Lecture 1: Sorting and Analysis

SORTING PROBLEM

Input: sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers.

Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

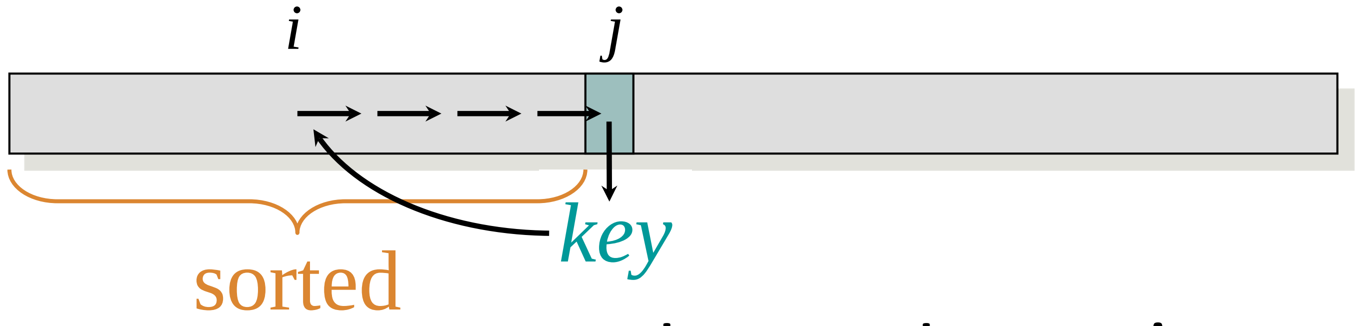
Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

INSERTION SORT

“pseudocode”

```
INSERTION-SORT (A, n)      ⊳ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```



Loop Invariant

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

for $j \leftarrow 2$ **to** n
do $key \leftarrow A[j]$
$i \leftarrow j - 1$
while $i > 0$ and $A[i] > key$
do $A[i+1] \leftarrow A[i]$
$i \leftarrow i - 1$
$A[i+1] = key$

EXAMPLE OF INSERTION SORT

8 **2** 4 9 3 6

for $j \leftarrow 2$ **to** n
do $key \leftarrow A[j]$
$i \leftarrow j - 1$
while $i > 0$ and $A[i] > key$
do $A[i+1] \leftarrow A[i]$
$i \leftarrow i - 1$
$A[i+1] = key$

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

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

do $key \leftarrow A[j]$

$i \leftarrow j - 1$

while $i > 0$ and $A[i] > key$

do $A[i+1] \leftarrow A[i]$

$i \leftarrow i - 1$

$A[i+1] = key$

EXAMPLE OF INSERTION SORT

8 2 4 9 3 6

2 8 4 9 3 6

```
for j ← 2 to n
    do key ← A[ j]
       i ← j – 1
       while  i > 0 and A[i] > key
            do A[i+1] ← A[i]
               i ← i – 1
       A[i+1] = key
```

EXAMPLE OF INSERTION SORT

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

  do $key \leftarrow A[j]$

   $i \leftarrow j - 1$

   while $i > 0$ and $A[i] > key$

     do $A[i+1] \leftarrow A[i]$

     $i \leftarrow i - 1$

   $A[i+1] = key$

8	2	4	9	3	6
2	8	4	9	3	6
2	4	8	9	3	6

EXAMPLE OF INSERTION SORT

```
for j ← 2 to n
    do key ← A[ j]
       i ← j – 1
       while  i > 0 and A[i] > key
           do A[i+1] ← A[i]
              i ← i – 1
       A[i+1] = key
```

8	**2**	4	9	3	6
2	8	**4**	9	3	6
2	4	8	**9**	3	6

EXAMPLE OF INSERTION SORT

```
for j ← 2 to n
    do key ← A[ j]
        i ← j – 1
        while  i > 0 and A[i] > key
            do A[i+1] ← A[i]
               i ← i – 1
        A[i+1] = key
```

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

```
for j ← 2 to n
    do key ← A[ j]
        i ← j – 1
        while   i > 0 and A[i] > key
                do A[i+1] ← A[i]
                i ← i – 1
        A[i+1] = key
```

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

```
for j ← 2 to n
    do key ← A[ j]
       i ← j – 1
       while  i > 0 and A[i] > key
           do A[i+1] ← A[i]
              i ← i – 1
       A[i+1] = key
```

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

for $j \leftarrow 2$ **to** n
  do $key \leftarrow A[j]$
    $i \leftarrow j - 1$
    while $i > 0$ and $A[i] > key$
      do $A[i+1] \leftarrow A[i]$
      $i \leftarrow i - 1$
    $A[i+1] = key$

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

for $j \leftarrow 2$ **to** n
    do $key \leftarrow A[j]$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do $A[i+1] \leftarrow A[i]$
            $i \leftarrow i - 1$
        $A[i+1] = key$

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

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

RUNNING TIME

- Parameterize the running time by the size of the input, since short sequences are easier to sort than long ones.
- $T(n)$ = runtime of algorithm on an input of size n.

RUNNING TIME

- The running time depends on the input: an already sorted sequence is easier to sort.
- ***Input:*** 2 4 6 7 8 9
- ***Input:*** 8 7 6 4 3 2
- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

KINDS OF ANALYSES

Worst-case: (usually)

- $T(n)$ = maximum time of algorithm on any input of size n.

Average-case: (sometimes)

- $T(n)$ = expected time of algorithm on any input of size n.

Best-case: (bogus)

- Cheat with a slow algorithm that works fast on *some* input.

MACHINE-INDEPENDENT TIME

What is insertion sort's worst-case time?

- It depends on the speed of our computer

Asymptotic Analysis:

- Ignore machine-dependent constants.
- Look at ***growth*** of $T(n)$ as $n \longrightarrow \infty$.

ASYMPTOTIC NOTATIONS

O-NOTATION

Math:

$O(g(n)) = \{ f(n)$:there exist positive constants c and n_0

such that $f(n) \leq c\, g(n)$ for all $n \geq n_0 \}$

Example: $100n^2 + 20n + 5 \leq 100n^2 + 20n + 5n^2$

$= 125n^2$

$= O(n^2)$

So $100n^2 + 20n + 5$ is $O(n^2)$ for $n_0 = 1$, $c=125$

Exercise: (i) $100n+5$ (ii) $3n^3 - 2n^2 + 5$

Ω-NOTATION

Math:

$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $f(n) \geq c\, g(n)$ for all $n \geq n_0$ $\}$

- Example : $3n^3 - 20n^2 + 5 \geq 3n^3 - 20n^2$

$$\geq 3n^3 - n^3 \text{ for } n \geq 20$$

$$= 2n^3$$

$$= \Omega(n^3)$$

So $3n^3 - 20n^2 + 5$ is $\Omega(n^3)$ for $n_0 = 20$, c=2

Excercise: (i) 100n+5 (ii) $100n^2 + 20n+5$

Θ-NOTATION

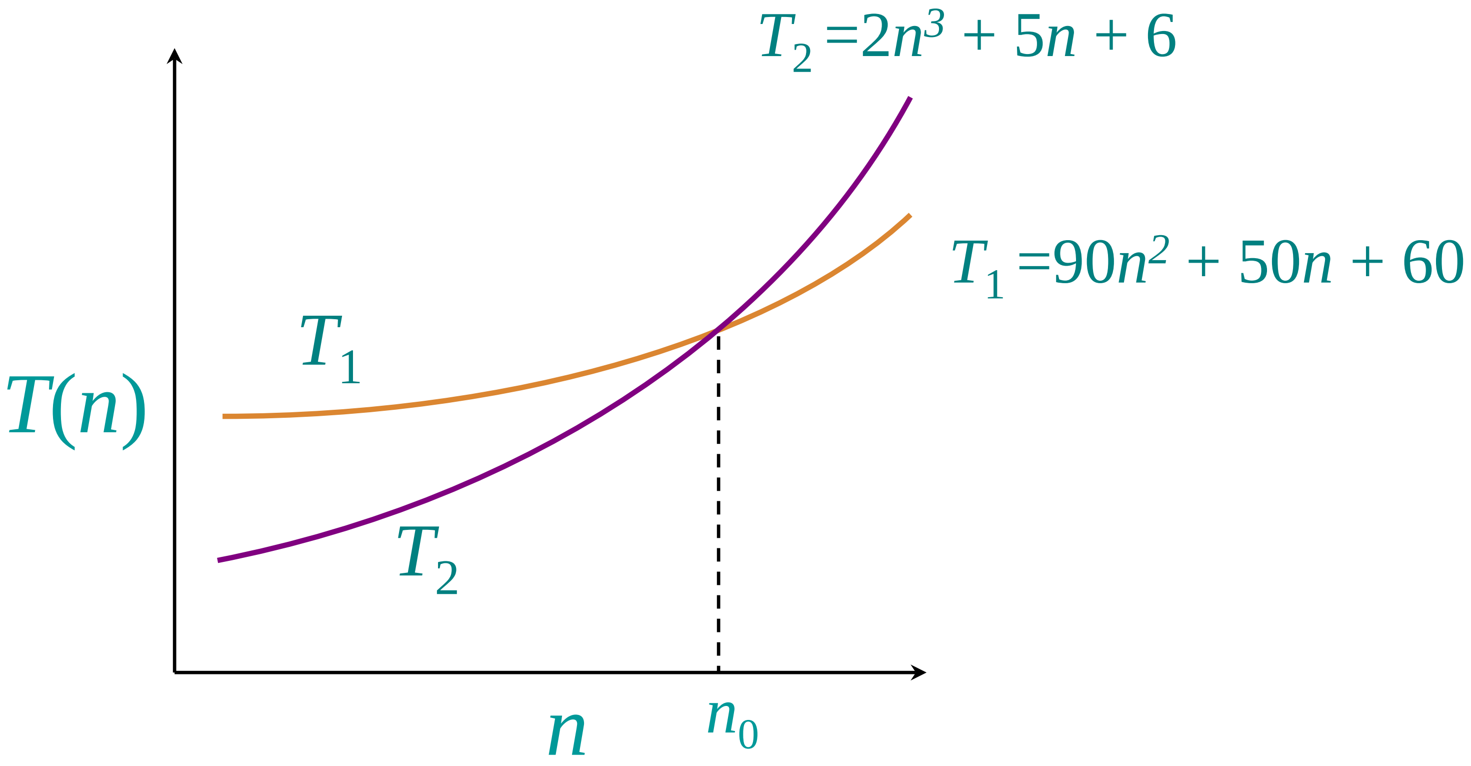
Math:

$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$
- Exercise: (i) 100n+5 (ii) $3n^3$ - $2n^2$ +5

ASYMPTOTIC PERFORMANCE

When n gets large enough, a $\Theta(n^2)$ algorithm *always* beats a $\Theta(n^3)$ algorithm.



INSERTION SORT ANALYSIS

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta\left(n^2\right)$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta\left(n^2\right)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

INSERTION SORT ANALYSIS

```
for j ← 2 to n
    do key ← A[j]
       i ← j – 1
       while i > 0 and A[i] > key
           do A[i+1] ← A[i]
              i ← i – 1
       A[i+1] = key
```

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

Average case: All permutations equally likely.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

Is insertion sort a fast sorting algorithm?

- Moderately so, for small n.
- Not at all, for large n.

MERGE SORT

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

To sort n numbers:

1. If $n = 1$, done.
2. Recursively sort $A[\, 1 \,.\,.\, n/2 \,]$ and $A[\, n/2+1 \,.\,.\, n \,]$.
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	

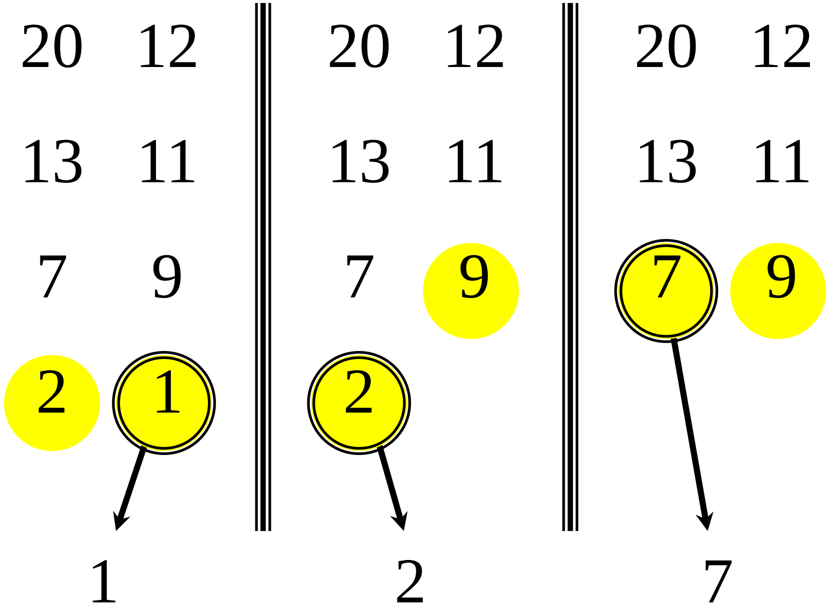
1

2

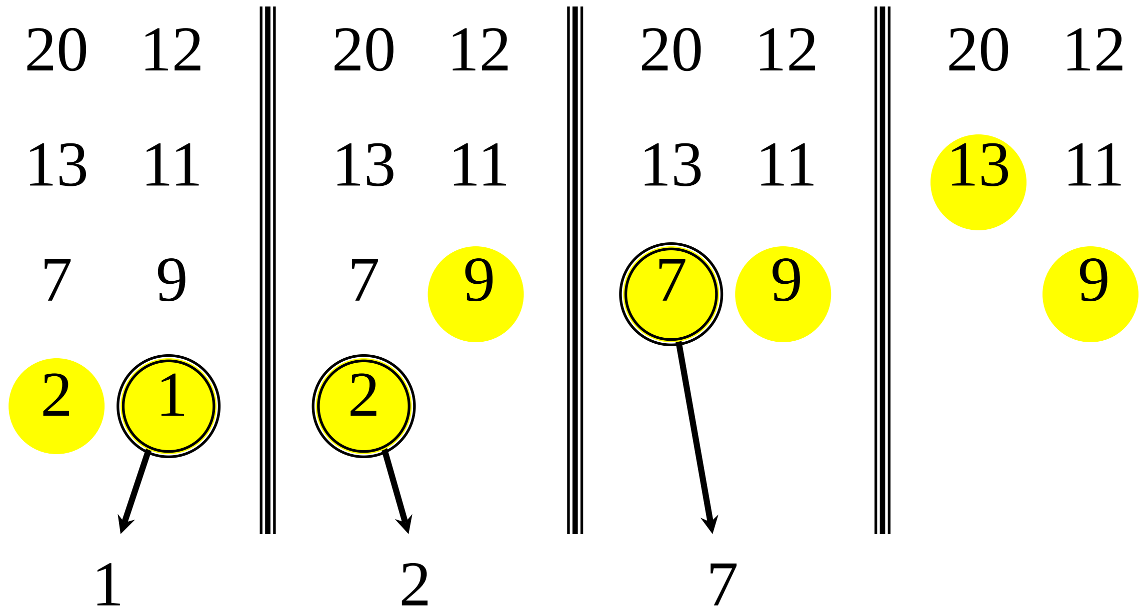
MERGING TWO SORTED ARRAYS

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

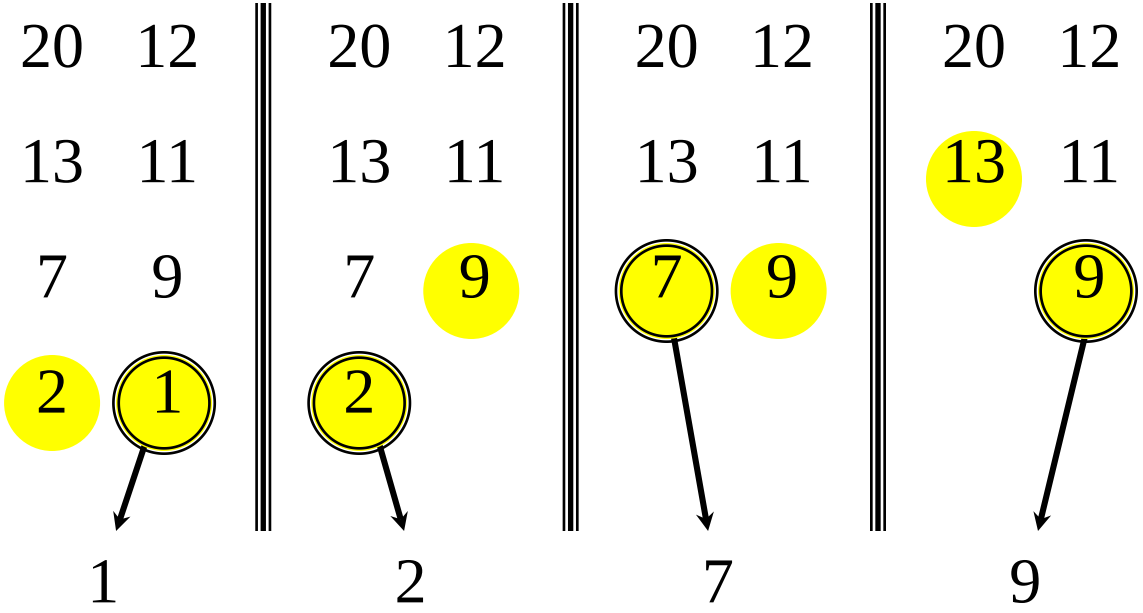
MERGING TWO SORTED ARRAYS



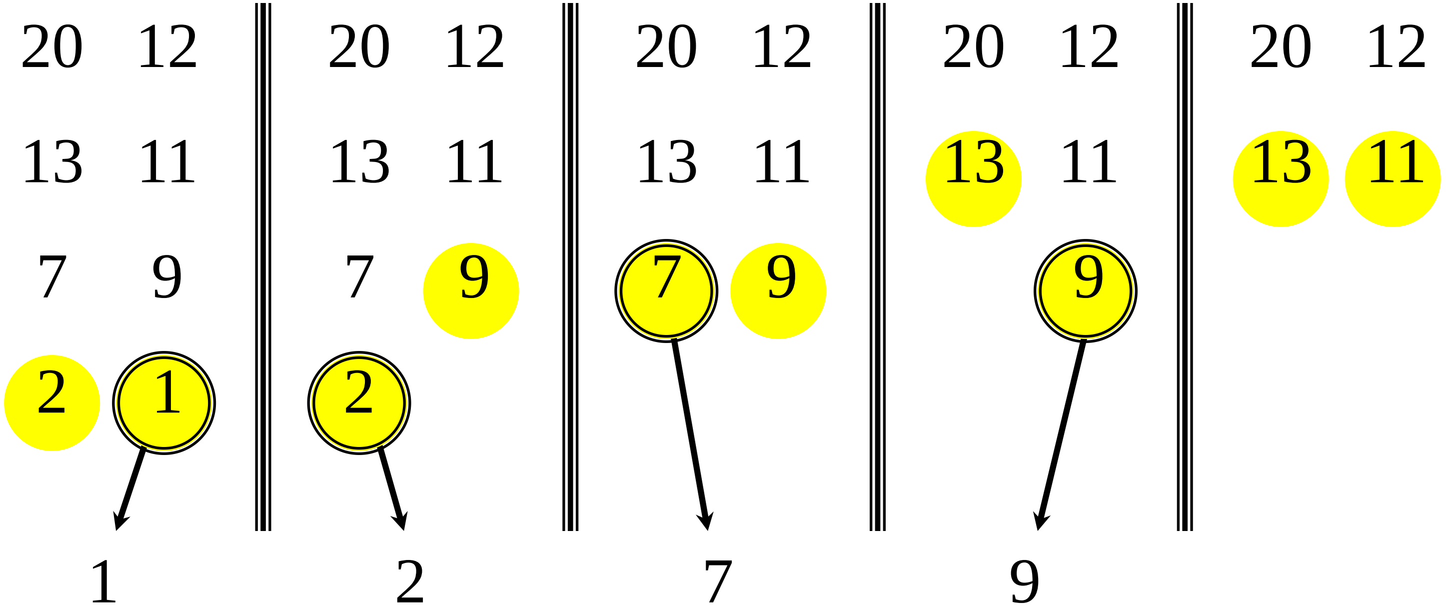
MERGING TWO SORTED ARRAYS



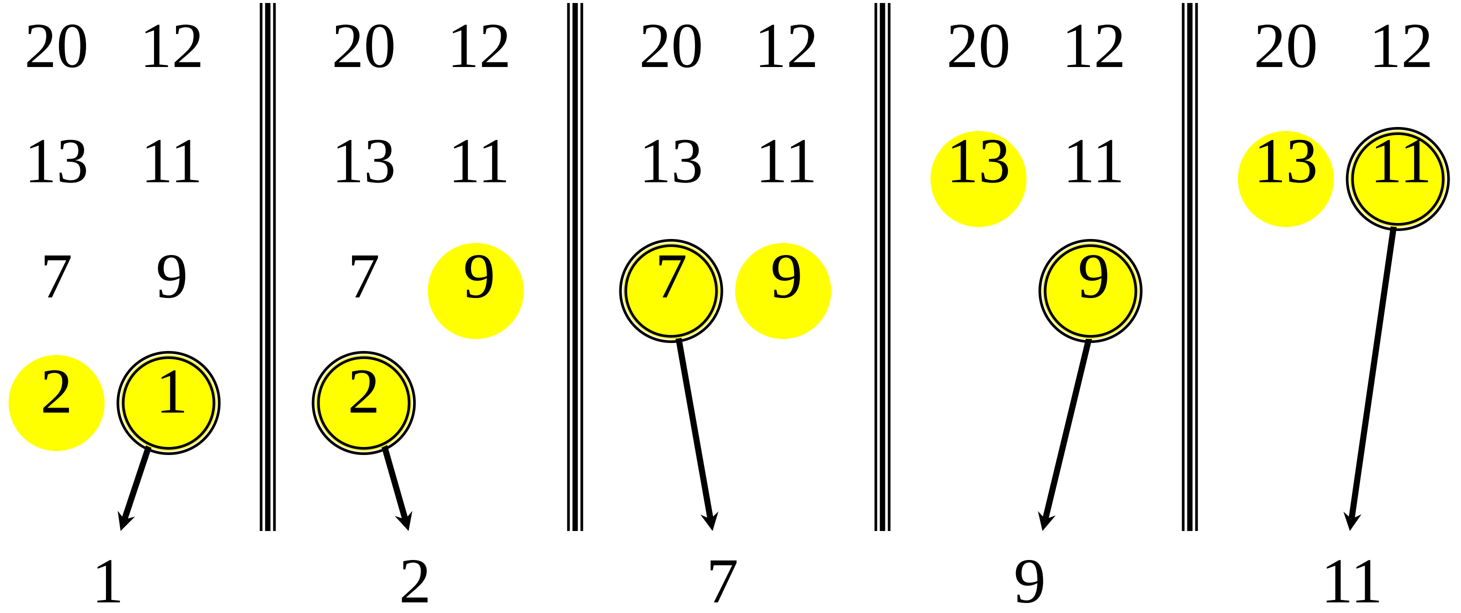
MERGING TWO SORTED ARRAYS



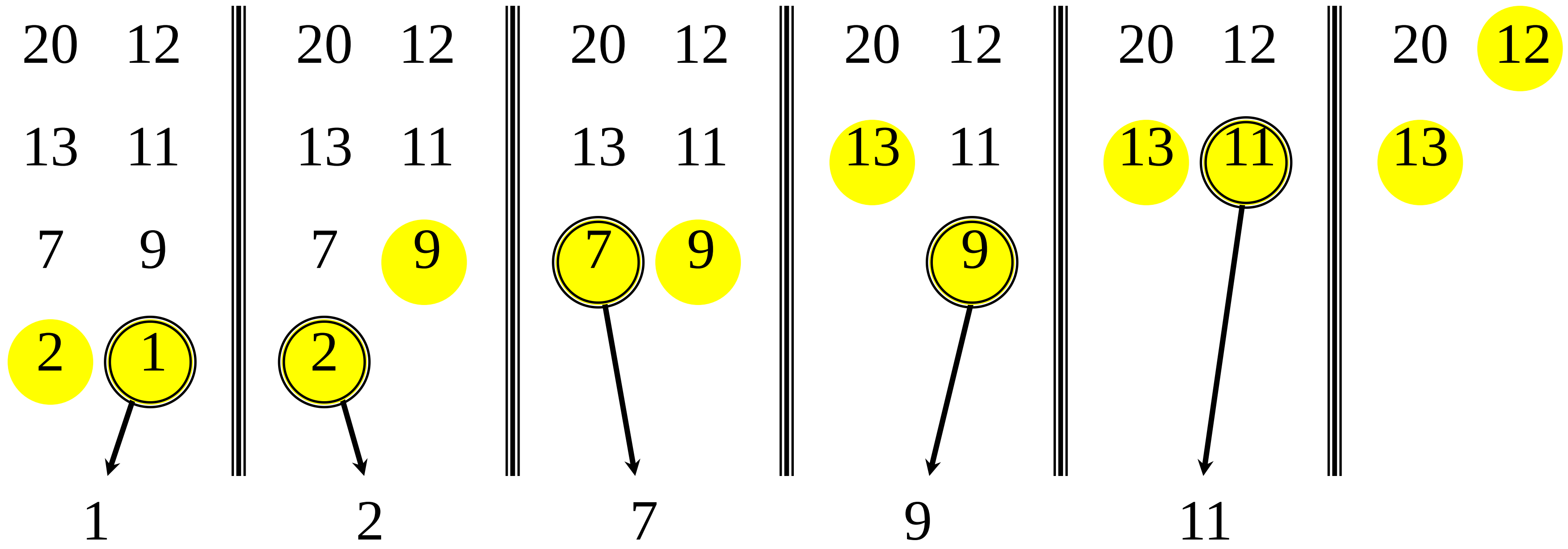
MERGING TWO SORTED ARRAYS



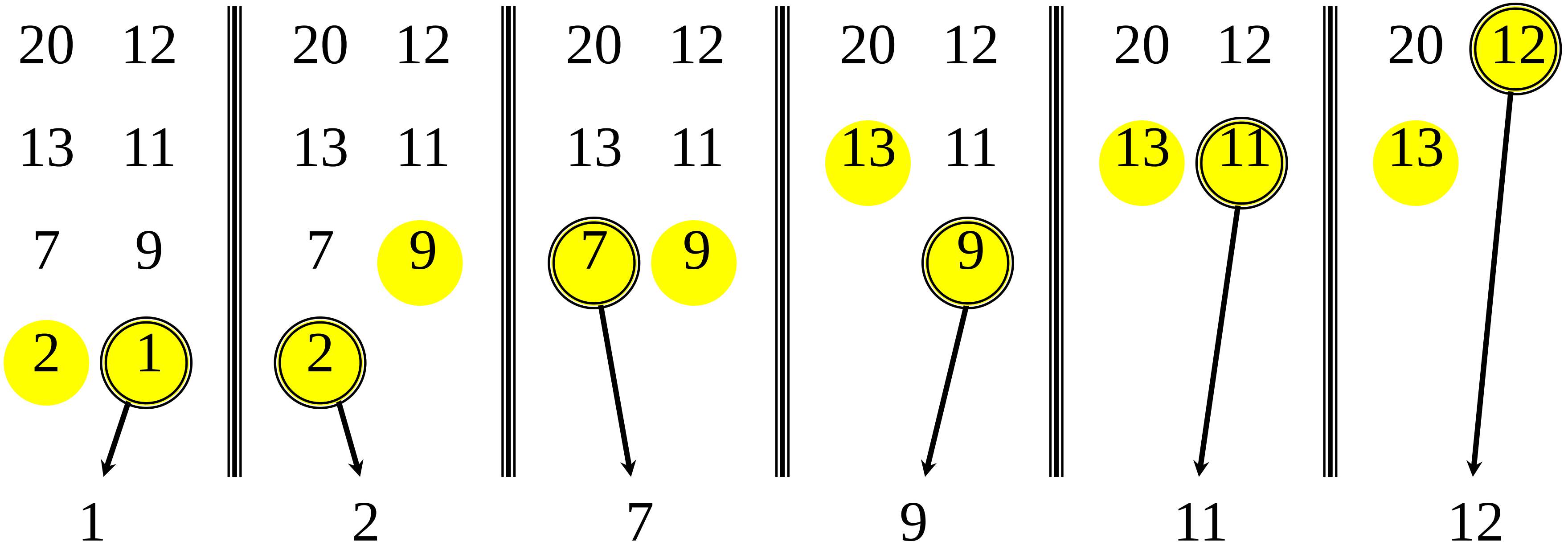
MERGING TWO SORTED ARRAYS



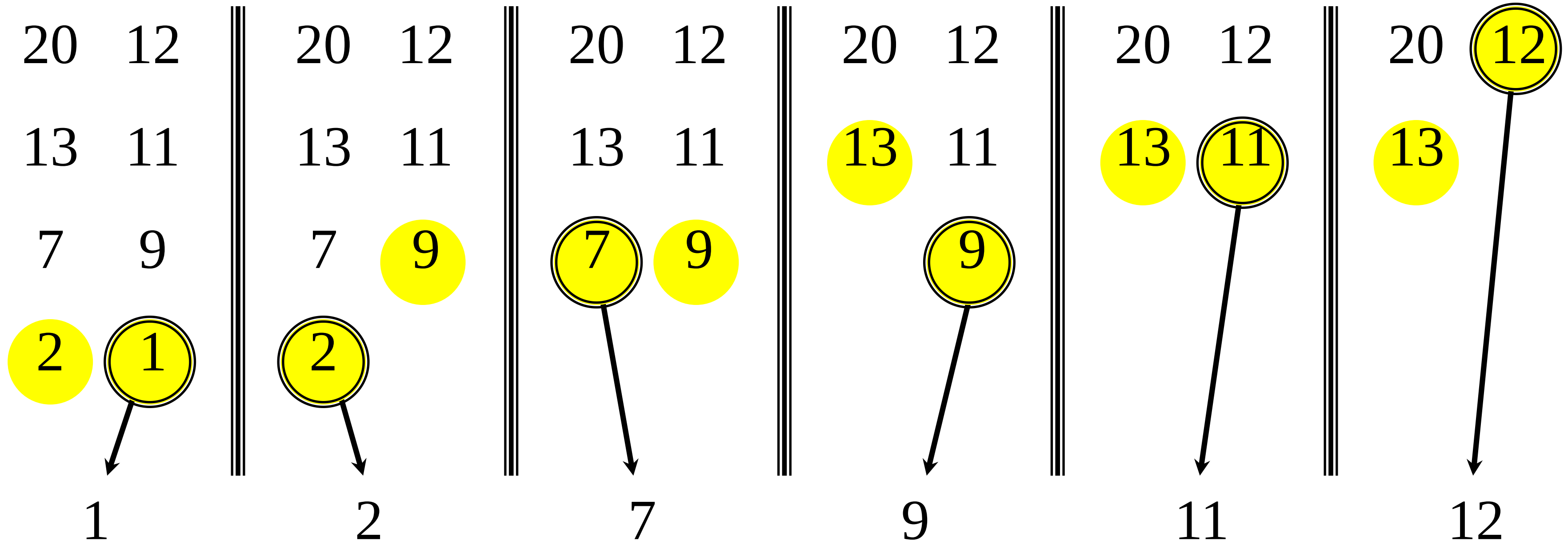
MERGING TWO SORTED ARRAYS



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, n) $\triangleright$ $A[1 \,..\, n]$

$T(n)$	To sort n numbers:
$\Theta(1)$	1. If $n = 1$, done.
$2T(n/2)$	2. Recursively sort $A[\,1 \,..\, n/2\,]$ and $A[\,n/2+1 \,..\, n\,]$.
$\Theta(n)$	***3. "Merge"*** the 2 sorted lists

Recurrence:

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

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.

$$T(n)$$

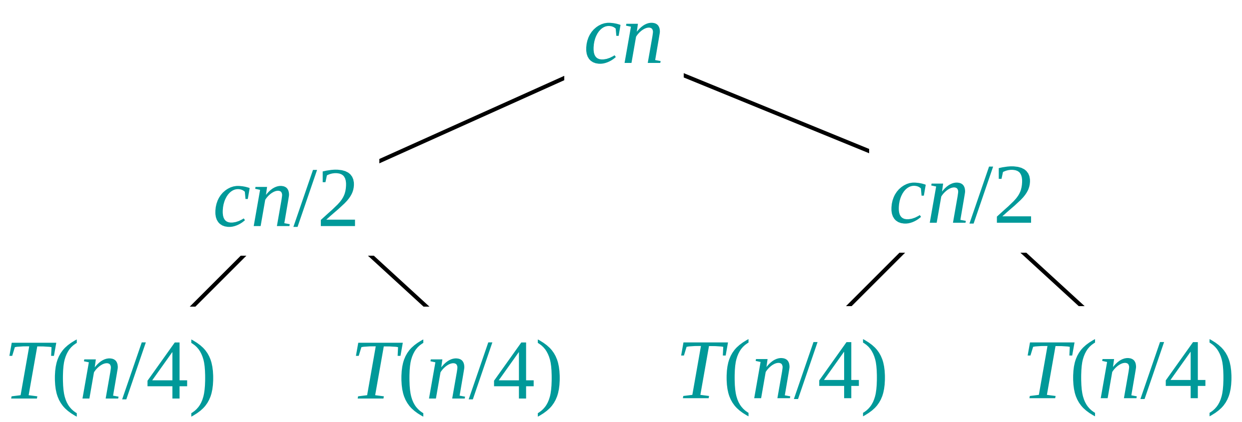
RECURSION TREE

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

$$
\begin{array}{ccc}
 & cn & \\
\swarrow & & \searrow \\
T(n/2) & & T(n/2)
\end{array}
$$

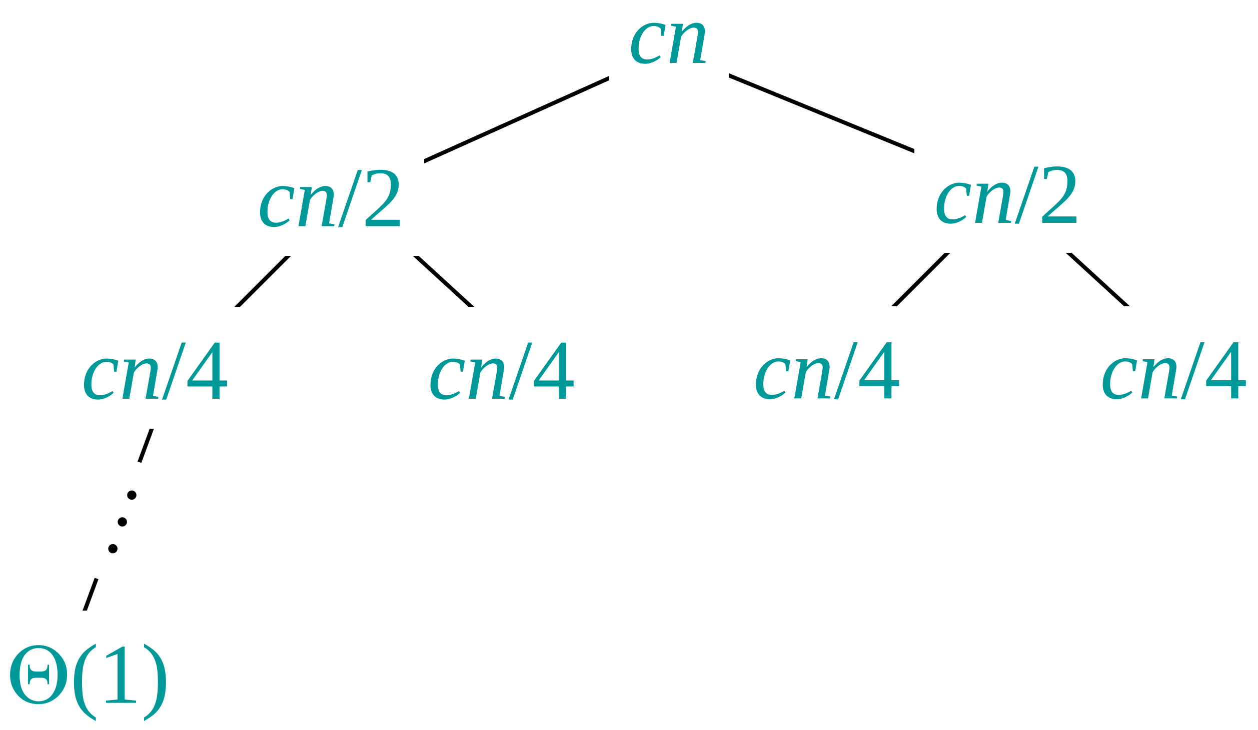
RECURSION TREE

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



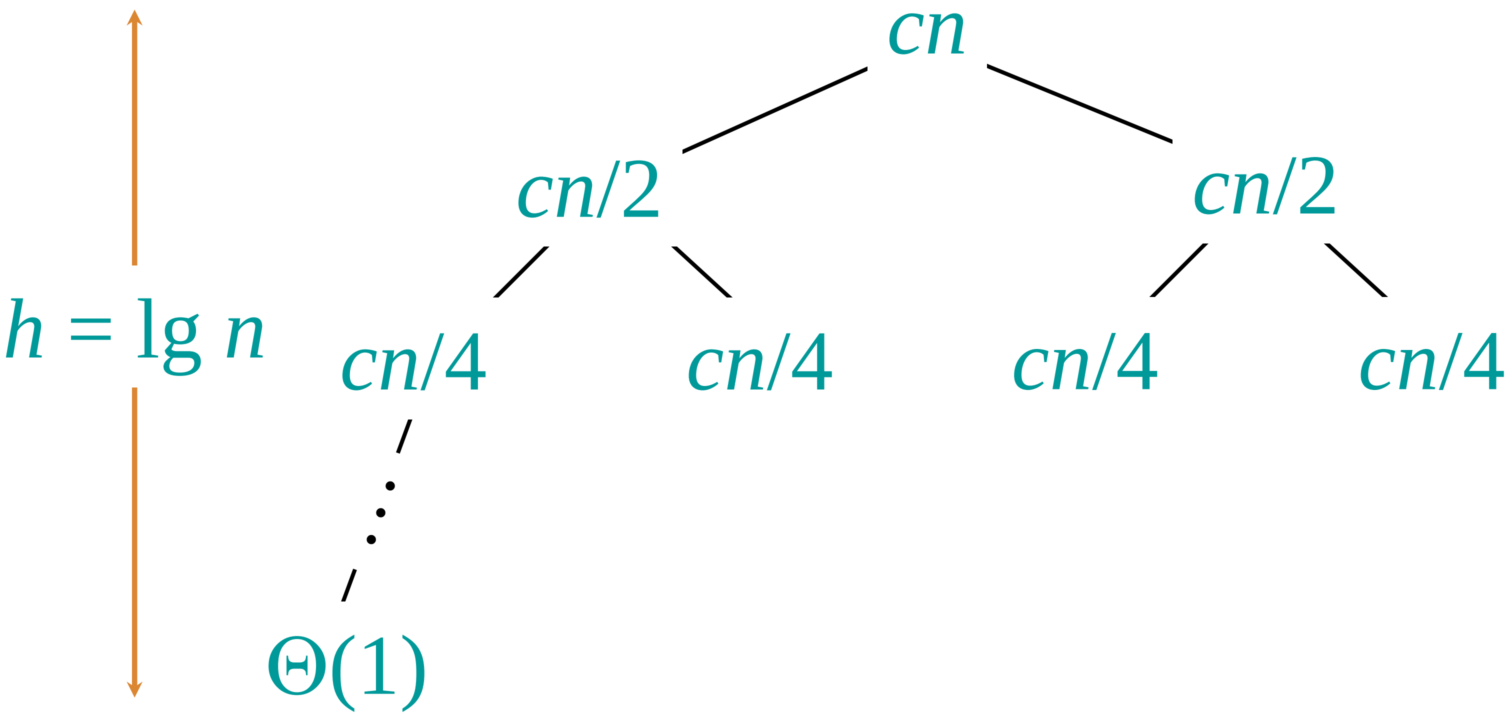
RECURSION TREE

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



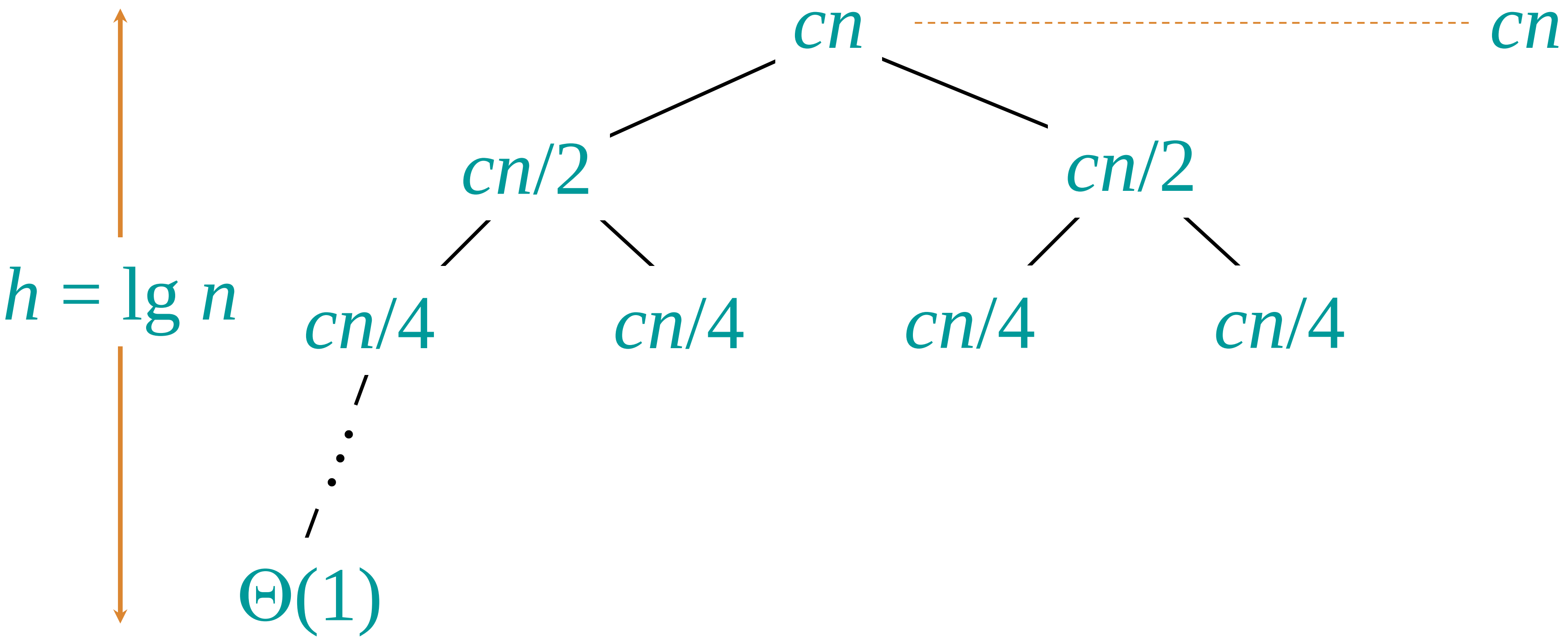
RECURSION TREE

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



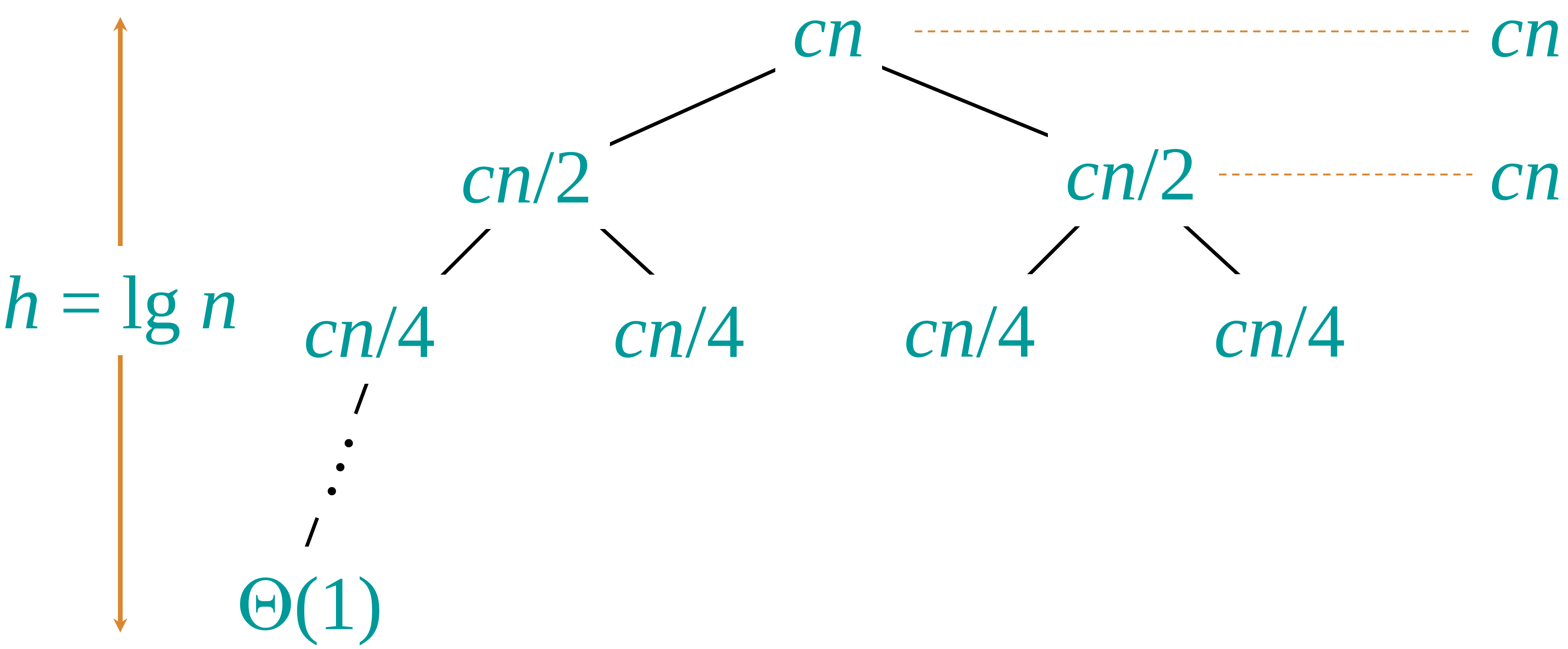
RECURSION TREE

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



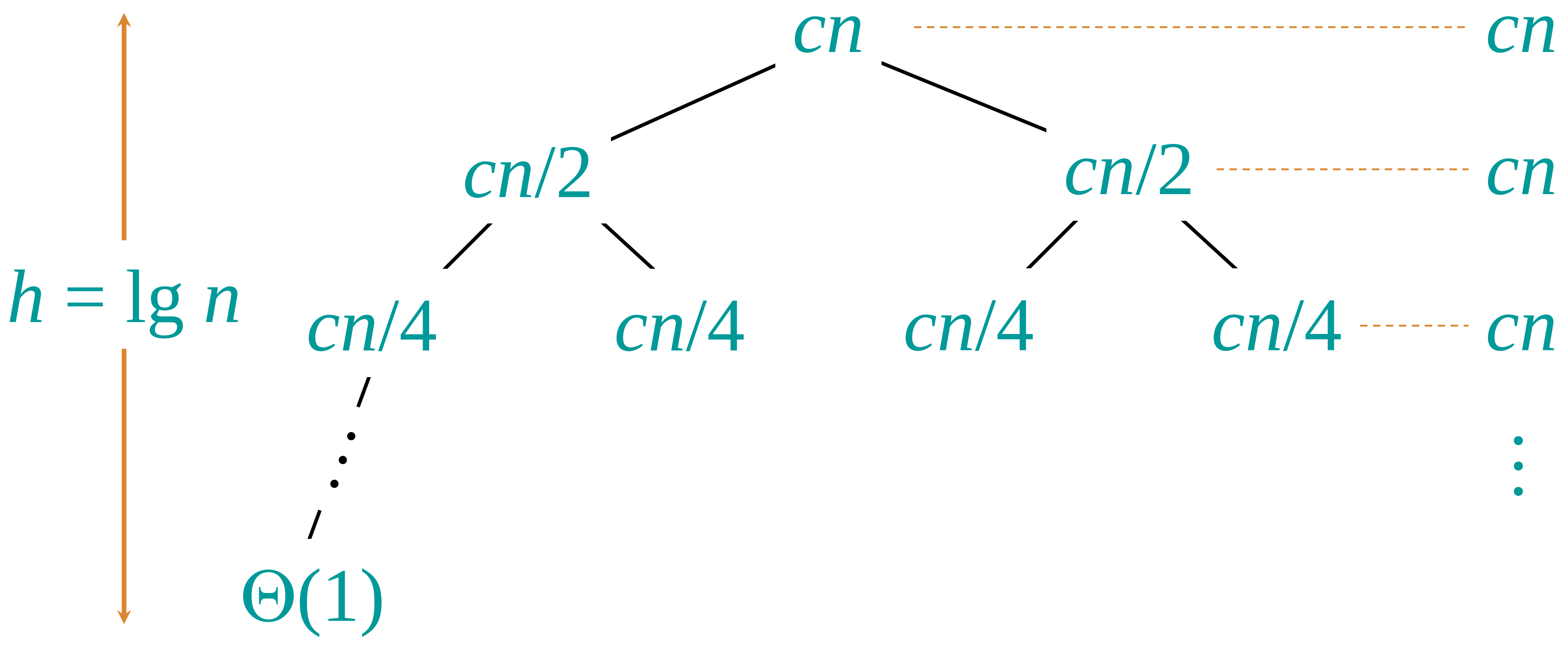
RECURSION TREE

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



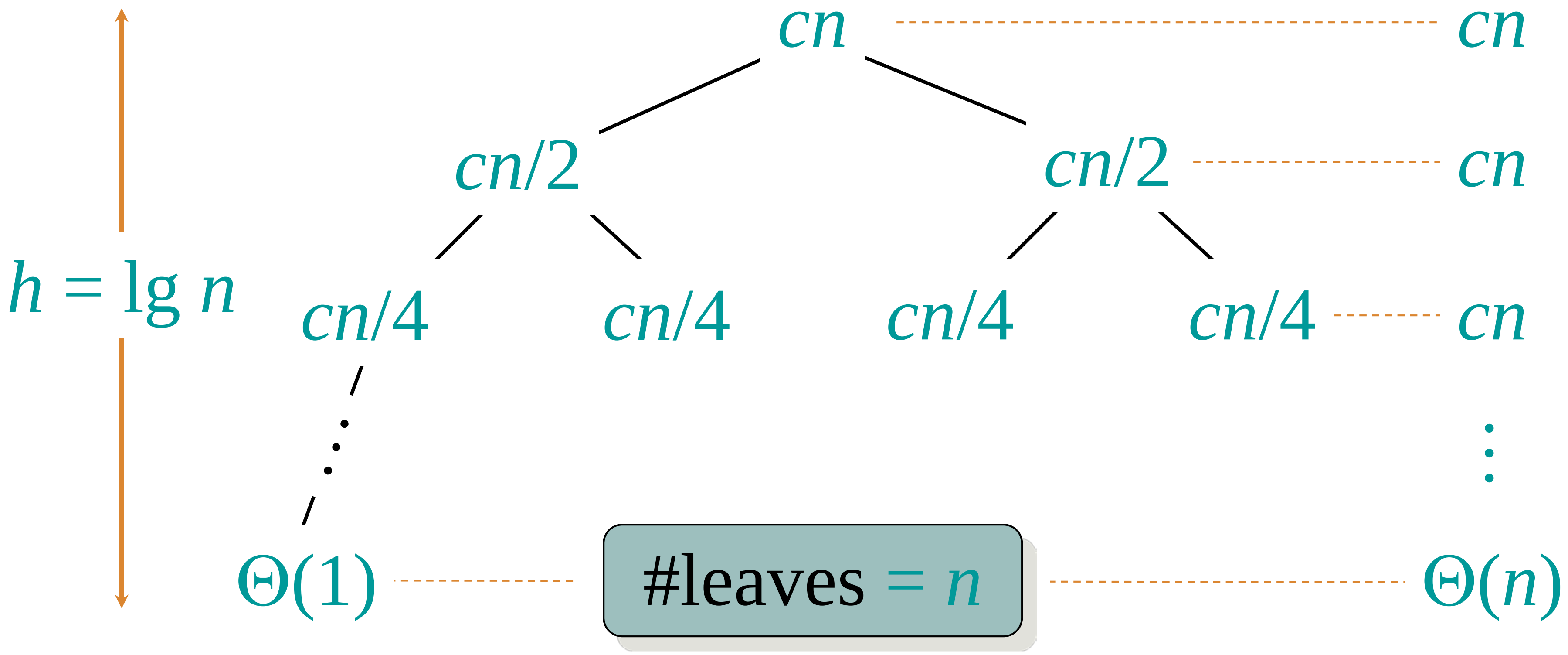
RECURSION TREE

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



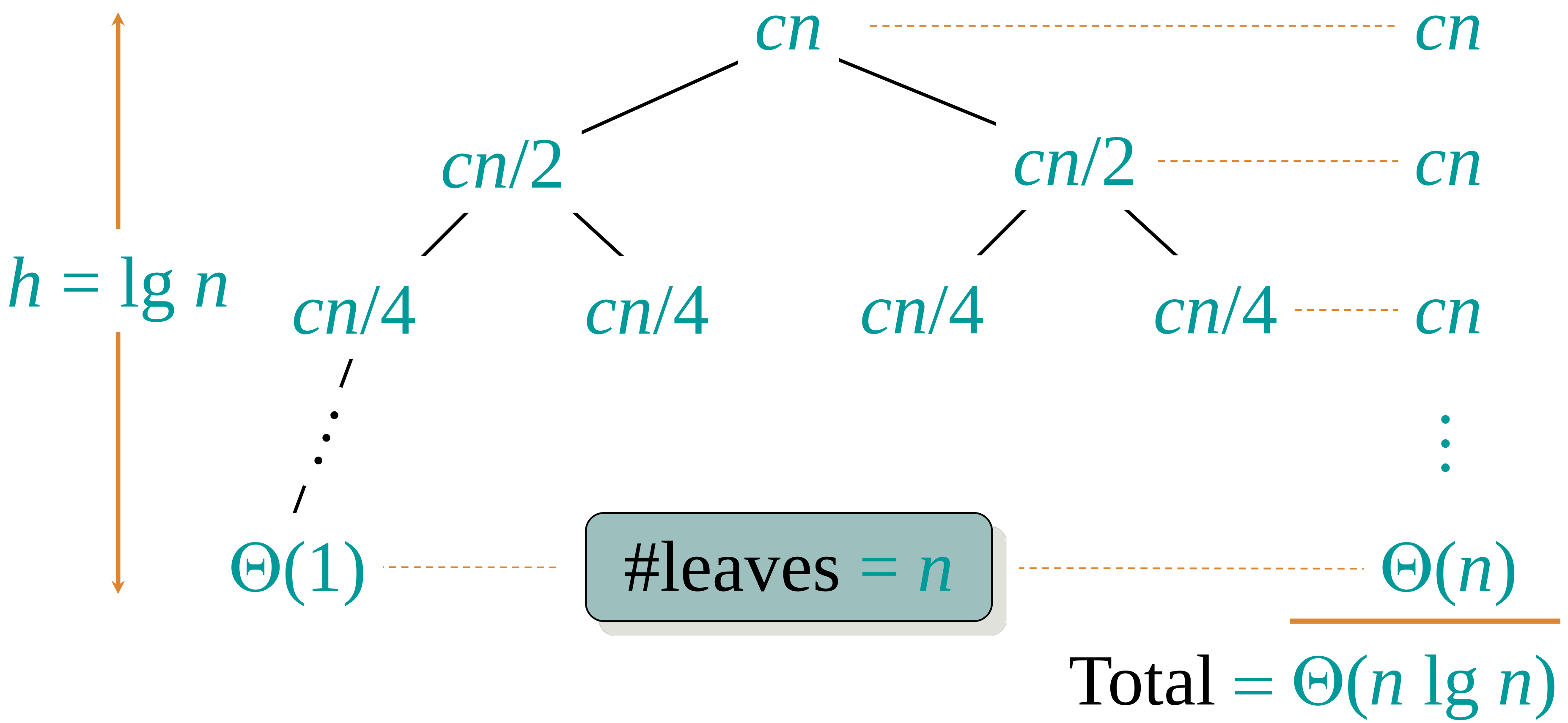
RECURSION TREE

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.



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!