Review of Running Time Analysis

Time Complexity

Given some algorithm A (for example, Bubble Sort):

What is the time complexity of A?

Worst-case Time Complexity

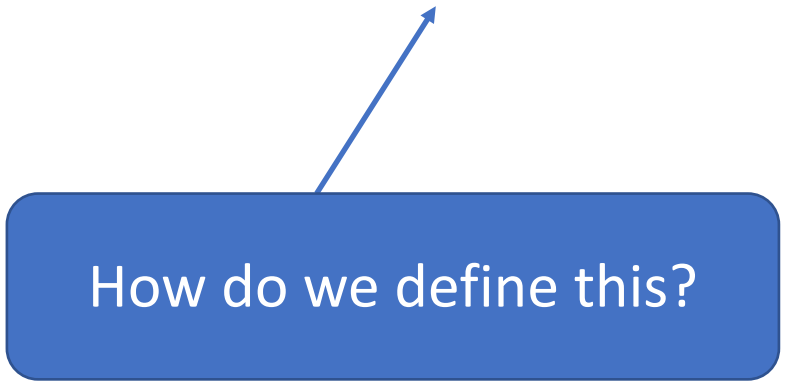
Given some algorithm A (for example, Bubble Sort):

What is the worst-case time complexity of A?

Worst-case Time Complexity

Given some algorithm A (for example, Bubble Sort):

What is the worst-case time complexity of A? ©



Worst-case Time Complexity

Given some algorithm A (for example, Bubble Sort):

What is the worst-case time complexity of A?

Intuitively it is:

maximum number of “steps” algorithm A takes

Worst-case Time Complexity

Given some algorithm A (for example, Bubble Sort):

What is the worst-case time complexity of A?

Intuitively it is:

maximum number of “steps” algorithm A takes on inputs of “size” n

Worst-case Time Complexity

Given some algorithm A (for example, Bubble Sort):

What is the worst-case time complexity of A?

Intuitively it is:

maximum number of "steps" algorithm A takes on inputs of "size" n

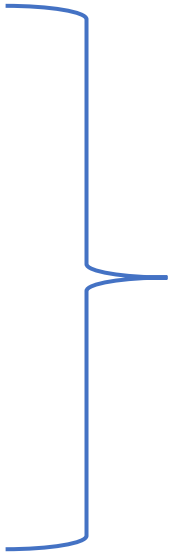
What do we mean by a “step”?

- Arithmetic operations: +, -, *, /, …
- Data Movement operations: load, store, copy, …
- Comparison operations: > , < , ==

.

.

.



Each operation takes a **constant** amount of time.

What do we mean by input “size”?

- It could be
  - Number of elements in the input array

    (e.g. sorting algorithms)

What do we mean by input “size”?

- It could be
  - Number of elements in the input array
    (e.g. sorting algorithms)
  - Number of edges and vertices of the input graph
    (e.g. graph algorithms)

What do we mean by input “size”?

- It could be
  - Number of elements in the input array
    (e.g. sorting algorithms)
  - Number of edges and vertices of the input graph
    (e.g. graph algorithms)
  - Number of bits used to represent the input
    (e.g. algorithms to test if the input number is a prime)

Why Worst-Case Time Complexity?

- We often want "worst-case guarantees" on an algorithm's running time

- Example of a worst-case guarantee:

"For **every** input of size n, algorithm A takes **at most** $7n^3$ steps"

Definition of Worst-Case Time Complexity

Let A be an algorithm (e.g. Bubble Sort)

Definition of Worst-Case Time Complexity

Let A be an algorithm (e.g. Bubble Sort)

Let t(x) = number of steps taken by algorithm A on input x

Definition of Worst-Case Time Complexity

Let A be an algorithm (e.g. Bubble Sort)

Let t(x) = number of steps taken by algorithm A on input x

Worst-Case Time Complexity of A is a function of the input size n:

$$T(n) = \max_{\text{All input x of size n}} t(x)$$

Definition of Worst-Case Time Complexity

Let A be an algorithm (e.g. Bubble Sort)

Let t(x) = number of steps taken by algorithm A on input x

Worst-Case Time Complexity of A is a function of the input size n:

$$T(n) = \max_{\text{All input x of size n}} t(x) = \max \{t(x) \mid x \text{ is an input of size n}\}$$

Definition of Worst-Case Time Complexity

Let A be an algorithm (e.g. Bubble Sort)

Let t(x) = number of steps taken by algorithm A on input x

Worst-Case Time Complexity of A is a function of the input size n:

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

T(n) = maximum number of steps taken by A on inputs of size n

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	
x_2	
.	
.	
x_i	
.	
.	

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	
.	
.	
x_i	
.	
.	

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	$t(x_2) = k_2$
.	
.	
x_i	
.	
.	

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	$t(x_2) = k_2$
.	
.	
x_i	$t(x_i) = k_i$
.	
.	

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	$t(x_2) = k_2$
.	
.	
x_i	$t(x_i) = k_i$
.	
.	

$$T(n) = \max \{t(x_1), t(x_2), \ldots, t(x_i), \ldots\}$$

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	$t(x_2) = k_2$
.	
.	
x_i	$t(x_i) = k_i$
.	
.	

$$T(n) = \max \{t(x_1), t(x_2), \ldots, t(x_i), \ldots\}$$

$$T(n) = \max \{k_1, k_2, \ldots, k_i, \ldots\}$$

$$T(n) = \max_{\text{All input } x \text{ of size } n} t(x) = \max \{t(x) \mid x \text{ is an input of size } n\}$$

Input of size n	# steps A takes
x_1	$t(x_1) = k_1$
x_2	$t(x_2) = k_2$
.	
.	
x_i	$t(x_i) = k_i$
.	
.	

$$T(n) = \max \{t(x_1), t(x_2), \ldots, t(x_i), \ldots\}$$

$$T(n) = \max \{k_1, k_2, \ldots, k_i, \ldots\}$$

T(n) = maximum number of steps algo A takes on inputs of size n

Worst-Case Time Complexity

How do we show the following?

$T(n) <= 7n^3$ (upper bound)

Worst-Case Time Complexity

How do we show the following?

$T(n) <= 7n^3$ (upper bound)

$T(n) >= 3n^2$ (lower bound)

An abstraction

Let S be a set of integers

An abstraction

Let S be a set of integers
Max(S): maximum element of S

An abstraction

Let S be a set of integers
Max(S): maximum element of S

Let c be some constant

An abstraction

Let S be a set of integers
Max(S): maximum element of S

Let c be some constant

Max(S) <= c

An abstraction

Let S be a set of integers
Max(S): maximum element of S

Let c be some constant

$$\text{Max}(S) <= c \quad \Leftrightarrow \quad \forall\, e \in S : e <= c$$

An abstraction

Let S be a set of integers
Max(S): maximum element of S

Let c be some constant

Max(S) <= c ⇔ ∀ e ∈ S : e <= c

Max(S) >= c ⇔

An abstraction

Let S be a set of integers
Max(S): maximum element of S

Let c be some constant

$$Max(S) <= c \quad \Leftrightarrow \quad \forall\, e \in S : e <= c$$

$$Max(S) >= c \quad \Leftrightarrow \quad \exists\, e \in S : e >= c$$

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) <= 7n^3$ (upperbound)

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) <= 7n^3$ (upperbound)

$\max \{t(x) \mid x \text{ is an input of size } n\} <= 7n^3$

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) <= 7n^3$ (upperbound)

$\max \{t(x) \mid x \text{ is an input of size } n\} <= 7n^3$

For **every** input of size n, A takes **at most** $7n^3$ steps.

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) >= 3n^2$ (lowerbound)

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) >= 3n^2$ (lowerbound)

$\max \{t(x) \mid x \text{ is an input of size } n\} >= 3n^2$

Recall that $T(n) = \max \{t(x) \mid x \text{ is an input of size } n\}$

How do we show the following? $T(n) >= 3n^2$ (lowerbound)

$\max \{t(x) \mid x \text{ is an input of size } n\} >= 3n^2$

For **some** input of size n, A takes **at least** $3n^2$ steps.

In Summary,

$T(n) <= 7n^3$ (upperbound) $\Leftrightarrow$ For **every** input of size n, A takes **at most** $7n^3$ steps.

$T(n) >= 3n^2$ (lowerbound) $\Leftrightarrow$ For **some** input of size n, A takes **at least** $3n^2$ steps.

Issue 1: Constant factors

- What if

$$T(n) <= 7n^3$$ **NOT TRUE**

Issue 1: Constant factors

- What if

$T(n) <= 7n^3$ **NOT TRUE**

$T(n) <= 100n^3$ **TRUE**

Issue 1: Constant factors

- What if

$T(n) <= 7n^3$ **NOT TRUE**

$T(n) <= 100n^3$ **TRUE**

- What if

$T(n) >= 3n^2$ **NOT TRUE**

$T(n) >= n^2/10$ **TRUE**

Issue 1: Constant factors

We would like to say something like

$$T(n) \underset{\text{within a constant factor}}{<=} n^3$$

$$T(n) \underset{\text{within a constant factor}}{>=} n^2$$

Issue 2: Quantifying over n

- What if

  For every n, $T(n) <= 7n^3$ **NOT TRUE**

Issue 2: Quantifying over n

- What if

For every n, $T(n) <= 7n^3$ **NOT TRUE**

For *sufficiently large* n, $T(n) <= 7n^3$ **TRUE**

Issue 2: Quantifying over n

- What if

Say, n >= 200

For every n, $T(n) <= 7n^3$ NOT TRUE

For *sufficiently large* n, $T(n) <= 7n^3$ TRUE

Issue 2: Quantifying over n

- What if

Say, n >= 200

For every n, $T(n) <= 7n^3$ **NOT TRUE**

For *sufficiently large* n, $T(n) <= 7n^3$ **TRUE**

- What if

For every n, $T(n) >= 3n^2$ **NOT TRUE**

For *sufficiently large* n, $T(n) >= 3n^2$ **TRUE**

Combining Issues 1 and 2

We would like to say something like

$T(n) <= n^3$

within a constant factor & for sufficiently large n

$T(n) >= n^2$

within a constant factor & for sufficiently large n

Combining Issues 1 and 2

We would like to say something like

$T(n) <= n^3$

within a constant factor & for sufficiently large n

Big O !

$T(n) >= n^2$

within a constant factor & for sufficiently large n

Combining Issues 1 and 2

We would like to say something like

$T(n) <= n^3$

within a constant factor & for sufficiently large n

$T(n)$ is $O(n^3)$

$T(n) >= n^2$

within a constant factor & for sufficiently large n

Combining Issues 1 and 2

We would like to say something like

$T(n) <= n^3$

within a constant factor & for sufficiently large n

$T(n)$ is $O(n^3)$

$T(n) >= n^2$

within a constant factor & for sufficiently large n

$T(n)$ is $\Omega(n^2)$

Big O notation

T(n) is O(g(n)) **Intuitively Means** T(n) <= g(n) within a constant factor & for sufficiently large n

Big O notation

T(n) is O(g(n)) **Intuitively Means** T(n) <= g(n) within a constant factor & for sufficiently large n

Formally,

T(n) is O(g(n)) $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
$T(n) <= c\ .\ g\ (n)$

Big O notation

T(n) is O(g(n)) **Intuitively Means** T(n) <= g(n) within a constant factor & for sufficiently large n

Formally,

T(n) is O(g(n)) $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
T(n) <= c . g (n)

$\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
For **every** input of size n,
the algorithm takes
at most c. g(n) steps

Big O notation

T(n) is $\Omega(g(n))$ **Intuitively Means** T(n) >= g(n) within a constant factor & for sufficiently large n

Big O notation

T(n) is $\Omega(g(n))$ **Intuitively Means** T(n) >= g(n) within a constant factor & for sufficiently large n

Formally,

T(n) is $\Omega(g(n))$ $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
$T(n) >= c\ .\ g\ (n)$

Big O notation

$T(n)$ is $\Omega(g(n))$ **Intuitively Means** $T(n)$ >= $g(n)$ within a constant factor & for sufficiently large n

Formally,

$T(n)$ is $\Omega(g(n))$ $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
$T(n) >= c\ .\ g\ (n)$

$\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
For **some** input of size n,
the algorithm takes
at least c. g(n) steps

In Summary,

T(n) is O(g(n)) (upperbound) $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
For **every** input of size n, A takes **at most** c.g(n) steps.

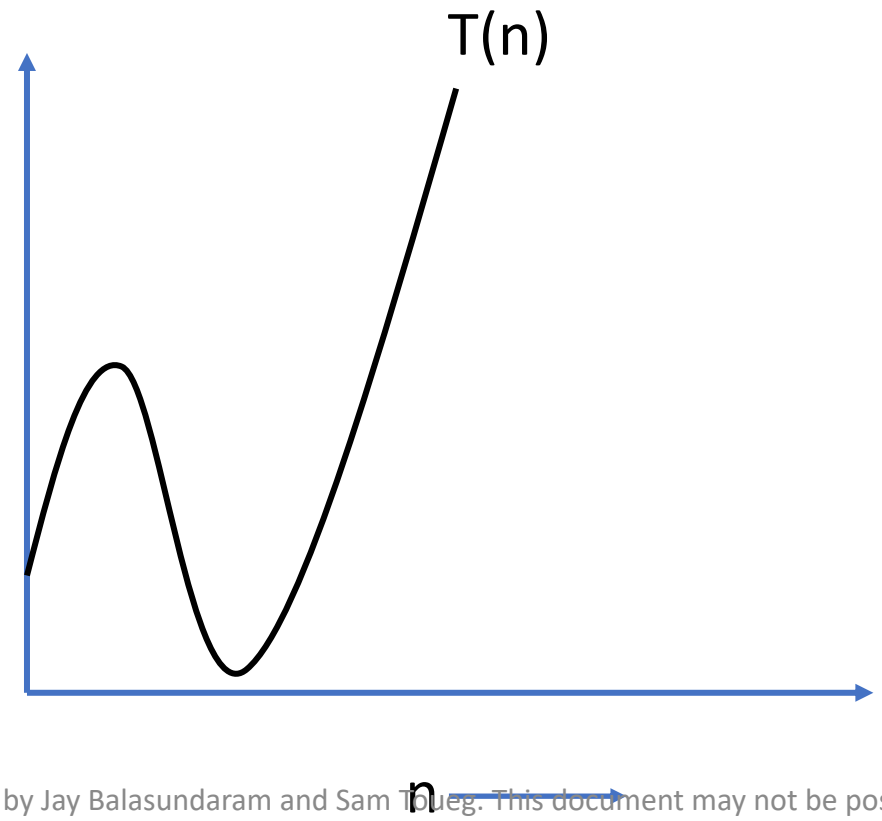
T(n) is Ω(g(n)) (lowerbound) $\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
For **some** input of size n, A takes **at least** c.g(n) steps.

Big O notation

$T(n)$ is $\Theta(g(n))$ $\Leftrightarrow$ $T(n)$ is $O(g(n))$ **and** $T(n)$ is $\Omega(g(n))$

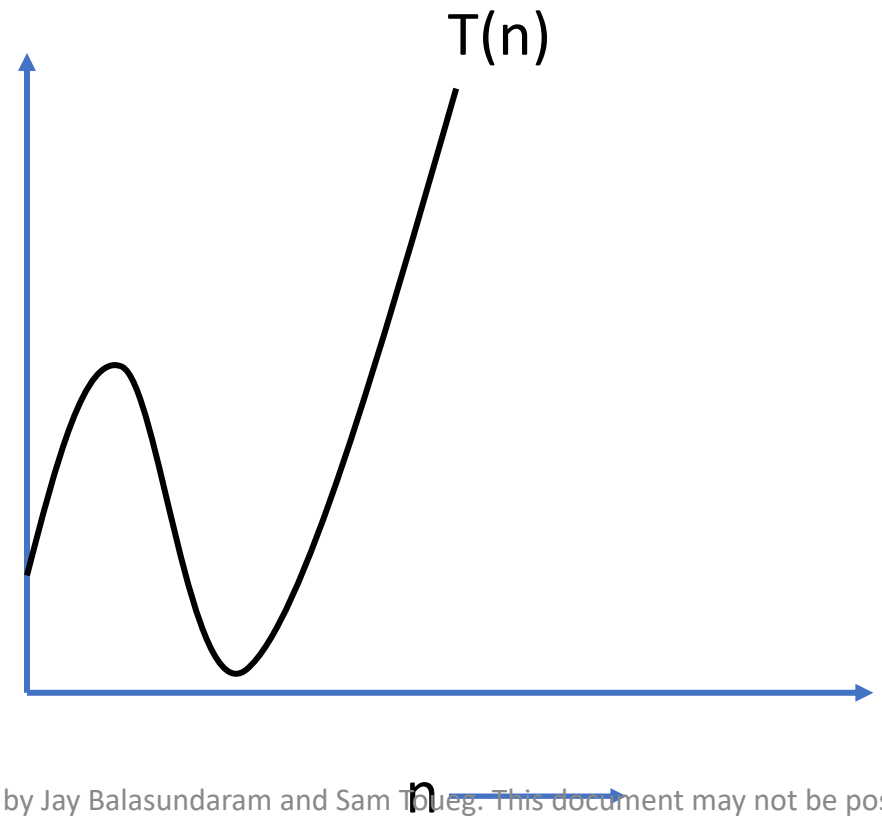
Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$



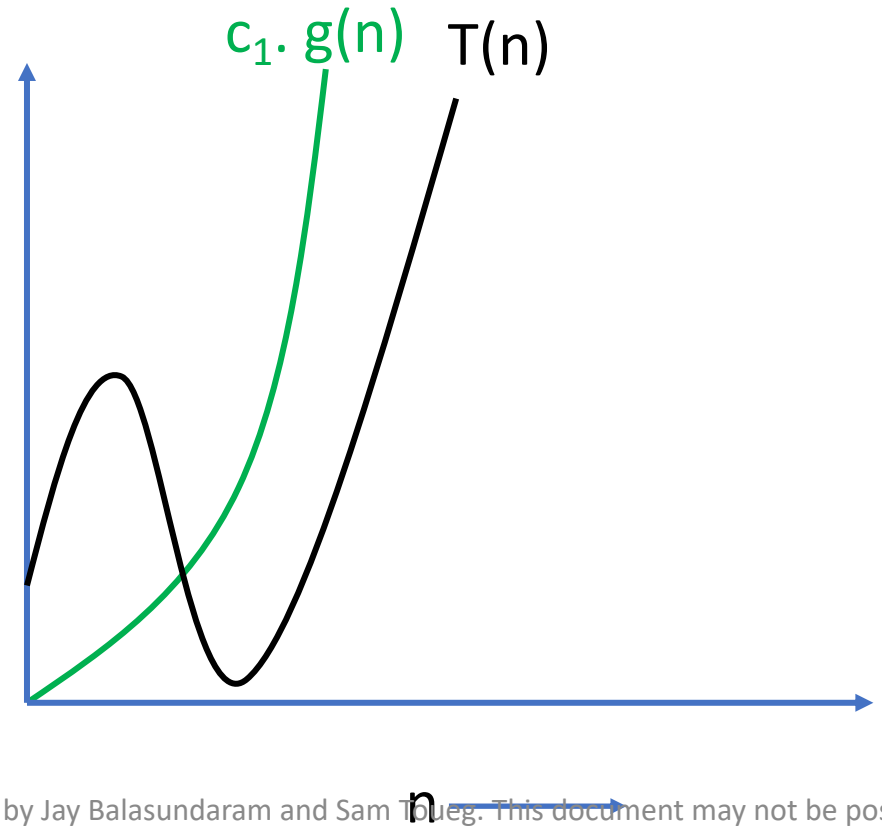
Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$



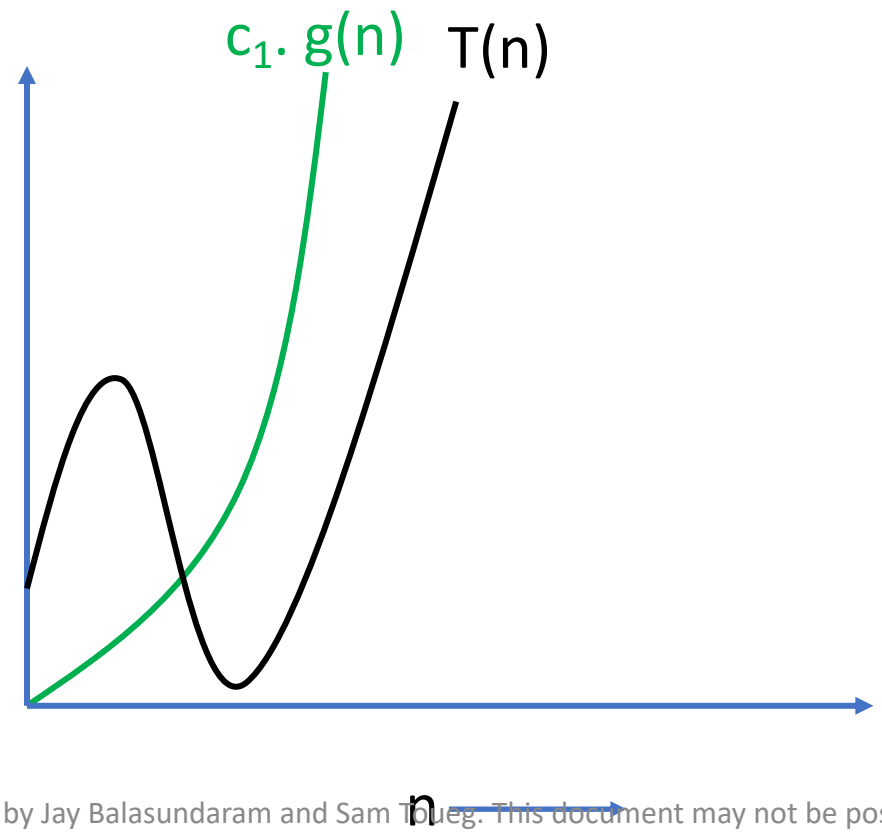
Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$



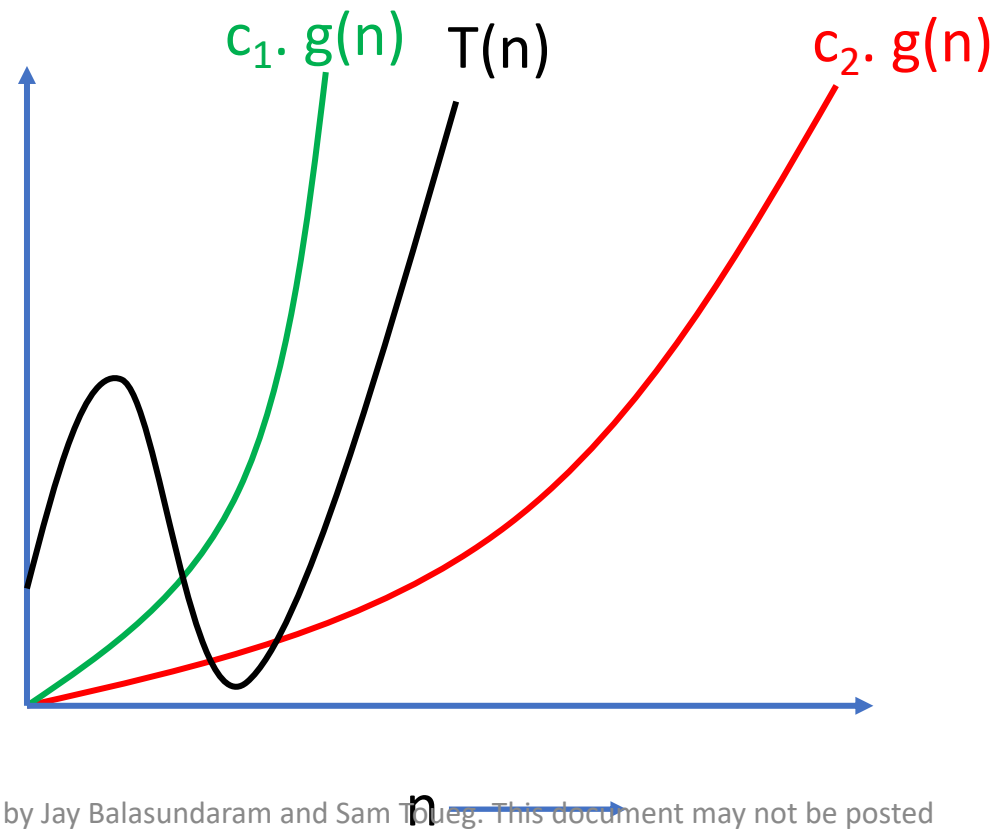
Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$



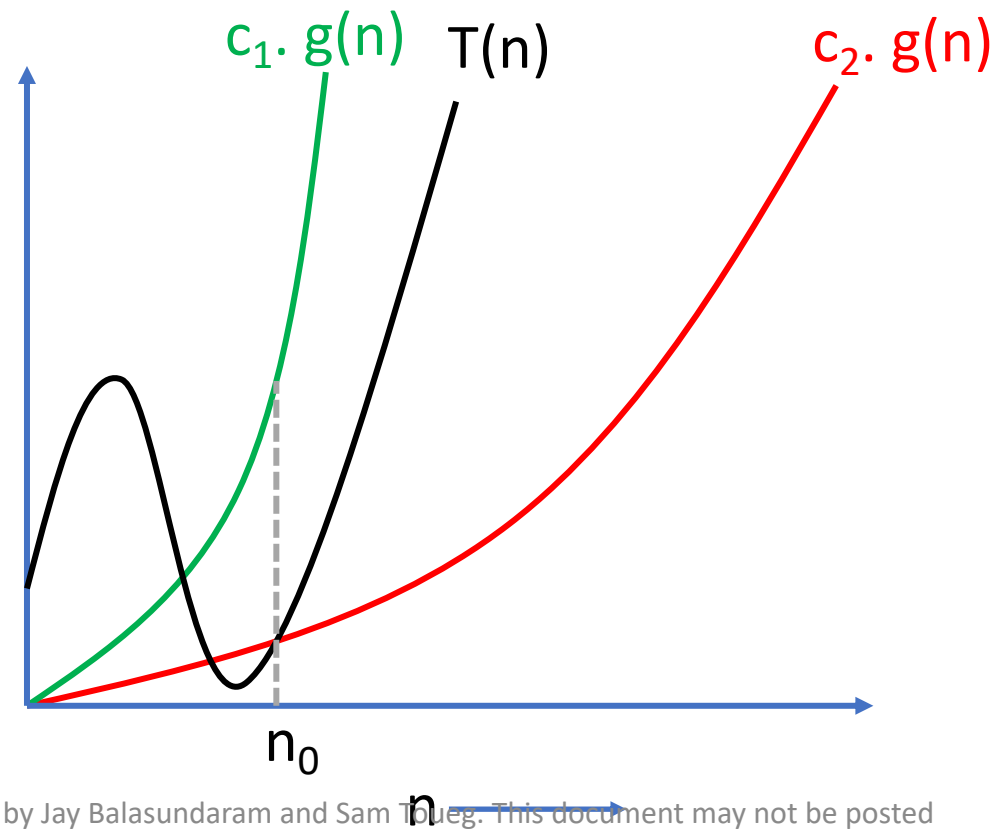
Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$



Big O notation

T(n) is $\Theta(g(n))$ $\Leftrightarrow$ T(n) is $O(g(n))$ **and** T(n) is $\Omega(g(n))$

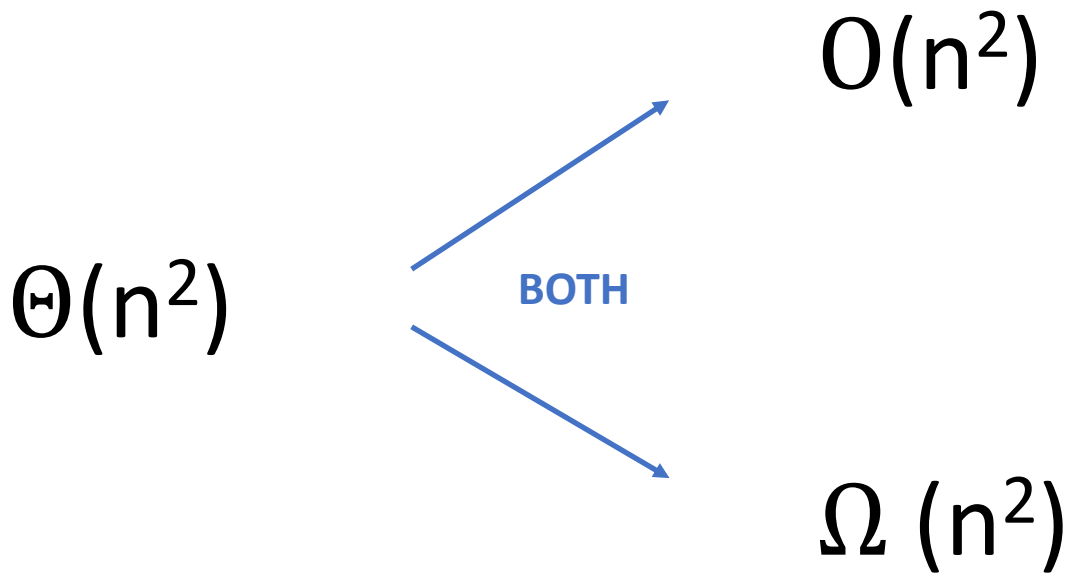


What is the **Worst-Case** Time Complexity of Bubble Sort?

What is the Worst-Case Time Complexity of Bubble Sort?

$\Theta(n^2)$

What is the Worst-Case Time Complexity of Bubble Sort?



What is the Worst-Case Time Complexity of Bubble Sort?

$\Theta(n^2)$

BOTH

$O(n^2)$

$\Omega(n^2)$

(ROUGHLY STATED)
For **every** input of size n,
the algorithm takes
at most $c_1 . n^2$ steps.

What is the Worst-Case Time Complexity of Bubble Sort?

$\Theta(n^2)$

BOTH

$O(n^2)$

(ROUGHLY STATED)
For **every** input of size n,
the algorithm takes
at most $c_1 . n^2$ steps.

$\Omega(n^2)$

(ROUGHLY STATED)
For **some** input of size n,
the algorithm takes
at least $c_2 . n^2$ steps.

What is the Worst-Case Time Complexity of Bubble Sort?

There exists $c_1 > 0$, $c_2 > 0$, such that for sufficiently large n

$\Theta(n^2)$

BOTH

$O(n^2)$

For **every** input of size n, the algorithm takes **at most** $c_1 . n^2$ steps.

$\Omega(n^2)$

For **some** input of size n, the algorithm takes **at least** $c_2 . n^2$ steps.