CMP461: Algorithms

Lecture 01: Algorithm Analysis

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

Agenda

- Analysis of Algorithms
- Insertion Sort
- Asymptotic Analysis
- Merge Sort
- Recurrences

Acknowledgment
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson

Algorithms

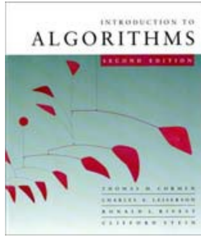
Algorithm

A computational procedure that takes some values as input and produces some values as output

Algorithm Analysis

Determining the *resources* required by the algorithm as a function of the input size.

Resources include *space* and *time*.



Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- Performance is the ***currency*** of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

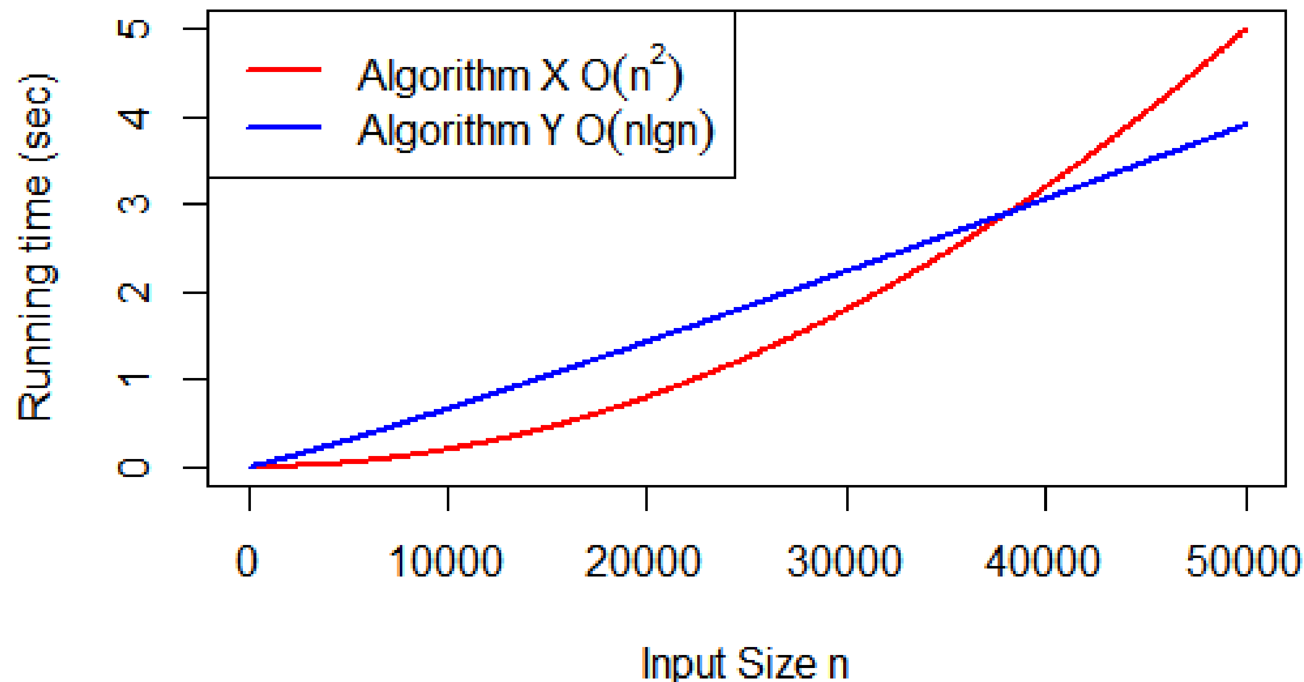
Running time

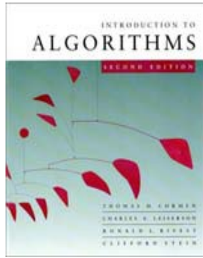
- Assume:
  - Algorithm X takes time $2n^2$ (written by best programmer) running on machine with 1000 MIPS
  - Algorithm Y takes time $50n \lg n$ (written by worst programmer) running on machine with 10 MIPS
- Running time for 10^6 numbers
  - Algorithm X takes 2000 seconds
  - Algorithm Y takes ~100 seconds

Complexity makes a huge difference!

Running time

- Assume:
  - Algorithm X takes time $2n^2$ (written by best programmer) running on machine with 1000 MIPS
  - Algorithm Y takes time $50n \lg n$ (written by worst programmer) running on machine with 10 MIPS





The problem of sorting

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

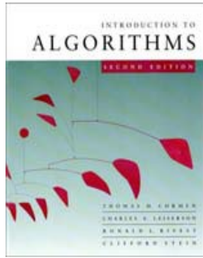
Output: permutation $\langle a'_1, a'_2, \ldots, a'_n \rangle$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Example:

Input: 8 2 4 9 3 6

Output: 2 3 4 6 8 9

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



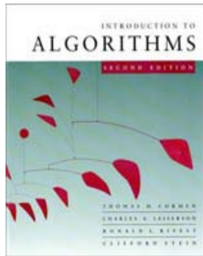
Insertion sort

"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

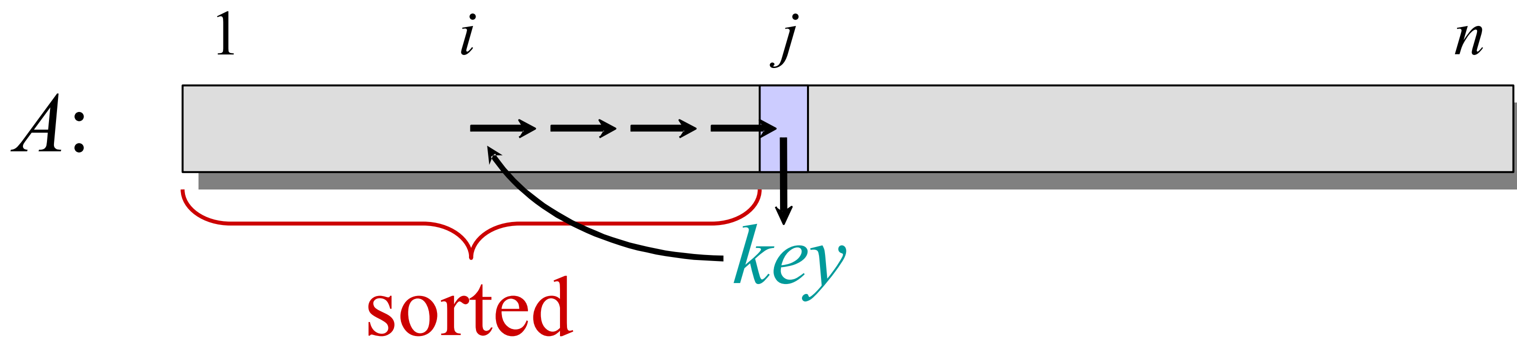
```
void insertion_sort(vector<int>& A) {
  for (int j = 1; j < A.size(); ++j) {
    int key = A[j];
    int i = j - 1;
   for (; i >= 0 && A[i] > key;) {
      A[i+1] = A[i--];
    }
    A[i+1] = key;
  }
}
```



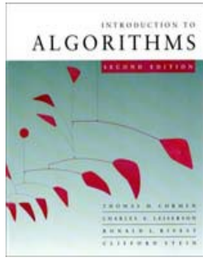
Insertion sort

"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
```

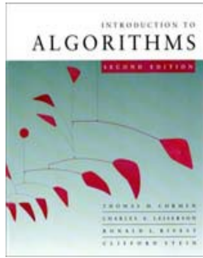


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



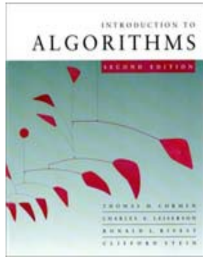
Example of insertion sort

8 2 4 9 3 6



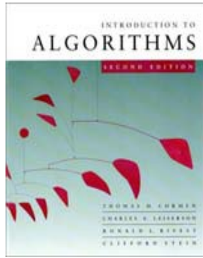
Example of insertion sort

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

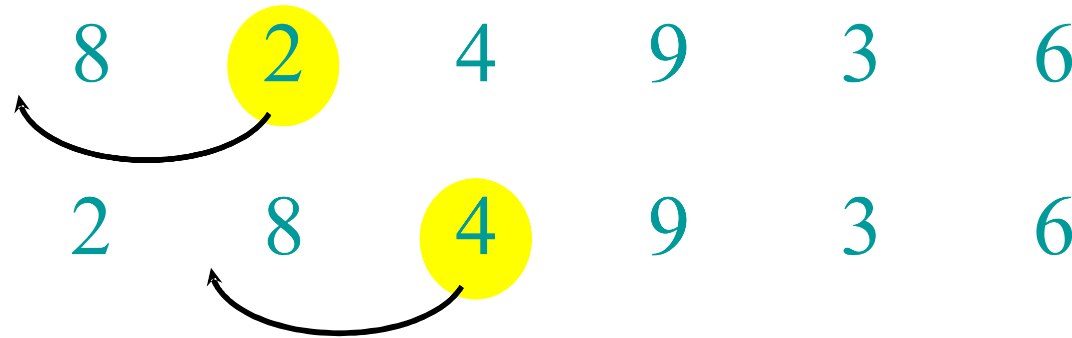


Example of insertion sort

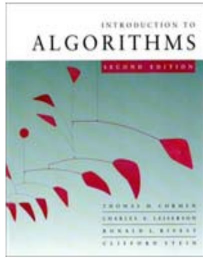
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



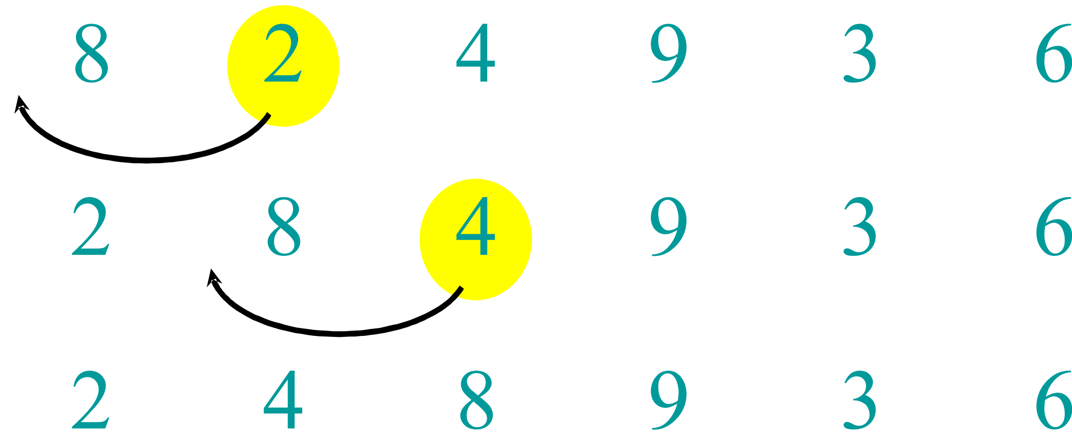
Example of insertion sort

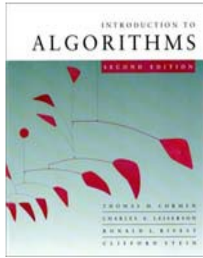


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

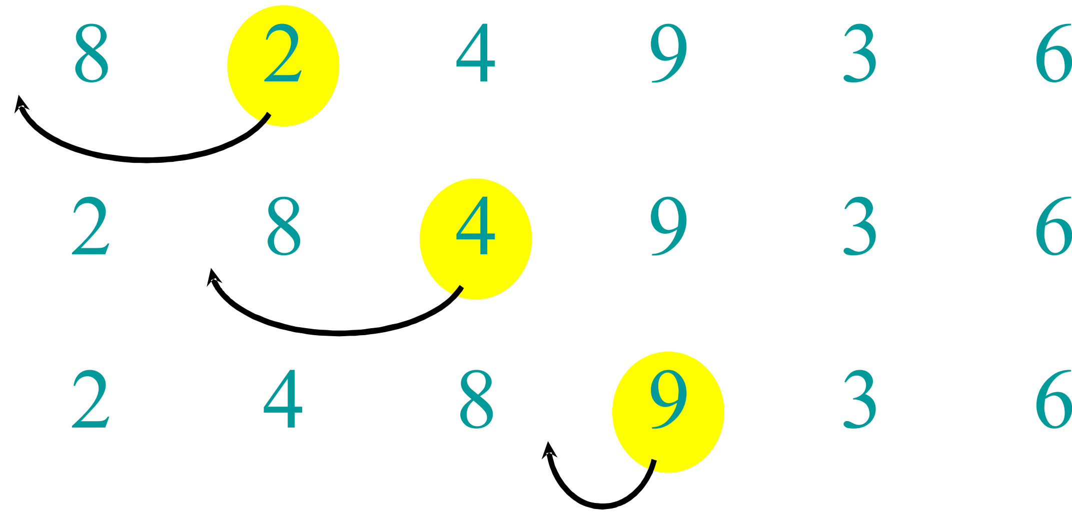


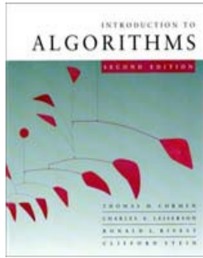
Example of insertion sort



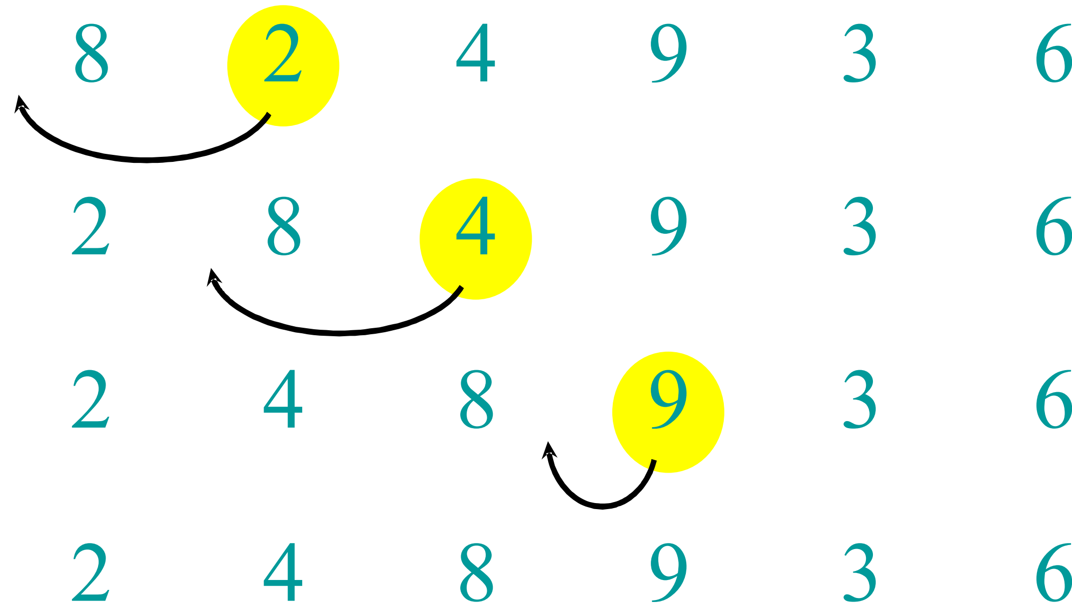


Example of insertion sort

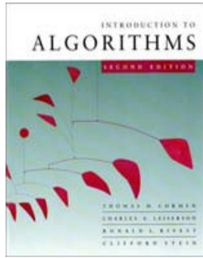




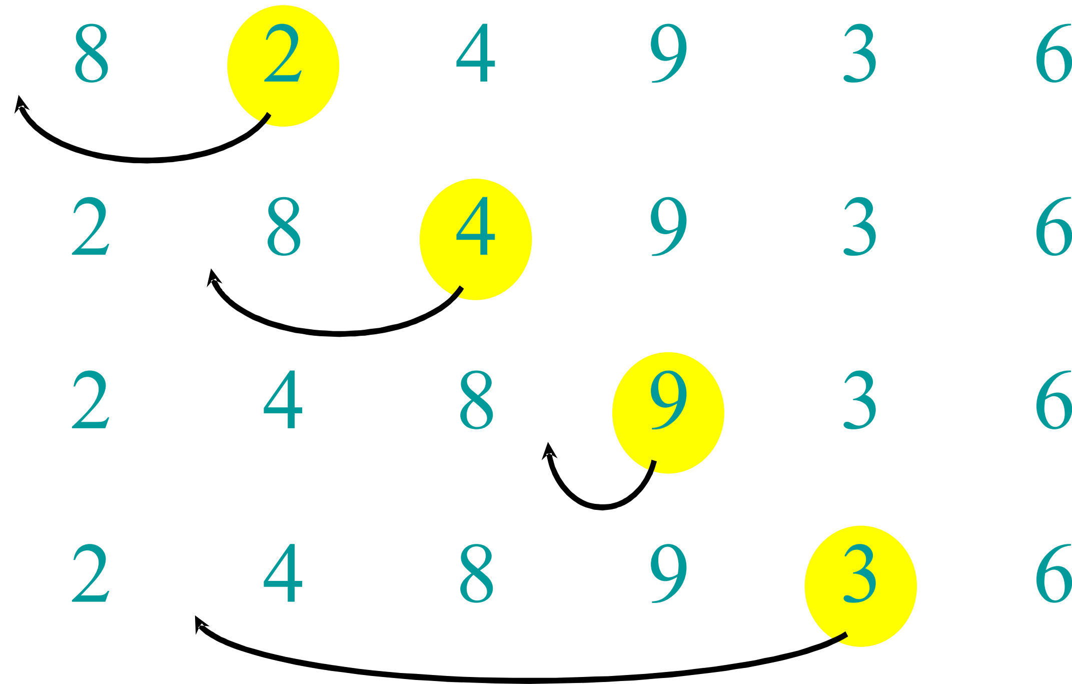
Example of insertion sort



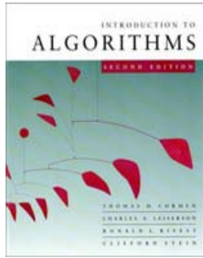
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



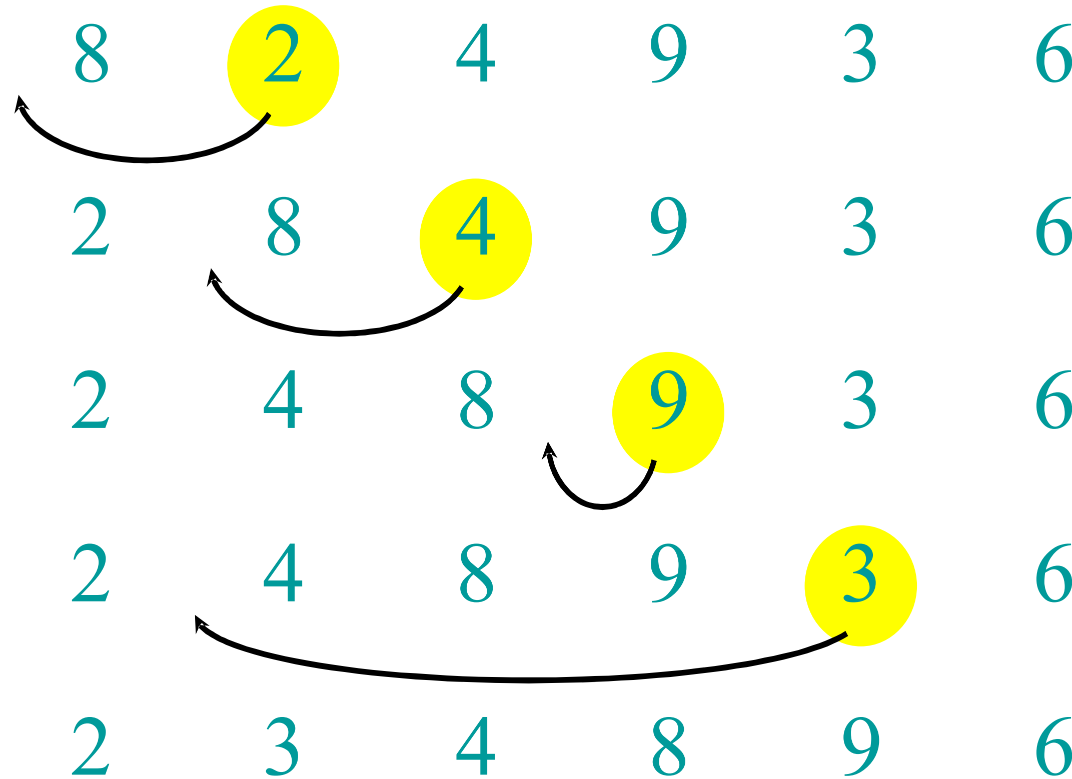
Example of insertion sort



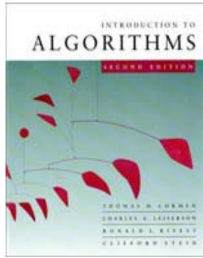
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



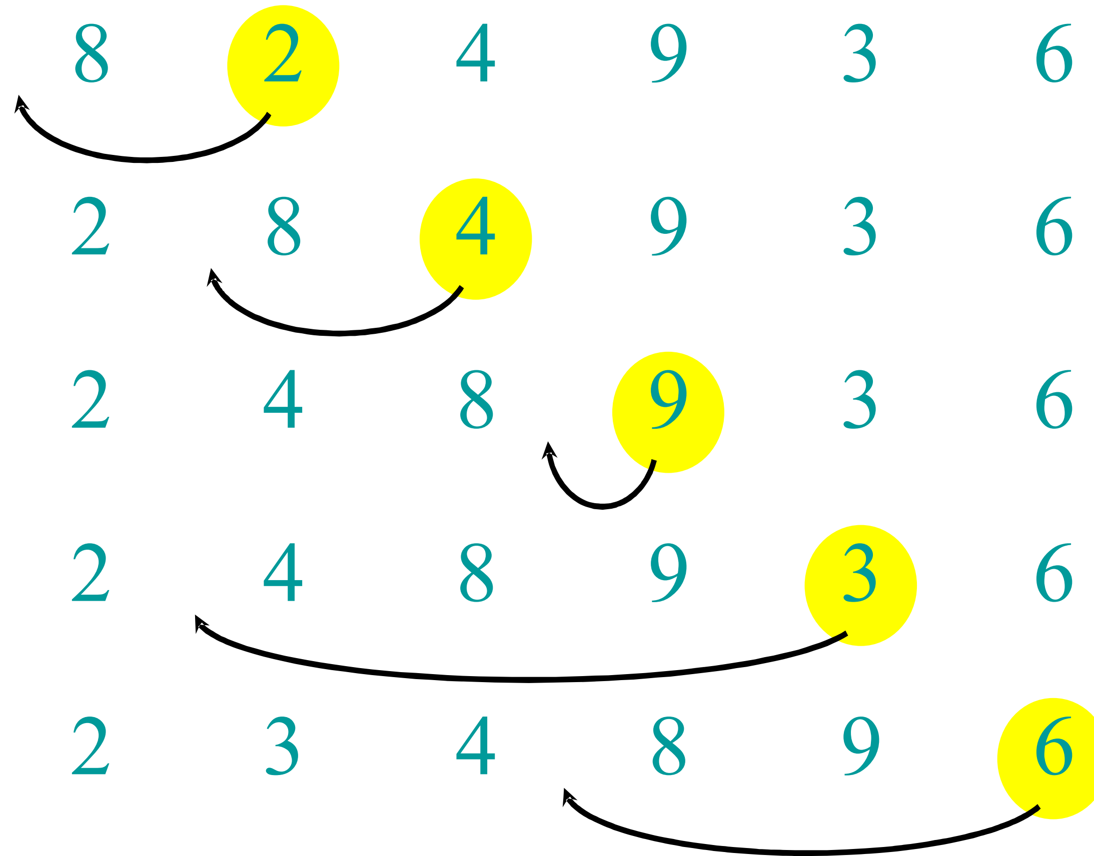
Example of insertion sort



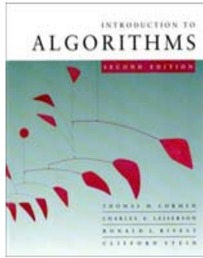
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Example of insertion sort



Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Example of insertion sort

8 2 4 9 3 6

2 8 4 9 3 6

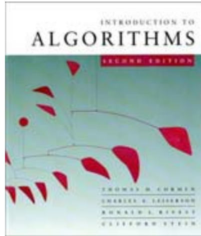
2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9 *done*

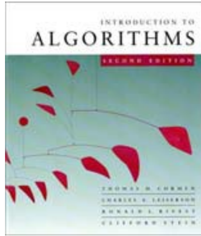
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



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.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Kinds of analyses

Worst-case: (usually)

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

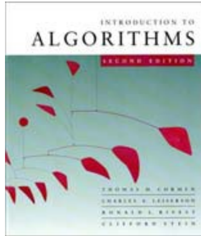
Average-case: (sometimes)

- $T(n)$ = expected time of algorithm 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.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Machine-independent time

What is insertion sort's worst-case time?

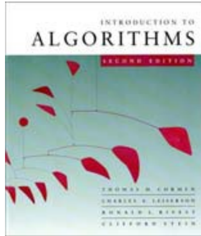
- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

BIG IDEA:

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

"Asymptotic Analysis"

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Θ-notation

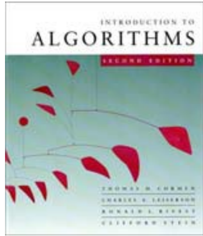
Math:

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

Engineering:

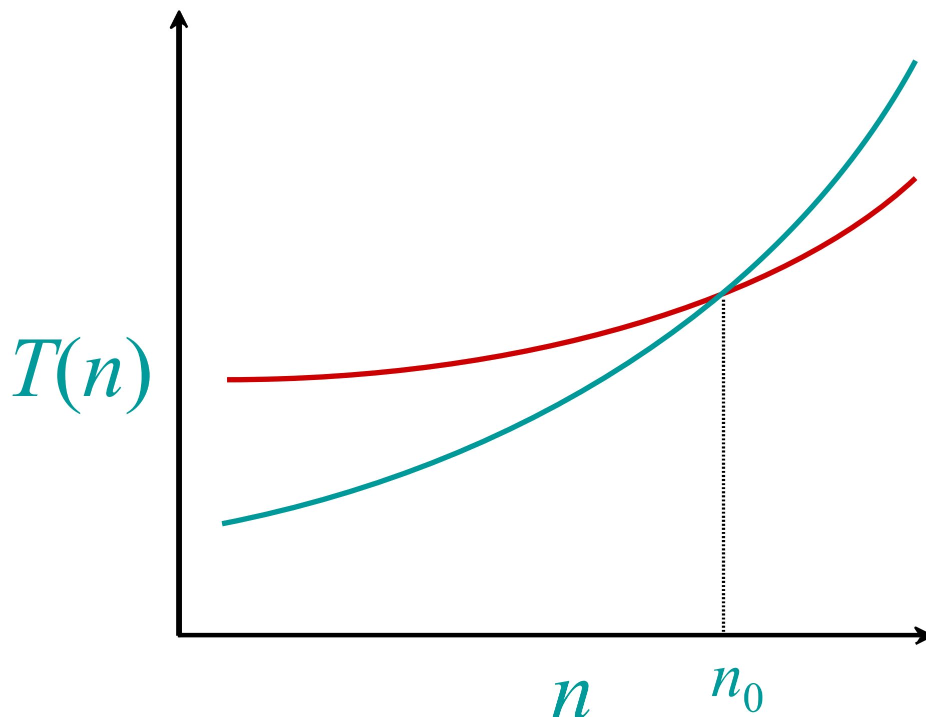
- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



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.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

Insertion Sort Analysis

INSERTION-SORT(A)		cost	times
1	**for** $j \leftarrow 2$ **to** $length[A]$	c_1	n
2	**do** $key \leftarrow A[j]$	c_2	$n-1$
3	$\triangleright$ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
4	$i \leftarrow j-1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	**do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] \leftarrow key$	c_8	$n-1$

Sum of all terms

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

Insertion Sort Analysis

INSERTION-SORT(A)	cost	times
1 **for** $j \leftarrow 2$ **to** $length[A]$	c_1	n
2 **do** $key \leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Best case: sorted array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} 1 = n-1$$

$$T(n) = an + b = \Theta(n)$$

Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*[A]	c_1	n
2 **do** *key* $\leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1\,..\,j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Worst case: reversed array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

Why? $$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$ Why?

Insertion Sort Analysis

$$\sum_{i=1}^{n} i = 1 + \quad 2 + \quad \cdots + \quad n-1 + \quad n$$

$$\sum_{i=1}^{n} i = n + \quad n-1 + \quad \cdots + \quad 2 + \quad 1$$

$$2\sum_{i=1}^{n} i = (n+1) + \quad (n+1) + \quad \cdots + \quad (n+1) + \quad (n+1)$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Review in Appendix A

Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
2 **do** *key* $\leftarrow A[j]$	c_2	$n-1$
3 $\triangleright$ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Worst case: reversed array

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$T(n) = a\,n^2 + bn + c = \Theta(n^2)$$

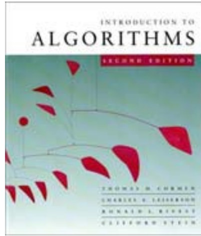
Insertion Sort Analysis

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
2 **do** *key* $\leftarrow A[j]$	c_2	$n-1$
3 ▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 **do** $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow key$	c_8	$n-1$

Average case: all permutations equally likely

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} \frac{j}{2}$$

$$T(n) = a\,n^2 + bn + c = \Theta(n^2)$$



Insertion sort analysis

Worst case: Input reverse sorted.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \qquad \text{[arithmetic series]}$$

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.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

Insertion Sort Analysis

- What about space?
- Insertion sorts "in place" as it does not copy the array anywhere
- It only takes a constant amount of extra storage, independent of n
- Therefore $S(n) = \Theta(1)$

Merge Sort

- Divide-and-Conquer
  - *Divide* the problem into a number of sub-problems
  - *Conquer* the smaller problems
  - *Combine* the results of the sub-problems into a solution for the big problem

```
MERGE-SORT(A, p, r)
1  if p < r
2     then q ← ⌊(p + r)/2⌋
3          MERGE-SORT(A, p, q)
4          MERGE-SORT(A, q + 1, r)
5          MERGE(A, p, q, r)
```

Merge Sort

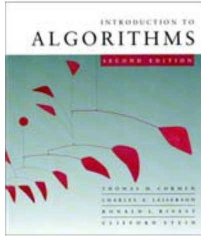
- Divide-and-Conquer
  - *Divide* the problem into a number of sub-problems
  - *Conquer* the smaller problems
  - *Combine* the results of the sub-problems into a solution for the big problem

```
void merge_sort(vector<int>& A, int p, int r) {
  if (p >= r) return;

  int q = (p + r) / 2;

  merge_sort(A, p, q);
  merge_sort(A, q+1, r);

  merge(A, p, q, r);
}
```



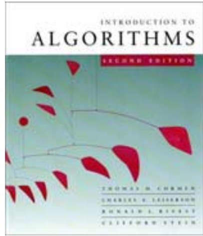
Merge sort

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

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

Key subroutine: **MERGE**

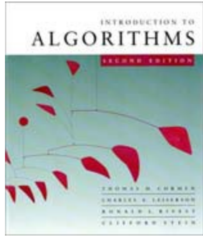
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



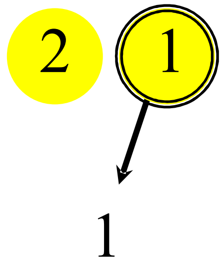
Merging two sorted arrays

20	12
13	11
7	9
2	1

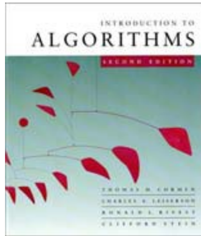
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



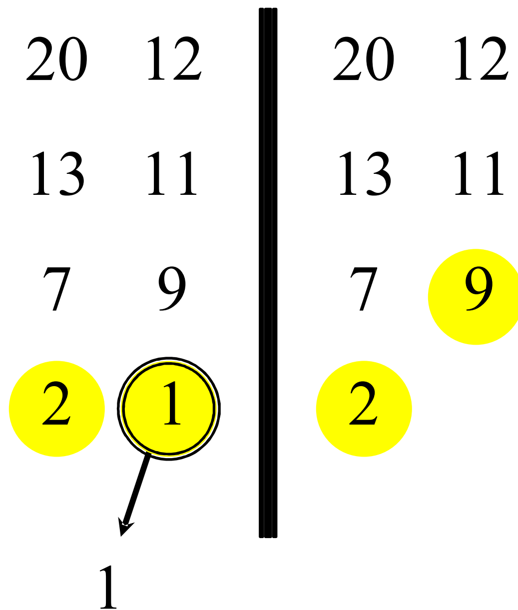
Merging two sorted arrays

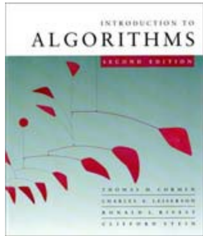


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

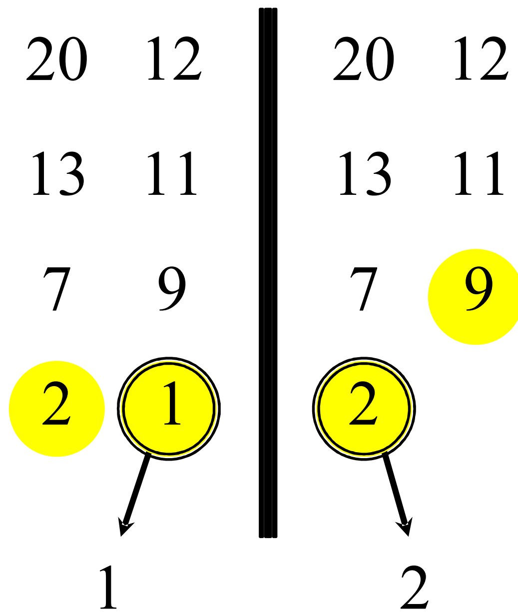


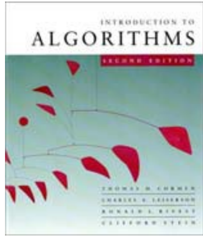
Merging two sorted arrays



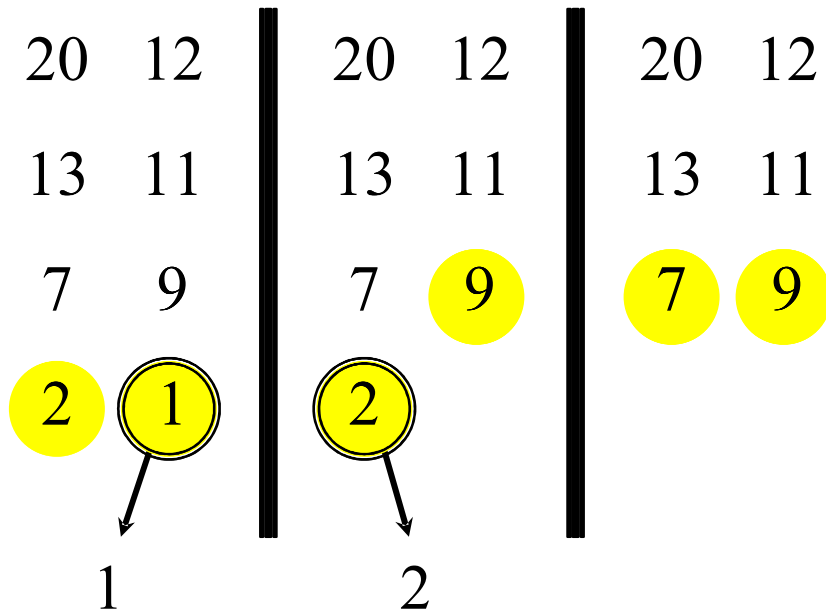


Merging two sorted arrays

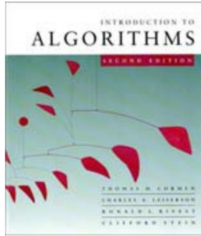




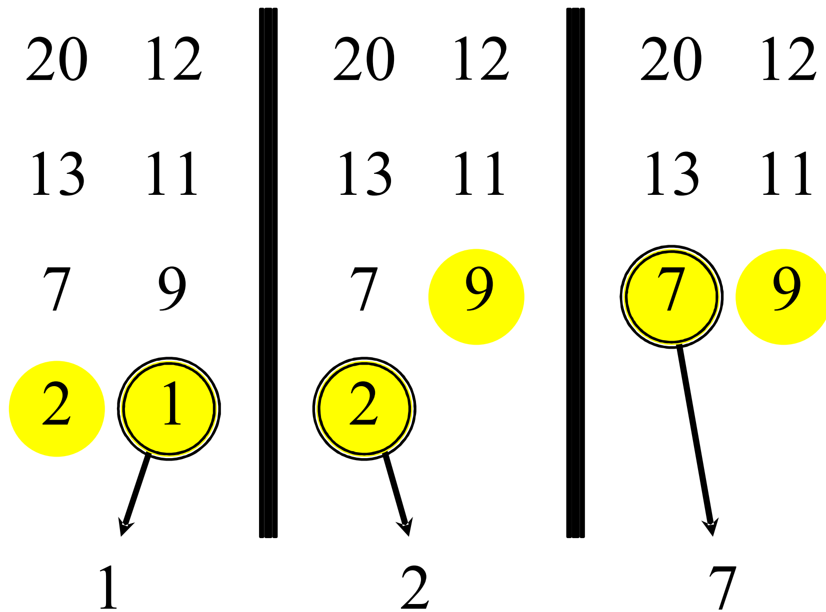
Merging two sorted arrays



Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Merging two sorted arrays

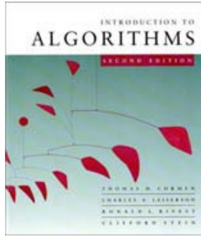


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

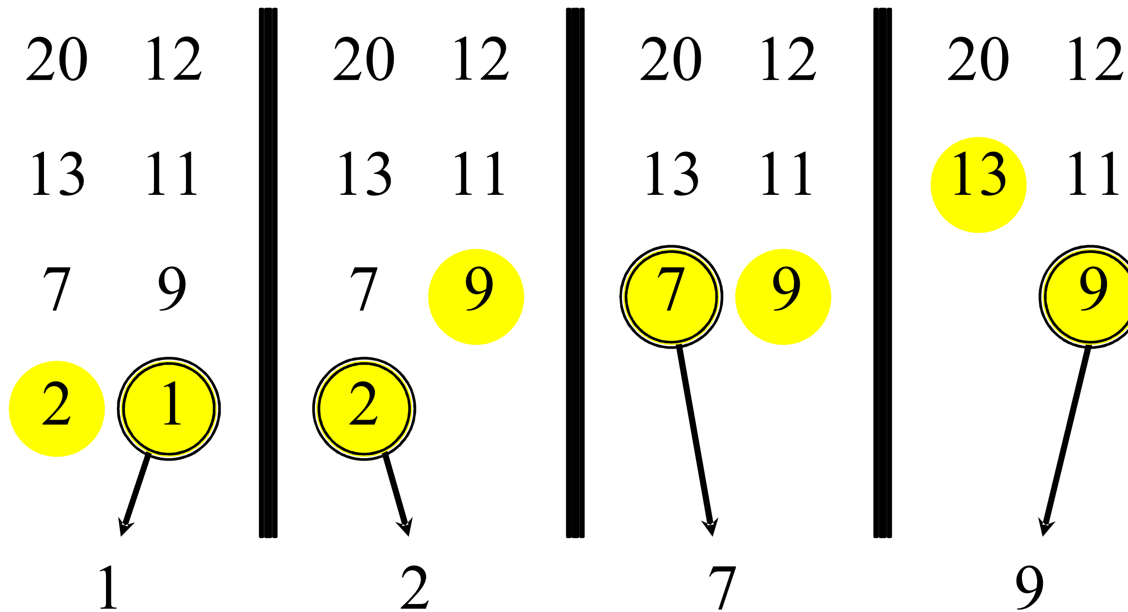
Merging two sorted arrays

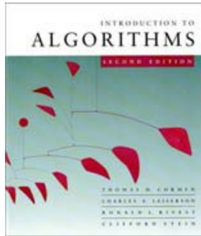
20	12		20	12		20	12		20	12
13	11		13	11		13	11		13	11
7	9		7	9		7	9			9
2	1		2							

1 2 7

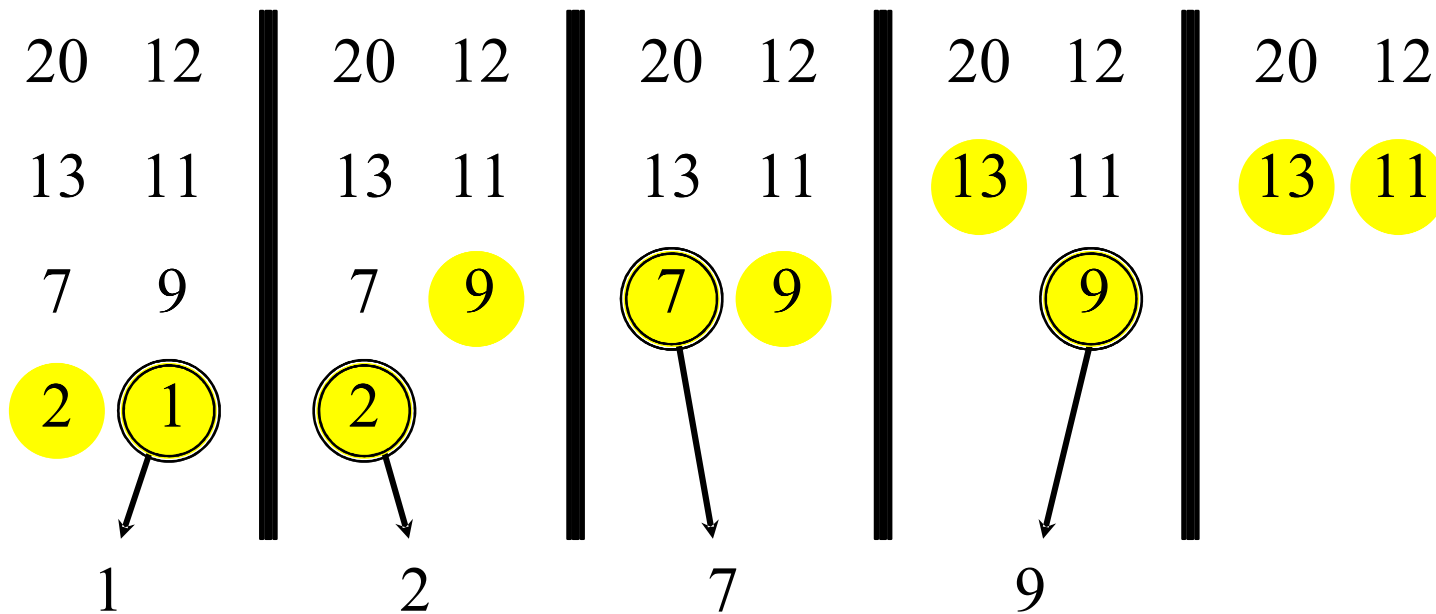


Merging two sorted arrays

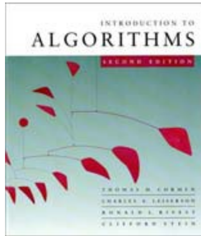




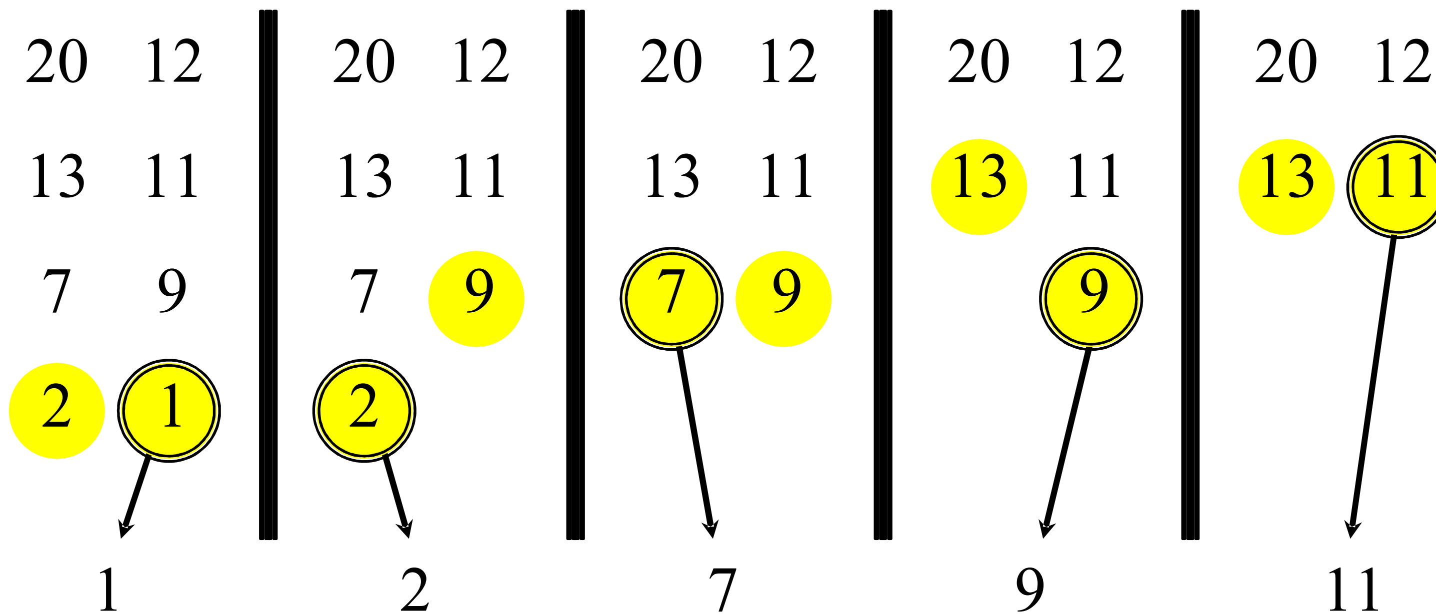
Merging two sorted arrays

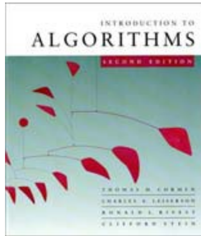


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

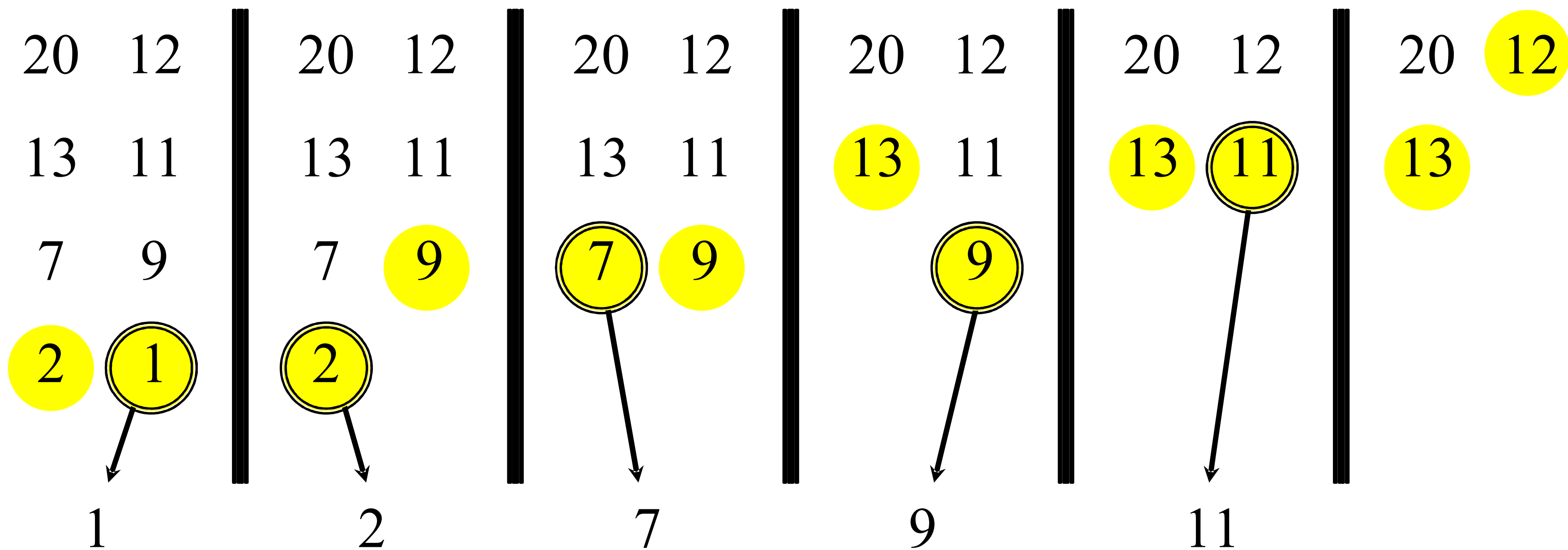


Merging two sorted arrays

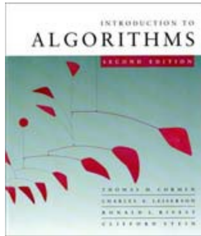




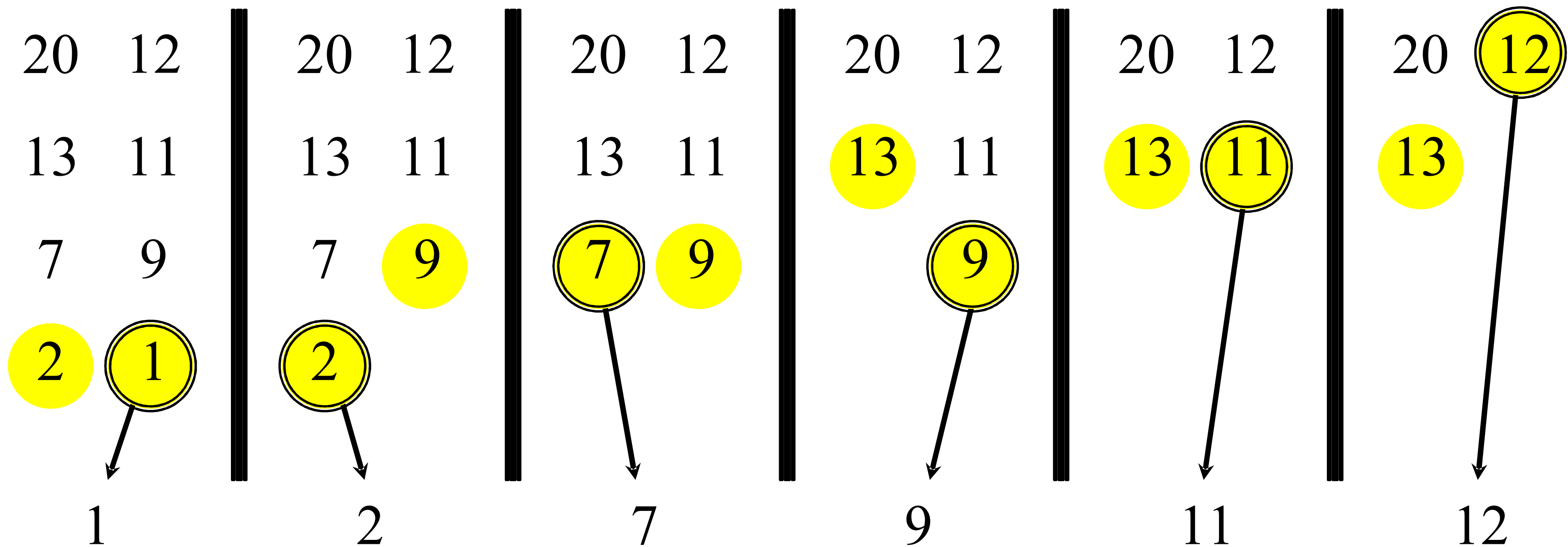
Merging two sorted arrays



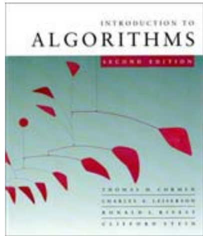
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



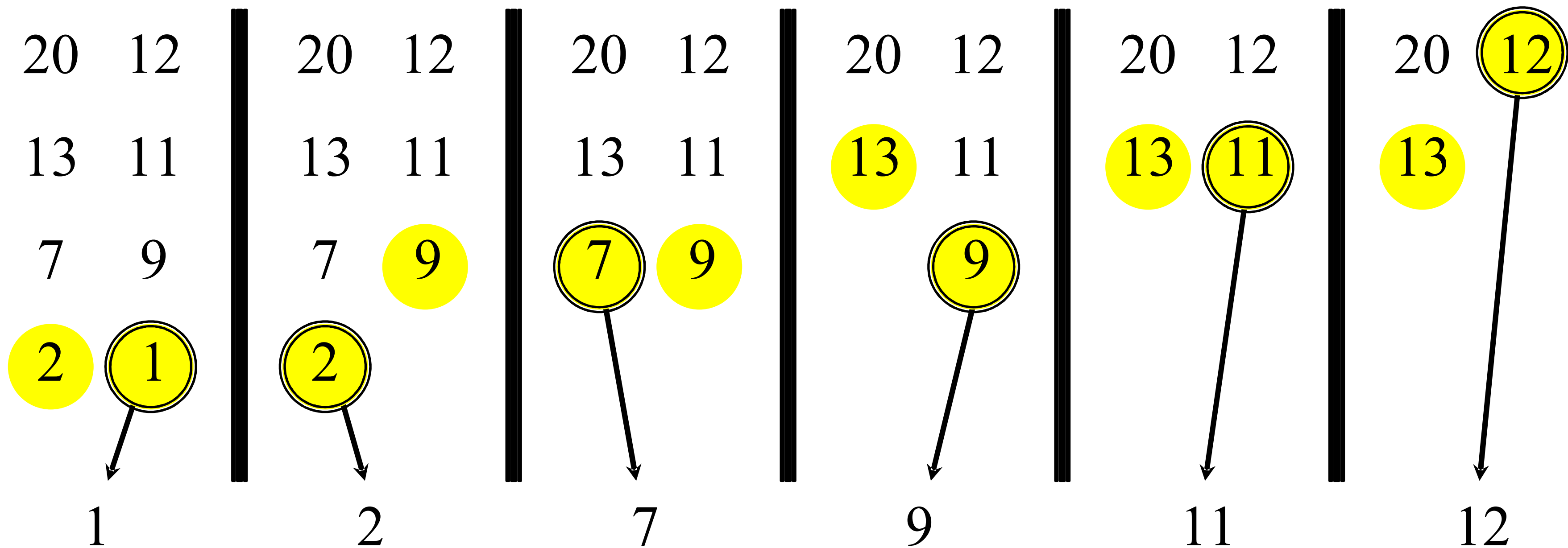
Merging two sorted arrays



Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Merging two sorted arrays



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

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

Merge

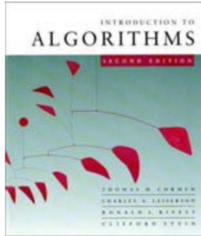
```
MERGE(A, p, q, r)
 1  n1 = q - p + 1
 2  n2 = r - q
 3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i - 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

compute sizes (lines 1–2)

copy arrays to be merged (lines 4–7)

adds "sentinel" (lines 8–9)

merge (lines 12–17)



Analyzing merge sort

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

Abuse

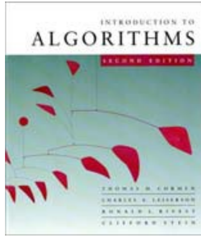
Sloppiness: Should be $T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$, but it turns out not to matter asymptotically.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

Recurrence for Merge Sort

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

- Next we need to solve this recurrence relation i.e. find $T(n)$ as a function of n without $T(n/2)$

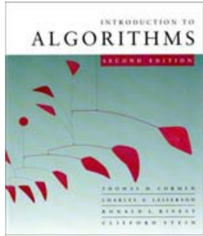


Recurrence for merge sort

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

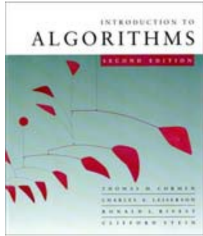
- We shall usually omit stating the base case when $T(n) = \Theta(1)$ for sufficiently small n, but only when it has no effect on the asymptotic solution to the recurrence.
- CLRS and Lecture 2 provide several ways to find a good upper bound on $T(n)$.

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

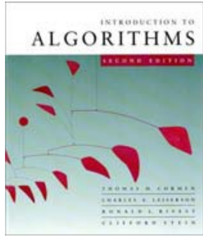


Recursion tree

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

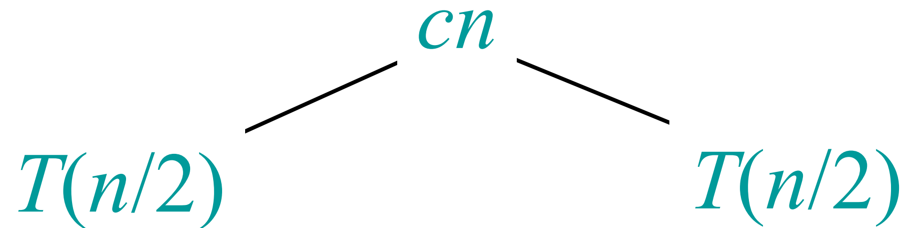
$$T(n)$$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

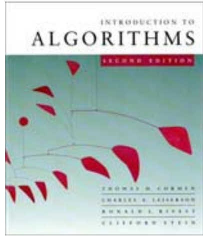


Recursion tree

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

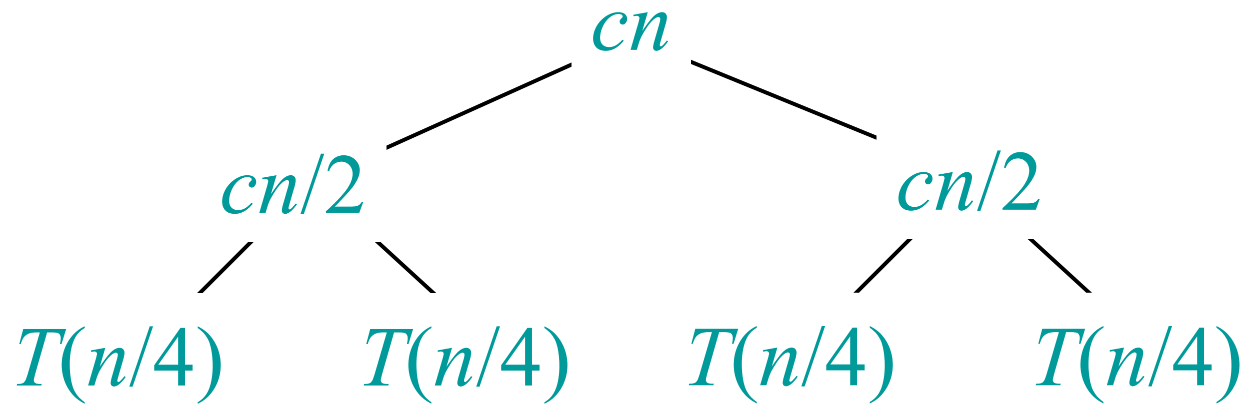


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

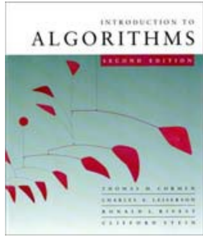


Recursion tree

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

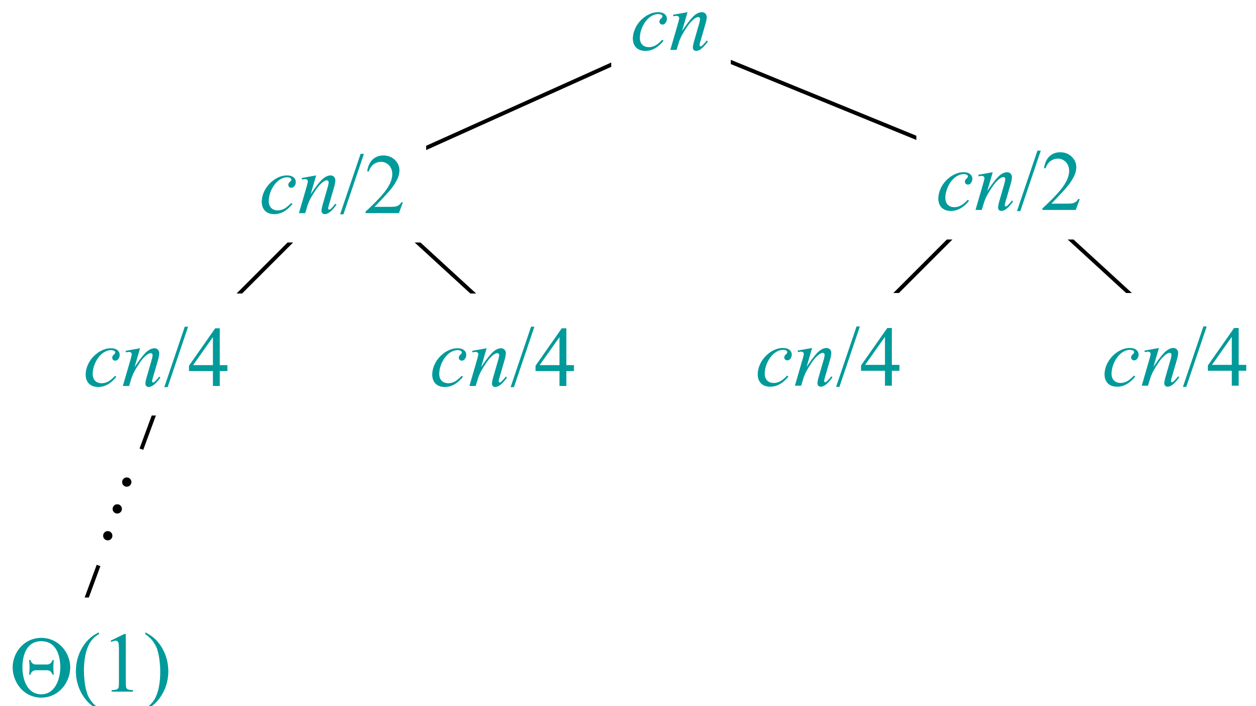


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

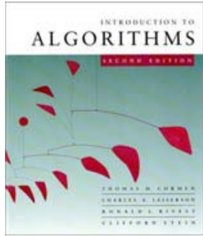


Recursion tree

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

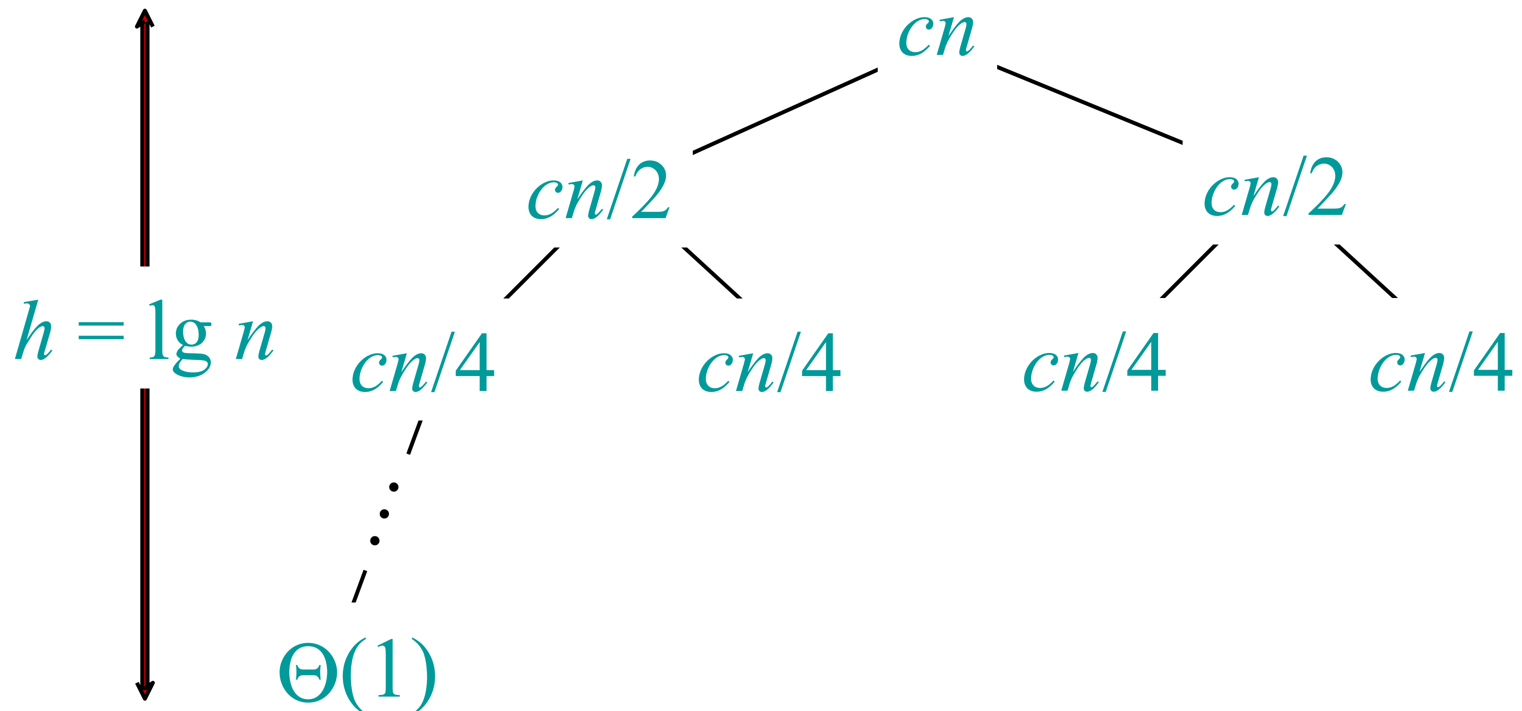


Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

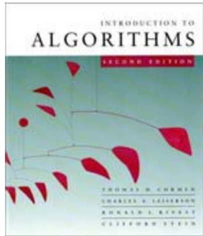


Recursion tree

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



Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

cn .... cn

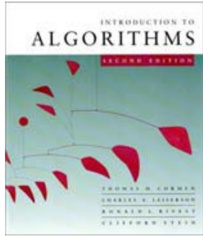
$cn/2$ $cn/2$

$cn/4$ $cn/4$ $cn/4$ $cn/4$

$h = \lg n$

$\Theta(1)$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

cn .... cn

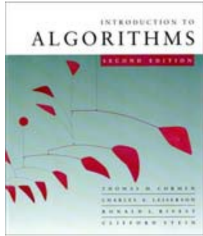
$cn/2$ $\quad$ $cn/2$ .... cn

$cn/4$ $\quad$ $cn/4$ $\quad$ $cn/4$ $\quad$ $cn/4$

$h = \lg n$

$\Theta(1)$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

cn .... cn

$cn/2$ $cn/2$ .... cn

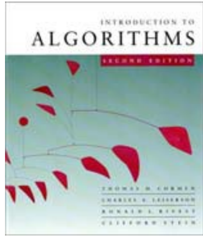
$cn/4$ $cn/4$ $cn/4$ $cn/4$ .... cn

$h = \lg n$

$\vdots$

$\Theta(1)$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

cn ········ cn

$cn/2$ $cn/2$ ········ cn

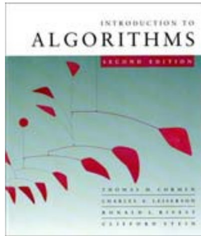
$cn/4$ $cn/4$ $cn/4$ $cn/4$ ········ cn

$h = \lg n$

$\vdots$

$\Theta(1)$ ········ #leaves $= n$ ········ $\Theta(n)$

Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson



Recursion tree

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

cn .... cn

$cn/2$ $cn/2$ .... cn

$cn/4$ $cn/4$ $cn/4$ $cn/4$ .... cn

$h = \lg n$

$\Theta(1)$ .... #leaves $= n$ .... $\Theta(n)$

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

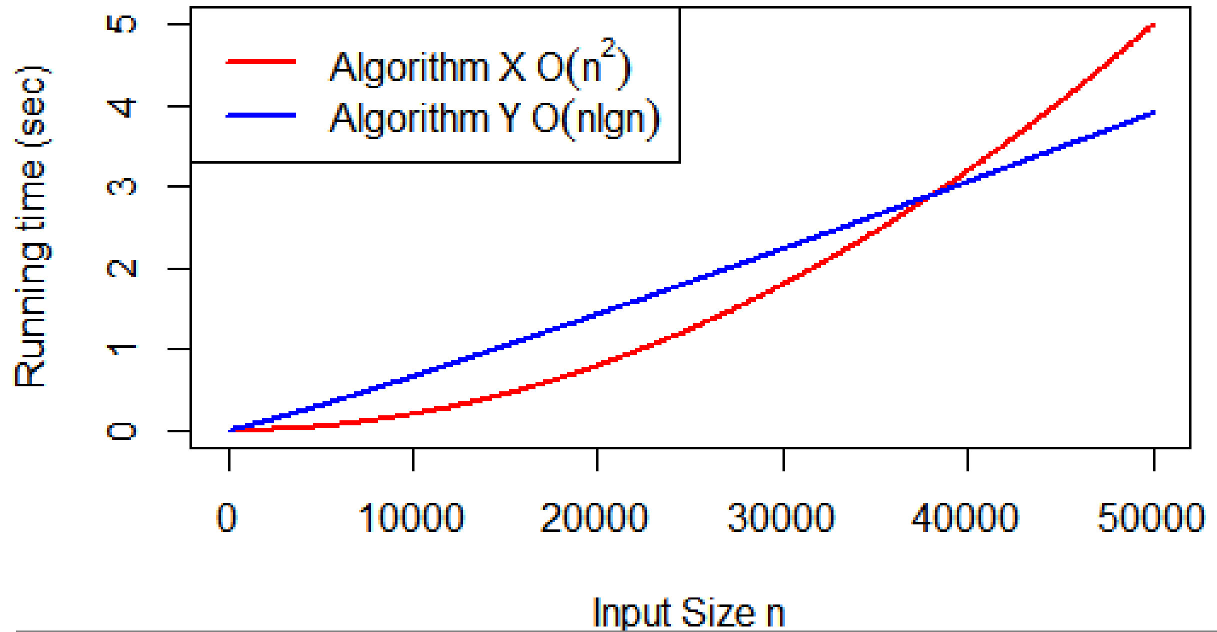
Copyright © 2001-5 Erik D. Demaine and Charles E. Leiserson

Merge Sort Analysis

- What about space?
- Merge Sort does *not* sort "in place" as it needs temporary space for the "Merge" subroutine
- It needs space: $S(n) = \Theta(n)$

Conclusion

- Insertion Sort:
  - $T(n) = \Theta(n^2)$
  - $S(n) = \Theta(1)$
- Merge Sort:
  - $T(n) = \Theta(n \lg n)$
  - $S(n) = \Theta(n)$



- Insertion sort better for smaller n, merge sort for larger n

Recap

- Insertion Sort
- Merge Sort
- Asymptotic Analysis
- Recurrences
- Next: More on recurrences and asymptotic analysis