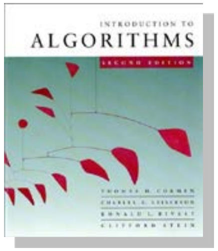
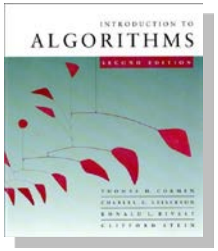


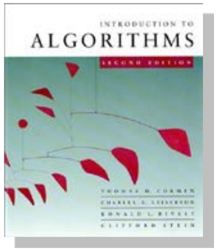

# The divide-and-conquer design paradigm

1. ***Divide*** the problem (instance) into subproblems.
2. ***Conquer*** the subproblems by solving them recursively.
3. ***Combine*** subproblem solutions.




# Merge sort

1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.




# Merge sort

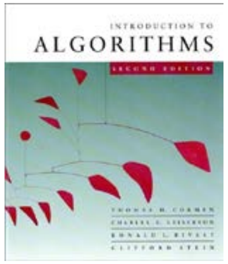
1. ***Divide:*** Trivial.
2. ***Conquer:*** Recursively sort 2 subarrays.
3. ***Combine:*** Linear-time merge.

$$T(n) = 2\,T(n/2) + \Theta(n)$$

*# subproblems*

*subproblem size*

*work dividing and combining*



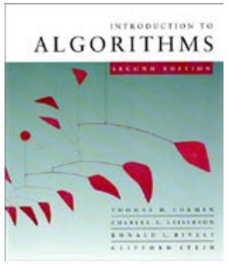

# Merge sort

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

1. If $n = 1$, done.
2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$ .
3. "***Merge***" the $2$ sorted lists.

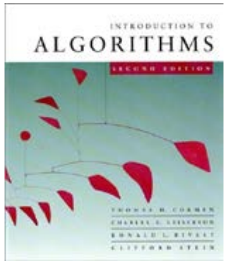
***Key subroutine:*** **MERGE**




# Merging two sorted arrays

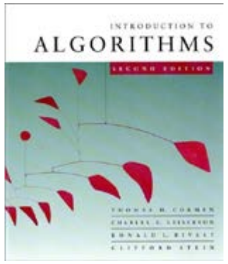
| | |
|---|---|
| 20 | 12 |
| 13 | 11 |
| 7 | 9 |
| 2 | 1 |




# Merging two sorted arrays

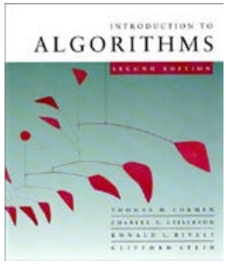
| | |
|---|---|
| 20 | 12 |
| 13 | 11 |
| 7 | 9 |
| 2 | 1 |

1




# Merging two sorted arrays

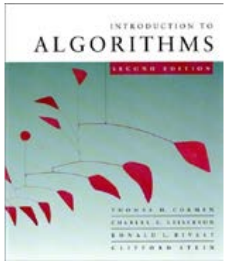
| | | | |
|---|---|---|---|
| 20 | 12 | 20 | 12 |
| 13 | 11 | 13 | 11 |
| 7 | 9 | 7 | 9 |
| 2 | 1 | 2 | |
| 1 | | | |




# Merging two sorted arrays

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



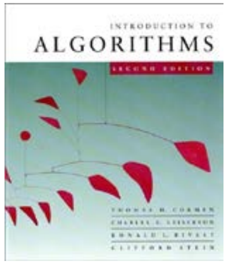

# Merging two sorted arrays

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

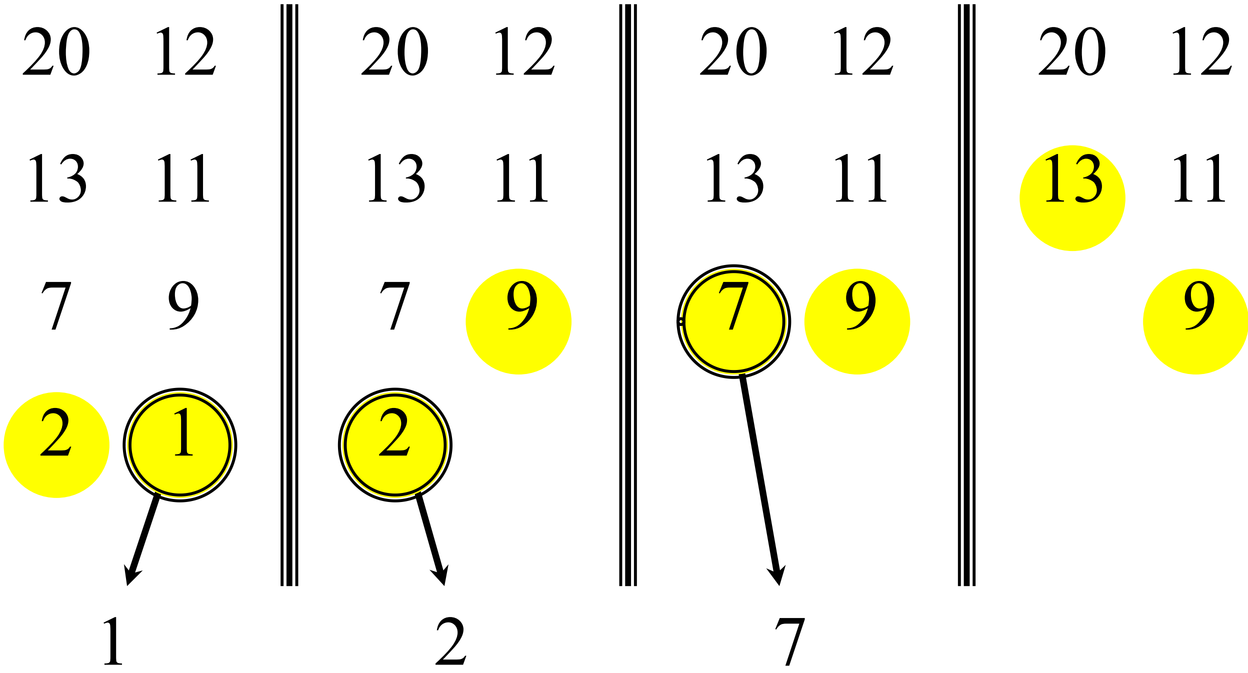
# Merging two sorted arrays

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

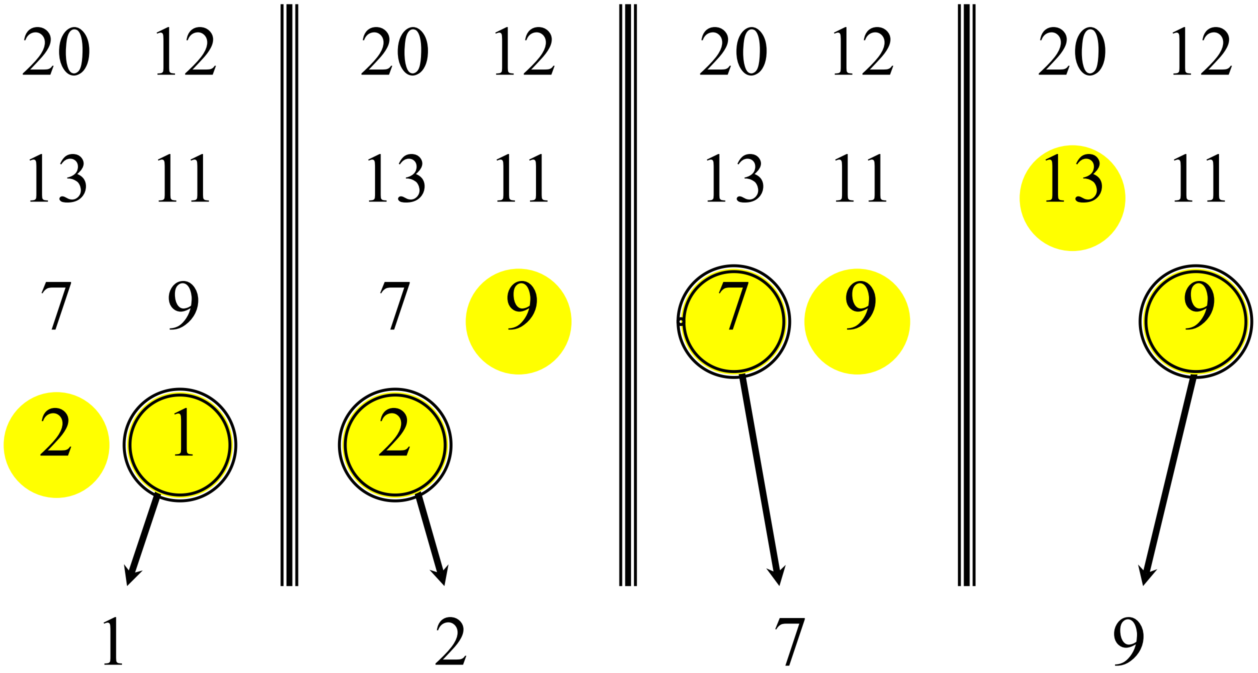
  

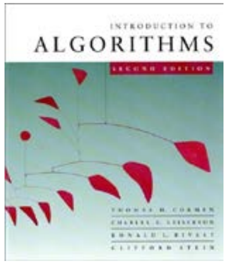



# Merging two sorted arrays

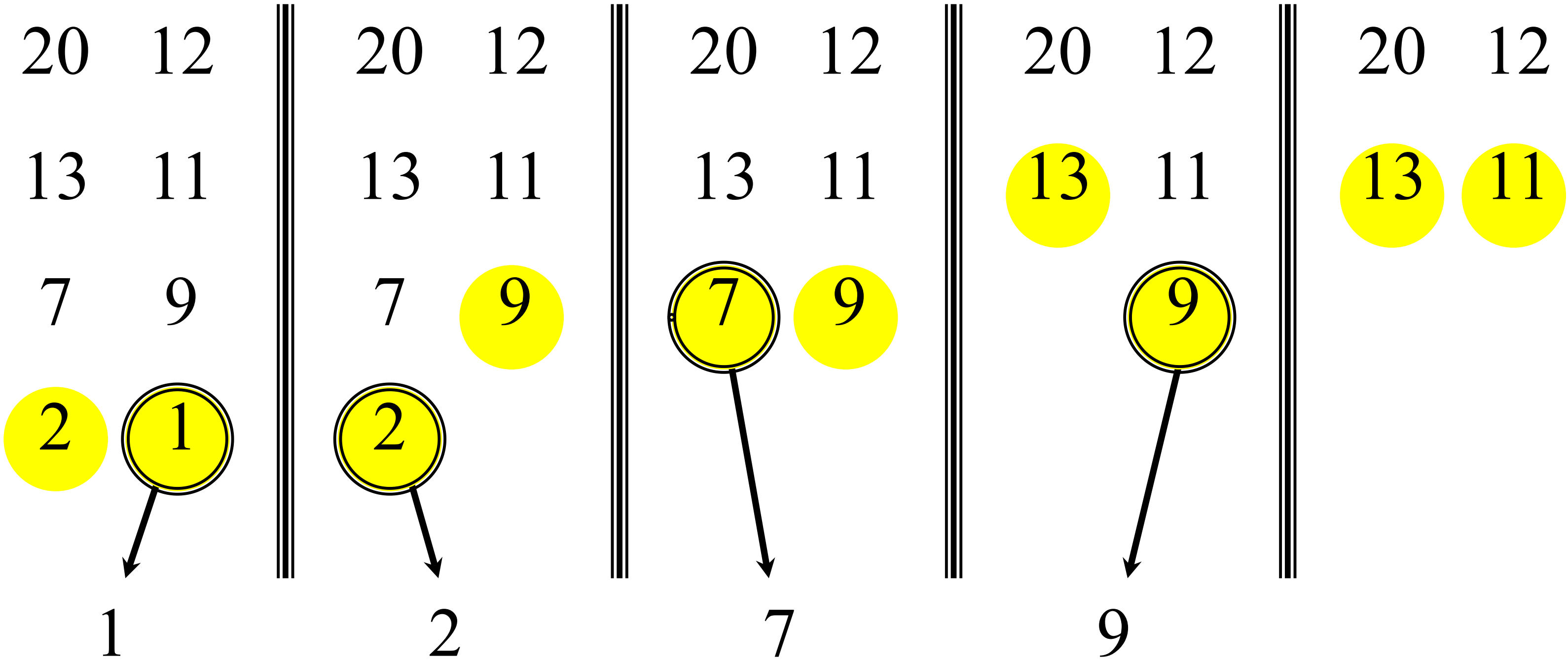


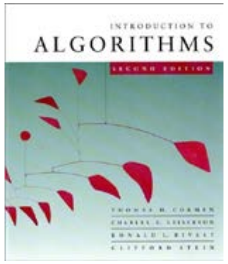
# Merging two sorted arrays



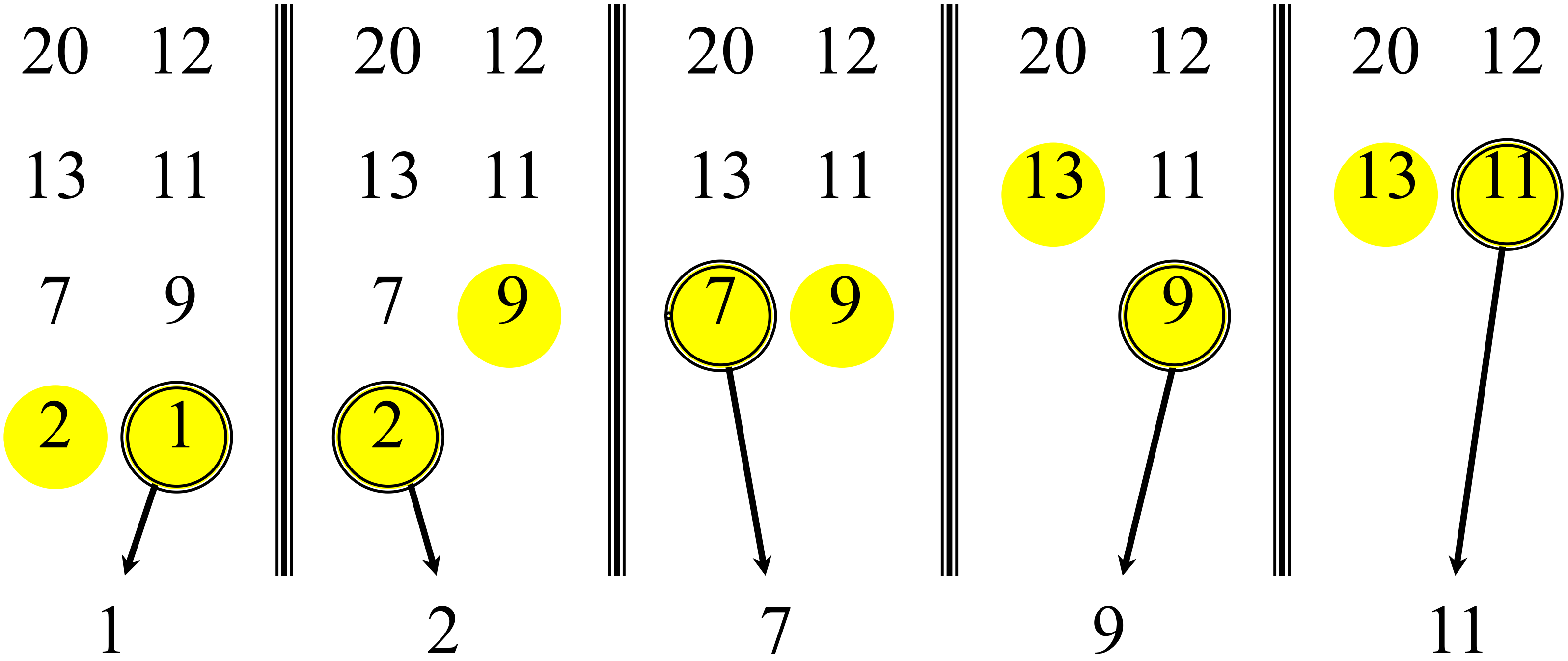



# Merging two sorted arrays

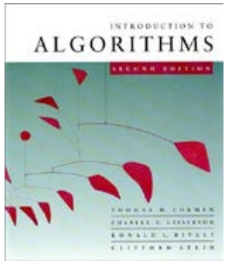




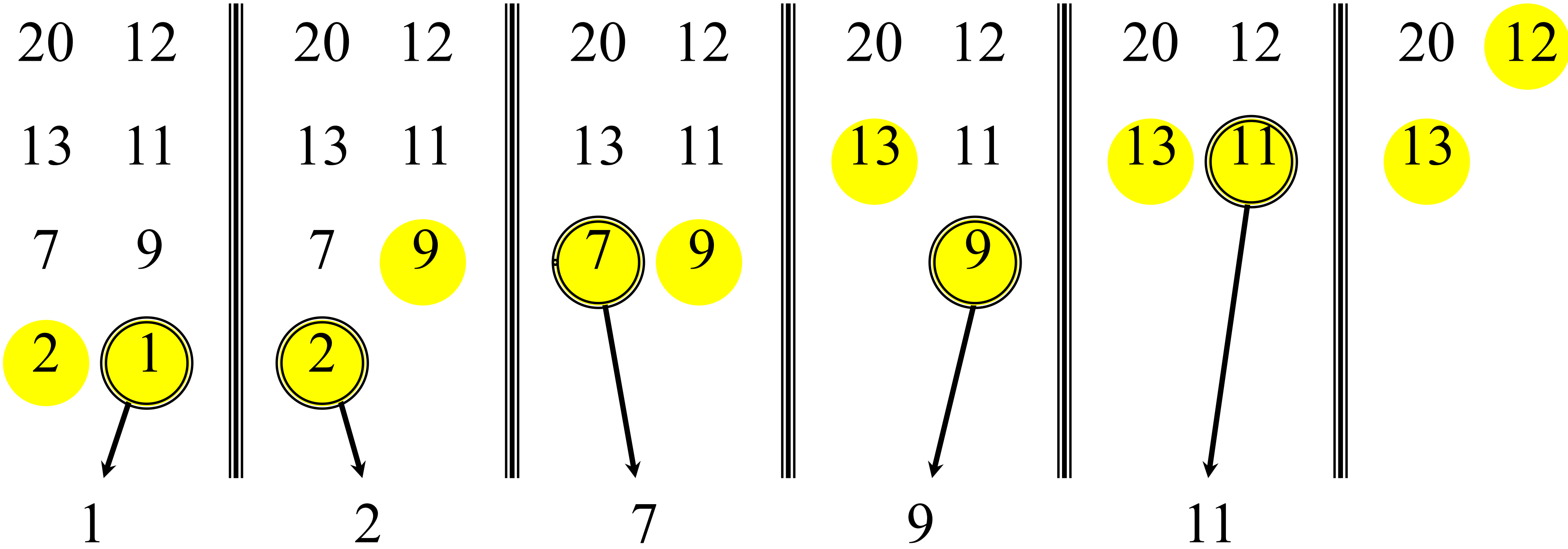

# Merging two sorted arrays



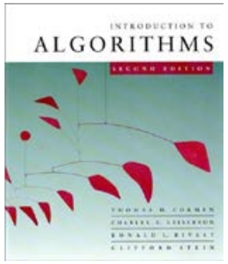

  



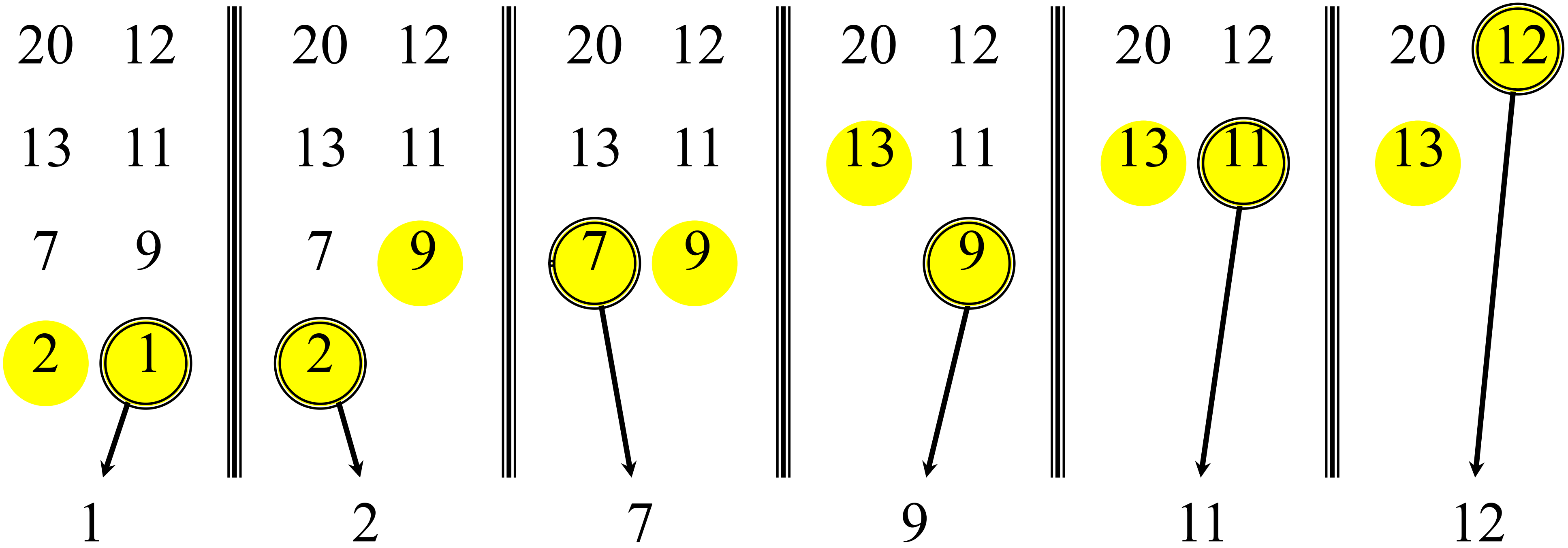

# Merging two sorted arrays

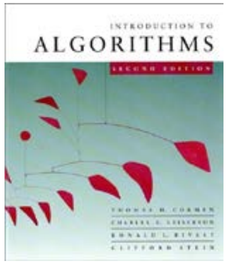


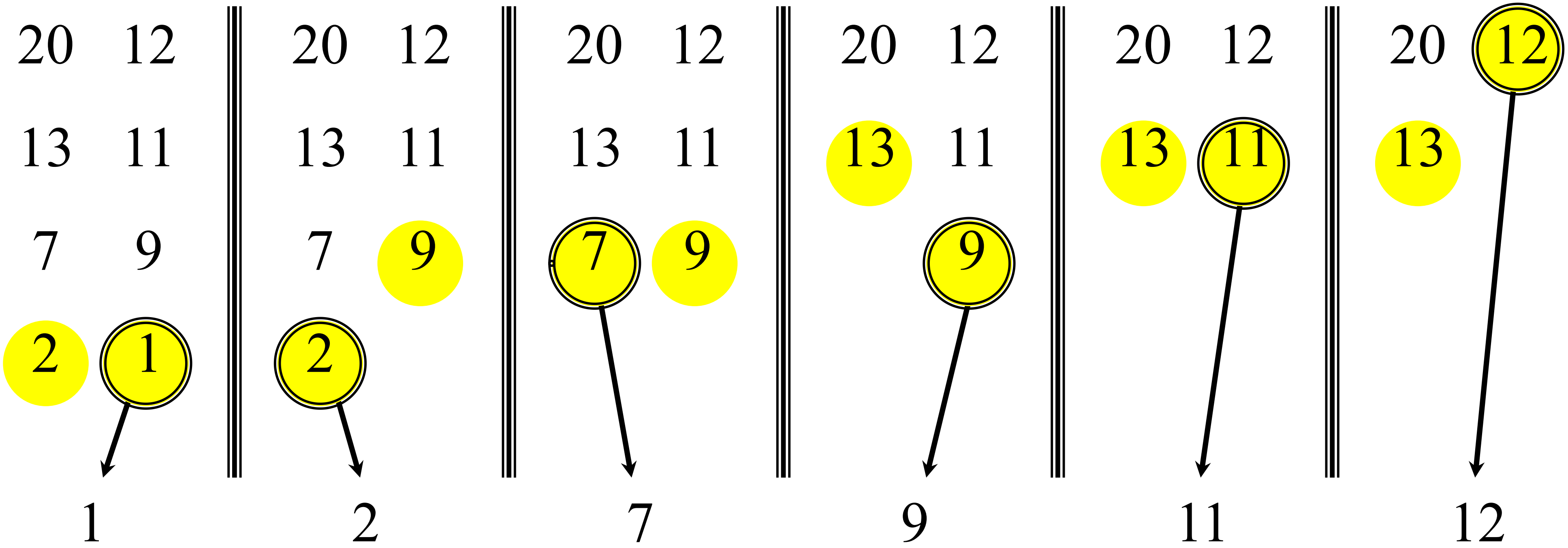



# Merging two sorted arrays

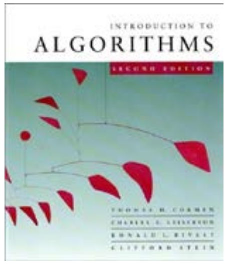





# Merging two sorted arrays




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

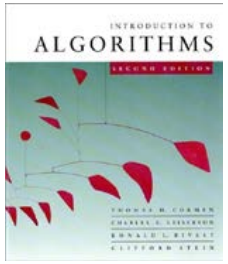



# Analyzing merge sort

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

***Abuse***

***Sloppiness:*** Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$ , but it turns out not to matter asymptotically.

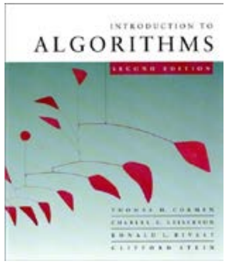



# Recurrence for merge sort

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

- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small $n$, but only when it has no effect on the asymptotic solution to the recurrence.
- CLRS and Lecture 2 provide several ways to find a good upper bound on $T(n)$.



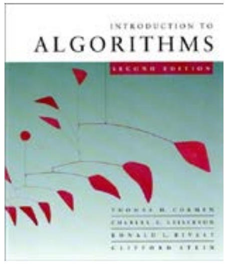

# Recursion tree

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.

$$T(n)$$




# Recursion tree

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

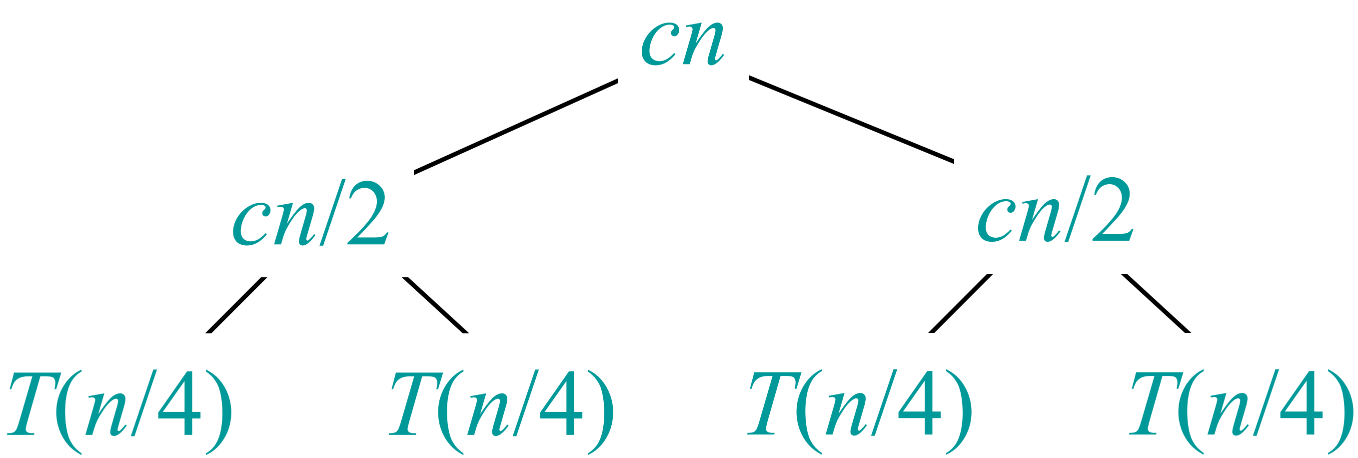
$$cn$$

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

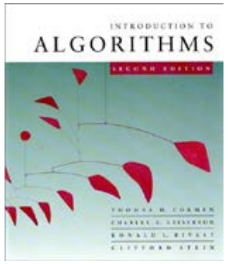
  

# Recursion tree

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

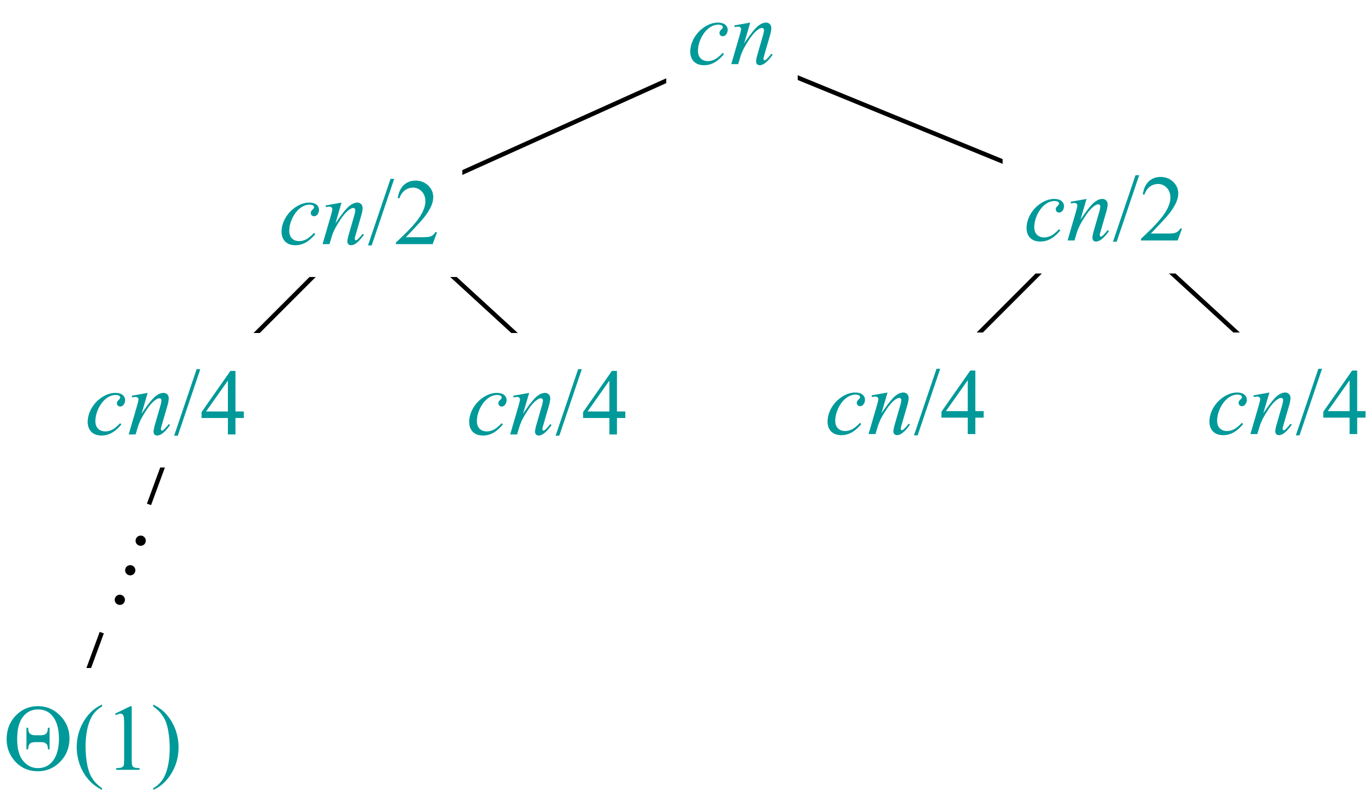


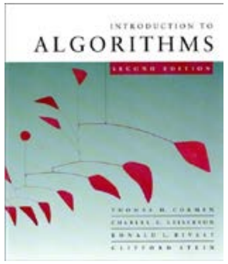



# Recursion tree

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

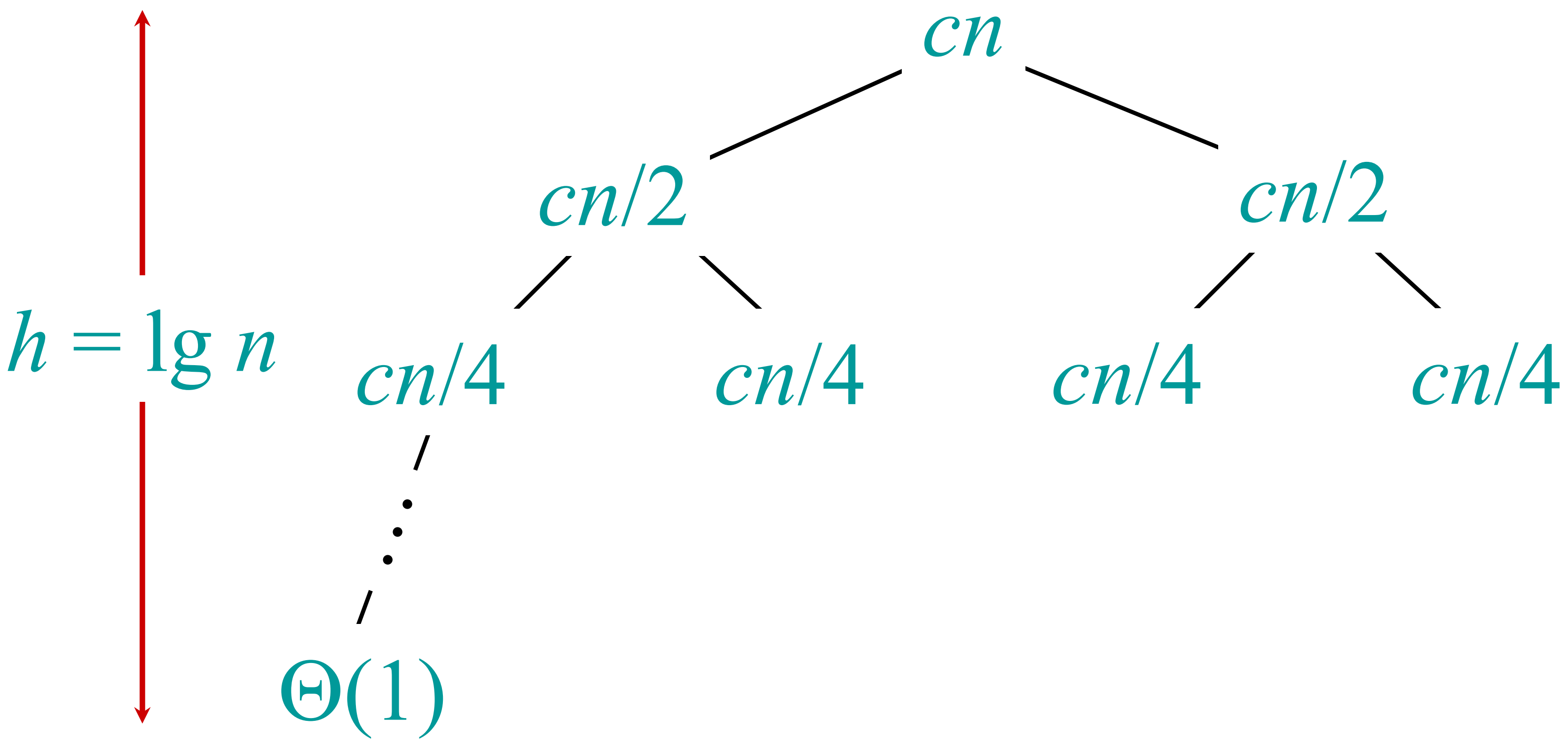


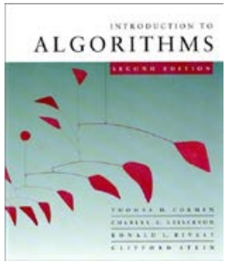

  




# Recursion tree

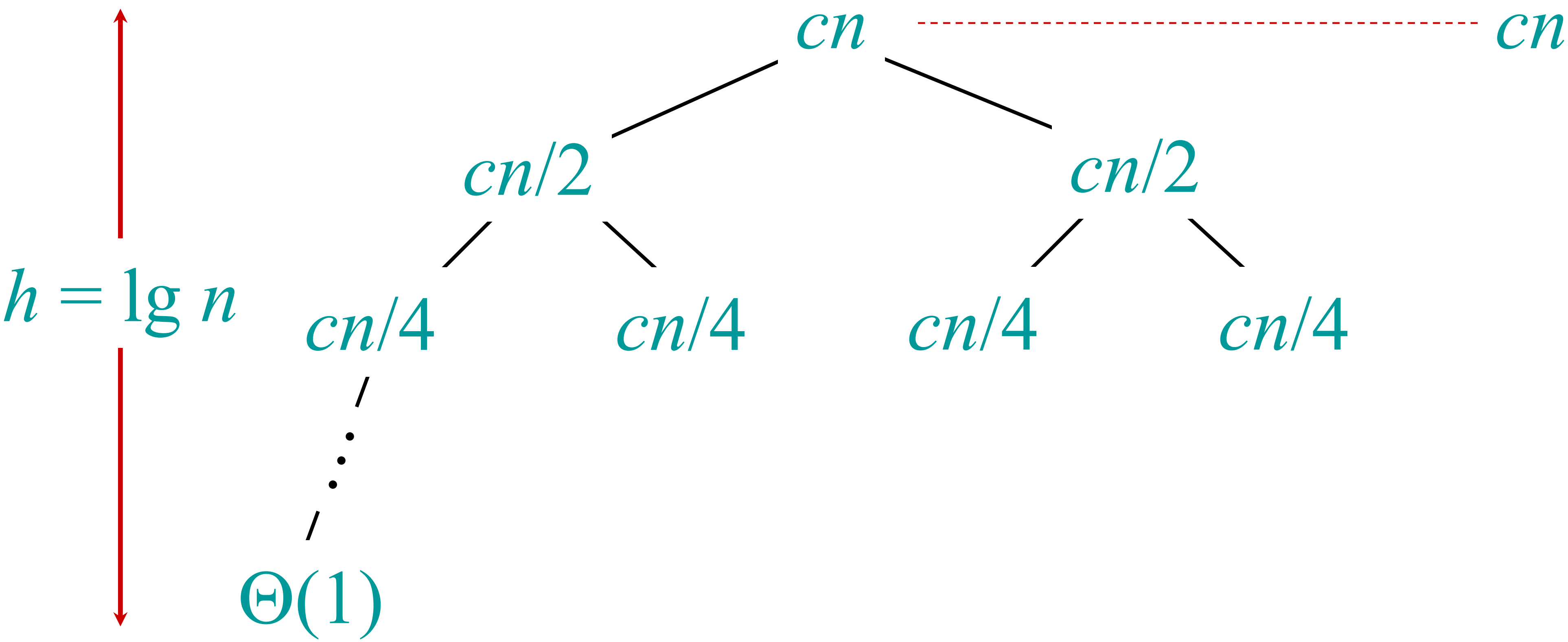
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

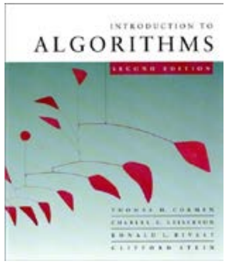





# Recursion tree

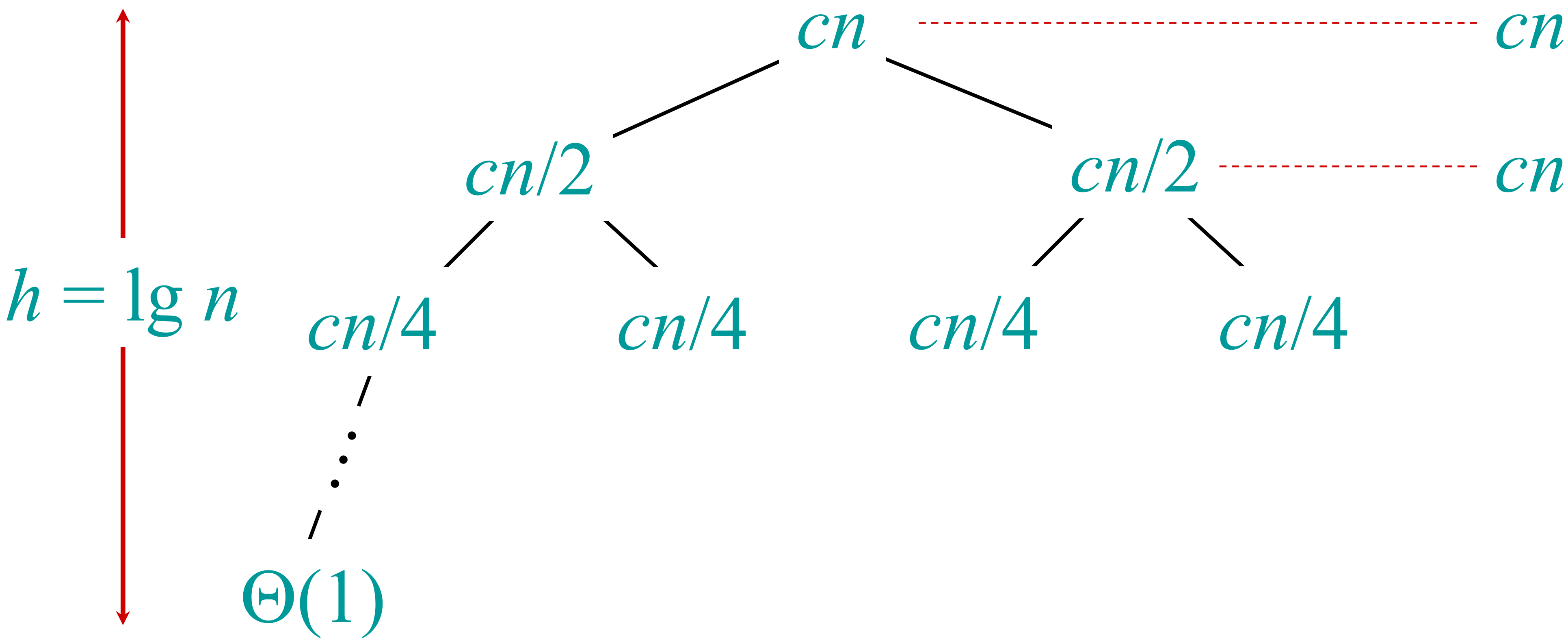
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.





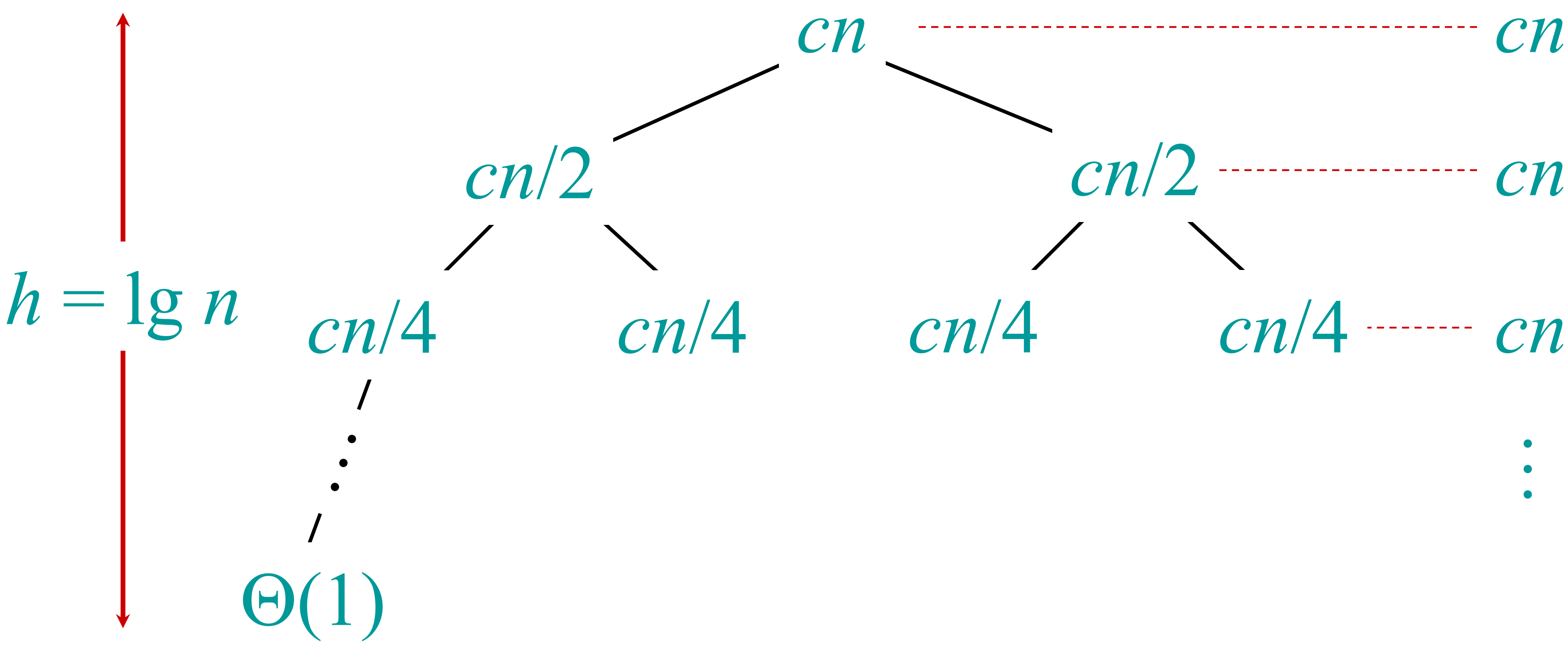
# Recursion tree

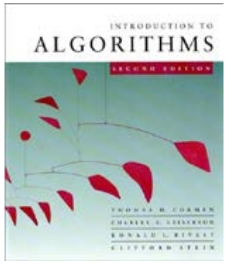
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



# Recursion tree

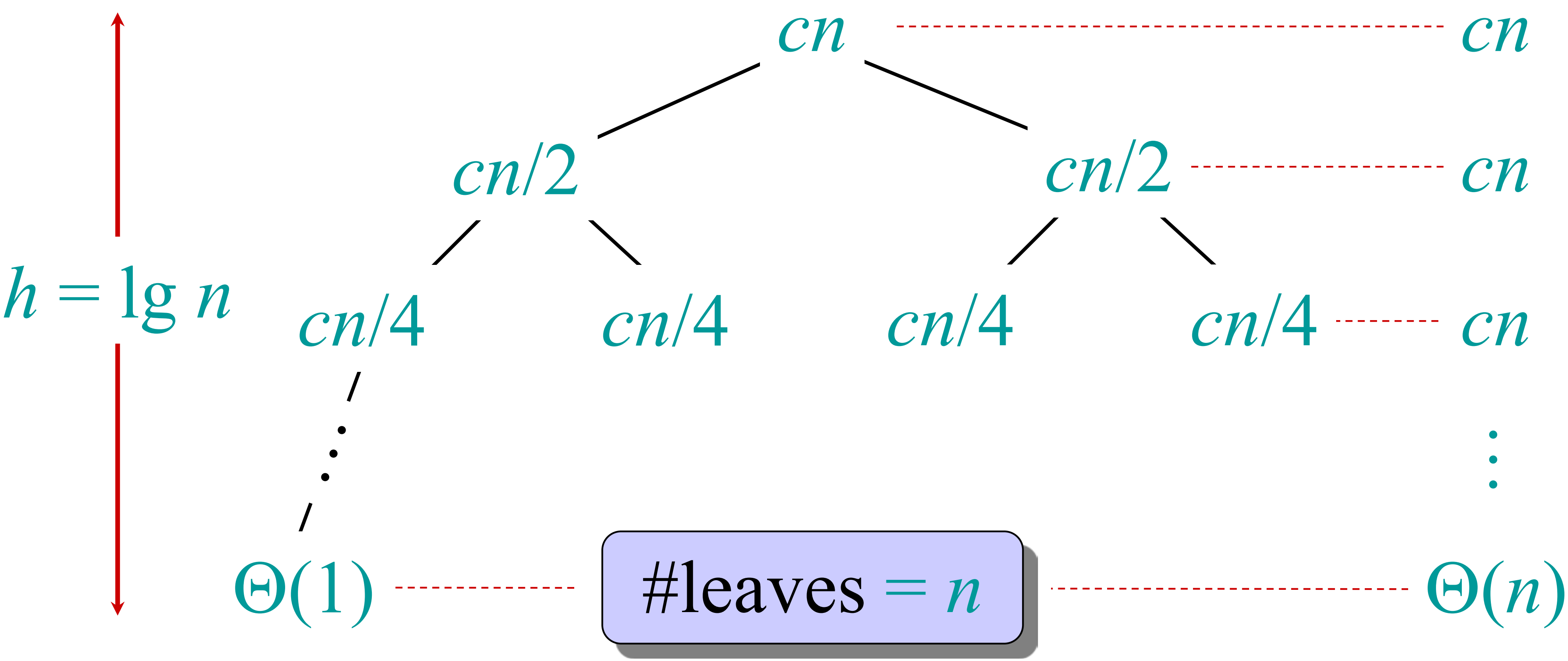
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.



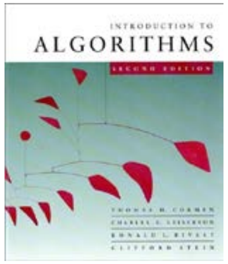



# Recursion tree

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

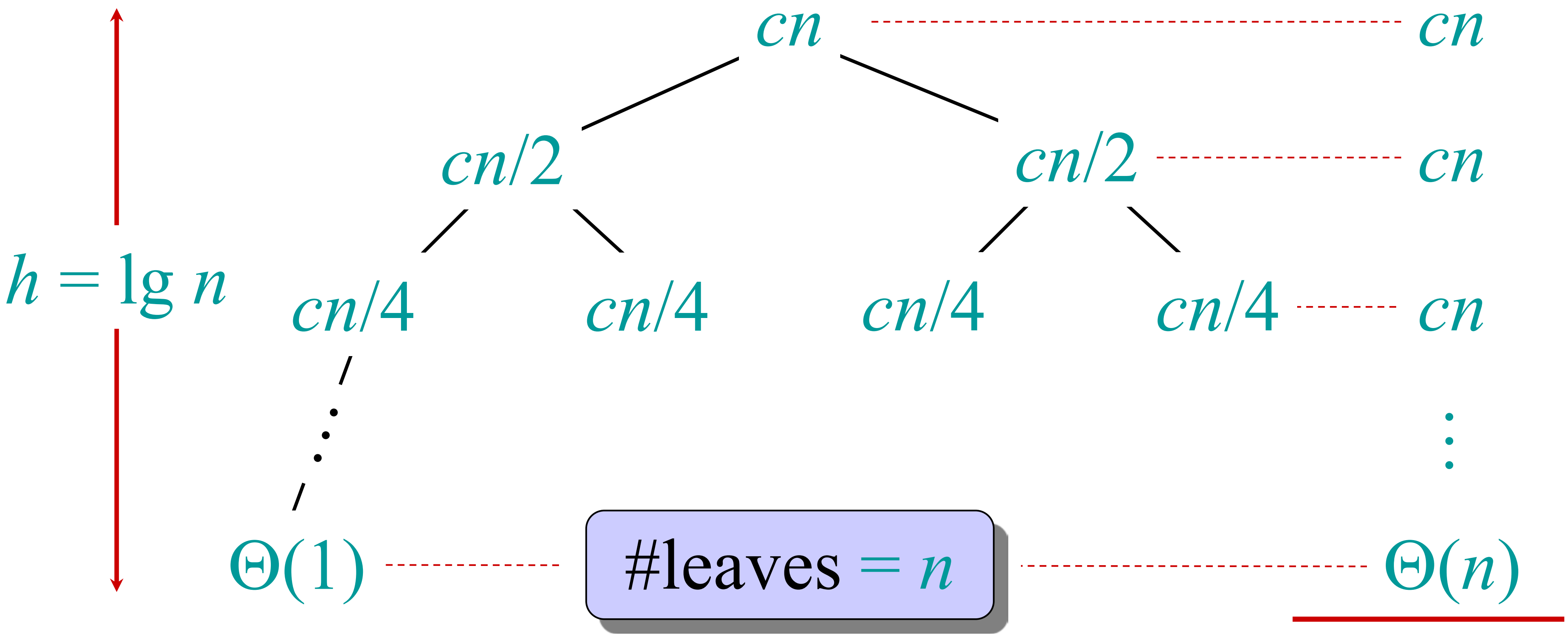




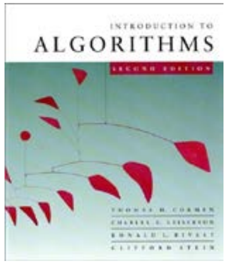

# Recursion tree

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




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




# Conclusions

- $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$.
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for $n > 30$ or so.
- Go test it out for yourself!