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.

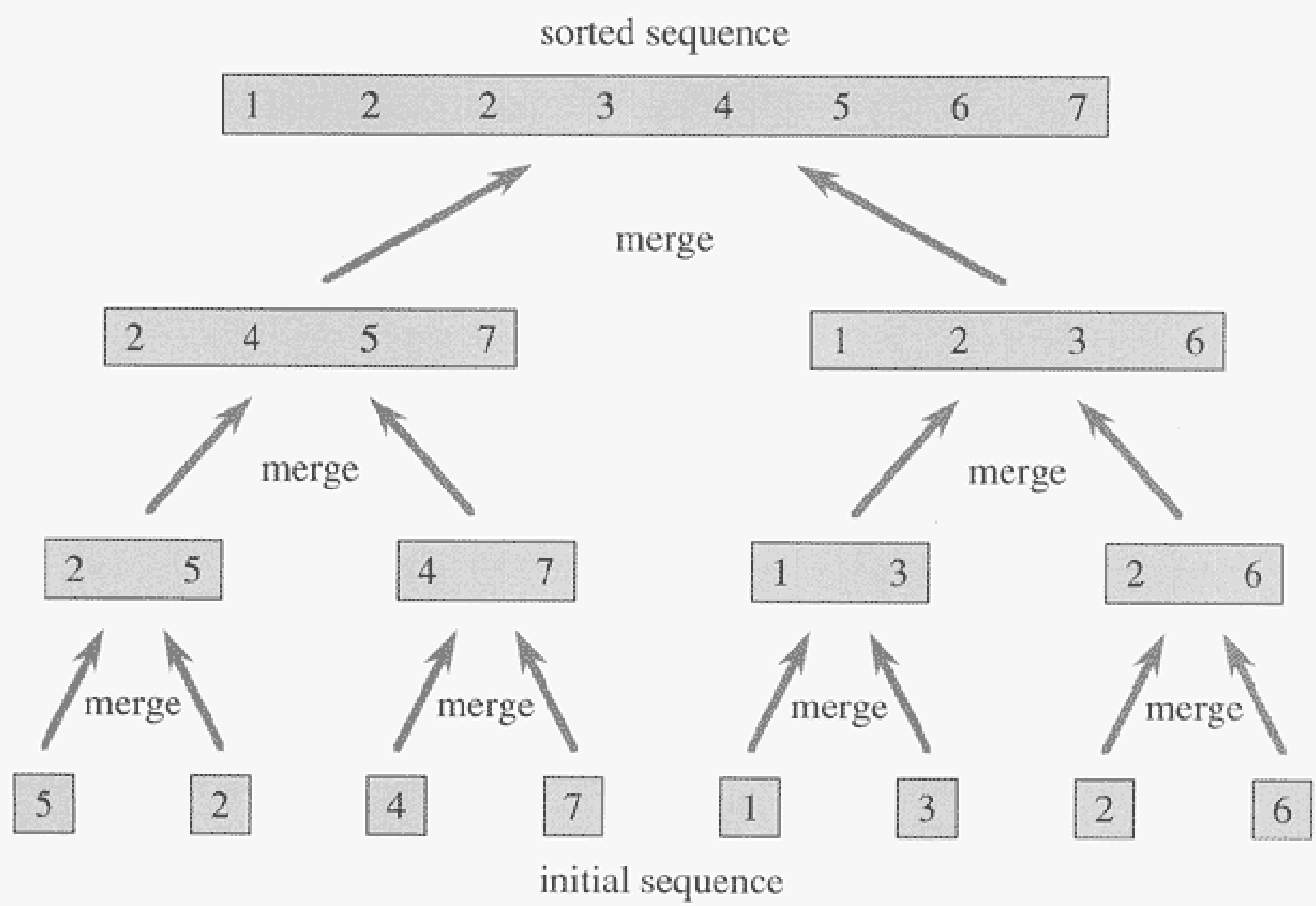
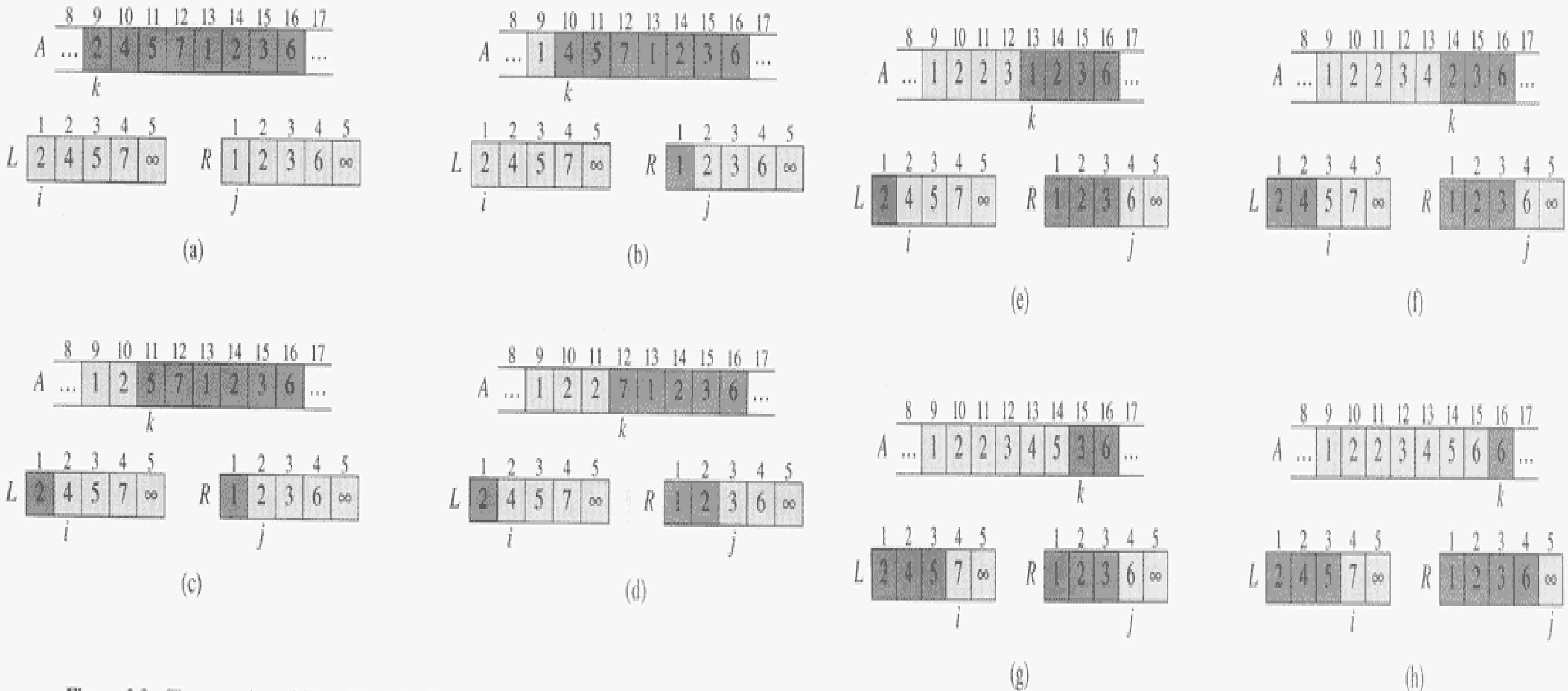


Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.



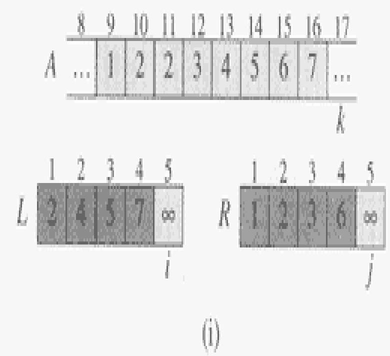


Figure 2.3 The operation of lines 10–17 in the call MERGE($A, 9, 12, 16$), when the subarray $A[9..16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$. After copying and inserting sentinels, the array L contains $\langle 2, 4, 5, 7, \infty \rangle$, and the array R contains $\langle 1, 2, 3, 6, \infty \rangle$. 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.

O(n) time

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

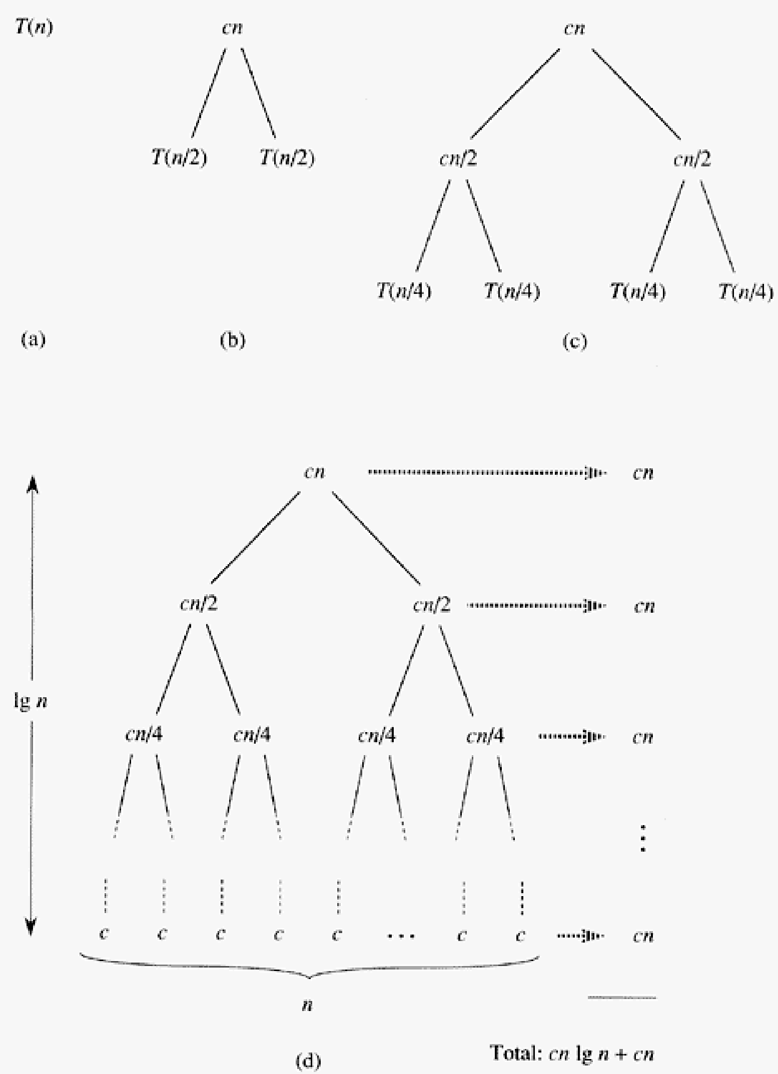


Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each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

Recursion Tree Method:
more examples

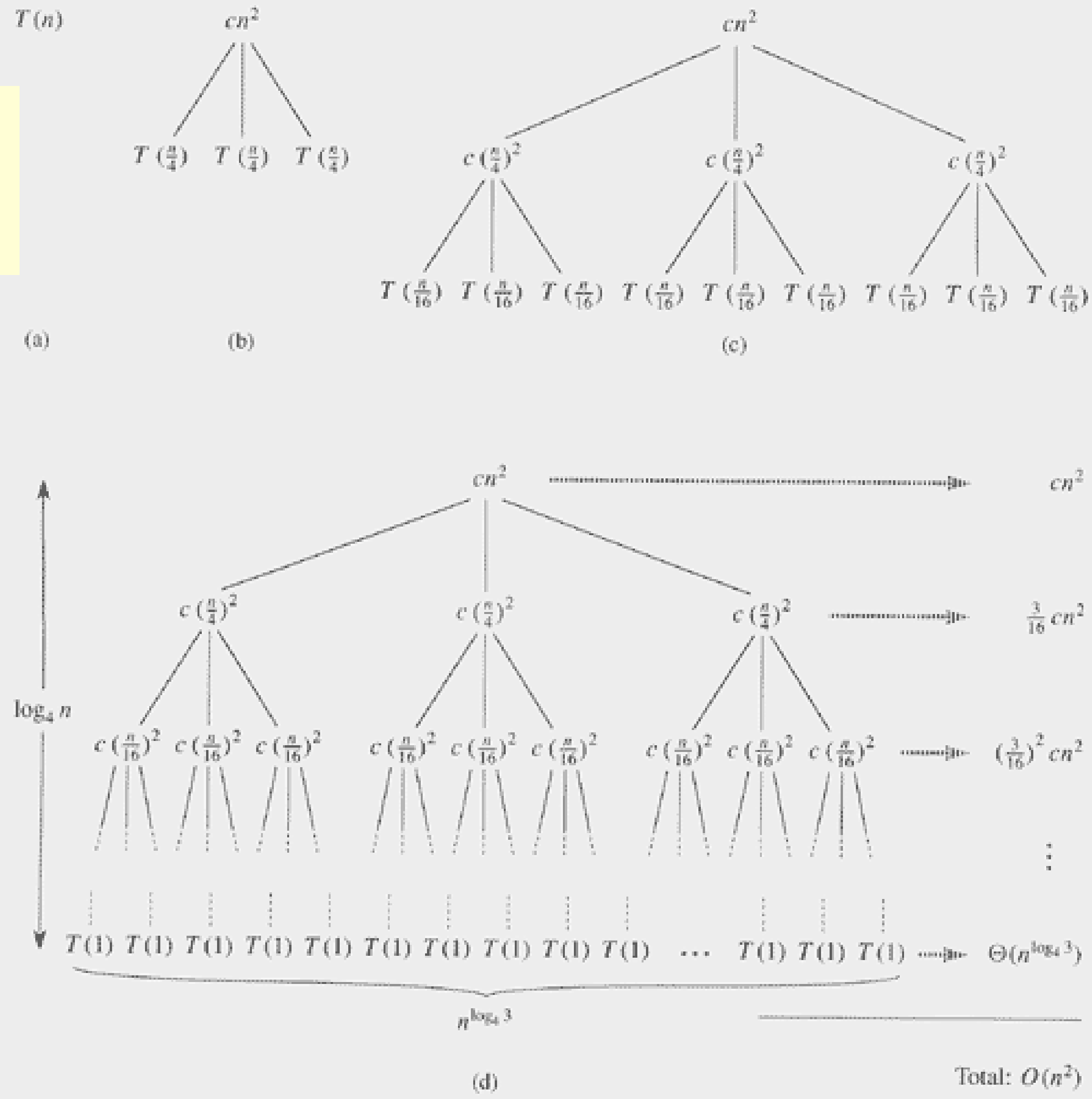


Figure 4.1 The construction of a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

Recursion Tree Method: more examples

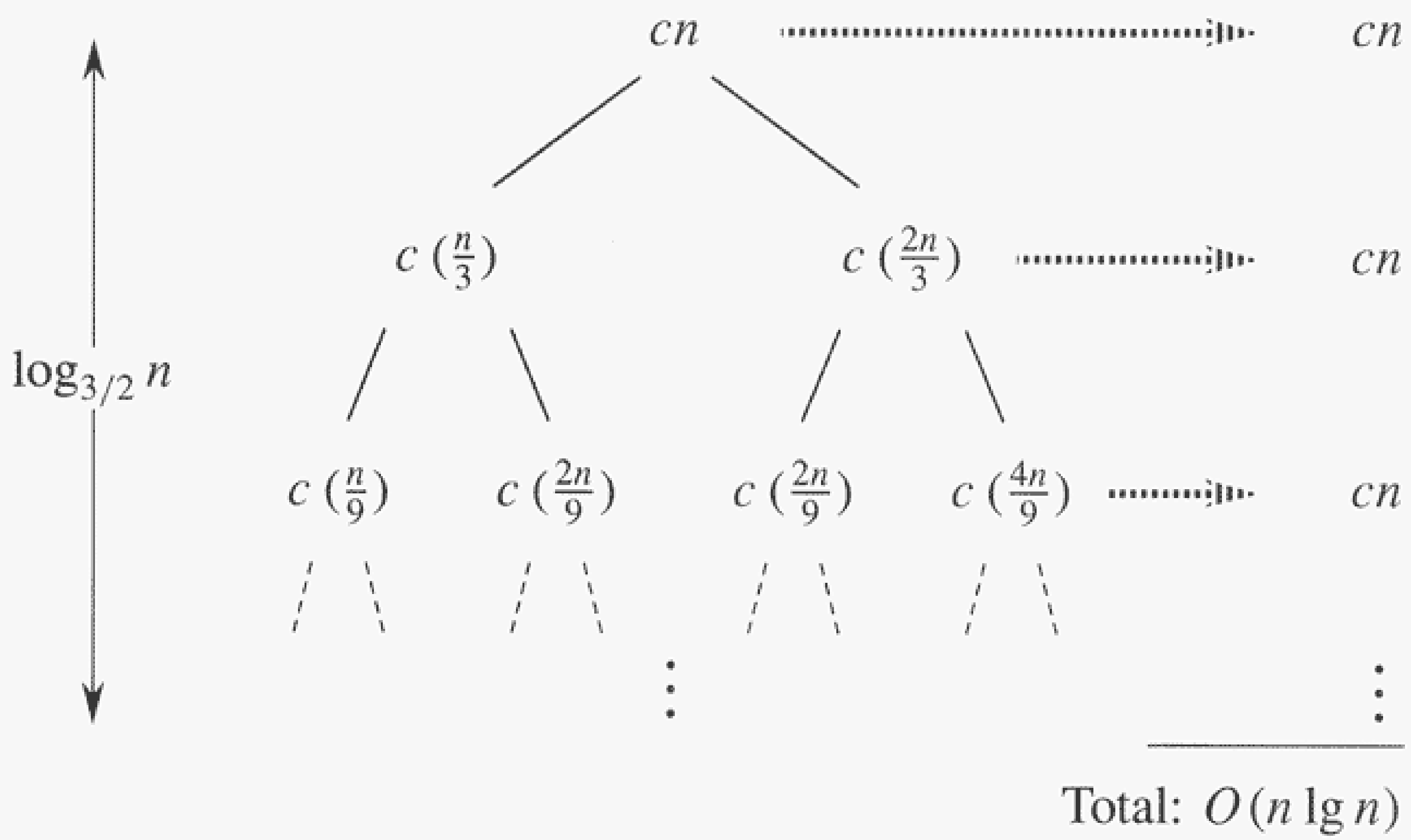


Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

MERGE-SORT(A, p, r)

if $p < r$

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

    MERGE-SORT(A, p, q)

    MERGE-SORT$(A, q + 1, r)$

    MERGE(A, p, q, r)

A recurrence relation for MergeSort

T(n) <= 2T(n/2) + O(n)

Solving Recurrence Relations

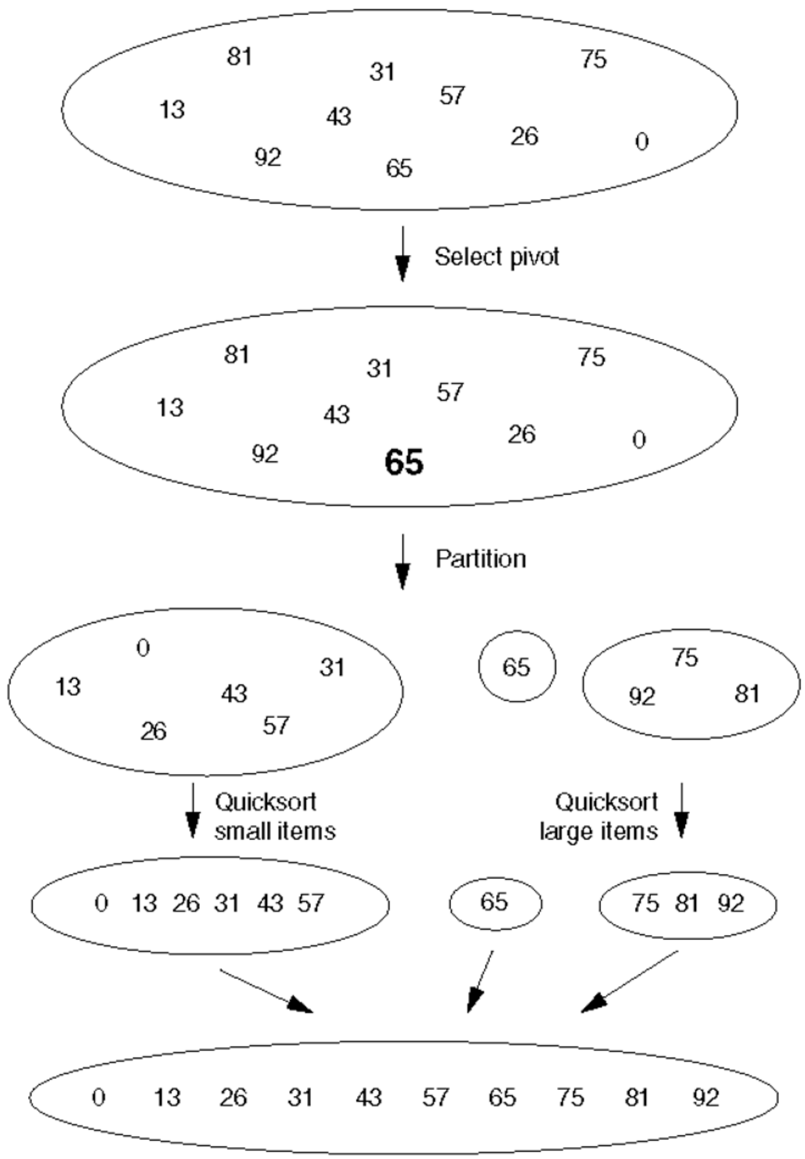
Page 62, [CLR]

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

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

```
QUICKSORT(array A, int p, int r)
if (p < r)
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

To sort array call QUICKSORT$(A, 1, length[A])$.

```
PARTITION(array A, int p, int r)
x ← A[r]                            ▷ Choose pivot
i ← p − 1
for j ← p to r − 1
    do if (A[j] ≤ x)
        then i ← i + 1
             exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ 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

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

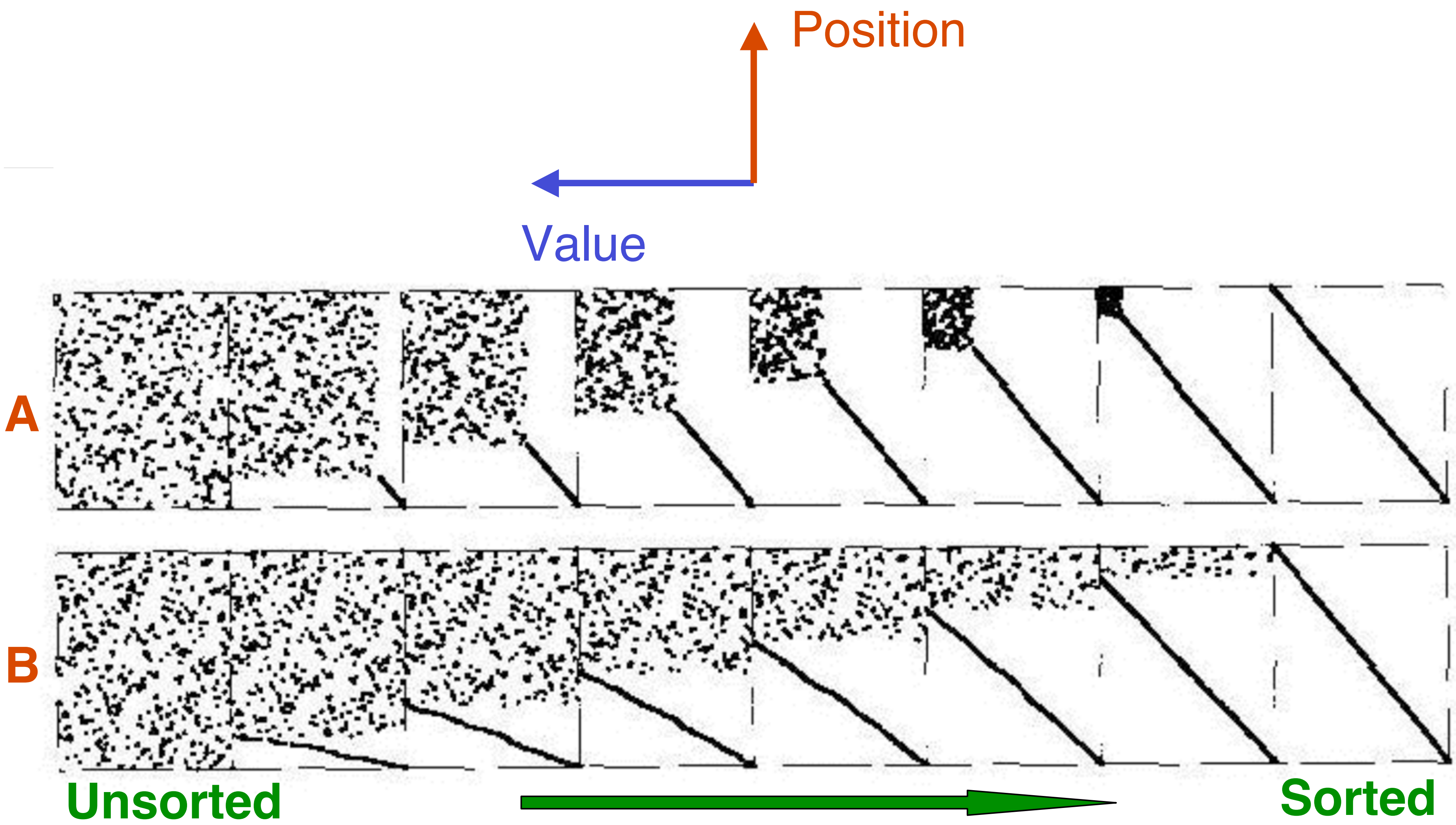
Master Theorem:

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

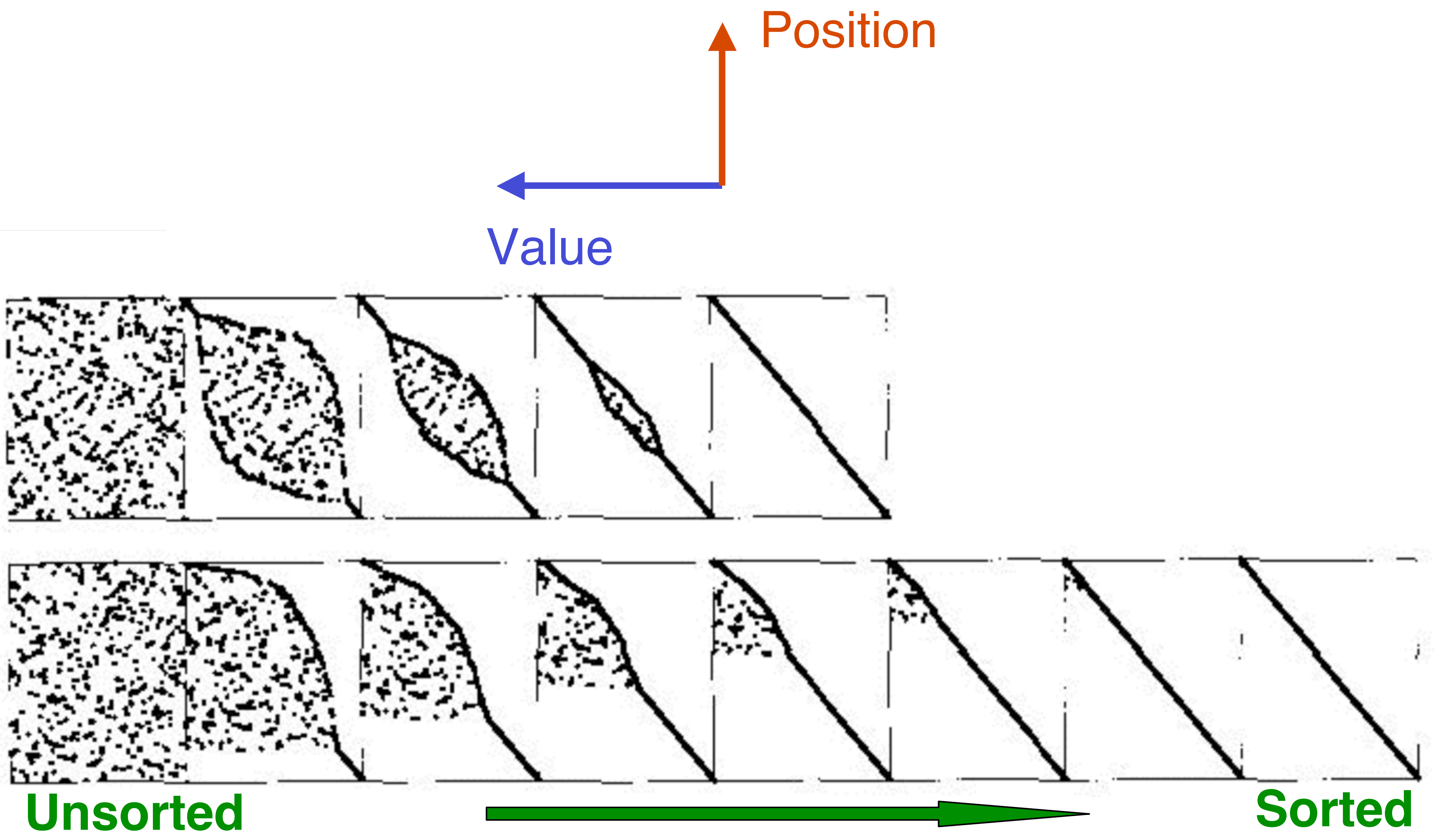
$$T(n) = aT(n/b) + f(n)$$

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))

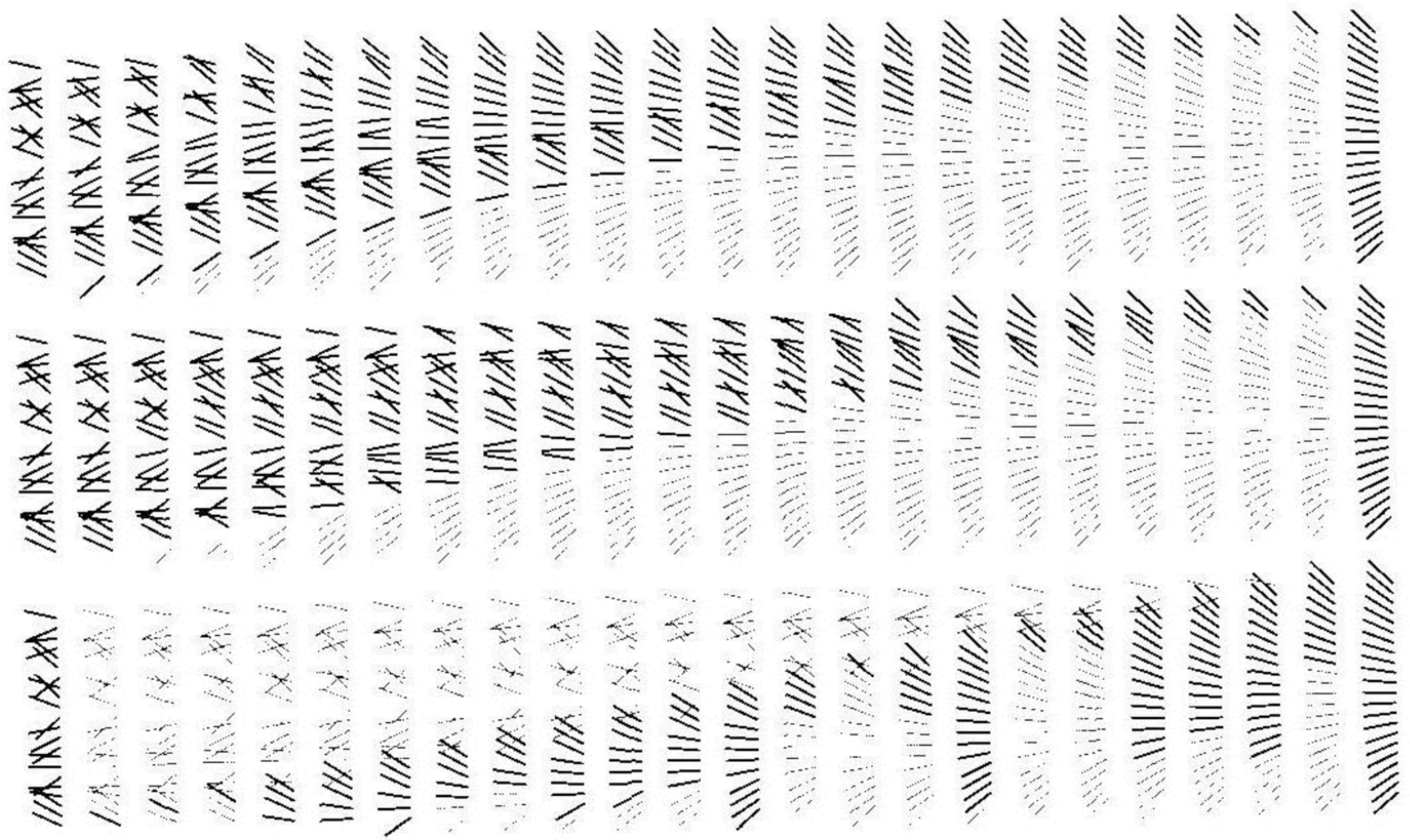
Visualizing Algorithms 1



Visualizing Algorithms 2



Visualizing Comparisons 3



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?