2003

Analyzing Algorithms & Asymptotic Notation

Dan E. Krane
Wright State University - Main Campus, dan.krane@wright.edu

Michael L. Raymer
Wright State University - Main Campus, michael.raymer@wright.edu

Analyzing algorithms

& Asymptotic Notation

Why Sorting Algorithms?

- Simple framework
- Sorting & searching
  - Without sorting, search is random
- Finding information
  - Sequence search
  - Similarity identification
  - Data structures & organization

Sorting algorithms

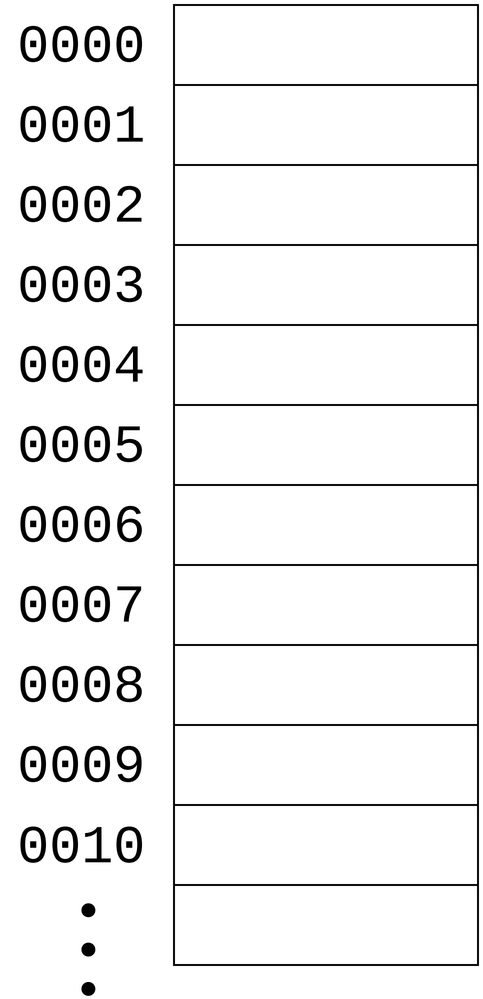
- SelectionSort
  - Looking at a “slot” = 1 operation
  - Moving a value = 1 operation

7	2	9	4	6

  - n items = $(n+1)(n)$ operations
  - n items = $2n$ memory positions

The RAM model of computing

- Linear, random access memory
- READ/WRITE = one operation
- Simple mathematical operations are also unit operations
- Can only read one location at a time, by address
- ***Registers***



“Naïve” Bubble Sort

- Simplifying assumption: compare/swap = 1 operation

$\boxed{7}$ $\boxed{2}$ $\boxed{9}$ $\boxed{4}$ $\boxed{6}$

- Each pass = (n-1) compare/swaps
- n passes = (n)(n-1) compare/swaps
- Space = n

“Smart” Bubble Sort

- MikeSort:

7	2	9	4	6

- First pass (n-1) compare/swaps
- Next pass (n-2) compare/swaps
- n inputs: (n-1) + (n-2) + (n-3) … + 1 $= \sum_{i=1}^{n} (n-i)$
  - We need a mathematical tool to solve this.

Series Sums

- The arithmetic series:
  - $1 + 2 + 3 + \ldots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Linearity: $\sum_{k=1}^{n} (ca_k + b_k) = c\sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$

Series Sums

- $0 + 1 + 2 + \ldots + n - 1 = \sum_{i=1}^{n} i - 1 = \frac{(n-1)n}{2}$
- Example:

$$\sum_{i=1}^{n} 3i + 5 = ?$$

$$\sum_{i=1}^{n} 3i + 5 = 3\left(\frac{n^2 + n}{2}\right) + 5n$$

More Series

- Geometric Series: $1 + x + x^2 + x^3 + \ldots + x^n$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

- Example:

$$\sum_{k=0}^{5} 3^k + 3k + 2$$

$$\sum_{k=0}^{5} 3^k = \frac{3^6-1}{2} = \frac{728}{2} = 364$$

$$\sum_{k=0}^{5} 3k = 3\left(\frac{5\times 6}{2}\right) = 45$$

$$\sum_{k=0}^{5} (2) = 12$$

$$364 + 45 + 12 = 421$$

Telescoping Series

- Consider the series:

$$\sum_{k=1}^{6}\left(\frac{2^k}{k+1}-\frac{2^{k-1}}{k}\right)$$

- Look at the terms:

$$\frac{2}{2}-\frac{1}{1}+\frac{2^2}{3}-\frac{2}{2}+\frac{2^3}{4}-\frac{2^2}{3}+\frac{2^4}{5}-\frac{2^3}{4}+\frac{2^5}{6}-\frac{2^4}{5}+\frac{2^6}{7}-\frac{2^5}{6}$$

$$=\frac{2^6}{7}-\frac{1}{1}$$

Telescoping Series

- In general:

$$\sum_{k=1}^{n}\left(a_k - a_{k-1}\right) = a_n - a_0$$

$$\sum_{k=0}^{n-1}\left(a_k - a_{k+1}\right) = a_0 - a_n$$

The Harmonic Series

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\ldots+\frac{1}{n} \quad = \quad \sum_{i=1}^{n}\frac{1}{n} \quad = \quad O(1)+\ln n$$

Time Complexity of MikeSort

- "Smart" BubbleSort

7	2	9	4	6

- *n* inputs: (n-1) + (n-2) + (n-3) … + 1

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

Exact Analysis of Algorithms

```
for ($i=1; $i<=$n; $i++)
{
     print $i;
}
```

```
for ($i=1; $i<=$n; $i++)
{
     print $i;
     print “Hi there\n”.
}
```

- To make it easy, we’ll ignore loop control structures, and worry about the code *in* the loops.
- Each line of code will be considered one “operation”.

Exact analysis

```
for ($i=1; $i<=$n; $i++) {
  for ($j=1; $j<=$n; $j++) {
     print “$i, $j\n”;
}
```

- \$i = 1
  - \$j =1 , 2, 3, … n
- \$i = 2
  - \$j = 1, 2, 3, … n etc.
- Total: n^2 operations

Exact Analysis of BubbleSort

```
# $i is the pass number
for ($i=0; $i<$n-1; $i++) {
  # $j is the current element looked at
  for ($j=0; $j<$n-1; $j++) {
     if ($array[$j] > $array[$j+1]) {
       swap($array[$j], $array[$j+1]);
     }
  }
}
```

What if the array is often already sorted or nearly sorted??

- Best case: n^2
- Worst case: $2n^2$
- Average case: $1.5(n^2)$

Exact Analysis of MikeSort

```
# $i is the pass number
for ($i=1; $i<=$n-1; $i++) {
  # $j is the current element looked at
  for ($j=1; $j<=$n-$i; $j++) {
     if ($array[$j] > $array[$j+1]) {
       swap($array[$j], $array[$j+1]);
     }
  }
}
```

- Best case: $= (n^2 - n)/2$
- Worst case: $n^2 - n$
- Average case: $1.5((n^2 - \mathrm{n})/2) = (3n^2 - 3n)/2$

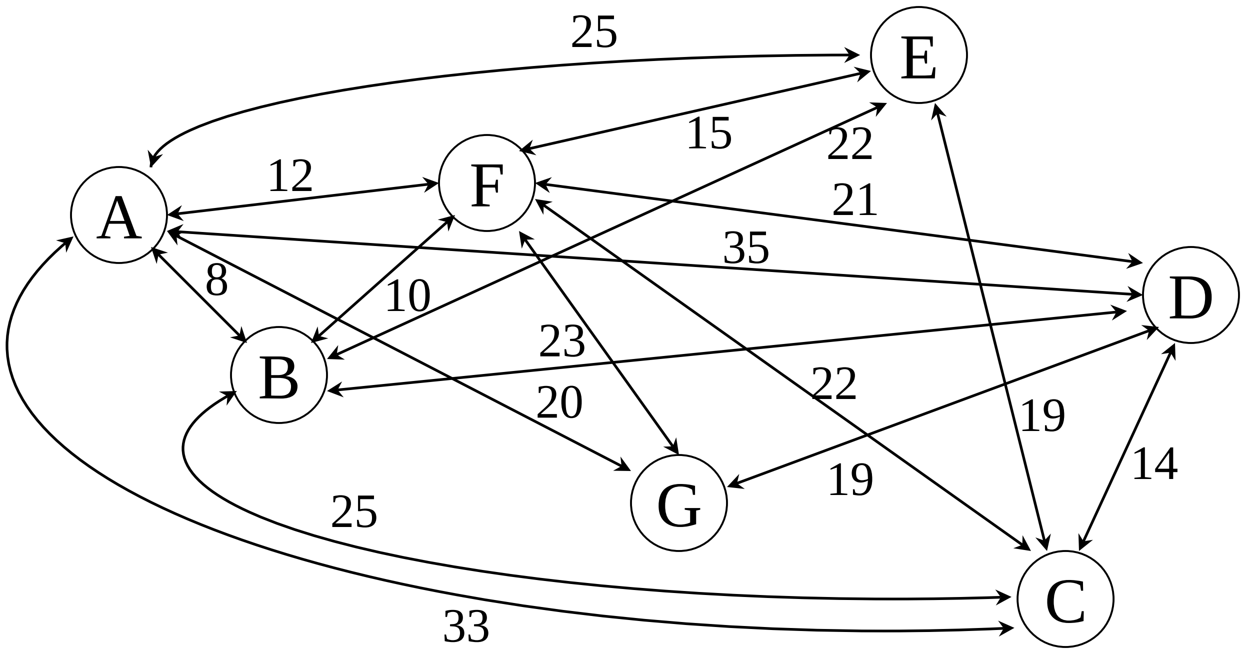
Exact Analysis of MikeSort

- Best case: $= (n^2 - n)/2$
- Worst case: $n^2 - n$
- Average case: $1.5((n^2 - \mathrm{n})/2) = (3n^2 - 3n)/2$

n	Worst	Average	Best
10	90	67.5	45
100	9900	7425	4950
500	249500	187125	124750
1000	999000	749250	499500

Traveling Salesman Problem

- *n* cities
  - Traveling distance between each pair is given
  - Find the circuit that includes all cities



Is there a “real difference”?

- 10^1
- 10^2
- 10^3 Number of students in the college of engineering
- 10^4 Number of students enrolled at Wright State University
- 10^6 Number of people in Dayton
- 10^8 Number of people in Ohio
- 10^10 Number of stars in the galaxy
- 10^20 Total number of all stars in the universe
- 10^80 Total number of particles in the universe
- 10^100 << Number of possible solutions to traveling salesman (100)

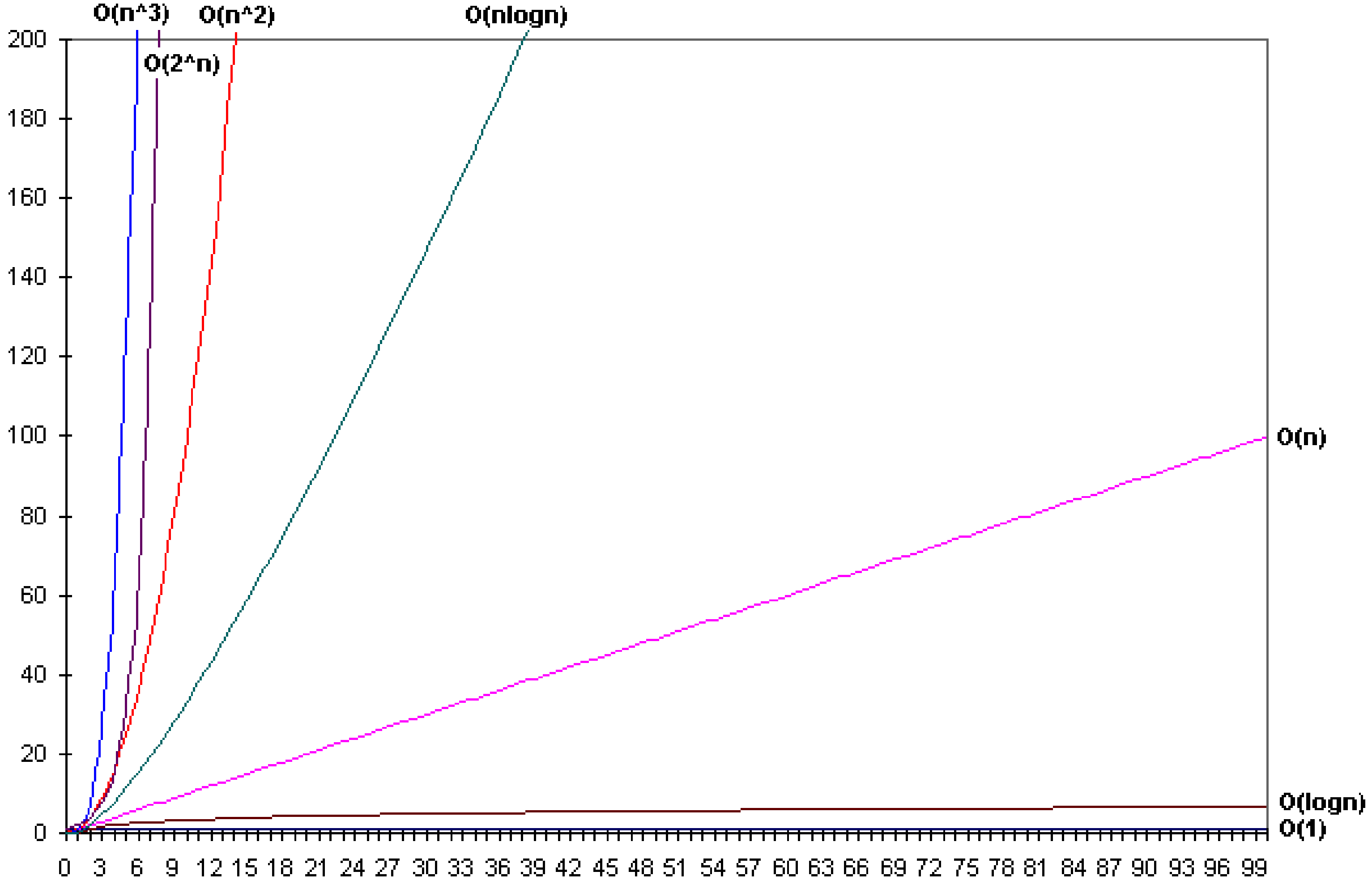
- Traveling salesman (100) is *computable* but it is NOT feasible.

Growth of Functions

n	1	lgn	n	nlgn	n^2	n^3	2^n
1	1	0.00	1	0	1	1	2
10	1	3.32	10	33	100	1,000	1024
100	1	6.64	100	664	10,000	1,000,000	1.2×10^{30}
1000	1	9.97	1000	9970	1,000,000	10^9	1.1×10^{301}

Is there a “real” difference?

- Growth of functions



Introduction to Asymptotic Notation

- We want to express the concept of "about", but in a mathematically rigorous way
- Limits are useful in proofs and performance analyses
- ***Talk about input size: sequence align***
- Θ notation: $\Theta(n^2)$ = "this function grows similarly to n^2".
- Big-O notation: O (n^2) = "this function grows at least as *slowly* as n^2".
  - Describes an *upper bound.*

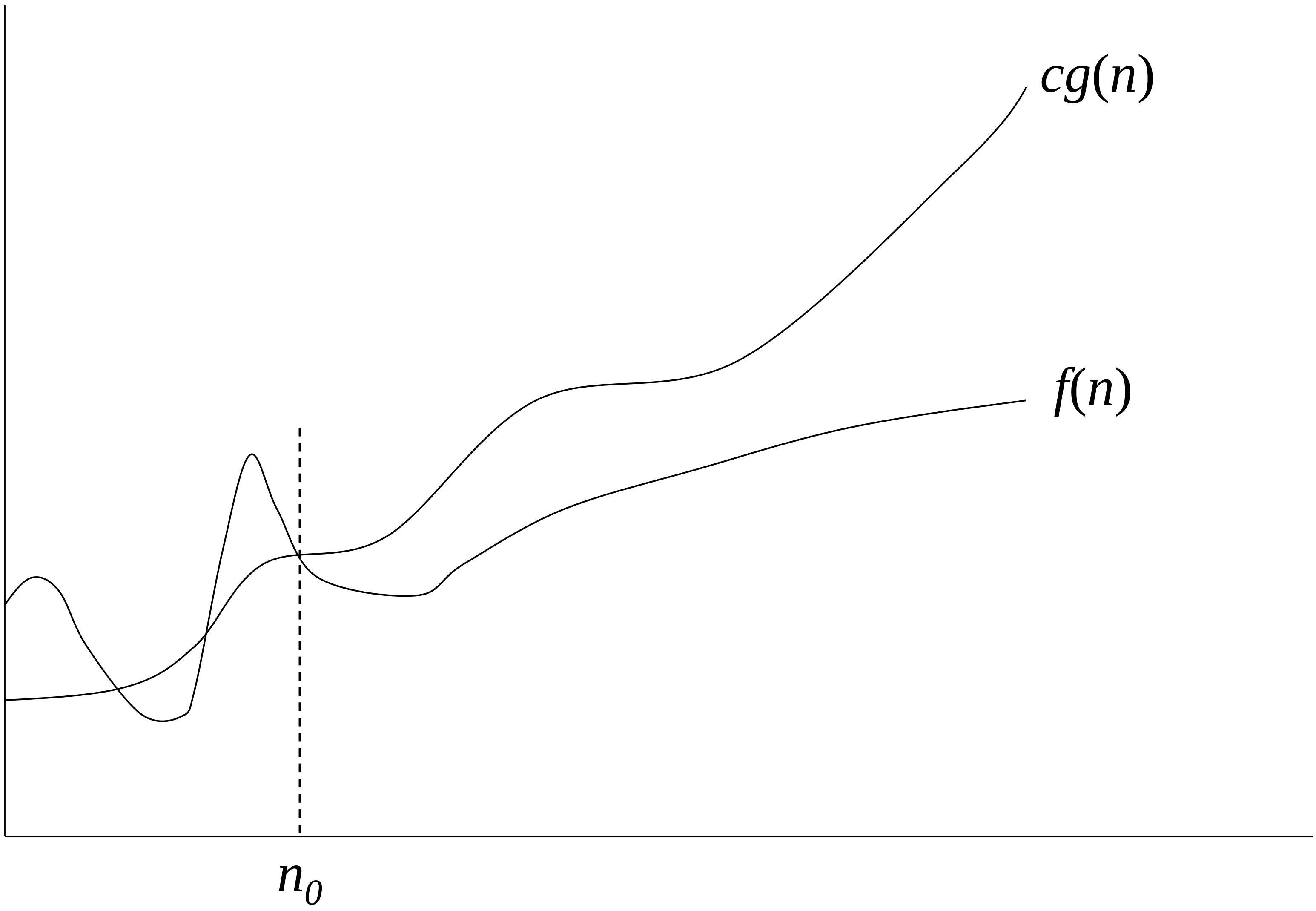
Big-O

$f(n) = O(g(n))$: there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0$$

- What does it mean?
  - If $f(n) = O(n^2)$, then:
    - $f(n)$ can be larger than n^2 sometimes, **but…**
    - I can choose some constant c and some value n_0 such that for **every** value of n larger than n_0 : $f(n) < cn^2$
    - That is, for values larger than n_0, $f(n)$ is never more than a constant multiplier greater than n^2
    - Or, in other words, $f(n)$ does not grow more than a constant factor faster than n^2.

Visualization of $O(g(n))$



Big-O

$$2n^2 = O(n^2)$$

$$1{,}000{,}000n^2 + 150{,}000 = O(n^2)$$

$$5n^2 + 7n + 20 = O(n^2)$$

$$2n^3 + 2 \neq O(n^2)$$

$$n^{2.1} \neq O(n^2)$$

More Big-O

- Prove that: $20n^2 + 2n + 5 = O(n^2)$
- Let c = 21 and n_0 = 4
- $21n^2 > 20n^2 + 2n + 5$ for all $n > 4$

  $n^2 > 2n + 5$ for all $n > 4$

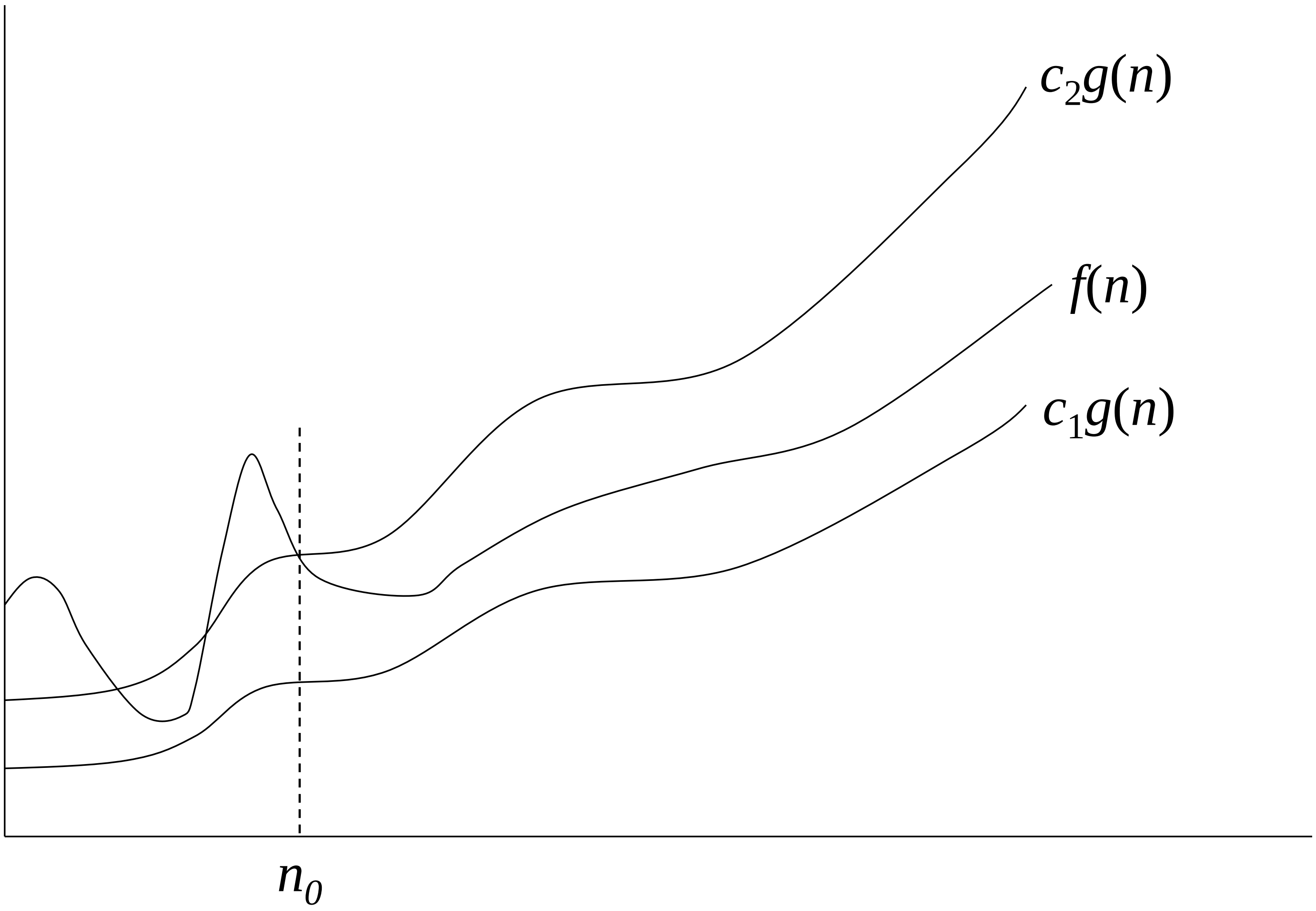
  TRUE

Θ-notation

- Big-O is not a tight upper bound. In other words $n = O(n^2)$
- Θ provides a tight bound

$$f(n) = \Theta(g(n)) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that}$$
$$0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0$$

Visualization of $\Theta(g(n))$



A Few More Examples

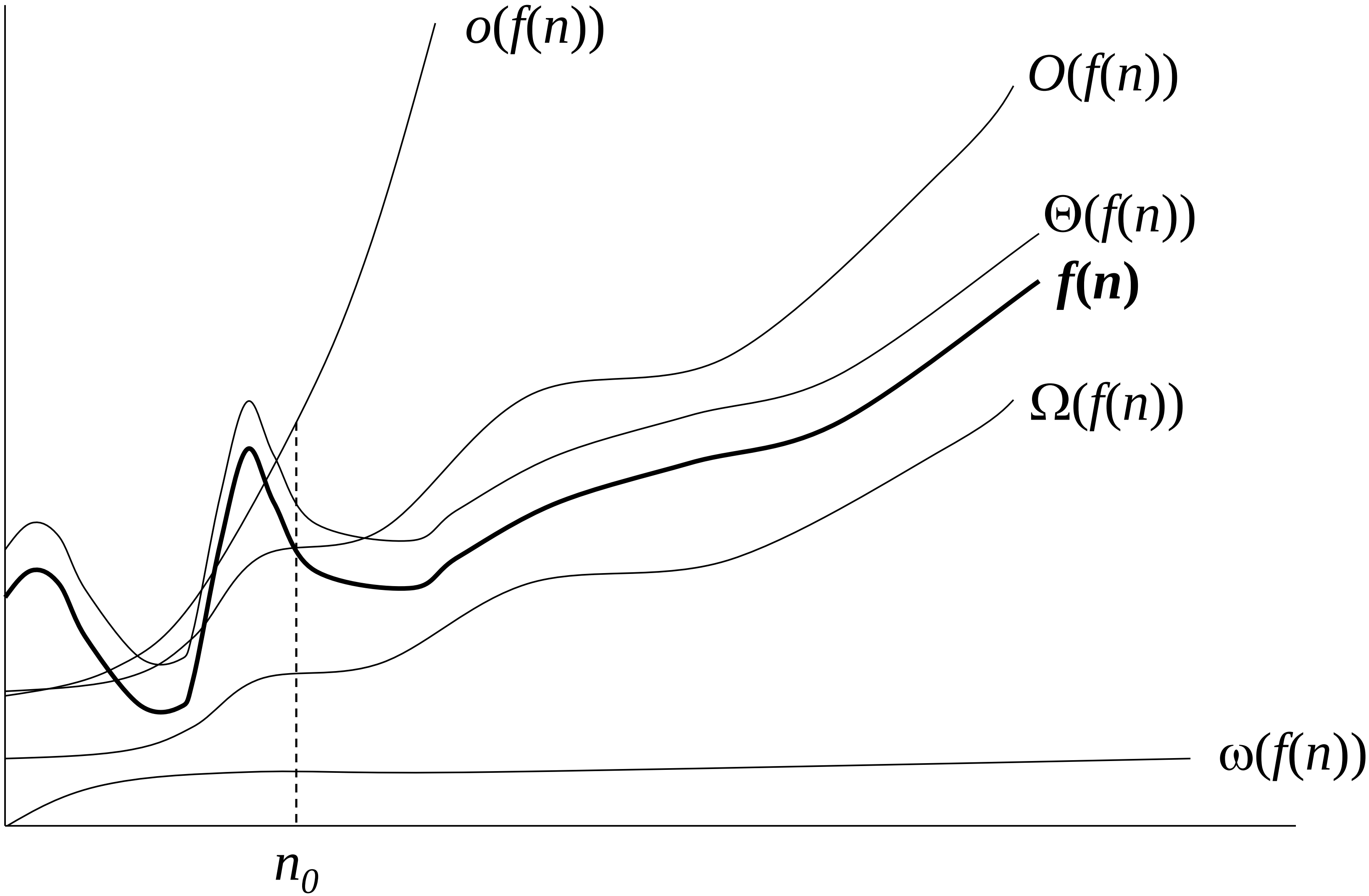
- $n = O(n^2) \neq \Theta(n^2)$
- $200n^2 = O(n^2) = \Theta(n^2)$
- $n^{2.5} \neq O(n^2) \neq \Theta(n^2)$

Some Other Asymptotic Functions

- Little o – A **non-tight** asymptotic upper bound
  - $n = o(n^2)$, $n = O(n^2)$
  - $3n^2 \neq o(n^2)$, $3n^2 = O(n^2)$
- $\Omega()$ – A **lower** bound
  - Similar definition to Big-O
  - $n^2 = \Omega(n)$
- $\omega()$ – A **non-tight** asymptotic lower bound

- $f(n) = \Theta(n) \Leftrightarrow f(n) = O(n)$ **and** $f(n) = \Omega(n)$

Visualization of Asymptotic Growth



Analogy to Arithmetic Operators

$$f(n) = O(g(n)) \qquad \approx \qquad a \leq b$$

$$f(n) = \Omega(g(n)) \qquad \approx \qquad a \geq b$$

$$f(n) = \Theta(g(n)) \qquad \approx \qquad a = b$$

$$f(n) = o(g(n)) \qquad \approx \qquad a < b$$

$$f(n) = \omega(g(n)) \qquad \approx \qquad a > b$$

Approaches to Solving Problems

- Direct/iterative
  - SelectionSort
  - Can by analyzed using series sums
- Divide and Conquer
  - Recursion and Dynamic Programming
  - Cut the problem in half
  - MergeSort

Recursion

- Computing factorials

```
fib(5)
```

```
sub fact($n) {
  if ($n <= 1) {
    return(1);
  }
  else {
    $temp = $fact($n-1);
    $result = $temp + 1;
    return($result);
  }
}

print(fact(4) . "\n");
```

Fibonacci Numbers

```
int fib(int N) {
  int prev, pprev;

  if (N == 1) {
      return 0;
  }
  else if (N == 2) {
      return 1;
  }
  else {
    prev = fib(N-1);
    pprev = fib(N-2);
    return prev + pprev;
  }
}
```

MergeSort

7	2	9	4	6	9	4	6

- Let M_n be the time to MergeSort *n* items
  - $M_n = 2(M_{n-1}) + n$