Analyzing the complexity of algorithms (II)

Slide credit: David Luebke(Virginia)

Review: Asymptotic Performance

- *Asymptotic performance*: How does algorithm behave as the problem size gets very large?
  - Running time
  - Memory/storage requirements
- Remember that we use the RAM model:
  - All memory equally expensive to access
  - No concurrent operations
  - All reasonable instructions take unit time
    - Except, of course, function calls
  - Constant word size
    - Unless we are explicitly manipulating bits

Review: Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
  - We can be more exact if need be
- Worst case vs. average case

An Example: Insertion Sort

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```

For any element: find its position in relation to previous elements and insert it there, shifting other elements.

An Example: Insertion Sort

30	*10*	*40*	*20*
1	2	3	4

$i = \varnothing \quad j = \varnothing \quad key = \varnothing$

$A[j] = \varnothing \qquad A[j+1] = \varnothing$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

30	*10*	*40*	*20*
1	2	3	4

i = 2   j = 1   key = 10
A[j] = 30   A[j+1] = 10

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

30	*30*	*40*	*20*
1	2	3	4

i = 2 j = 1 key = 10
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

30	*30*	*40*	*20*
1	2	3	4

i = 2 j = 1 key = 10
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

30	*30*	*40*	*20*
1	2	3	4

$i = 2 \quad j = 0 \quad key = 10$

$A[j] = \varnothing \quad A[j+1] = 30$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

30	*30*	*40*	*20*
1	2	3	4

$i = 2$ $j = 0$ $key = 10$

$A[j] = \varnothing$ $A[j+1] = 30$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*20*

$i = 2 \quad j = 0 \quad key = 10$

$A[j] = \emptyset \quad A[j+1] = 10$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*20*

i = 3 j = 0 key = 10

A[j] = ∅ A[j+1] = 10

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

$i = 3 \quad j = 0 \quad key = 40$

$A[j] = \emptyset \quad A[j+1] = 10$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	30	40	20
1	2	3	4

$i = 3 \quad j = 0 \quad key = 40$

$A[j] = \varnothing \quad A[j+1] = 10$

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 3   j = 2   key = 40

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 3 j = 2 key = 40

A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 3 j = 2 key = 40
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*20*

i = 4	j = 2	key = 40
A[j] = 30		A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 2 key = 20

A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 4 j = 3	key = 20
A[j] = 40	A[j+1] = 20

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*20*
1	2	3	4

i = 4 j = 3 key = 20
A[j] = 40 A[j+1] = 20

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20

A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20

A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*40*	*40*

i = 4 j = 3 key = 20

A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*40*
1	2	3	4

i = 4   j = 2   key = 20

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*40*	*40*
1	2	3	4

i = 4   j = 2   key = 20
A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*30*	*40*
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*30*	*40*

i = 4 j = 2 key = 20

A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*30*	*30*	*40*
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

1	2	3	4
10	*30*	*30*	*40*

i = 4   j = 1   key = 20
A[j] = 10   A[j+1] = 30

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*20*	*30*	*40*
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

An Example: Insertion Sort

10	*20*	*30*	*40*
1	2	3	4

i = 4	j = 1	key = 20
A[j] = 10		A[j+1] = 20

```
InsertionSort(A, n) {
   for i = 2 to n {
   key = A[i]
   j = i - 1;
   while (j > 0) and (A[j] > key) {
   A[j+1] = A[j]
   j = j - 1
   }
   A[j+1] = key
   }
}
```

Done!

Insertion Sort

What is the *precondition* *for this loop?*

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
          A[j+1] = A[j]
          j = j - 1
      }
      A[j+1] = key
  }
}
```

Insertion Sort

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
           A[j+1] = A[j]
           j = j - 1
      }
      A[j+1] = key
  }
}
```

How many times will this loop execute?

Insertion Sort

<u>Statement</u> <u>Effort</u>

```
InsertionSort(A, n) {
  for i = 2 to n {                              c1n
  key = A[i]                                    c2(n-1)
  j = i - 1;                                    c3(n-1)
  while (j > 0) and (A[j] > key) {              c4T
  A[j+1] = A[j]                                 c5(T-(n-1))
  j = j - 1                                     c6(T-(n-1))
  }                                             0
  A[j+1] = key                                  c7(n-1)
  }                                             0
}
```

$T = t_2 + t_3 + \ldots + t_n$ where t_i is number of while expression evaluations for the i^{th} for loop iteration

Analyzing Insertion Sort

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 T + c_5(T - (n-1)) + c_6(T - (n-1)) + c_7(n-1)$

  $= c_8 T + c_9 n + c_{10}$
- What can T be?
  - Best case -- inner loop body never executed
    - o $t_i = 1 \rightarrow T(n)$ is a linear function
  - Worst case -- inner loop body executed for all previous elements
    - o $t_i = i \rightarrow T(n)$ is a quadratic function
  - Average case
    - o ???

Analysis

- Simplifications
  - Ignore actual and abstract statement costs
  - *Order of growth* is the interesting measure:
    - Highest-order term is what counts
      - Remember, we are doing asymptotic analysis
      - As the input size grows larger it is the high order term that dominates

Upper Bound Notation

- We say InsertionSort's run time is $O(n^2)$
  - Properly we should say run time is *in* $O(n^2)$
  - Read O as "Big-O" (you'll also hear it as "order")
- In general a function
  - f(n) is O(g(n)) if there exist positive constants c and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
- Formally
  - $O(g(n)) = \{ f(n)\colon \exists$ positive constants c and n_0 such that $f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0$

Insertion Sort Is $O(n^2)$

- Proof
  - Suppose runtime is $an^2 + bn + c$
    - If any of a, b, and c are less than 0 replace the constant with its absolute value
  - $an^2 + bn + c \le (a + b + c)n^2 + (a + b + c)n + (a + b + c)$
  - $\le 3(a + b + c)n^2$ for $n \ge 1$
  - Let $c' = 3(a + b + c)$ and let $n_0 = 1$
- Question
  - Is InsertionSort $O(n^3)$?
  - Is InsertionSort $O(n)$?

Big O Fact

- A polynomial of degree k is $O(n^k)$
- Proof:
  - Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$
    - Let $a_i = | b_i |$
  - $f(n) \le a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

$$\le \; n^k \sum a_i \frac{n^i}{n^k} \; \le \; n^k \sum a_i \; \le \; cn^k$$

Lower Bound Notation

- We say InsertionSort's run time is $\Omega(n)$
- In general a function
  - $f(n)$ is $\Omega(g(n))$ if $\exists$ positive constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n) \ \forall\, n \geq n_0$
- Proof:
  - Suppose run time is $an + b$
    - Assume a and b are positive (what if b is negative?)
  - $an \leq an + b$

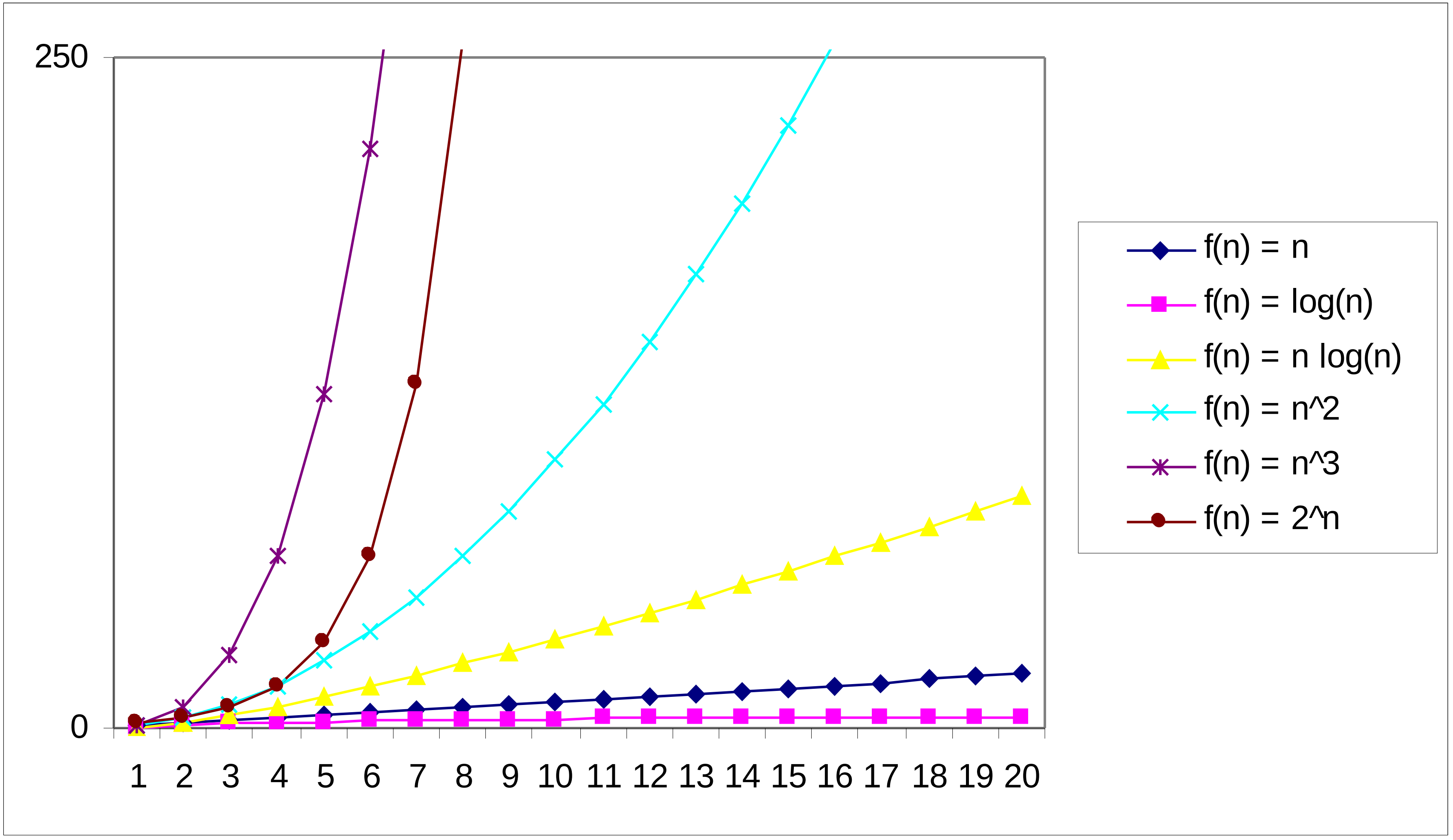
Asymptotic Tight Bound

- A function f(n) is Θ(g(n)) if ∃ positive constants c_1, c_2, and n_0 such that

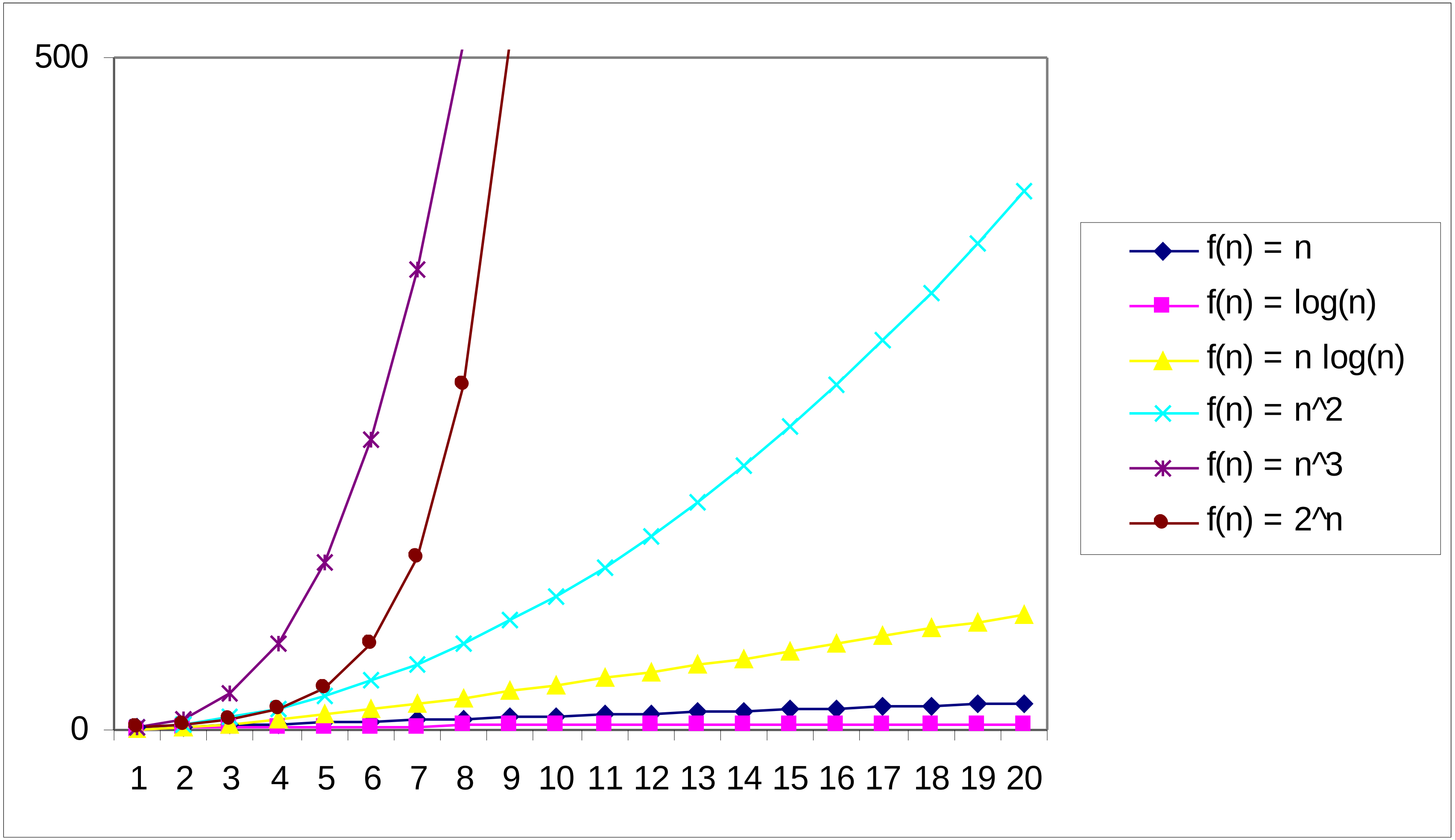
$$c_1\ g(n) \le f(n) \le c_2\ g(n)\ \forall\ n \ge n_0$$

- Theorem
  - f(n) is Θ(g(n)) iff f(n) is both O(g(n)) and Ω(g(n))
  - Proof: someday

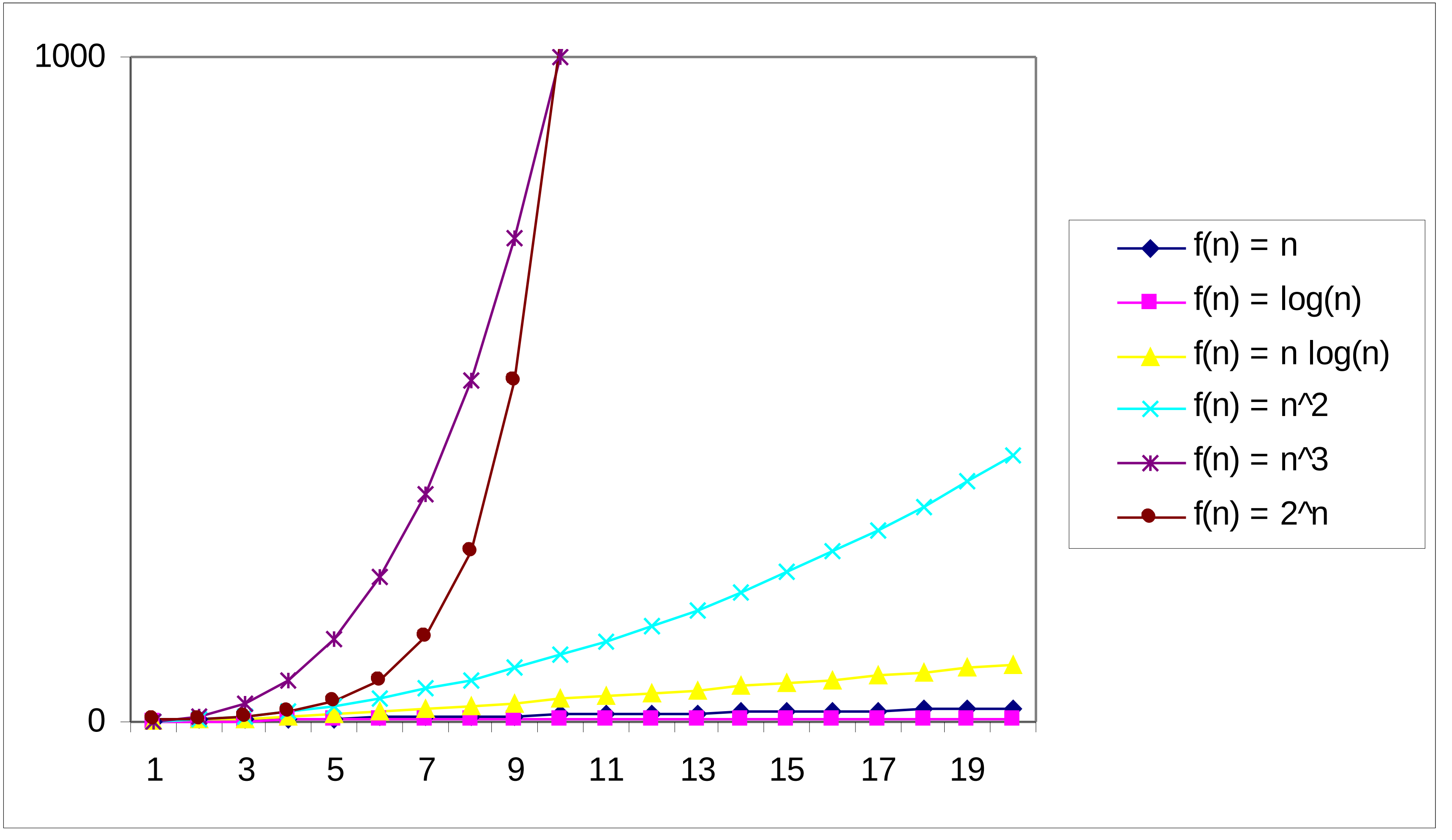
Practical Complexity



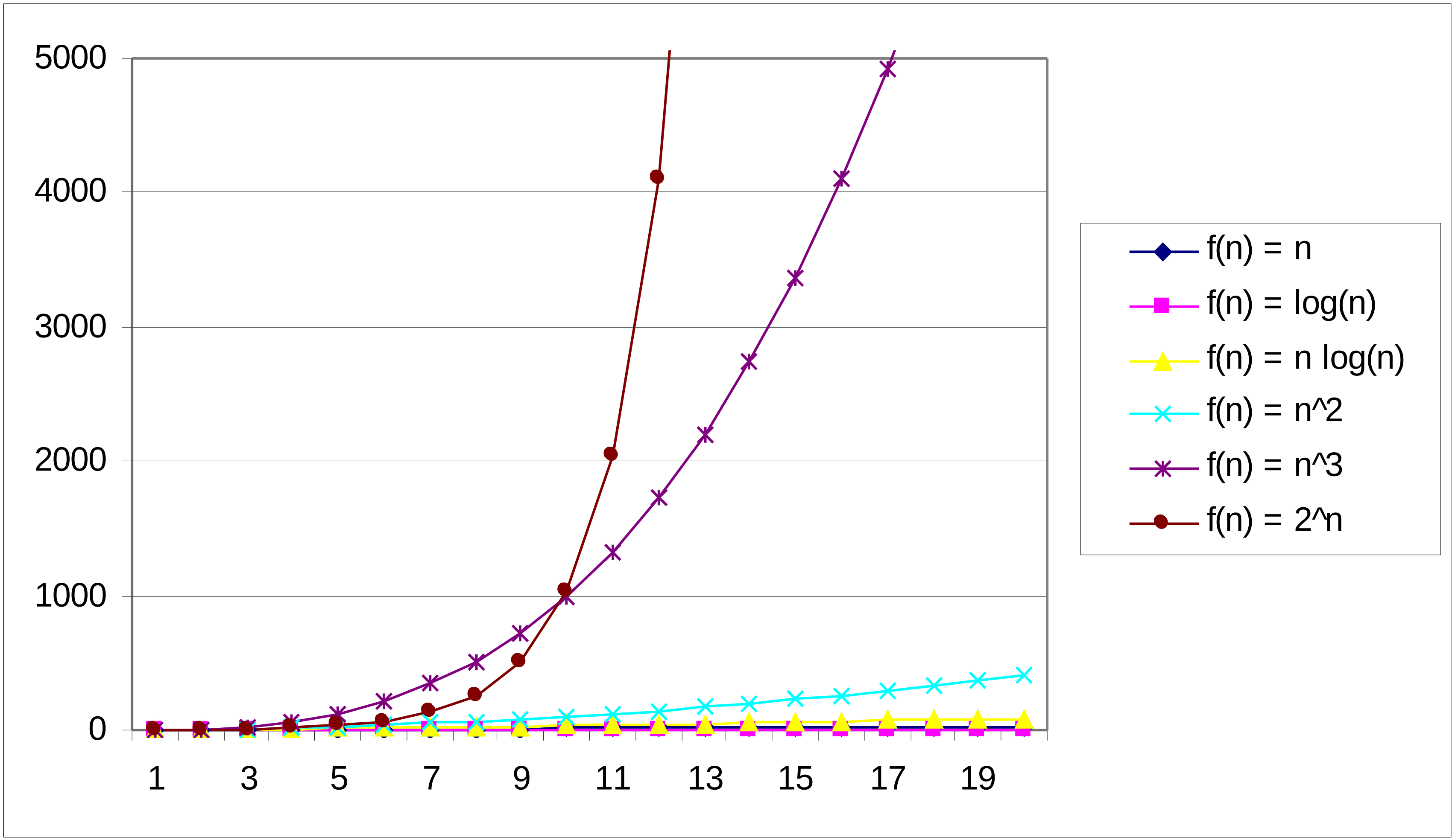
Practical Complexity



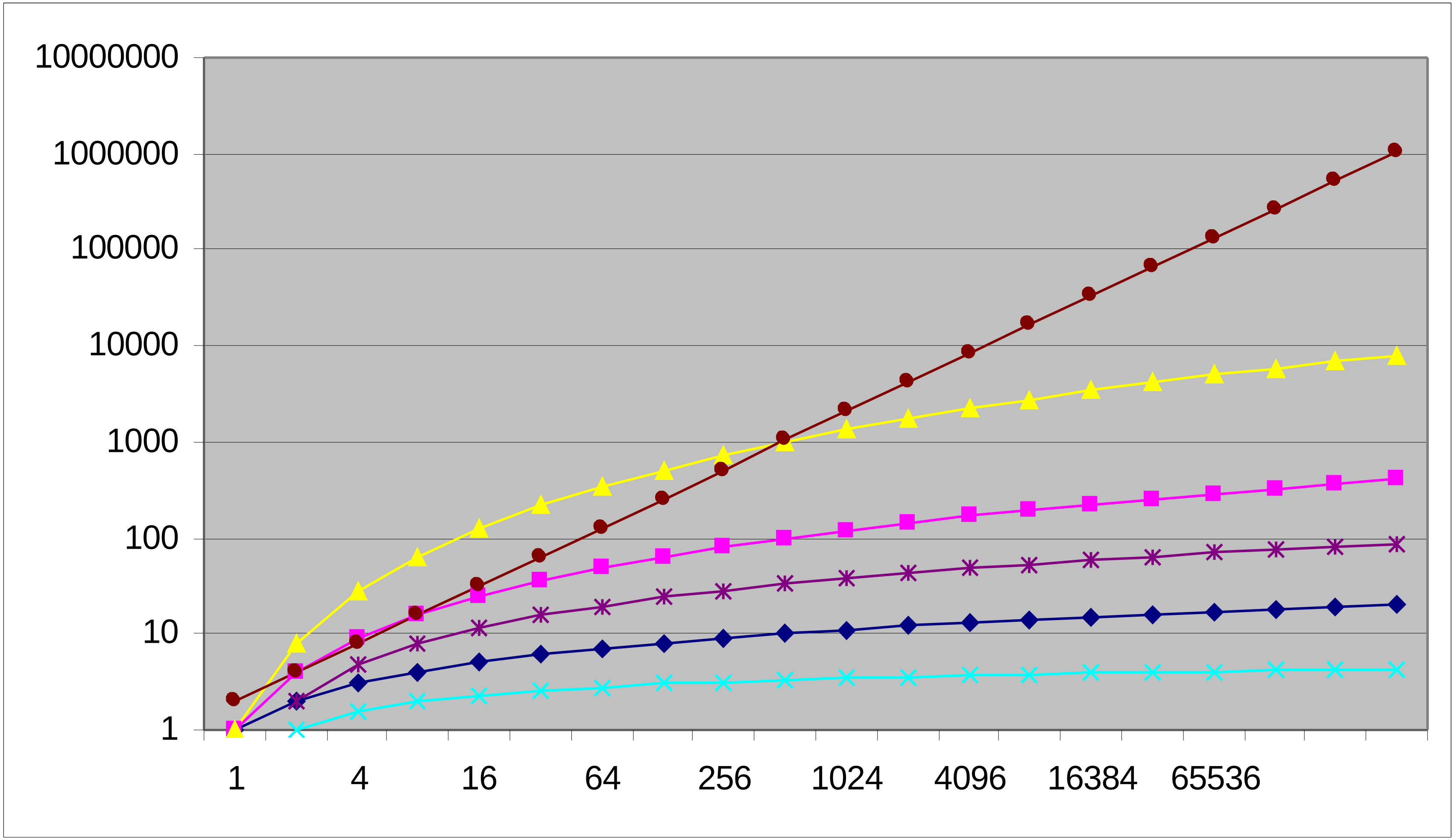
Practical Complexity



Practical Complexity



Practical Complexity



Other Asymptotic Notations

- A function f(n) is o(g(n)) if $\exists$ positive constants c and n_0 such that

  $$f(n) < c\, g(n) \;\forall\; n \geq n_0$$

- A function f(n) is ω(g(n)) if $\exists$ positive constants c and n_0 such that

  $$c\, g(n) < f(n) \;\forall\; n \geq n_0$$

-
  - o() is like $<$
  - ω() is like $>$
  - Θ() is like $=$
  - O() is like $\leq$
  - Ω() is like $\geq$

Merge Sort

```
MergeSort(A, left, right) {
  if (left < right) {
  mid = floor((left + right) / 2);
  MergeSort(A, left, mid);
  MergeSort(A, mid+1, right);
  Merge(A, left, mid, right);
  }
}

// Merge() takes two sorted subarrays of A and
// merges them into a single sorted subarray of A
//    (how long should this take?)
```

Merge Sort: Example

- Show MergeSort() running on the array

```
A = {10, 5, 7, 6, 1, 4, 8, 3, 2, 9};
```

Analysis of Merge Sort

Statement Effort

```
MergeSort(A, left, right) {                   T(n)
   if (left < right) {                        Θ(1)
      mid = floor((left + right) / 2);           Θ(1)
      MergeSort(A, left, mid);                   T(n/2)
      MergeSort(A, mid+1, right);                T(n/2)
      Merge(A, left, mid, right);                Θ(n)
   }
}
```

- So T(n) = $\Theta(1)$ when n = 1, and
  $2T(n/2) + \Theta(n)$ when n > 1
- So what (more succinctly) is T(n)?

Recurrences

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\dfrac{n}{2}\right) + cn & n > 1 \end{cases}$$

  is a *recurrence*.
  - Recurrence: an equation that describes a function in terms of its value on smaller functions

Recurrence Examples

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Solving Recurrences

- Substitution method
- Iteration method
- Master method

Solving Recurrences

- The substitution method (CLR 4.1)
  - A.k.a. the "making a good guess method"
  - Guess the form of the answer, then use induction to find the constants and show that solution works
  - Examples:
    - $T(n) = 2T(n/2) + \Theta(n) \rightarrow T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \rightarrow$???

Solving Recurrences

- The substitution method (CLR 4.1)
  - A.k.a. the "making a good guess method"
  - Guess the form of the answer, then use induction to find the constants and show that solution works
  - Examples:
    - $T(n) = 2T(n/2) + \Theta(n) \rightarrow T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \rightarrow T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + 17) + n \rightarrow$???

Solving Recurrences

- The substitution method (CLR 4.1)
  - A.k.a. the “making a good guess method”
  - Guess the form of the answer, then use induction to find the constants and show that solution works
  - Examples:
    - $T(n) = 2T(n/2) + \Theta(n) \rightarrow \quad T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \rightarrow T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \rightarrow \Theta(n \lg n)$

Solving Recurrences

- Another option is what the book calls the "iteration method"
  - Expand the recurrence
  - Work some algebra to express as a summation
  - Evaluate the summation
- We will show several examples

$$s(n)=\begin{cases}0 & n=0\\ c+s(n-1) & n>0\end{cases}$$

- s(n) =

  c + s(n-1)

  c + c + s(n-2)

  2c + s(n-2)

  2c + c + s(n-3)

  3c + s(n-3)

  …

  kc + s(n-k) = ck + s(n-k)

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have
  - s(n) = ck + s(n-k)
- What if k = n?
  - s(n) = cn + s(0) = cn

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have
  - s(n) = ck + s(n-k)
- What if k = n?
  - s(n) = cn + s(0) = cn
- So

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

- Thus in general
  - s(n) = cn

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- s(n)

= n + s(n-1)

= n + n-1 + s(n-2)

= n + n-1 + n-2 + s(n-3)

= n + n-1 + n-2 + n-3 + s(n-4)

= …

= n + n-1 + n-2 + n-3 + … + n-(k-1) + s(n-k)

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

- So far for n >= k we have

$$\sum_{i=n-k+1}^{n} i \quad + \quad s(n-k)$$

- What if k = n?

$$\sum_{i=1}^{n} i \quad + \quad s(0) \quad = \quad \sum_{i=1}^{n} i \quad + \quad 0 \quad = \quad n\frac{n+1}{2}$$

- Thus in general

$$s(n) \quad = \quad n\frac{n+1}{2}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

- $T(n) =$

  $2T(n/2) + c$

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

  $2^2T(n/2^2) + 2c + c$

  $2^2(2T(n/2^2/2) + c) + 3c$

  $2^3T(n/2^3) + 4c + 3c$

  $2^3T(n/2^3) + 7c$

  $2^3(2T(n/2^3/2) + c) + 7c$

  $2^4T(n/2^4) + 15c$

  …

  $2^kT(n/2^k) + (2^k - 1)c$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\dfrac{n}{2}\right) + c & n > 1 \end{cases}$$

- So far for n > 2k we have
  - $T(n) = 2^k T(n/2^k) + (2^k - 1)c$
- What if k = lg n?
  - $T(n) = 2^{\lg n}\, T(n/2^{\lg n}) + (2^{\lg n} - 1)c$

    $= n\, T(n/n) + (n - 1)c$

    $= n\, T(1) + (n-1)c$

    $= nc + (n-1)c = (2n - 1)c$