Asymptotic Notation (Big O)

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Discuss total running time
- Discuss asymptotic running time
- Learn about asymptotic notation

Exercise

- Running time

Extra Resources

- Chapter 3: asymptotic notation

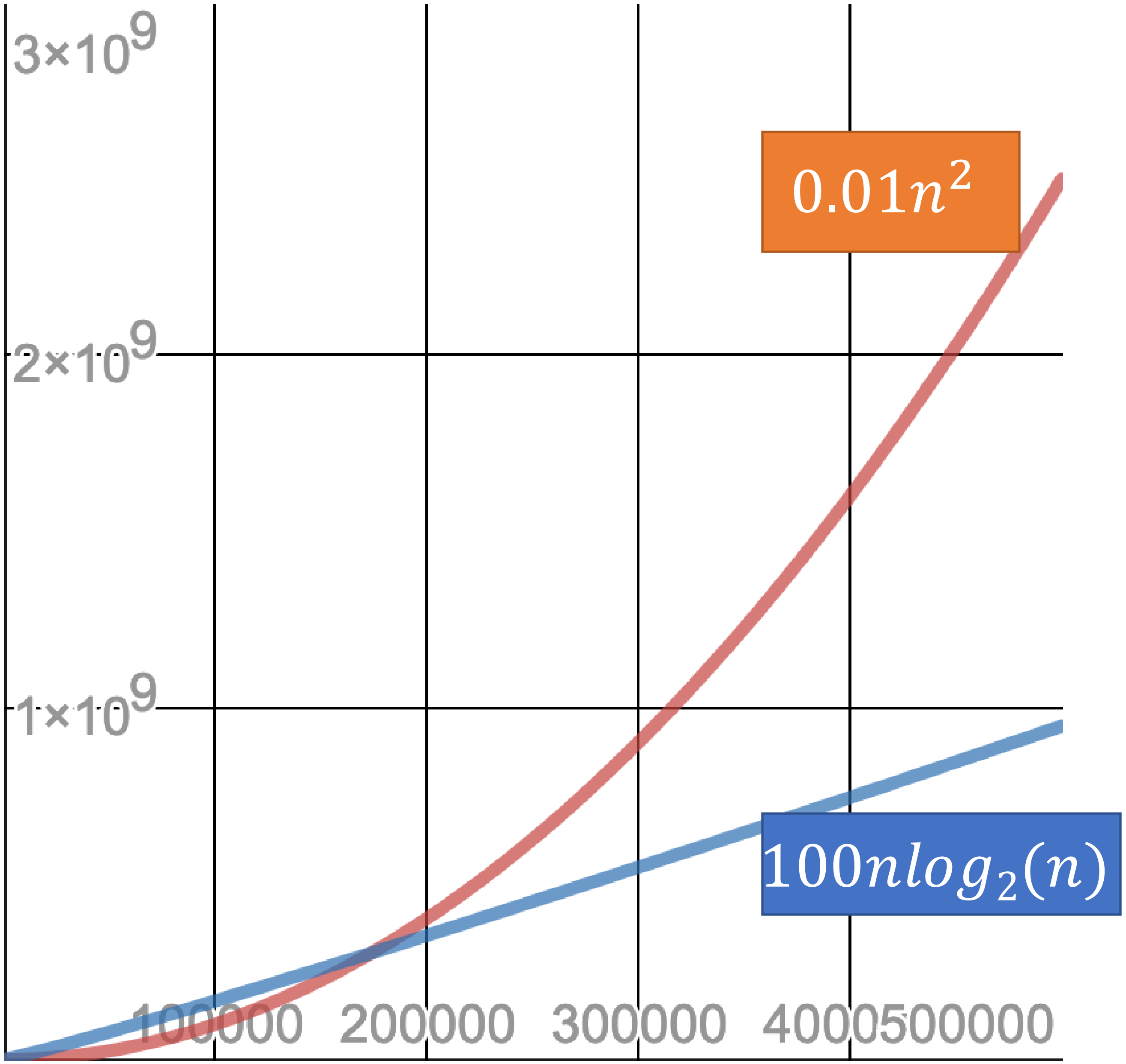
Comparing Algorithms and Data Structures

We like to compare algorithms and data structures
- Speed
- Memory usage

We don’t always need to care about little details

We ignore some details anyway
- Data locality
- Differences among operations

Constants



3×10^9
2×10^9
1×10^9
100000 200000 300000 4000500000
$0.01n^2$
$100nlog_2(n)$

Big-O Example Code (ODS 1.3.3)

```
# function_one has a total running time of 2nlogn + 2n - 250
a = function_one(input_one)

# function_two has a total running time of 3nlogn + 6n + 48
b = function_two(input_two)
```

- The total running time of the code above is:

$$2n \log n + 2n - 250 + 1 + 3n \log n + 6n + 48 + 1$$

$$5n \log n + 8n - 200$$

Big-O Example Math (ODS 1.3.3)

$$5n \log n + 8n - 200$$

- We don't care about most of these details
- We want to be able to quickly glance at the running time of an algorithm and know how it compares to others
- So we say the following

$$5n \log n + 8n - 200 = O(n \log n)$$

Big-O (Asymptotic Running Time)

$$T(n) = O(f(n))$$

If and only if (iff) we can find values for c, $n_0 > 0$, such that

$$T(n) \leq c\ f(n), \text{ where } n \geq n_0$$

Note: c, n_0 cannot depend on n

Searching an array for a given number?

TTYN

Write an algorithm (in pseudocode):

What is the total running time?

$T(n) \leq c\, f(n)$, where $n \geq n_0$

Searching an array for a given number?

What is the asymptotic running time? $T(n) = 2n + 1$

Search two separate arrays (sequentially) for a given number?

Write an algorithm (in pseudocode):

What is the total running time?

$T(n) \leq c\, f(n)$, where $n \geq n_0$

Search two separate arrays (sequentially) for a given number?

What is the asymptotic running time? $T(n) = 4n + 3$

Searching two arrays for any common number?

Write an algorithm (in pseudocode):

What is the total running time?

$T(n) \le c\ f(n)$, where $n \ge n_0$

Searching two arrays for any common number?

What is the asymptotic running time? $T(n) = 2n^2 + 2n + 1$

$T(n) \le c\, f(n)$, where $n \ge n_0$

Searching two arrays for any common number?

TTYN

What is the asymptotic running time? $T(n) = 2n^2 + 2n + 1$

Searching a single array for duplicate numbers?

TTYN

Write an algorithm (in pseudocode):

What is the total running time?

$T(n) \leq c\ f(n)$, where $n \geq n_0$

Searching a single array for duplicate numbers?

TTYN

What is the asymptotic running time? $T(n) = 21n\lg n + 25n + 1$

$T(n) \leq c\, f(n)$, where $n \geq n_0$

Searching a single array for duplicate numbers?

What is the asymptotic running time? $T(n) = 21n\lg n + 25n + 1$

Big-O Examples

$T(n) = O(f(n))$
If and only if we can find values for c, $n_0 > 0$, such that
$T(n) \le c\ f(n)$, where $n \ge n_0$
Note: c, n_0 cannot depend on n

TTYN

- Claim: $2^{n+10} = O(2^n)$

Big-O Examples

$T(n) = O(f(n))$

If and only if we can find values for c, $n_0 > 0$, such that

$T(n) \le c\ f(n)$, where $n \ge n_0$

Note: c, n_0 cannot depend on n

- Claim: $2^{10n} = O(2^n)$

Big-O Examples

$T(n) = O(f(n))$
If and only if we can find values for c, $n_0 > 0$, such that
$T(n) \leq c\ f(n)$, where $n \geq n_0$
Note: c, n_0 cannot depend on n

- Claim: for every k >= 1, n^k is **not** $O(n^{k-1})$

Θ Examples

$T(n) = \Theta(f(n))$

If and only if we can find values for c, $n_0 > 0$, such that

$$c_1\ f(n) \le T(n) \le c_2\ f(n), \text{ where } n \ge n_0$$

Note: c_1, c_2, n_0 <u>cannot</u> depend on n

- Claim: $21n(\log_2(n) + 1) = \Theta(n\log_2 n)$

Other Notations

- Big-O (≤) : $T(n) = O(f(n))$ if $T(n) \le c\, f(n)$, where $n \ge n_0$
- Big-Omega (≥) : $T(n) = \Omega(f(n))$ if $T(n) \ge c\, f(n)$, where $n \ge n_0$

- Theta (=) : $T(n) = \Theta(f(n))$ if $T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$

$$c_1 f(n) \le T(n) \le c_2 f(n), \text{ where } n \ge n_0$$

Other Notations

- Big-O (≤) : $T(n) = O(f(n))$ if $T(n) \leq c\ f(n)$, where $n \geq n_0$
- little-o (<)

- Big-Omega (≥) : $T(n) = \Omega(f(n))$ if $T(n) \geq c\ f(n)$, where $n \geq n_0$
- Little-omega (>)

Θ Examples

$T(n) = \Theta(f(n))$

If and only if we can find values for c, $n_0 > 0$, such that

$$c_1\ f(n) \le T(n) \le c_2\ f(n), \text{ where } n \ge n_0$$

Note: c_1, c_2, n_0 <u>cannot</u> depend on n

- Claim: $21n\ (\log_2(n) + 1) = \Theta(n\log_2 n)$

Θ Examples

$T(n) = \Theta(f(n))$

If and only if we can find values for c, $n_0 > 0$, such that

$$c_1\ f(n) \leq T(n) \leq c_2\ f(n),\ \text{where } n \geq n_0$$

Note: c_1, c_2, n_0 cannot depend on n

- Claim: $21n\ (\log_2(n) + 1) = \Theta(n\log_2 n)$

Θ Examples

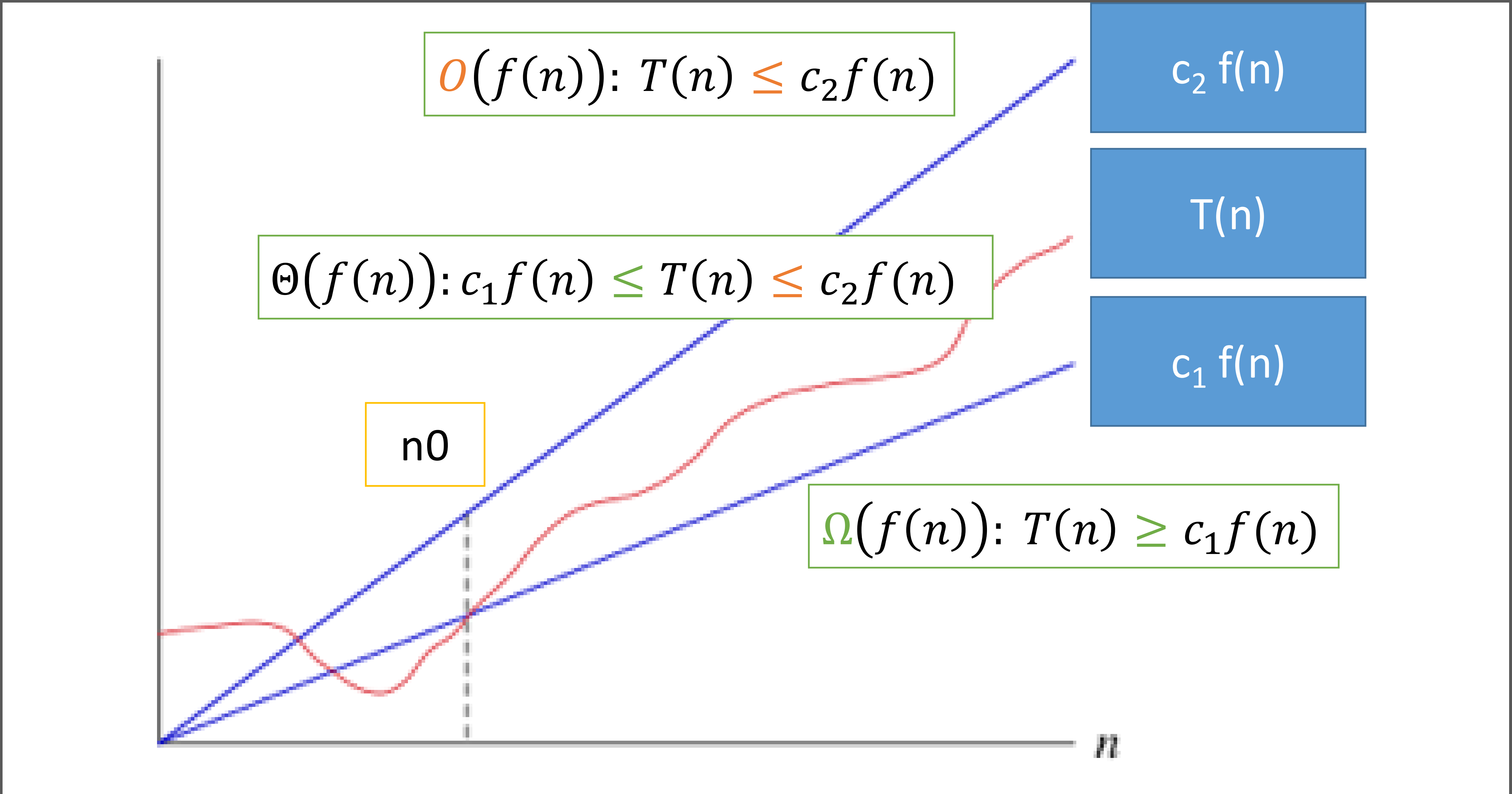
$T(n) = \Theta(f(n))$

If and only if we can find values for c, $n_0 > 0$, such that

$$c_1 f(n) \leq T(n) \leq c_2 f(n), \text{ where } n \geq n_0$$

Note: c_1, c_2, n_0 <u>cannot</u> depend on n

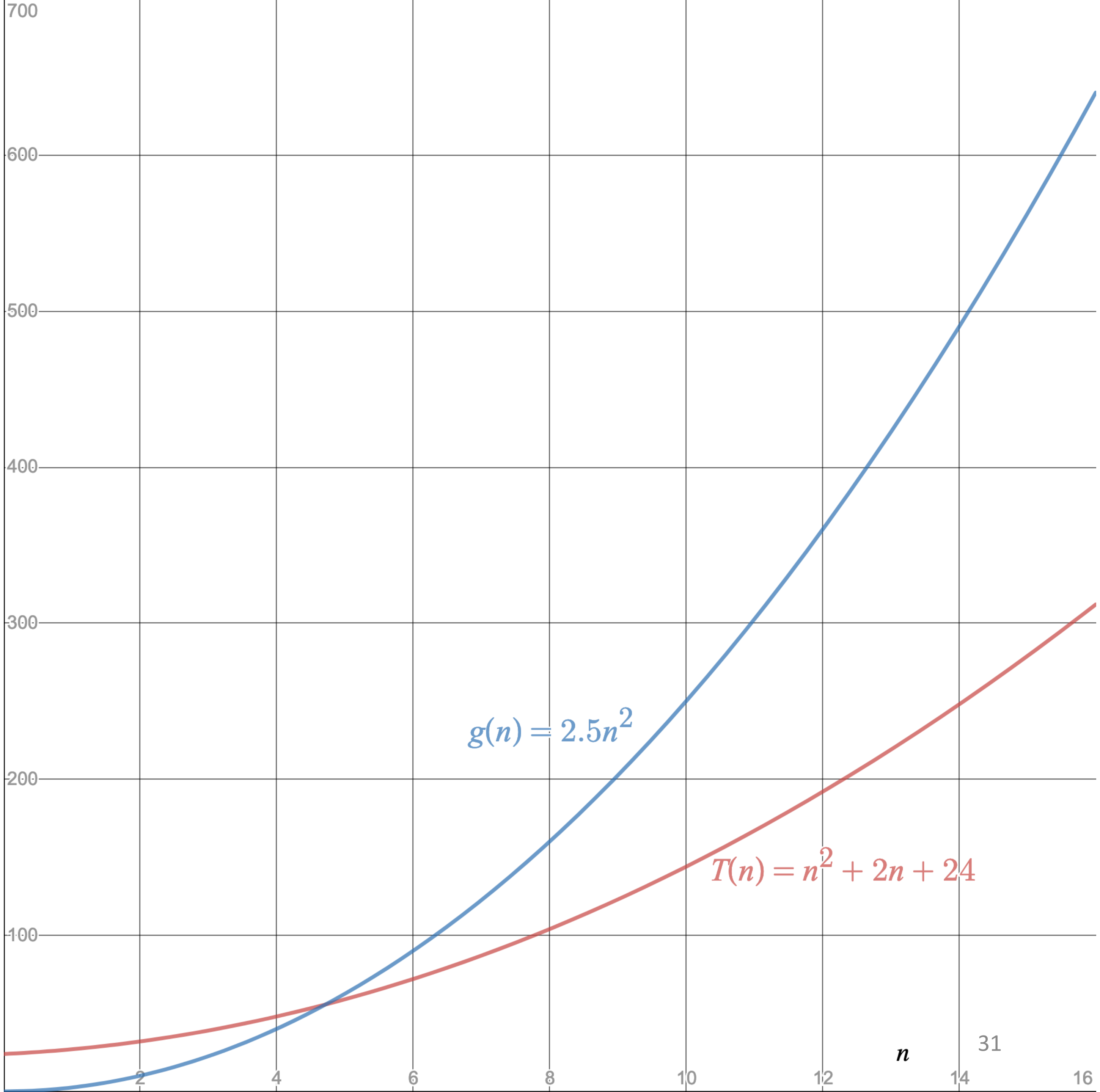
- Claim: $21n(\log_2(n) + 1) = \Theta(n\log_2 n)$


$O(f(n))$: $T(n) \leq c_2 f(n)$
$\Theta(f(n))$: $c_1 f(n) \leq T(n) \leq c_2 f(n)$
$\Omega(f(n))$: $T(n) \geq c_1 f(n)$
n0
c_2 f(n)
T(n)
c_1 f(n)
n

What is f(n)?

What are good values for:

- c
- n_0



Insertion Sort vs Merge Sort

Computer A :
Insertion Sort

10,000 MIPS
$2n^2$

Computer B :
Merge Sort

10 MIPS
$50 \, n \lg n$

5.5 hours	10 million numbers	20 minutes
23 days	100 million numbers	4 hours

Simplifying the Comparison

- Why can we remove leading coefficients?

- Why can we remove lower order terms?

- They are both insignificant when compared with the growth of the function.
- They both get factored into the constant “c”