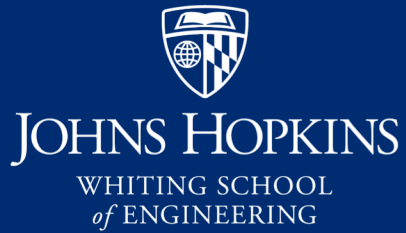


Chapter 2
Getting Started

Outline

- Selection Sort
- Insertion Sort
- Asymptotic Analysis
- Merge Sort
- Recurrences

Analyzing Selection Sort

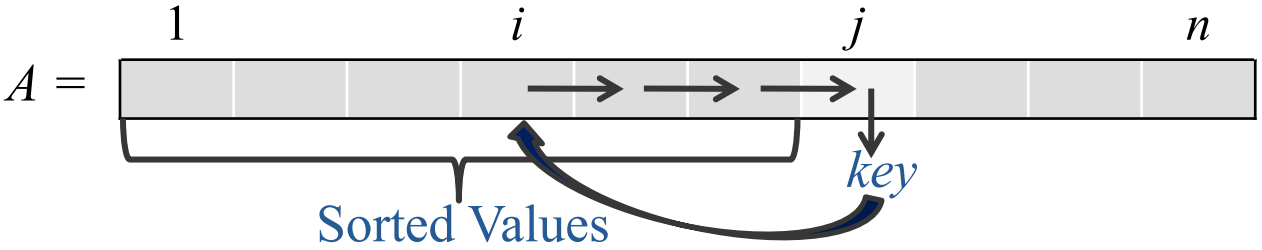
- SelectionSort, finds the largest number in the list and places it last. It then finds the largest number remaining and places it next to last, and so on until the list contains only a single number. The number of comparisons is n-1 for the first iteration, n-2 for the second iteration, and so on. Let $T(n)$ denote the complexity for selection sort and c denote the total number of other operations such as assignments and additional comparisons in each iteration. So,

$$T(n) = (n - 1) + c + (n - 2) + c \cdots + 2 + c + 1 + c = n^2/2 - n/2 + cn$$

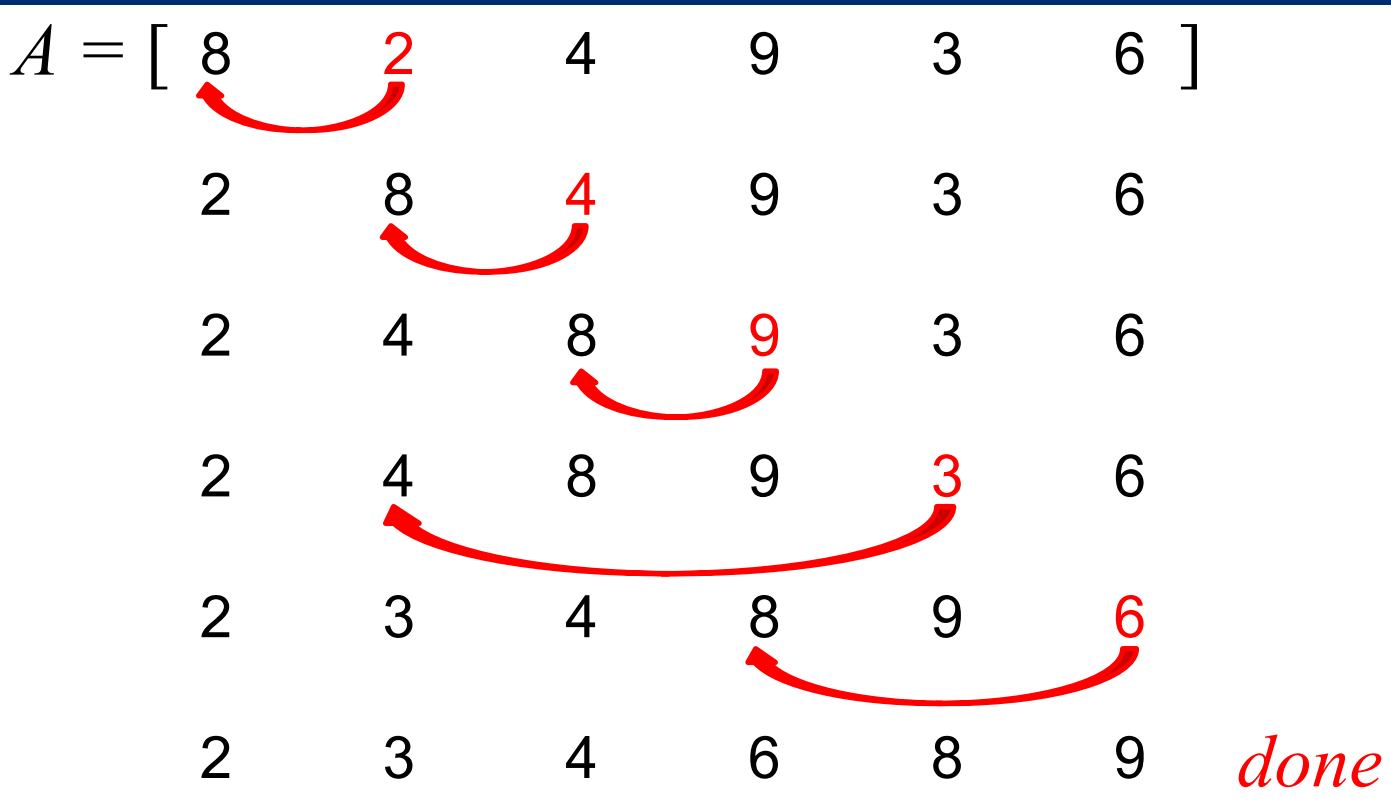
- Ignoring constants and smaller terms, the complexity of the selection sort algorithm is $O(n^2)$.

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
```



INSERTION-SORT Example



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.

Types 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 over all inputs of size n.
- Need assumption of statistical distribution of inputs.

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:
- relative speed (on the same machine),
- absolute speed (on different machines).

BIG IDEA:

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

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 $n \geq n_0\}$

Engineering:

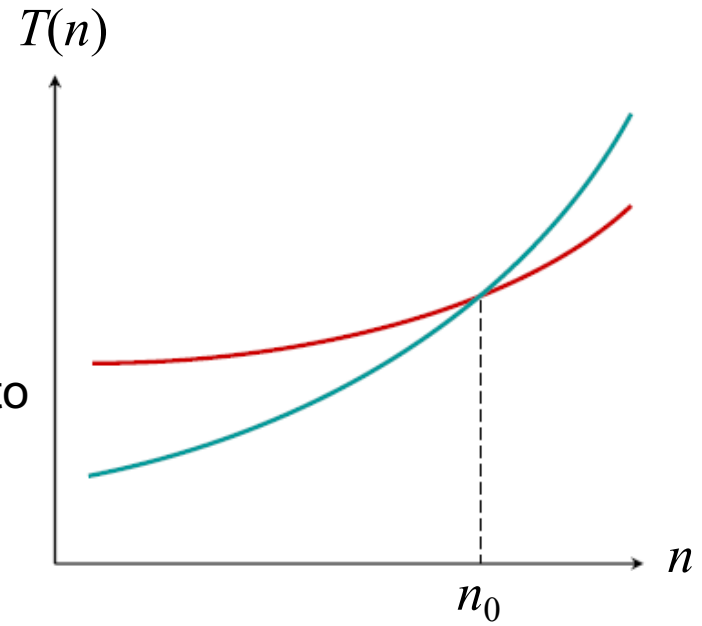
Drop low-order terms; ignore leading constants.

Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Asymptotic performance

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

- We shouldn't ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.



Insertion sort analysis

Is insertion sort a fast sorting algorithm?

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

Worst case: Input reverse sorted.

$$T(n) = \Sigma^{n}_{j=2}\Theta(n) = \Theta(n^2)$$ [arithmetic series]

Average case: All permutations equally likely.

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

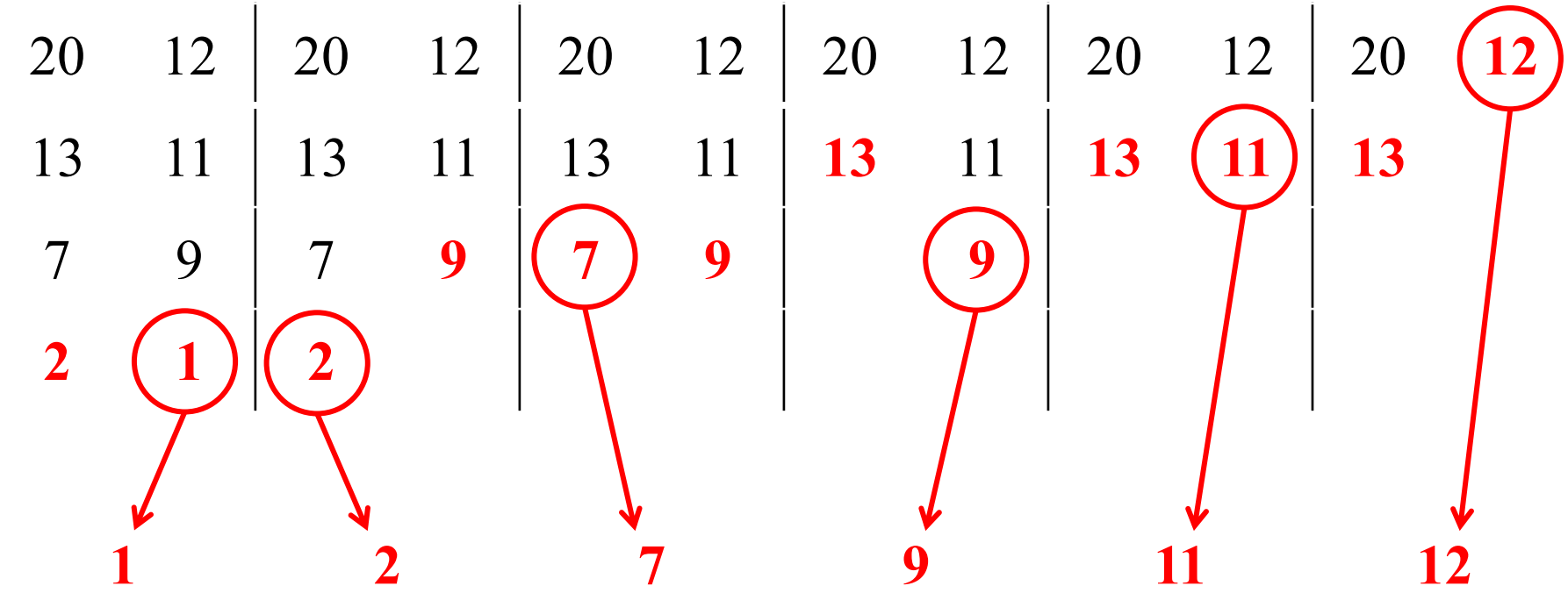
Merge sort

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.

The subroutine used by MERGE-SORT is MERGE

Merging 2 sorted arrays



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

Analyzing MERGE-SORT

	Cost
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 2 sorted lists.	$\Theta(n)$

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

Recurrence for MERGE-SORT

- 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.

- Chapter 3 provide several ways to find a good upper bound on $T(n)$.

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

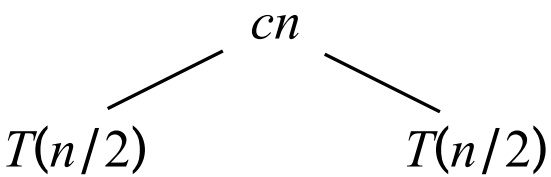
Recursion Tree

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

$T(n)$

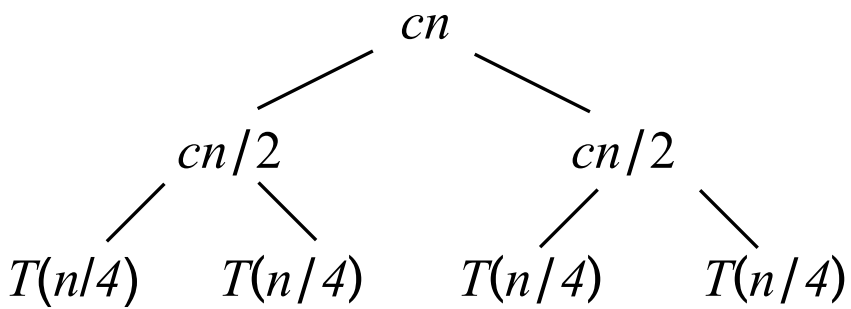
Recursion Tree

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



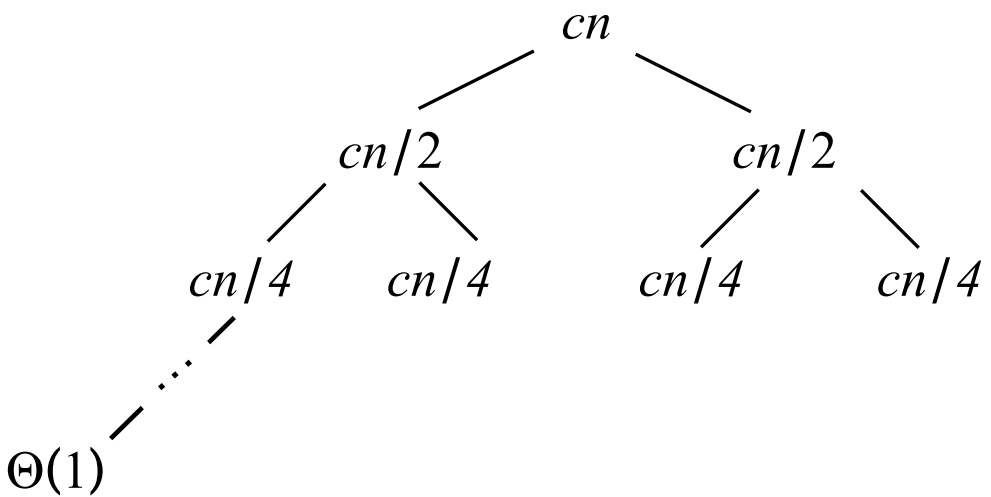
Recursion Tree

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



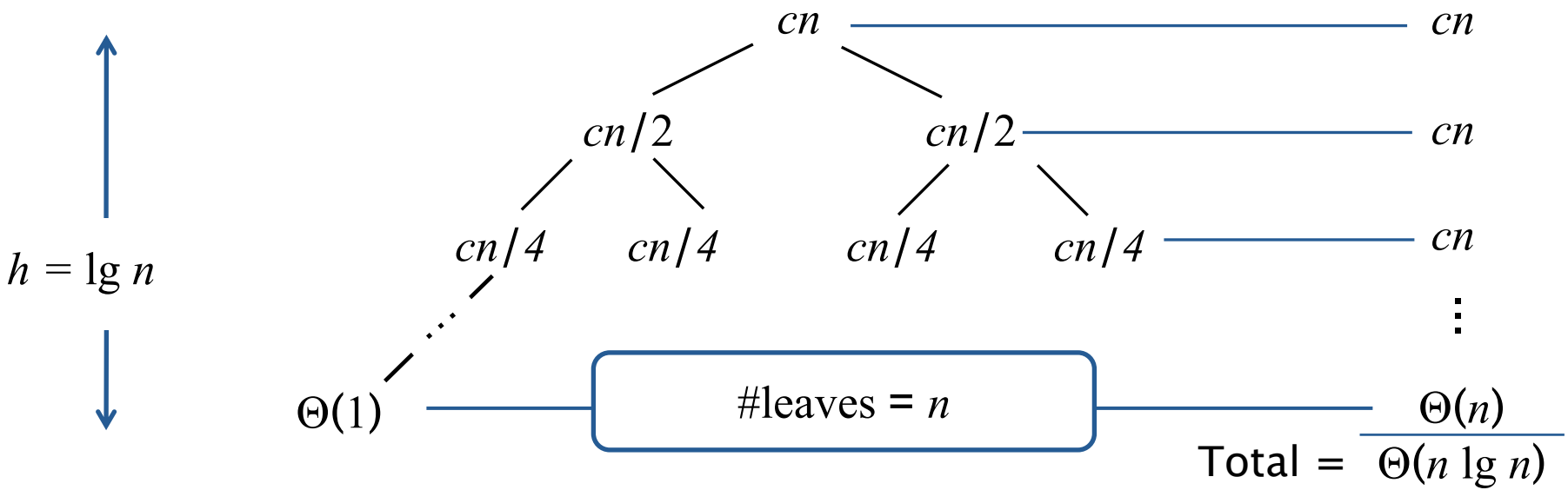
Recursion Tree

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.*



Conclusions

- $\Theta(n \lg n)$ grows slower 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.

References

- T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, "*Introduction to Algorithms*, *Third Edition*", 2009
- MIT Prof. Erik D. Demaine and MIT Prof. Charles E. Leiserson, Lecture Note, Slides and Videos
- Y. D. Liang, "Introduction to JAVA Programming, Comprehensive Version, Eighth Edition", 2011