ASYMPTOTIC PERFORMANCE

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
  }
}
```

An Example: Insertion Sort

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

$i = \emptyset$ $j = \emptyset$ $key = \emptyset$

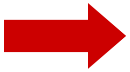
$A[j] = \emptyset$ $A[j+1] = \emptyset$

```
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

$i = 2$ $j = 0$ $key = 10$
$A[j] = \varnothing$ $A[j+1] = 30$

```
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

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

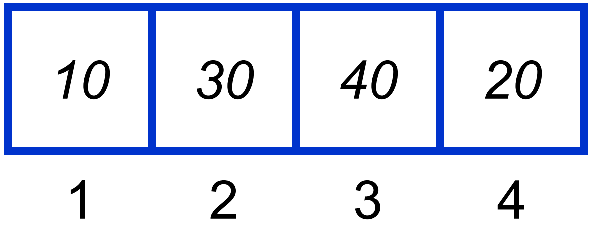
10	*30*	*40*	*20*
1	2	3	4

i = 2   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

$i = 3$ $j = 0$ $key = 10$

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

```
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

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

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

```
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

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

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

```
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

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

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

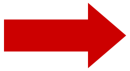
```
InsertionSort(A, n) {
   for i = 2 to n {
        key = A[i]
        j = i - 1;
        while (j > 0) and (A[j] > key) {
⇨              A[j+1] = A[j]
               j = j - 1
        }
        A[j+1] = key
   }
}
```

An Example: Insertion Sort

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

$i = 4$ $j = 2$ $key = 20$

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

```
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

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

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

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

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!

Animating Insertion Sort

- Check out the Animator, a java applet at:

  http://www.cs.hope.edu/~alganim/animator/Animator.html

- Try it out with random, ascending, and descending inputs

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
  }
}
```

What is the precondition for this loop?

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

```
    Statement                                            Effort
InsertionSort(A, n) {
   for i = 2 to n {                                      $c_1 n$
        key = A[i]                                       $c_2(n-1)$
        j = i - 1;                                       $c_3(n-1)$
        while (j > 0) and (A[j] > key) {                 $c_4 T$
              A[j+1] = A[j]                              $c_5(T-(n-1))$
              j = j - 1                                  $c_6(T-(n-1))$
        }                                                0
        A[j+1] = key                                     $c_7(n-1)$
   }                                                     0
}
```

$T = t_2 + t_3 + \dots + t_n$ 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
    - $t_i = 1$ ➔ T(n) is a linear function
  - Worst case -- inner loop body executed for all previous elements
    - $t_i = i$ ➔ T(n) is a quadratic function
  - Average case
    - ???

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
    - $O(g(n)) = \{ f(n)\text{: } \exists$ 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
  - $an^2 + bn + c \leq (a + b + c)n^2 + (a + b + c)n + (a + b + c)$
  - $\leq 3(a + b + c)n^2$ for $n \geq 1$
  - 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 $a_i = | b_i |$
  - $f(n) \leq a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

$$\leq \quad n^k \sum a_i \frac{n^i}{n^k} \quad \leq \quad n^k \sum a_i \quad \leq \quad cn^k$$

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 $0 \le c \cdot g(n) \le f(n) \quad \forall\ n \ge n_0$
- Proof:
  - Suppose run time is $an + b$
    - Assume a and b are positive (what if b is negative?)
  - $an \le an + b$

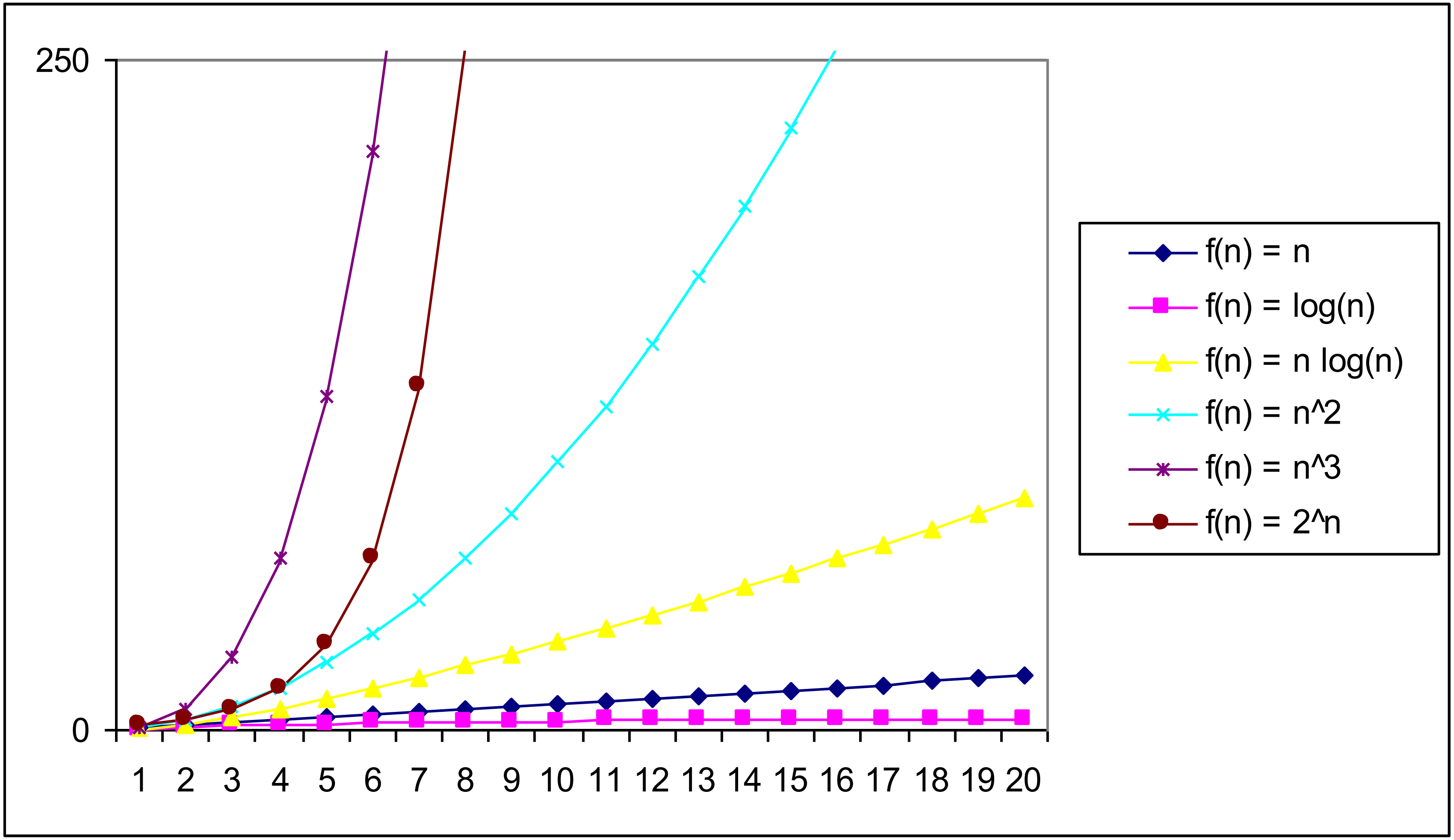
Asymptotic Tight Bound

- A function f(n) is $\Theta(g(n))$ if $\exists$ positive constants c_1, c_2, and n_0 such that

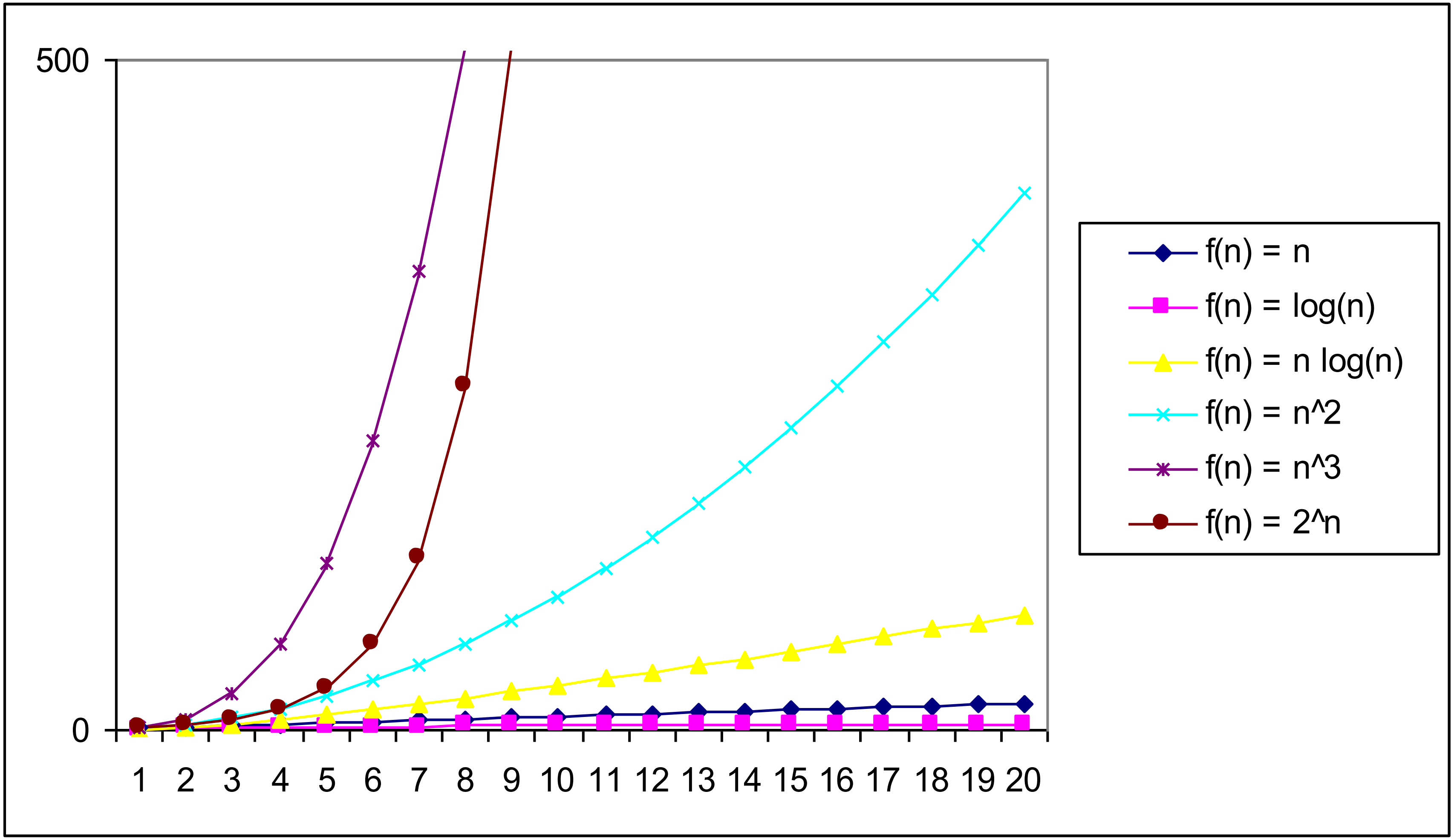
$$c_1\ g(n) \le f(n) \le c_2\ g(n)\ \forall\ n \ge n_0$$

- Theorem
  - f(n) is $\Theta(g(n))$ iff f(n) is both $O(g(n))$ and $\Omega(g(n))$
  - Proof: someday

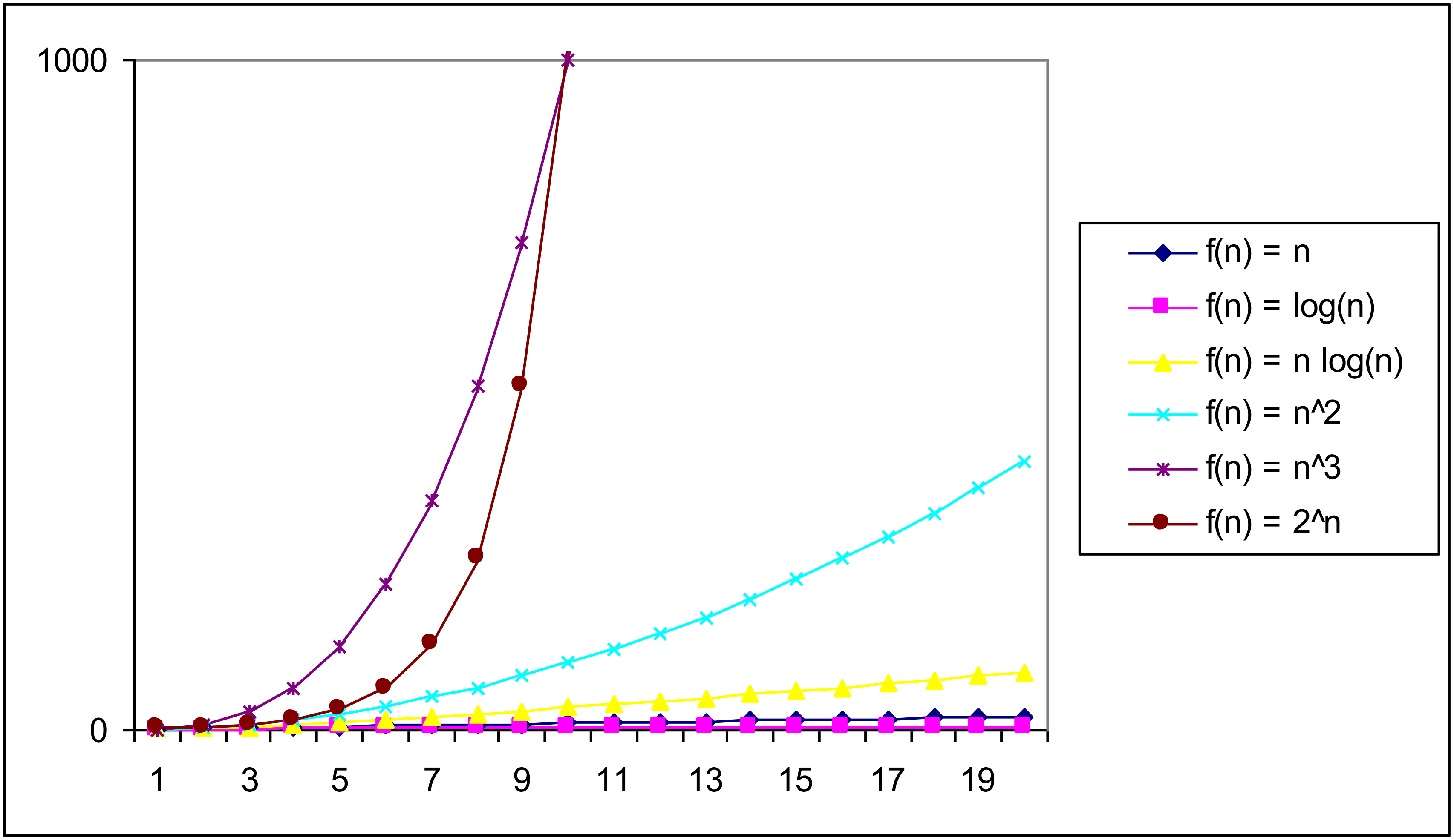
Practical Complexity



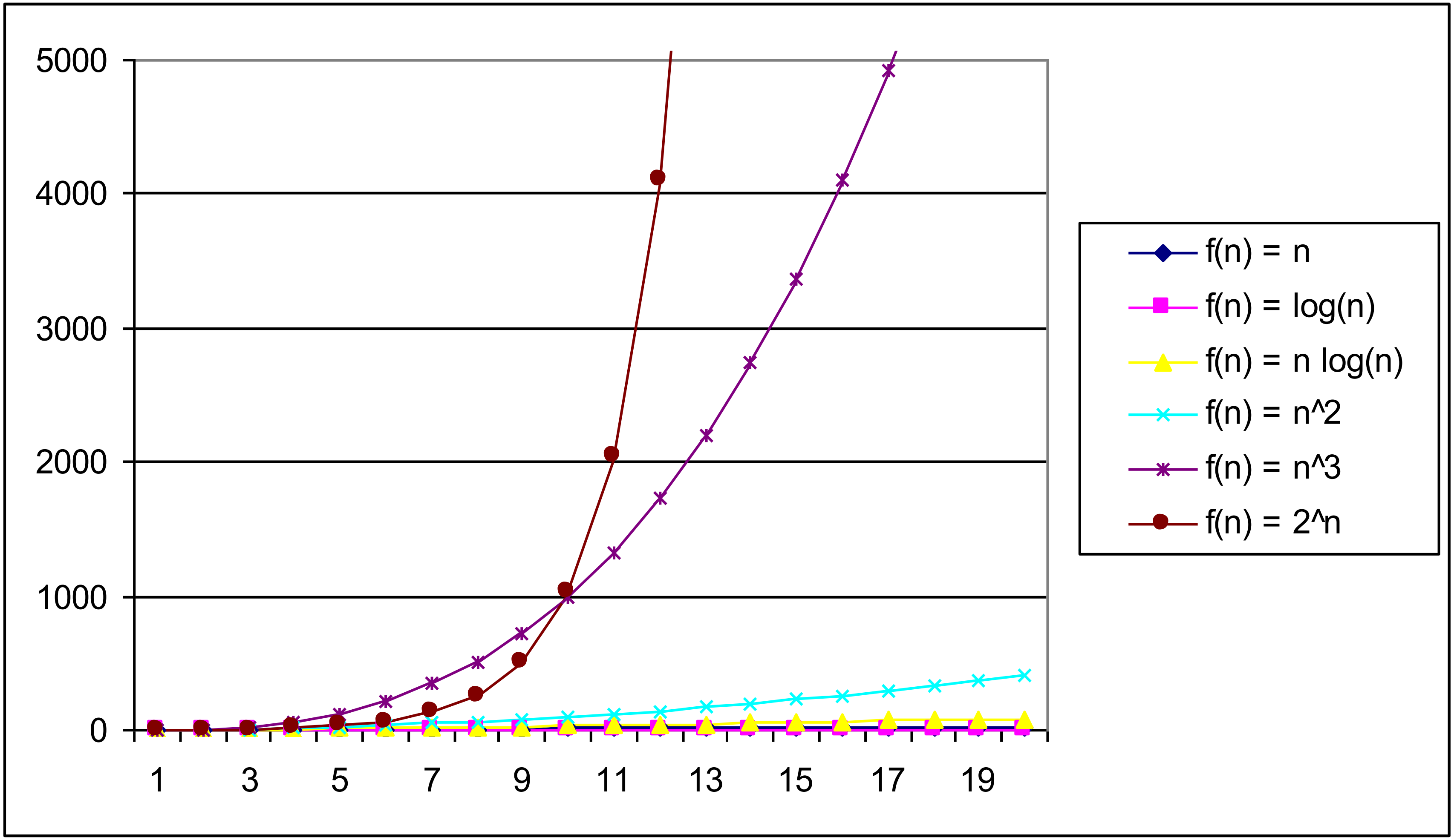
Practical Complexity



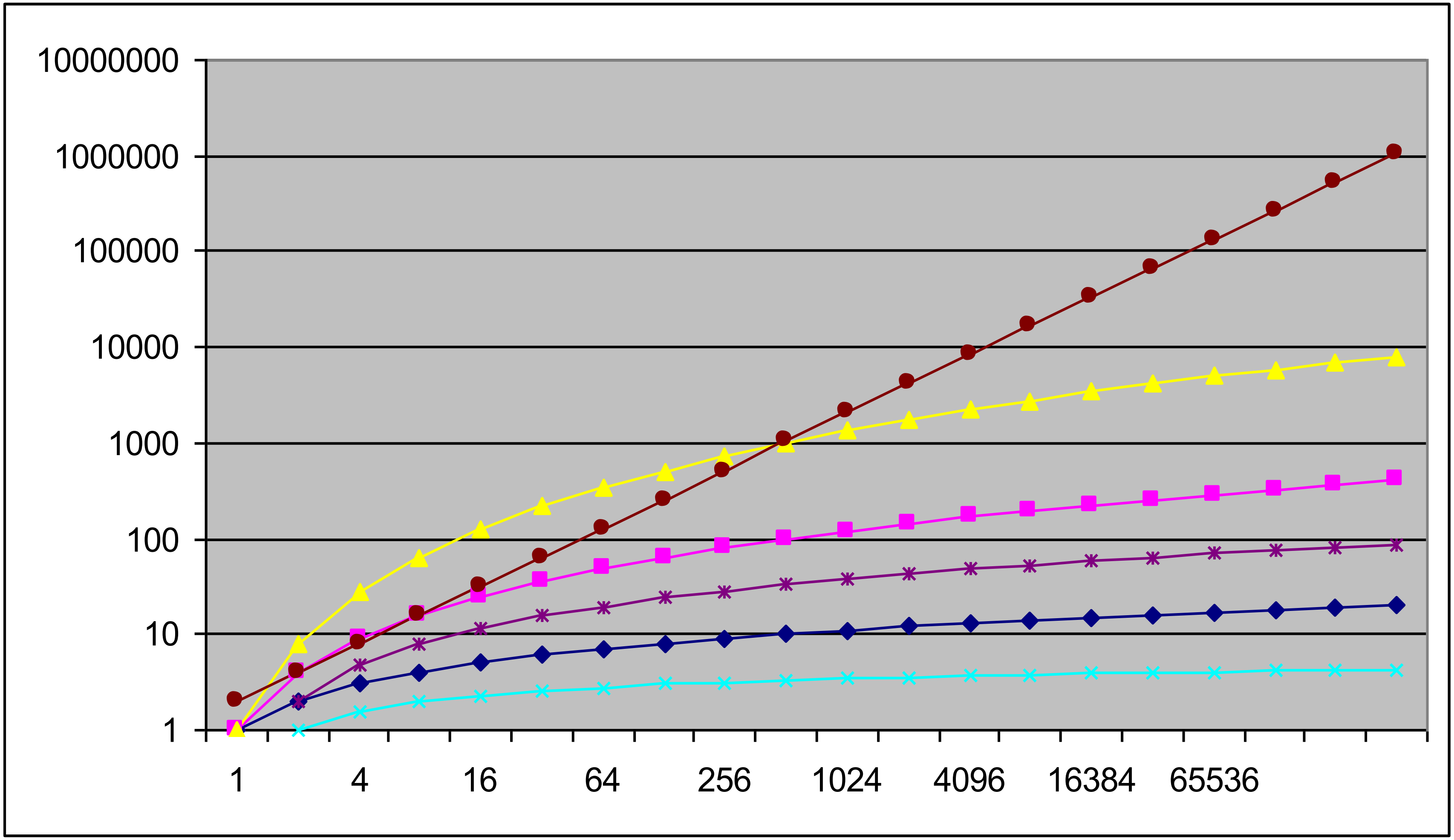
Practical Complexity



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

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

- Intuitively,
  - o() is like <
  - ω() is like >
  - Θ() is like =
  - O() is like ≤
  - Ω() is like ≥

Up Next

- Solving recurrences
    - Substitution method
    - Master theorem