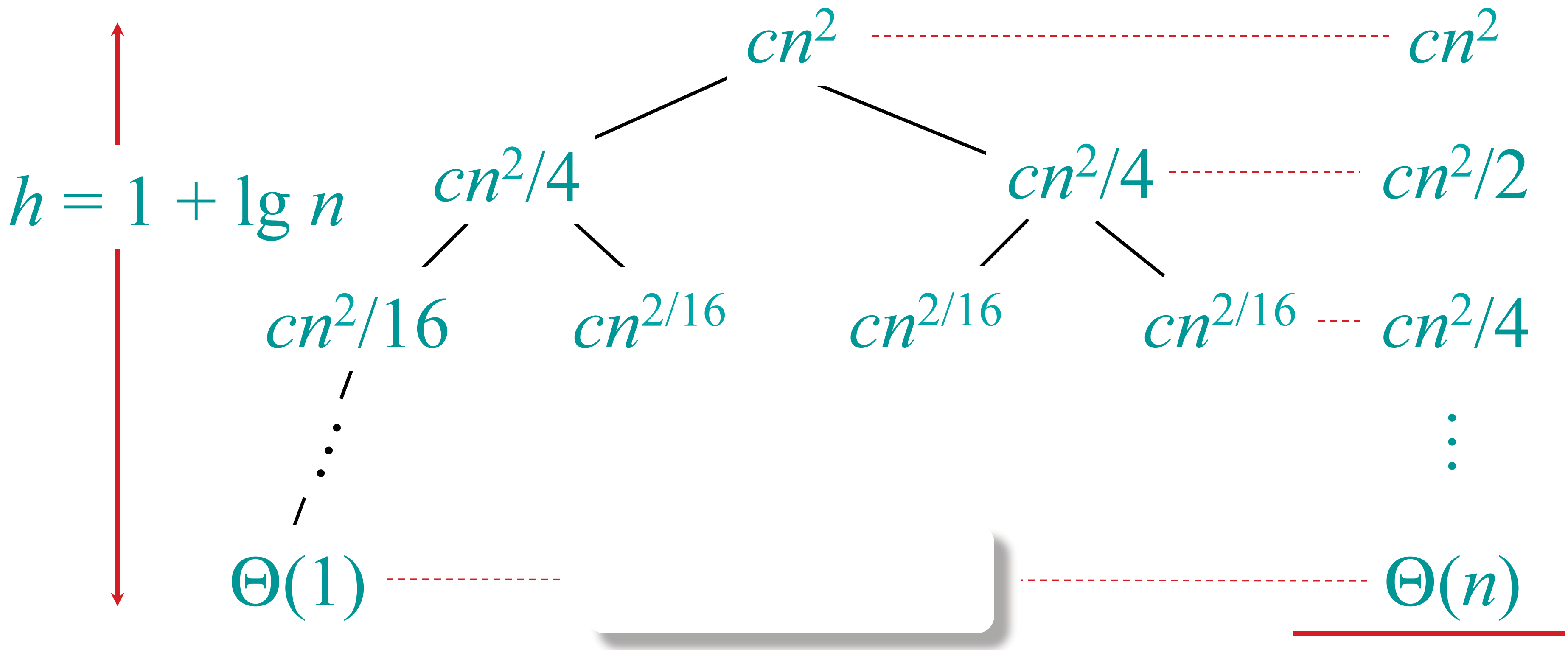

Note that $1 + ½ + ¼ + \ldots < 2$

Total $= \Theta(n)$

All the work done at the leaves

# Tree for yet another recurrence

Solve $T(n) = 2T(n/2) + cn^2$, $c > 0$ is constant.




Note that $1 + ½ + ¼ + \ldots < 2$

Total $= \Theta(n^2)$

All the work done at the root

6.006 Introduction to Algorithms
Fall 2011