Exam Dates (Tentative)

- Midterm — October 9
- Final Exam — December 11 (??)
- Homework Assignments
  - Sep 11, Sep 23, Oct 2, Oct 14, Oct 23, Nov 4, Nov 18
- Quizzes
  - Sep 23, Oct 2, Oct 14, Oct 23, Nov 4, Nov 18,
- Semester Project — October 1

Sorting

- Input is a list of n items that can be compared.
- Output is an ordered list of those n items.
- Fundamental problem that has received a lot of attention over the years.
- Used in many applications.
- Scores of different algorithms exist.
- Task: To compare algorithms
  - On what bases?
    - Time
    - Space
    - Other

Sorting Algorithms

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

SelectionSort

Array Position		0	1	2	3	4	5
Initial State		8	5	9	2	6	3
After Iteration 1		2	5	9	8	6	3
After Iteration 2		2	3	9	8	6	5
After Iteration 3		2	3	5	8	6	9
After Iteration 4		2	3	5	6	8	9
After Iteration 5		2	3	5	6	8	9

How to prove invariants & correctness

- **Initialization**: prove it is true at start
- **Maintenance**: prove it is maintained within iterative control structures
- **Termination**: show how to use it to prove correctness

Algorithm Analysis

- Worst-case time complexity
- (Worst-case) space complexity
- Average-case time complexity

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

```
INSERTION-SORT(A)
  for j ← 2 to length[A]
       do key ← A[j]
          ▷ Insert A[j] into the sorted sequence A[1 .. j − 1].
          i ← j − 1
          while i > 0 and A[i] > key
              do A[i + 1] ← A[i]
                 i ← i − 1
          A[i + 1] ← key
```

Loop invariants and the correctness of insertion sort

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
do *key* $\leftarrow A[j]$	c_2	$n-1$
▷ 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

InsertionSort: Algorithm Invariant

- iteration k:
  - the first k items are in sorted order.

Figure 8.3

Basic action of insertion sort (the shaded part is sorted)

Array Position	0	1	2	3	4	5
Initial State	8	5	9	2	6	3
After a[0..1] is sorted	5	8	9	2	6	3
After a[0..2] is sorted	5	8	9	2	6	3
After a[0..3] is sorted	2	5	8	9	6	3
After a[0..4] is sorted	2	5	6	8	9	3
After a[0..5] is sorted	2	3	5	6	8	9

Figure 8.4

A closer look at the action of insertion sort (the dark shading indicates the sorted area; the light shading is where the new element was placed).

Array Position	0	1	2	3	4	5
Initial State	8	**5**				
After a[0..1] is sorted	5	8	**9**			
After a[0..2] is sorted	5	8	9	**2**		
After a[0..3] is sorted	2	5	8	9	**6**	
After a[0..4] is sorted	2	5	6	8	9	**3**
After a[0..5] is sorted	2	3	5	6	8	9

```
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]
```

$O(n^2)$ time

$O(1)$ space

BubbleSort: Algorithm Invariant

- In each pass, a scan is made in one direction and every item that does not have a smaller item after it, is moved as far up in the list as possible ("bubbled" up).
- Iteration k:
  - k smallest items are in the correct location.

ShakerSort

- In each pass, two scans are made first in one direction and then in the opposite direction;
- Every item that does not have a smaller item after it, is moved up in the list as far as possible ("bubbled" up) .
- Every item that does not have a larger item before it, is moved down in the list as far as possible ("bubbled" down) .

Animation Demos

http://cg.scs.carleton.ca/~morin/misc/sortalg/

Comparing $O(n^2)$ Sorting Algorithms

- InsertionSort and SelectionSort (and ShakerSort) are roughly twice as fast as BubbleSort for small files.
- InsertionSort is the best for very small files.
- $O(n^2)$ sorting algorithms are **NOT** useful for large random files.
- If comparisons are very expensive, then among the $O(n^2)$ sorting algorithms, InsertionSort is best.
- If data movements are very expensive, then among the $O(n^2)$ sorting algorithms, ?? is best.

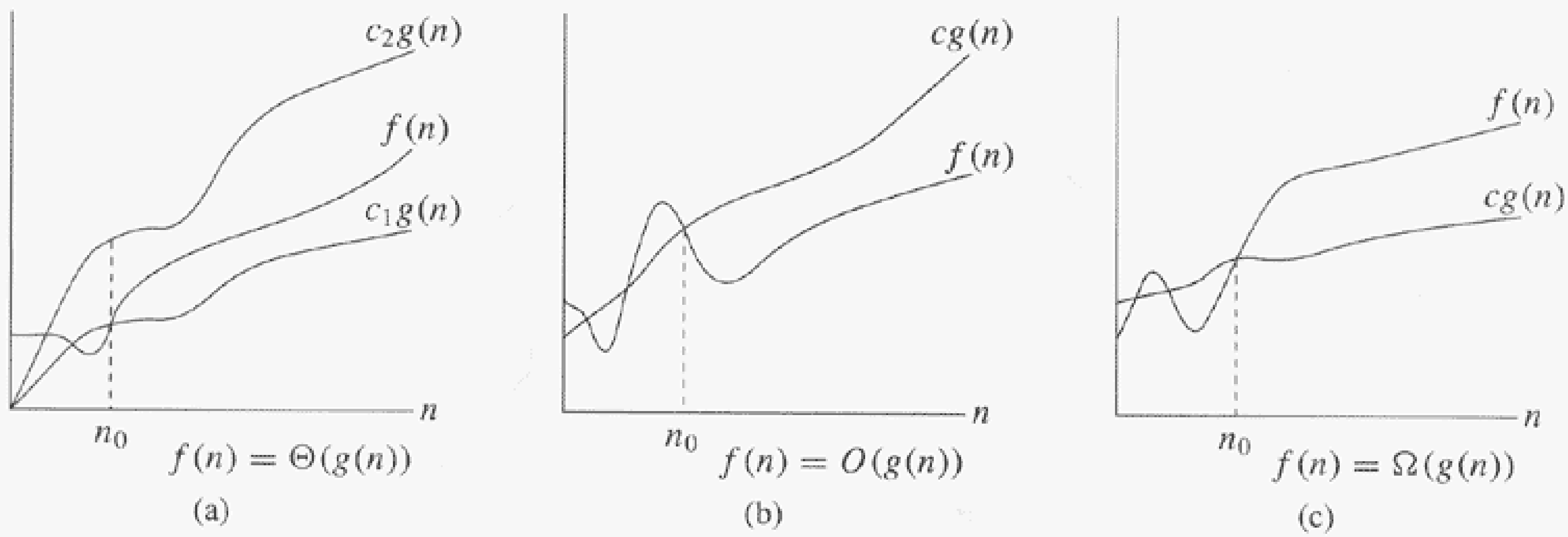

Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, 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 $cg(n)$.

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 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 **$T(n) \le cn\log n$** for some constant $c>0$

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

Thus $T(n) \le 2\, c(n/2)(\log(n/2)) + n$

$\le cn\log n - cn + n \le cn\log n$

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

This is the correct guess. WHY?

Show $T(n) \ge c'n\log n$ 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

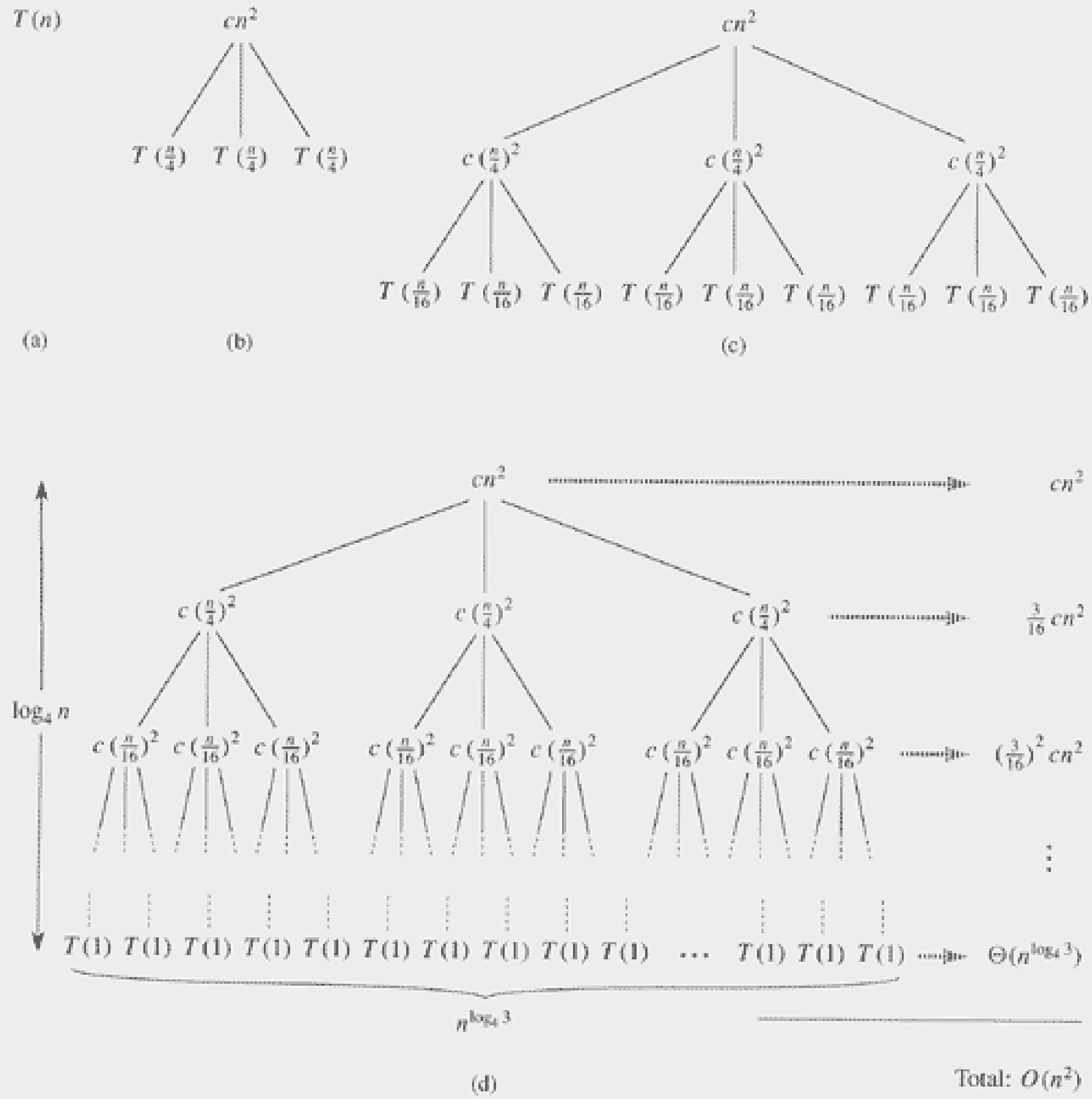

Figure 4.1 The construction of a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

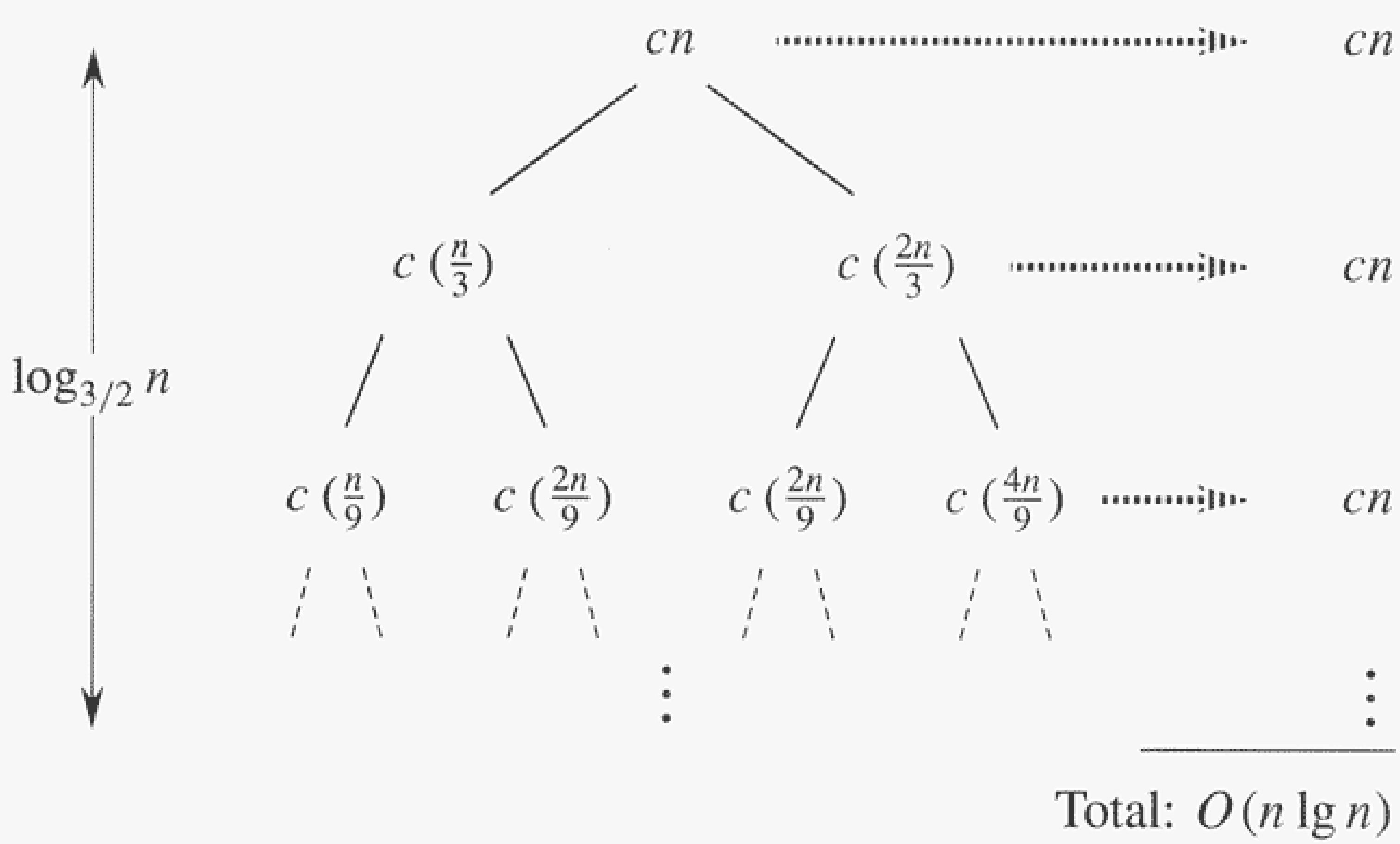

Figure 4.2 A recursion tree for the recurrence $T(n) = T(n/3) + T(2n/3) + cn$.

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?