Animation Demos

- http://cg.scs.carleton.ca/~morin/misc/sortalg/
- http://home.westman.wave.ca/~rhenry/sort/
  - Shows time complexities on best, worst and average case
- http://vision.bc.edu/~dmartin/teaching/sorting/anim-html/quick3.html
  - runs on almost sorted, reverse, random, and unique inputs; shows code with invariants
- http://www.brian-borowski.com/Sorting/
  - Shows comparisons and movements, and animations in stepwise fashion; it allows users to input their own data
- http://maven.smith.edu/~thiebaut/java/sort/demo.html
  - Shows comparisons and data movements and step by step execution.

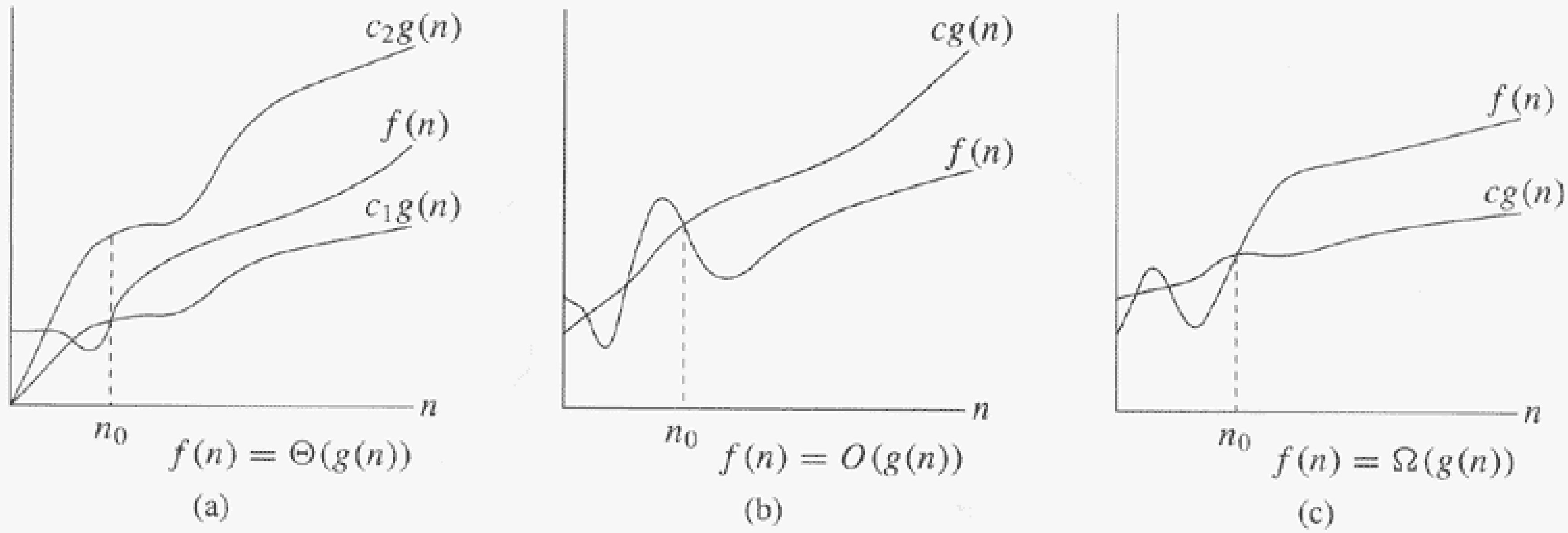


Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of $f(n)$ always lies between $c_1g(n)$ and $c_2g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

Examples

$3n^3 + 16n^2 - 7n + 299$

$= O(n^3)$ Why?

Because

$$3n^3 + 16n^2 - 7n + 299 < 3n^3 + 16n^3 + 7n^3 + 299n^3$$

$$< 325n^3$$

Thus for $c = 325$ and $n_0 = 1$, the definition of big-Oh is satisfied.

SelectionSort

```
SelectionSort(array A)
N ← length[A]
for p ← 1 to N
      do ▷ Compute j
         j ← p
         for m ← p + 1 to N
             do if (A[m] < A[j])
                   then j ← m
         ▷ Swap A[p] and A[j]
         temp ← A[p]
         A[p] ← A[j]
         A[j] ← temp
```

$O(n^2)$ time

$O(1)$ space

$T(n) <= T(n-1) + 7n$

Solving Recurrences by Substitution

- Guess the form of the solution
- (Using mathematical induction) find the constants and show that the solution works

Example

$$T(n) <= T(n-1) + 3n$$

Guess (#1)	T(n) = O(n)	
Need	**T(n) <= cn**	for some constant c>0
Assume	T(n-1) <= c(n-1)	Inductive hypothesis
Thus	T(n) <= c(n-1) + 3n <= (c+3) n	
	Our guess was wrong!!	

Solving Recurrences by Substitution: 2

$$T(n) <= T(n-1) + 3n$$

Guess (#2) $T(n) = O(n^2)$

Need **$T(n) <= cn^2$** for some constant $c>0$

Assume $T(n-1) <= c(n-1)^2$ Inductive hypothesis

Thus $T(n) <= c(n-1)^2 + 3n = cn^2 - 2cn + c + 3n$

$= cn^2 - (2c - 3)n + c$

$<= cn^2$

Works as long as $c>=2$ for all $n > c/(2c-3)$!!

This is the correct guess. WHY?

Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

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

$O(n^2)$ time

$O(1)$ space

$T(n) <= T(n-1) + 6n$

```
BUBBLESORT(A)
for i ← 1 to length[A]
    do for j ← length[A] downto i + 1
        do if A[j] < A[j − 1]
            then exchange A[j] ↔ A[j − 1]
```

$T(n) <= T(n-1) + 6n$

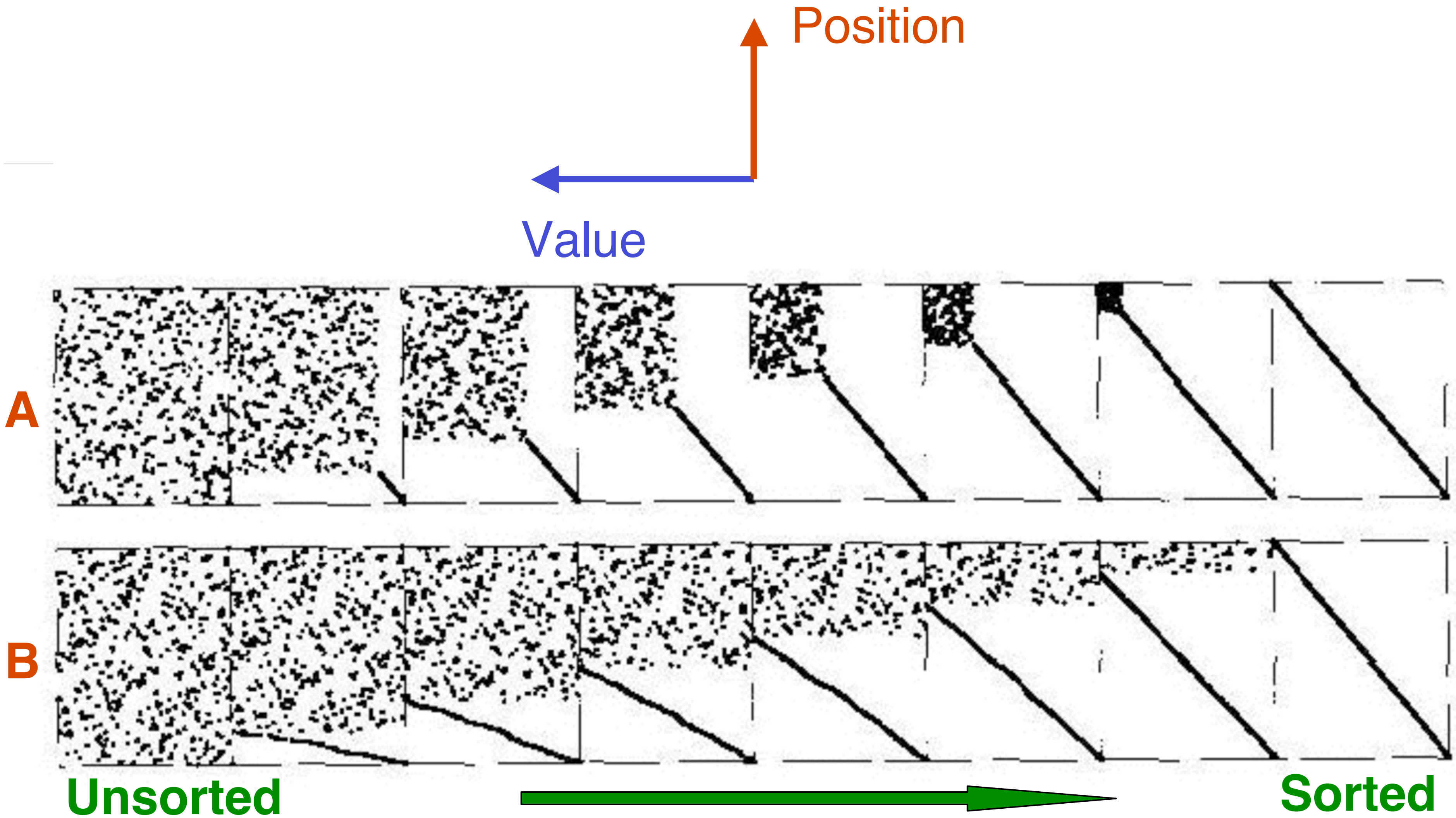
$O(n^2)$ time

$O(1)$ space

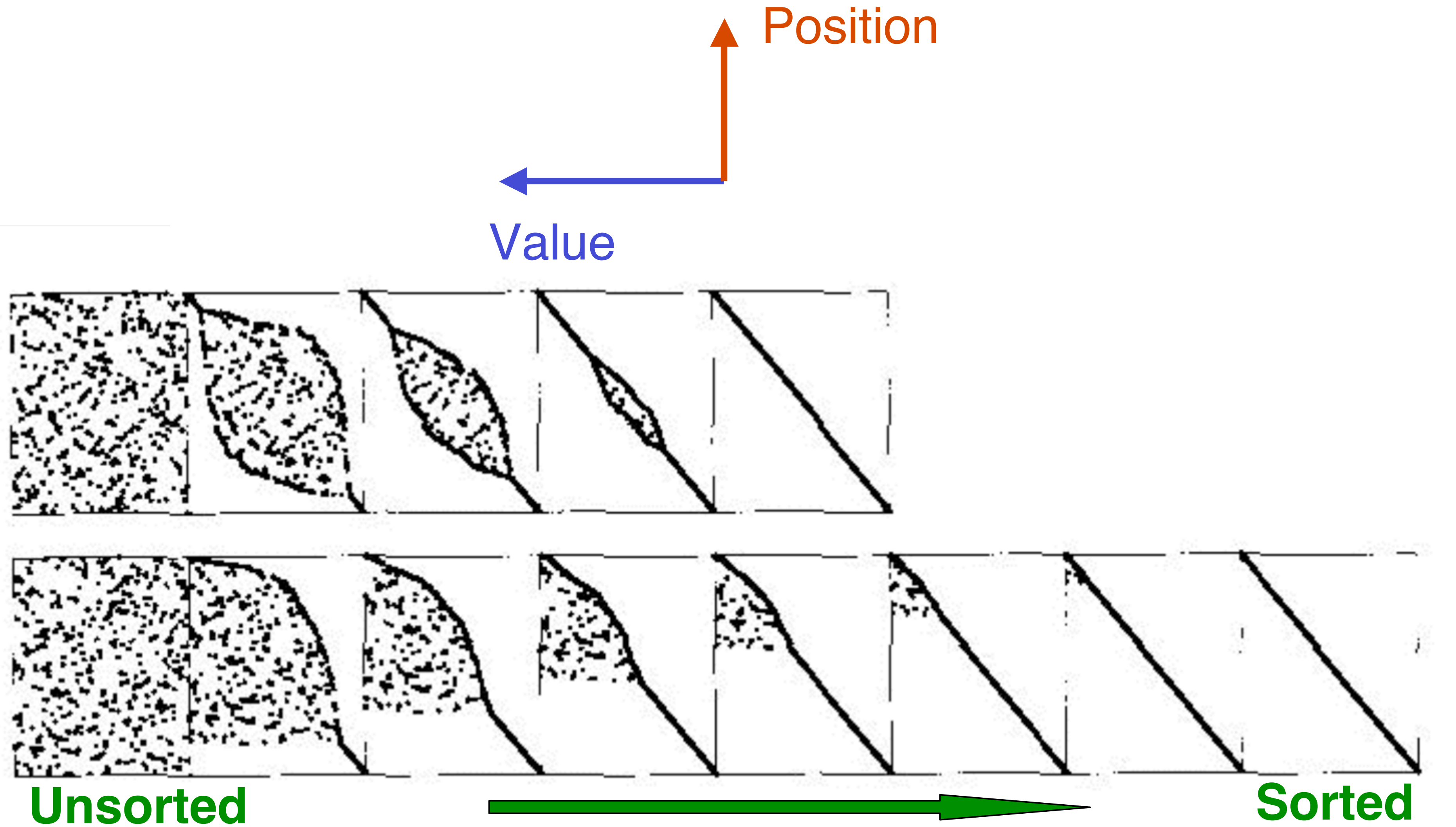
Sorting Algorithms

- SelectionSort
- InsertionSort
- BubbleSort
- ShakerSort
- MergeSort
- HeapSort
- QuickSort
- Bucket & Radix Sort
- Counting Sort

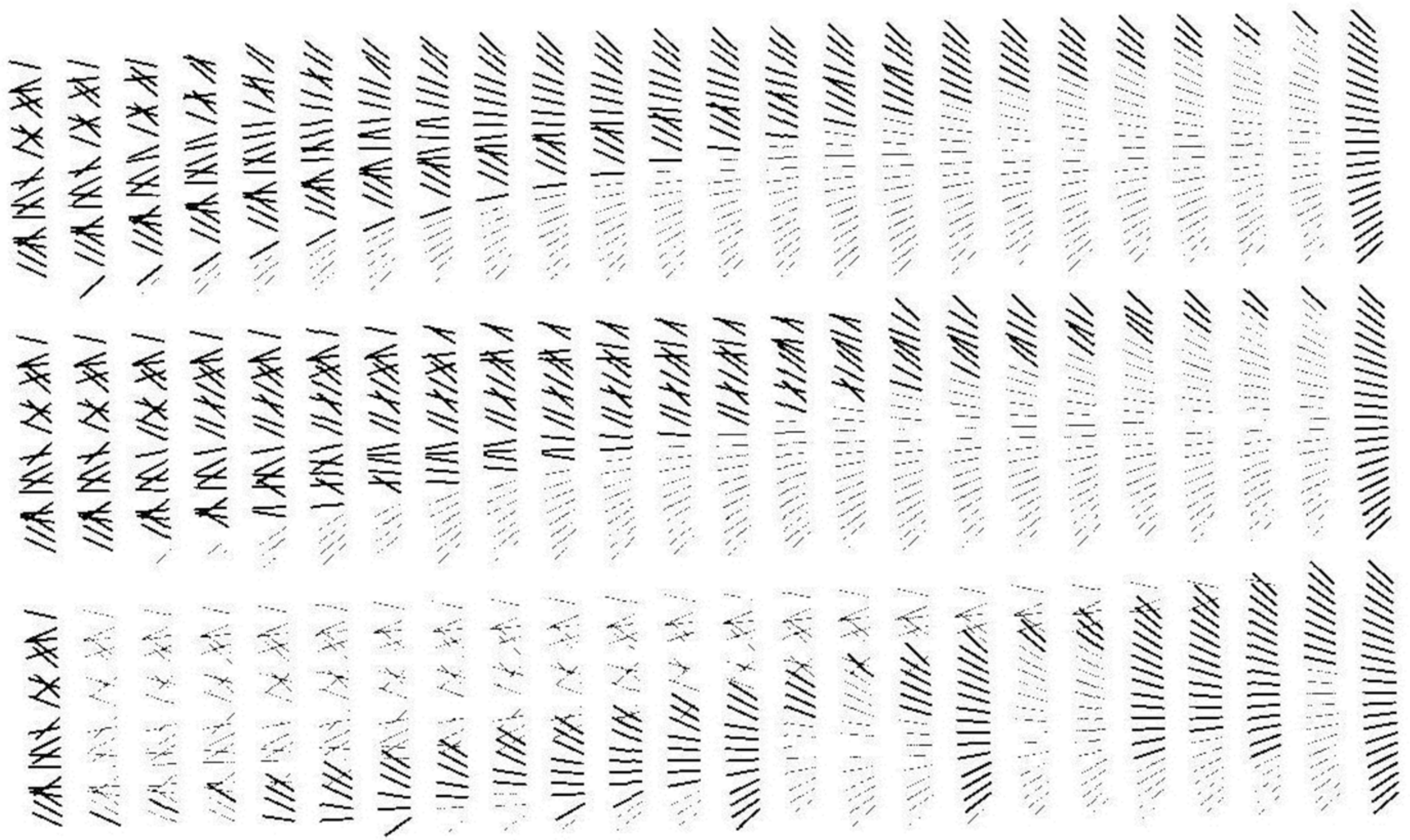
Visualizing Algorithms 1



Visualizing Algorithms 2



Visualizing Comparisons 3



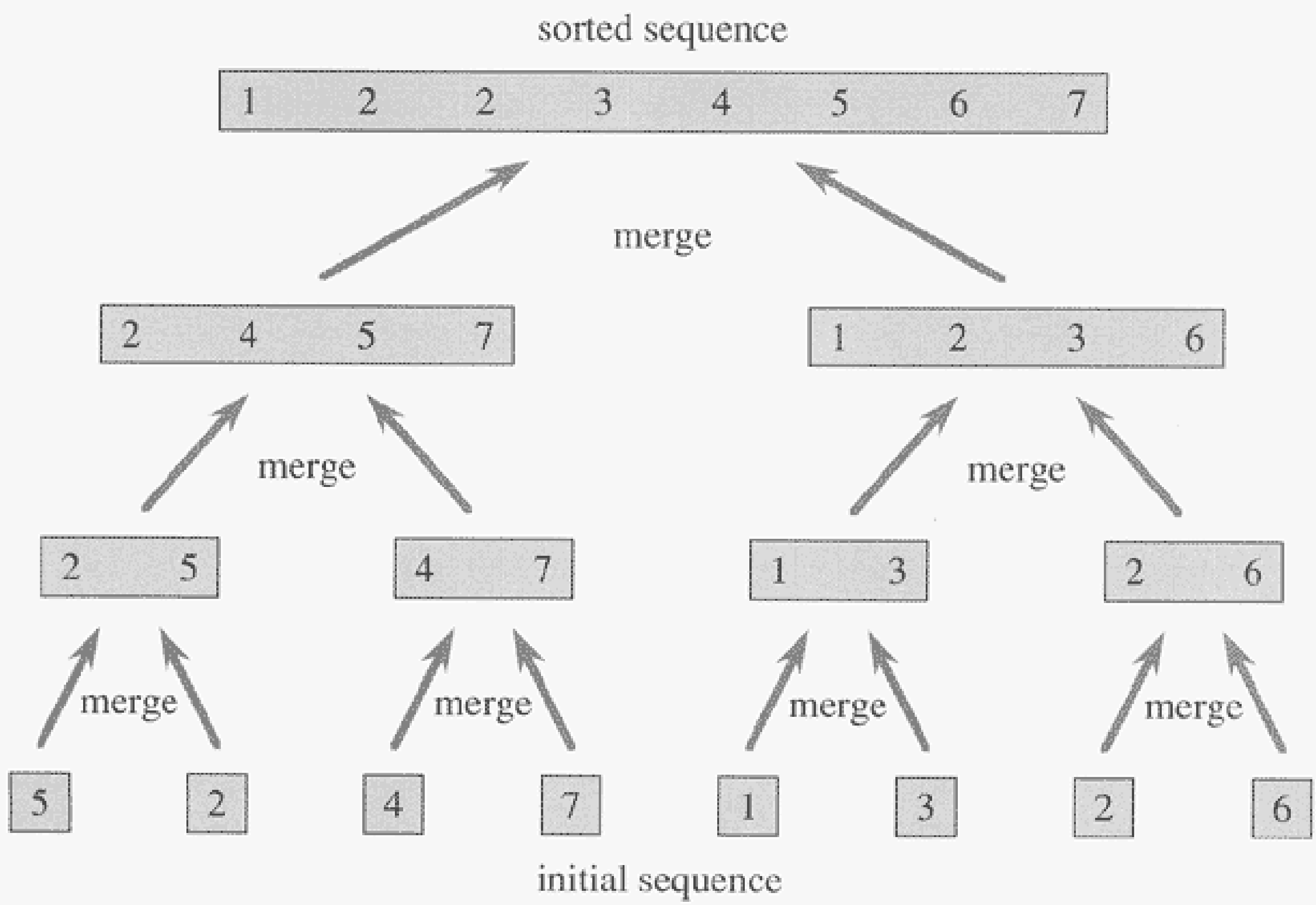
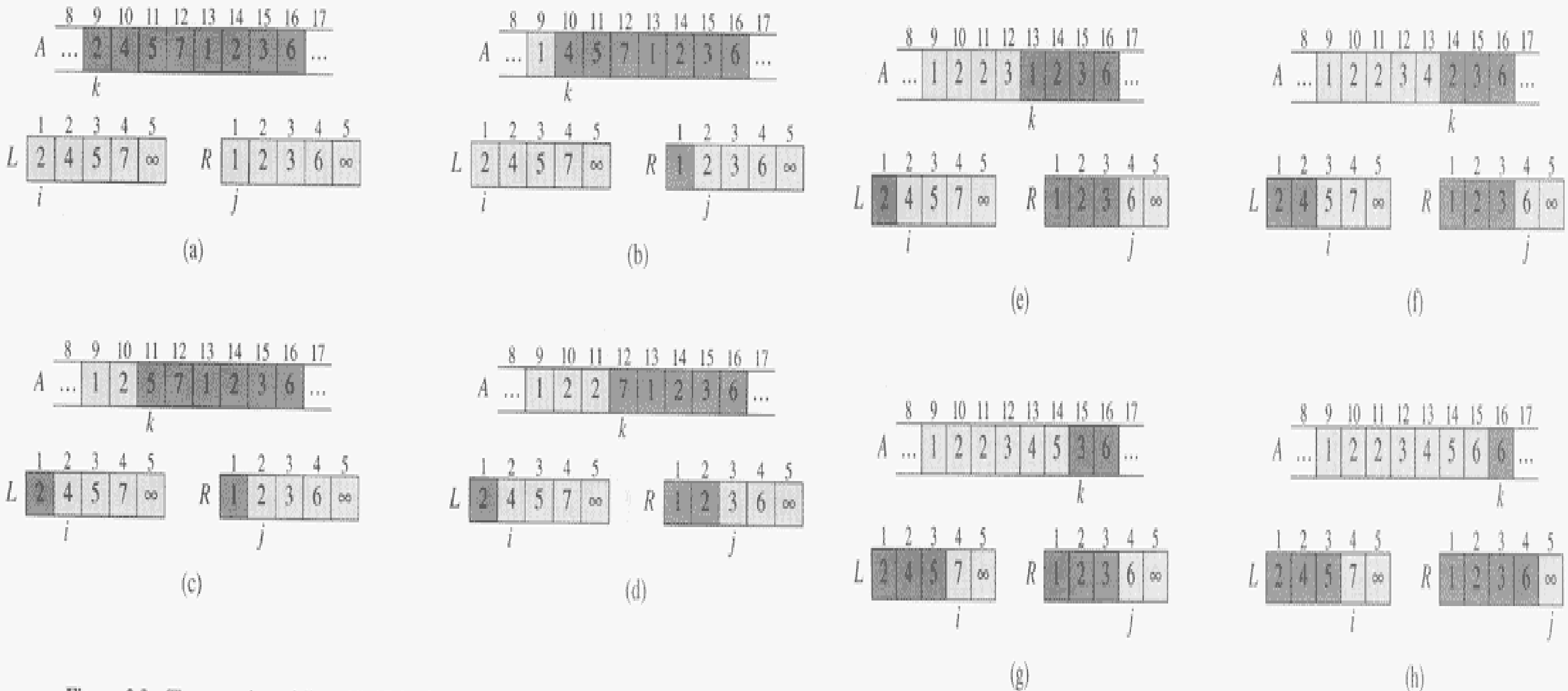


Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.



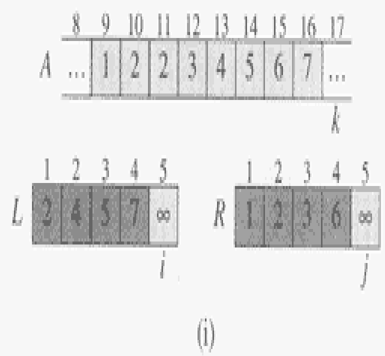


Figure 2.3 The operation of lines 10–17 in the call MERGE$(A, 9, 12, 16)$, when the subarray $A[9\,..\,16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6\rangle$. After copying and inserting sentinels, the array L contains $\langle 2, 4, 5, 7, \infty\rangle$, and the array R contains $\langle 1, 2, 3, 6, \infty\rangle$. Lightly shaded positions in A contain their final values, and lightly shaded positions in L and R contain values that have yet to be copied back into A. Taken together, the lightly shaded positions always comprise the values originally in $A[9\,..\,16]$, along with the two sentinels. Heavily shaded positions in A contain values that will be copied over, and heavily shaded positions in L and R contain values that have already been copied back into A. **(a)–(h)** The arrays A, L, and R, and their respective indices k, i, and j prior to each iteration of the loop of lines 12–17. **(i)** The arrays and indices at termination. At this point, the subarray in $A[9\,..\,16]$ is sorted, and the two sentinels in L and R are the only two elements in these arrays that have not been copied into A.

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
      do L[i] ← A[p + i − 1]
for j ← 1 to n2
      do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
      do if L[i] ≤ R[j]
            then A[k] ← L[i]
                 i ← i + 1
            else A[k] ← R[j]
                 j ← j + 1
```

Assumption: Array A is sorted from positions p to q and also from positions q+1 to r.

$\text{MERGE-SORT}(A, p, r)$

if $p < r$

  then $q \leftarrow \lfloor (p + r)/2 \rfloor$

    $\text{MERGE-SORT}(A, p, q)$

    $\text{MERGE-SORT}(A, q + 1, r)$

    $\text{MERGE}(A, p, q, r)$

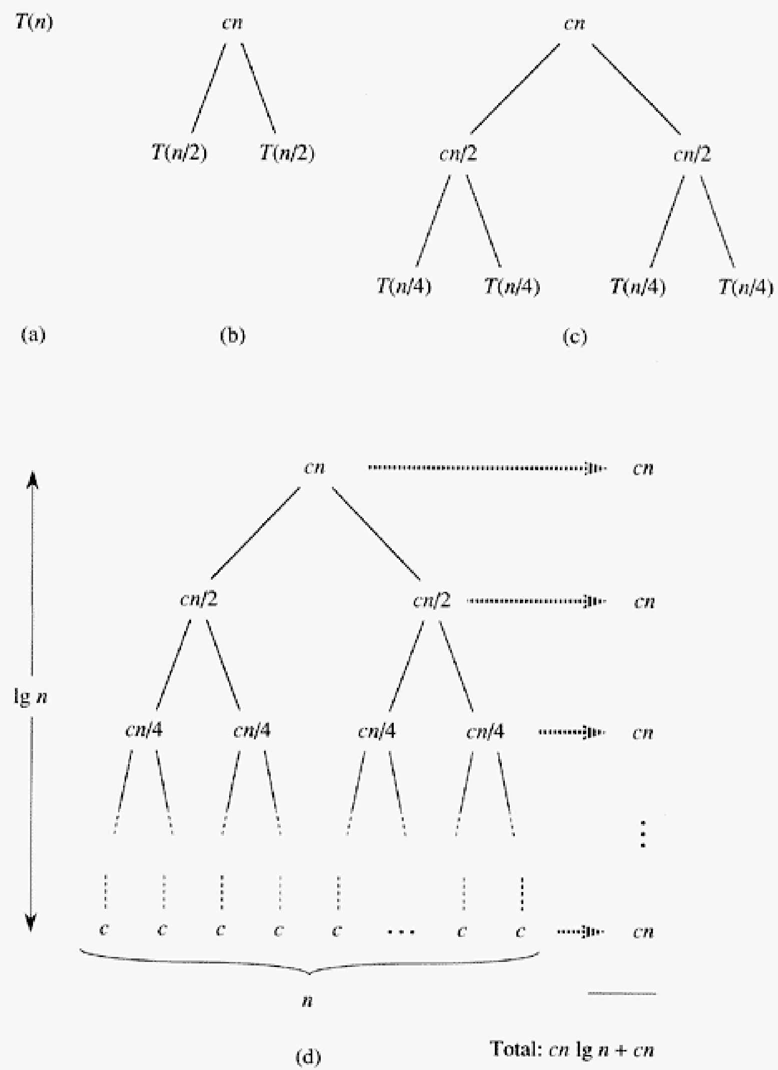
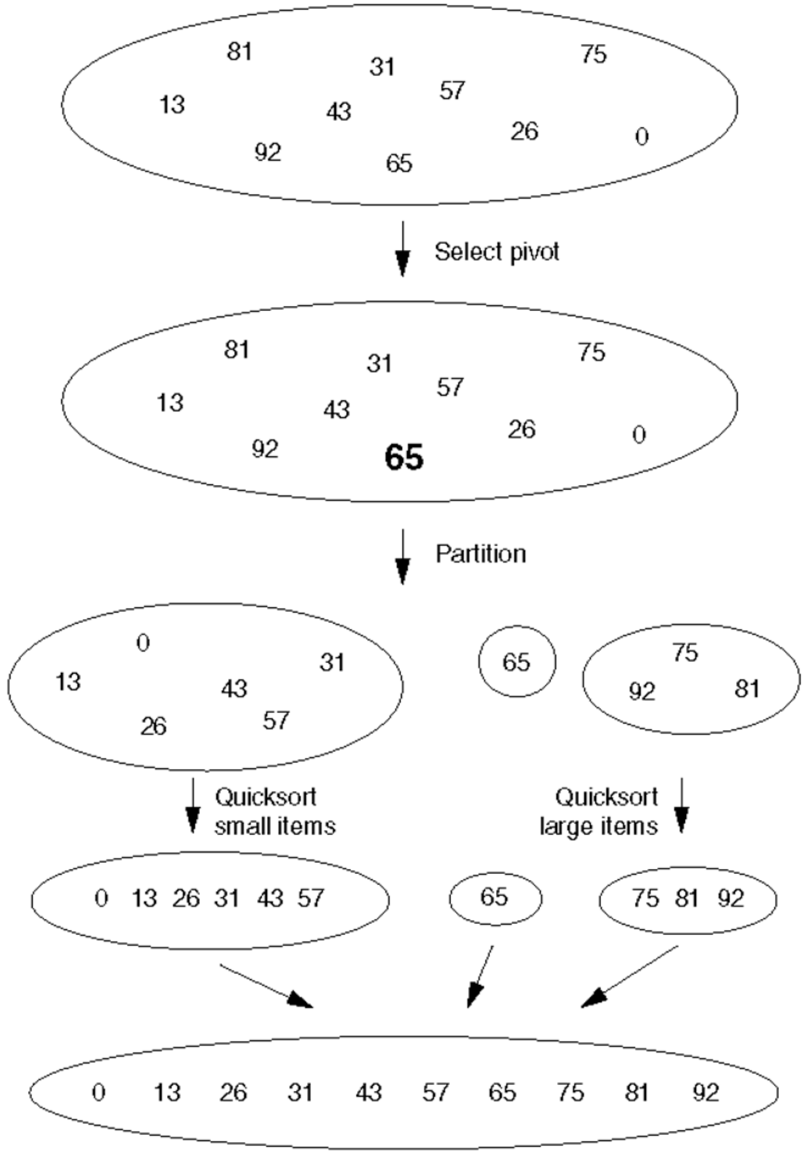


Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of cn. The total cost, therefore, is $cn \lg n + cn$, which is $\Theta(n \lg n)$.

Merge: Algorithm Invariants

- Merge (many lists)
  - ??

Figure 8.10 Quicksort



Partition

Figure A If 6 is used as pivot, the end result after partitioning is as shown in the Figure B.

2	1	4	5	0	3	9	8	7	6

Figure B Result after Partitioning

2	1	4	5	0	3	6	8	7	9

QuickSort

$\textsc{QuickSort}(array\ A, int\ p, int\ r)$

if $(p < r)$
  then $q \leftarrow \textsc{Partition}(A, p, r)$
    $\textsc{QuickSort}(A, p, q-1)$
    $\textsc{QuickSort}(A, q+1, r)$

To sort array call $\textsc{QuickSort}(A, 1, length[A])$.

$\textsc{Partition}(array\ A, int\ p, int\ r)$

$x \leftarrow A[r]$ $\quad\triangleright$ Choose **pivot**
$i \leftarrow p - 1$
for $j \leftarrow p$ **to** $r - 1$
  do if $(A[j] \leq x)$
    then $i \leftarrow i + 1$
      exchange $A[i] \leftrightarrow A[j]$
exchange $A[i+1] \leftrightarrow A[r]$
return $i + 1$

Page 146, CLRS

Solving Recurrences by Substitution

- Guess the form of the solution
- (Using mathematical induction) find the constants and show that the solution works

Example

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

Guess (#1)	$T(n) = O(n)$	
Need	$T(n) <= cn$	for some constant $c>0$
Assume	$T(n/2) <= cn/2$	Inductive hypothesis
Thus	$T(n) <= 2cn/2 + n = (c+1)\, n$	
	Our guess was wrong!!	

Solving Recurrences by Substitution: 2

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

Guess (#2)	$T(n) = O(n^2)$	
Need	$T(n) <= cn^2$	for some constant $c>0$
Assume	$T(n/2) <= cn^2/4$	Inductive hypothesis
Thus	$T(n) <= 2cn^2/4 + n = cn^2/2 + n$	

Works for all n as long as c>=2 !!

But there is a lot of "slack"

Solving Recurrences by Substitution: 3

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

Guess (#3) $T(n) = O(n\log n)$

Need **$T(n) <= cn\log n$** for some constant $c>0$

Assume $T(n/2) <= c(n/2)(\log(n/2))$ Inductive hypothesis

Thus $T(n) <= 2\ c(n/2)(\log(n/2)) + n$

$<= cn\log n - cn + n <= cn\log n$

Works for all n as long as c>=1 !!

This is the correct guess. WHY?

Show $T(n) >= c'n\log n$ for some constant $c'>0$

Solving Recurrences: Recursion-tree method

- Substitution method fails when a good guess is not available
- Recursion-tree method works in those cases
  - Write down the recurrence as a tree with recursive calls as the children
  - Expand the children
  - Add up each level
  - Sum up the levels
- Useful for analyzing divide-and-conquer algorithms
- Also useful for generating good guesses to be used by substitution method

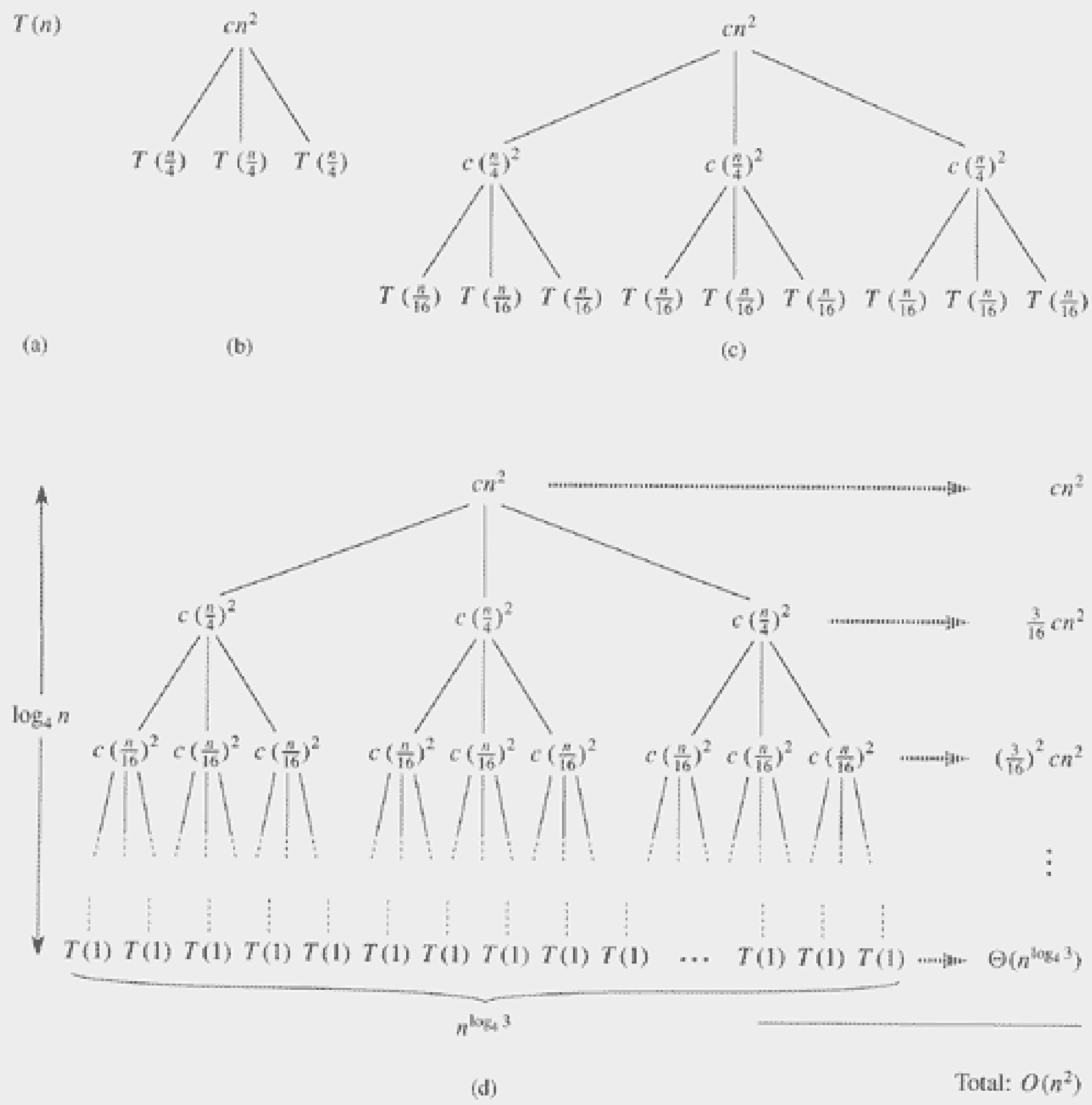


Figure 4.1 The construction of a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

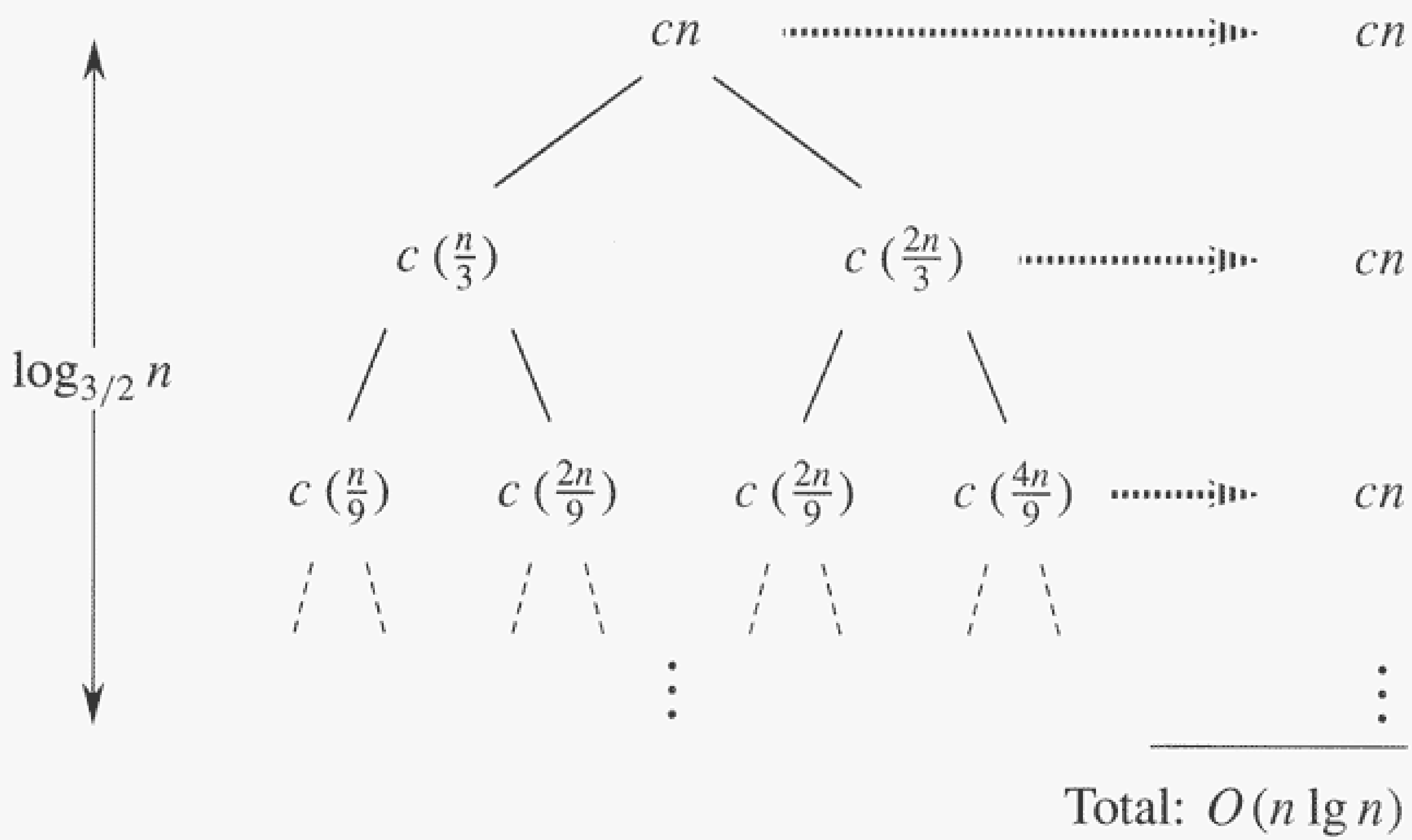


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

Solving Recurrences using Master Theorem

Master Theorem:

Let a,b >= 1 be constants, let f(n) be a function, and let

$$T(n) = aT(n/b) + f(n)$$

1. If $f(n) = O(n^{\log_b a-e})$ for some constant e>0, then
   $T(n) = Theta(n^{\log_b a})$
2. If $f(n) = Theta(n^{\log_b a})$, then
   $T(n) = Theta(n^{\log_b a} \log n)$
3. If $f(n) = Omega(n^{\log_b a+e})$ for some constant e>0, then
   $T(n) = Theta(f(n))$

Problems to think about!

- What is the least number of comparisons you need to sort a list of 3 elements? 4 elements? 5 elements?
- How to arrange a tennis tournament in order to find the tournament champion with the least number of matches? How many tennis matches are needed?