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

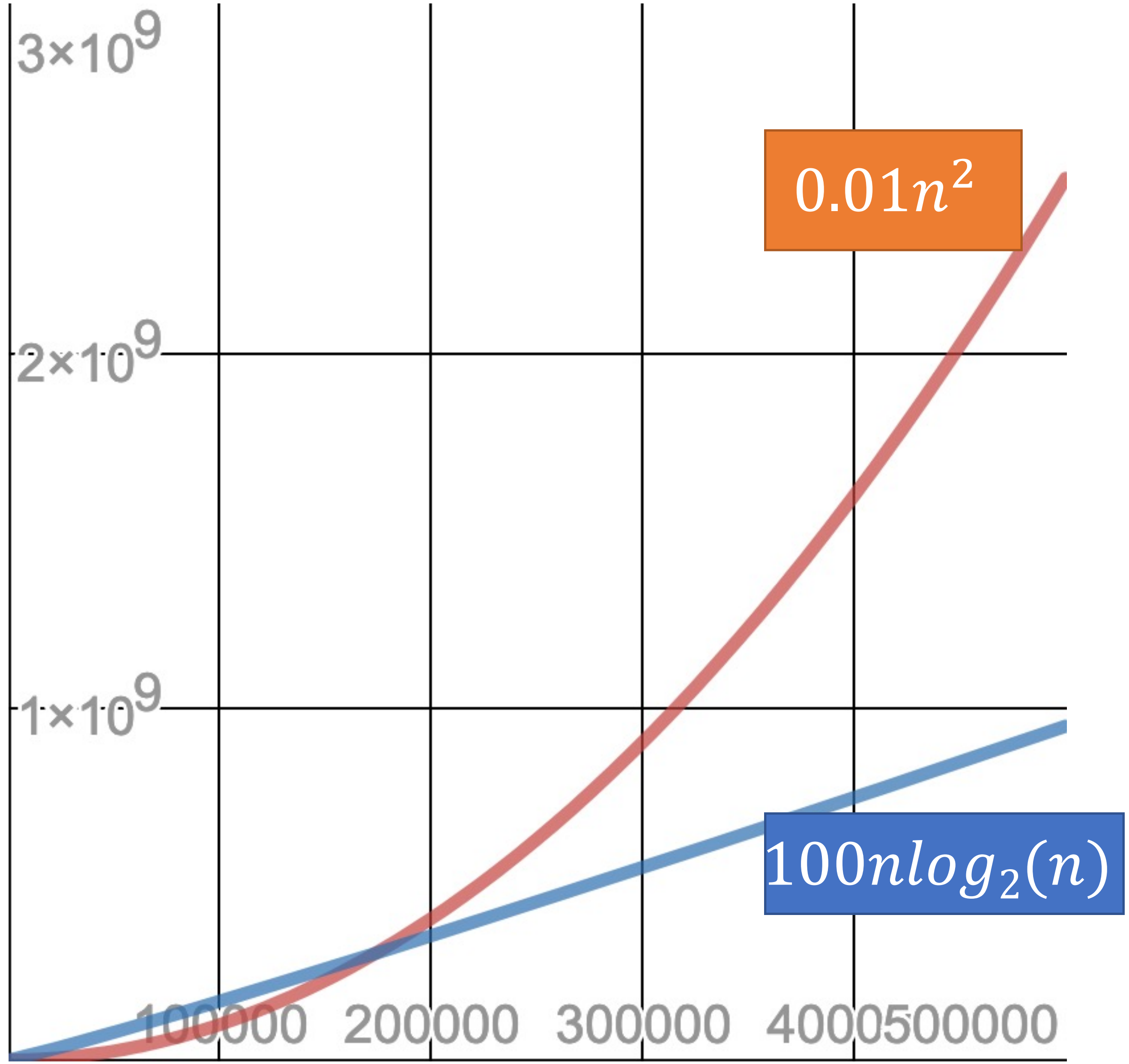
- Speed
- Memory usage

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
400000
500000
$0.01n^2$
$100nlog_2(n)$

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

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

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

TTYN

Searching an array for a given number?

Write an algorithm (in pseudocode):

What is the total running time?

Function FindNum(array, num);

{For val In array

{ If val == num

{Return True

{Return False

What is the total running time?

$T(n) = 2n + 1$

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

Searching an array for a given number?

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

TTYN

$T(n) = O(?)$

$f(n)$

$T(n) = O(n)$

$3n \leq cn$

$\forall\ n \geq n_0$

$T(n) \leq c\,n$ $\forall\ n \geq n_0$

$2n + 1 \leq cn$ $\forall\ n \geq n_0$

$2n + 1 \leq 2n + n$ $\forall\ n \geq 1$

$c = 3$

$n_0 = 1$

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

Write an algorithm (in pseudocode): **What is the total running time?**

TTYN

Function FindNumIn2(array1, array2, num)

return FindNum(array1, num) OR FindNum(array2, num)

$T(n) = 4n+3$

n?

n = max(array1.length, array2.length)

$2n + 1 + 2n + 1 + 1 + \cancel{1}$

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

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

TTYN

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

$T(n) = O(n)$

$4n + 3 \le \boxed{c}\, n \quad \forall\, n \ge \boxed{n_0}$

$4n + 3 \le 4n + 3n \le cn \quad \forall\, n \ge n_0$

1. $4n + 3 \le 4n + 3n$

   $3 \le 3n \rightarrow n \ge 1$

2. $4n + 3n \le cn$

   $7\cancel{n} \le c\cancel{n}$

   $c = 7,\ n_0 = 1$

Naïve

Hash Table → O(n)

Searching two arrays for any common number?

TTYN

Write an algorithm (in pseudocode):

What is the total running time?

Function FindCommon(array1, array2)

n For val1 In array1

n If FindNum(array2, val1) (2n+1)

Return True

Return False

$$T(n) = n + n(2n+1) + 1$$

$$= 2n^2 + 2n + 1$$

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

Searching two arrays for any common number?

TTYN

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

$T(n) \ne O(n)$

$\frac{2n^2}{n} + \frac{2n}{n} + \frac{1}{n} \le \frac{cn}{n} \quad \forall\ \underbrace{n \ge n_0}$

1,000,000

$202 + 1 = 203$

$\underbrace{2n + 2 + \frac{1}{n}} \le c$

$n_0 = ?$

$= 200$

`T(n) ≤ c f(n), where n ≥ n₀`

Searching two arrays for any common number?

TTYN

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

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

$2n^2 + 2n + 1 \le c n^2$

$\underbrace{2n^2 + 2n + 1 \le 2n^2 + 2n^2 + 1n^2}_{①} \le c n^2 \quad \forall n \ge n_0$ ②

$\cancel{2n^2} + 2n + 1 \le \cancel{2n^2} + 2n^2 + n^2$

$\cancel{2}n \le \cancel{2}n^2 \qquad 1 \le n^2$

$1 \le n \qquad n \ge 1$

$2n^2 + 2n^2 + n^2 \le c n^2 \quad \forall n \ge n_0$

$5\cancel{n^2} \le c\cancel{n^2}$

$c = 5, \; n_0 = 1$

Searching a single array for duplicate numbers?

TTYN

Write an algorithm (in pseudocode):

Function FindDuplicate(array)
array = MergeSort(array)
For i In [1 .. < array.length]
If array[i-1] == array[i]
Return True
Return False

What is the total running time?

$21n\lg n + 21n$

$1n$

$3n$

$+1$

$21n\lg n + 21n + 4n + 1$

$21n\lg n + 25n + 1$

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

Searching a single array for duplicate numbers?

TTYN

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

$$T(n) = O(n \lg n)$$

$$\frac{21\,\cancel{n \lg n}}{\cancel{n \lg n}} + \frac{25\cancel{n}}{\cancel{n}\lg n} + \frac{1}{n \lg n} \le \frac{c\,\cancel{n \lg n}}{\cancel{n \lg n}} \quad \forall\, n \ge n_0$$

$$\rightarrow 21 + \frac{25}{\lg n} + \frac{1}{n \lg n} \le c \quad \forall\, n \ge n_0$$

$$n \ge 2^{25}$$

$$\frac{25}{\lg n} \le 1 \rightarrow \frac{25}{\lg 2^{25}} \rightarrow \frac{\cancel{25}}{\cancel{25}\lg 2} \rightarrow \frac{1}{\lg 2} \rightarrow 1 \le 1$$

26 26

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

Searching a single array for duplicate numbers?

TTYN

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

$$21 + 1 + \frac{1}{\underbrace{n \lg n}} \le c \quad \forall\, n \ge n_0$$

$$n_0 \ge 2^{25}$$

$$\frac{1}{\underbrace{n \lg n}_{\ge 1}} \le 1 \qquad 21 + 1 + 1 \le c \quad \forall\, n \ge 2^{25}$$

$$n \ge 2 \checkmark$$

$$c = 22, \underbrace{n_0 = 2^{25}}$$

Exercise

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

$2^{n+10} \le c\, 2^n \quad \forall\, n \ge n_0$

$\cancel{2^n}\, 2^{10} \le c\, \cancel{2^n} \quad \forall\, n \ge n_0$

$c = 2^{10}, \quad n_0 = 1$

Big-O Examples

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

TTYN

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

$$2^{-n} \quad \frac{2^{10n}}{2^n} \le c\,\cancel{2^n} \quad \forall\, n \ge n_0$$

$$2^{10n-n} \le c$$

$$2^{9n} \le \boxed{c} \quad \forall\, n \ge n_0$$

$$2^{10n} \ne O(2^n)$$

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

TTYN

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

$\forall k \ge 1 \quad n^k \ne O(n^{k-1})$

$n^k \le c\, n^{k-1} \quad \forall n \ge n_0$

$\cancel{n^k} \le c\, \cancel{n^k}\, n^{-1} \quad \forall n \ge n_0$

$\underbrace{n \le c}$ $\quad \forall n \ge n_0$

$\therefore$ Claim is true

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 cannot depend on n

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

Other Notations

Upper

- 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 $n \ge n_0$

Lower

- 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 $T(n) \le c\, f(n)$, where $n \ge n_0$
- little-o (<)

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

Θ Examples

$T(n) = \Theta(f(n))$
If and only if we can find values for c, $n_0 > 0$, such that
$c_1\ f(n) \le T(n) \le c_2\ f(n)$, where $n \ge n_0$
Note: c_1, c_2, n_0 <u>cannot</u> depend on n

Big-O upper bound

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

TTYN

$21n\lg n + 21n \le c_2 n\lg n \quad \forall n \ge n_0$

$21\,n\lg n + 21n \le 21n\lg n + 21n\lg n \le c\,n\lg n$

$21n\lg n \le 21\lg n$ ✓

$\cancel{21n} \le \cancel{21n}\lg n$

$1 \le \lg n \quad n \ge 2$

$42\,\cancel{n\lg n} \le c_2\,\cancel{n\lg n}$

$c_2 = 42, \quad n_0 = 2$

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 cannot depend on n

TTYN

Big-O

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

$c_2 = 22$

$n_0 = 2^{21}$

Θ Examples

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

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

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

Note: c_1, c_2, n_0 cannot depend on n

42 2

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

TTYN

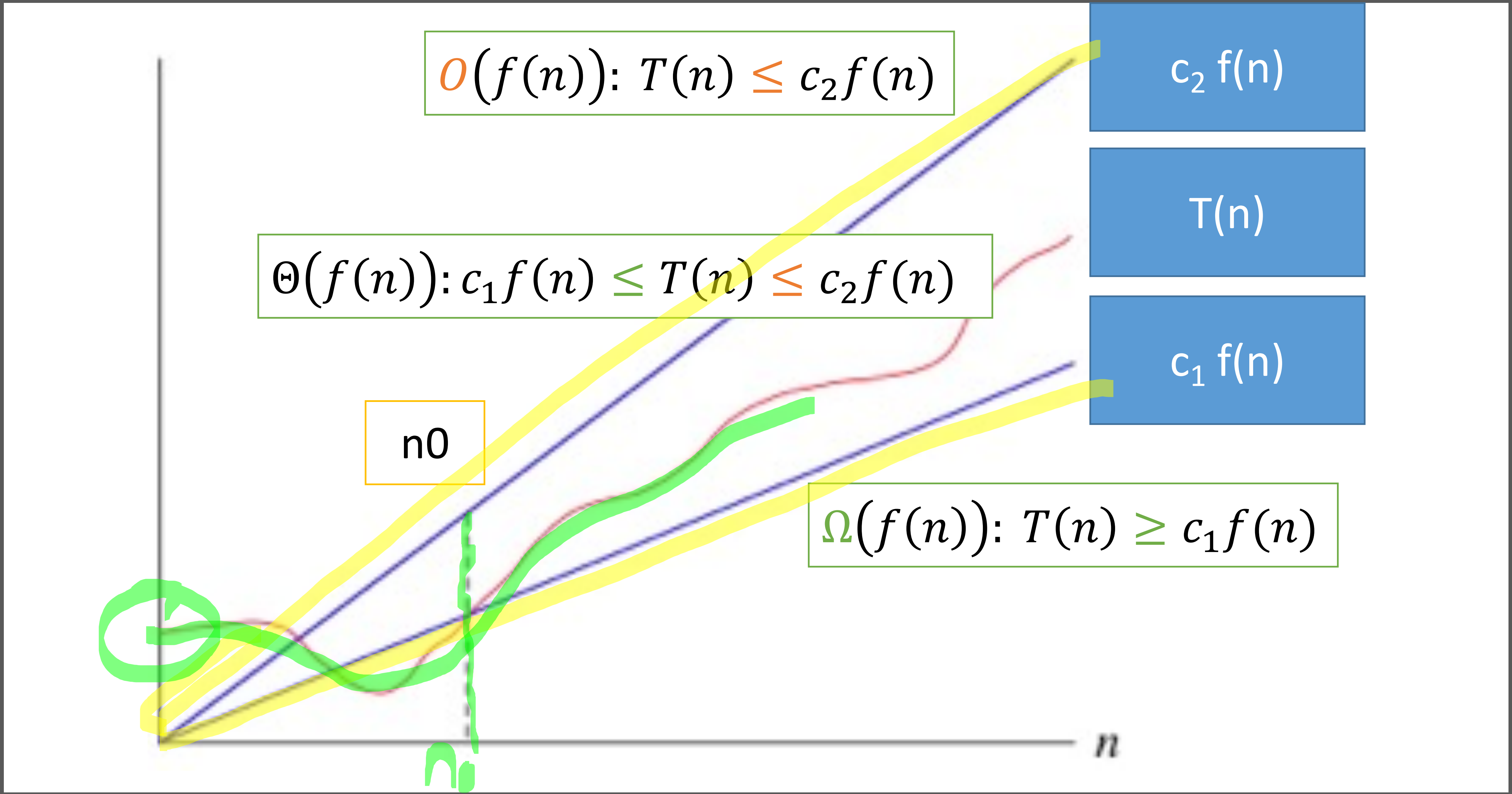
$$c_1 n \lg n \le 21 n \lg n + 21n \quad \forall n \ge n_0$$

$$c_1 n \lg n \le 21 n \lg n \le 21 n \lg n + 21 n$$

$$n \ge 1$$

$$c_1 \cancel{n \lg n} \le 21 \cancel{n \lg n}$$

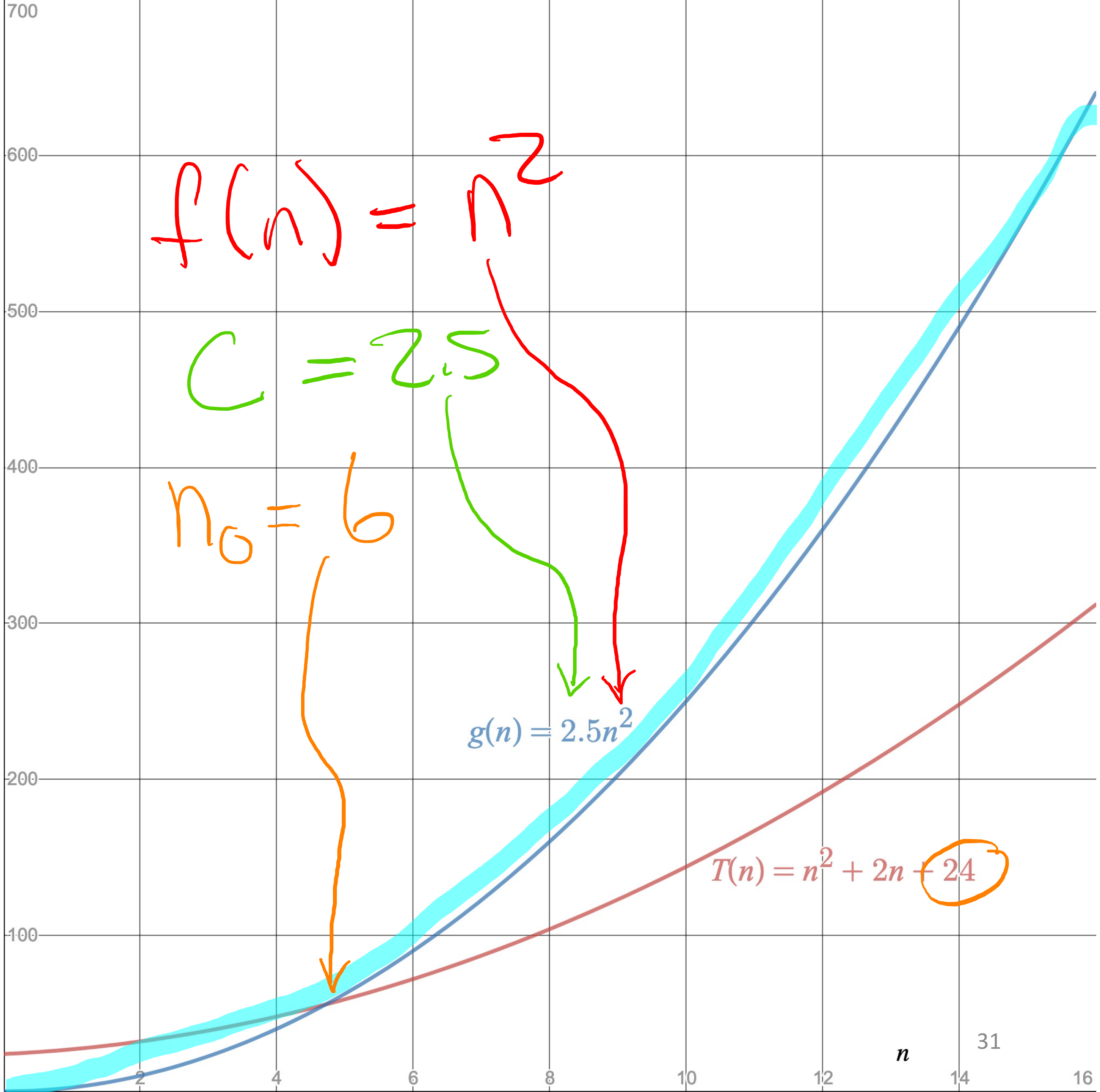
$$c_1 = 21, \quad c_2 = 42, \quad n_0 = 2$$


$O(f(n))$: $T(n) \leq c_2 f(n)$
$\Theta(f(n))$: $c_1 f(n) \leq T(n) \leq c_2 f(n)$
$\Omega(f(n))$: $T(n) \geq c_1 f(n)$
c_2 f(n)
T(n)
c_1 f(n)
n0
n
n_0

What is f(n)?

What are good values for:

- c
- n_0



Insertion Sort vs Merge Sort

Computer A :
Insertion Sort

10,000 MIPS

$2n^2$

$O(n^2)$

$O(n \lg n)$

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?

$21 n \lg n + 21 n$

- 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”