Review of Running Time Analysis

What is the Time Complexity of Bubble Sort?

How do we define this?

What is the Time Complexity of Bubble Sort?

Informal Definition of **Worst-Case** Time Complexity

The maximum number of “steps” an algorithm 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 a "step"?

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

.

.

.

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

Strictly speaking, we have to fix a model of computation–
One processor RAM model (Refer CLRS section 2.2)

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?

- Often, we would like to have “worst-case guarantees” on the time complexity of the algorithms we design.

- 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)

t(x) = Number of steps taken by A on input x

Worst-Case Time Complexity of A is a function T : ℕ -> ℕ of input size n

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

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

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

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

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

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

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

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

Worst-Case Time Complexity

- Rephrasing Worst-Case guarantee using our definition of T(n)

  “For every input of size n, Algorithm A takes at most $7n^3$ steps”

Worst-Case Time Complexity

- Rephrasing Worst-Case guarantee using our definition of T(n)

“$T(n) <= 7n^3$”

Worst-Case Time Complexity

- Rephrasing Worst-Case guarantee using our definition of T(n)

“$T(n) <= 7n^3$” (Upper Bound)

Worst-Case Time Complexity

- Rephrasing Worst-Case guarantee using our definition of T(n)

  "$T(n) <= 7n^3$" (Upper Bound)

- We would like to have both upper and lower bounds

Worst-Case Time Complexity

- Rephrasing Worst-Case guarantee using our definition of T(n)

“$T(n) <= 7n^3$” (Upper Bound)

- We would like to have both upper and lower bounds:

“$T(n) <= 7n^3$” (upper bound)
AND
“$T(n) >= 3n^2$” (lower bound)

Worst-Case Time Complexity

How do we show the following?

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

AND

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

An abstraction

Let S be a set of integers

Max(S): maximum element of S

Let c be some constant

How would you prove the following?

Max(S) <= c

An abstraction

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

Let c be some constant

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

An abstraction

Let S be a set of integers

Max(S): maximum element of S

Let c be some constant

How would you prove the following?

Max(S) >= c

An abstraction

Let S be a set of integers

Max(S): maximum element of S

Let c be some constant

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

An abstraction

Let S be a set of integers

Max(S): maximum element of S

Let c be some constant

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

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)

$\Updownarrow$

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

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)

$\Updownarrow$

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

$\Updownarrow$

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)

$$\Updownarrow$$

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

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)

$\Updownarrow$

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

$\Updownarrow$

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

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

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

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, $T(n) \le 7n^3$ **NOT TRUE**

For *sufficiently large* n, $T(n) \le 7n^3$ **TRUE**

- What if

For every n, $T(n) \ge 3n^2$ **NOT TRUE**

For *sufficiently large* n, $T(n) \ge 3n^2$ **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

Big Ω !

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,

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

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 Ω(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** $T(n)$ >= within a constant factor & for sufficiently large n $g(n)$

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 $\forall\ n >= n_0$:
$T(n) >= c \cdot g(n)$

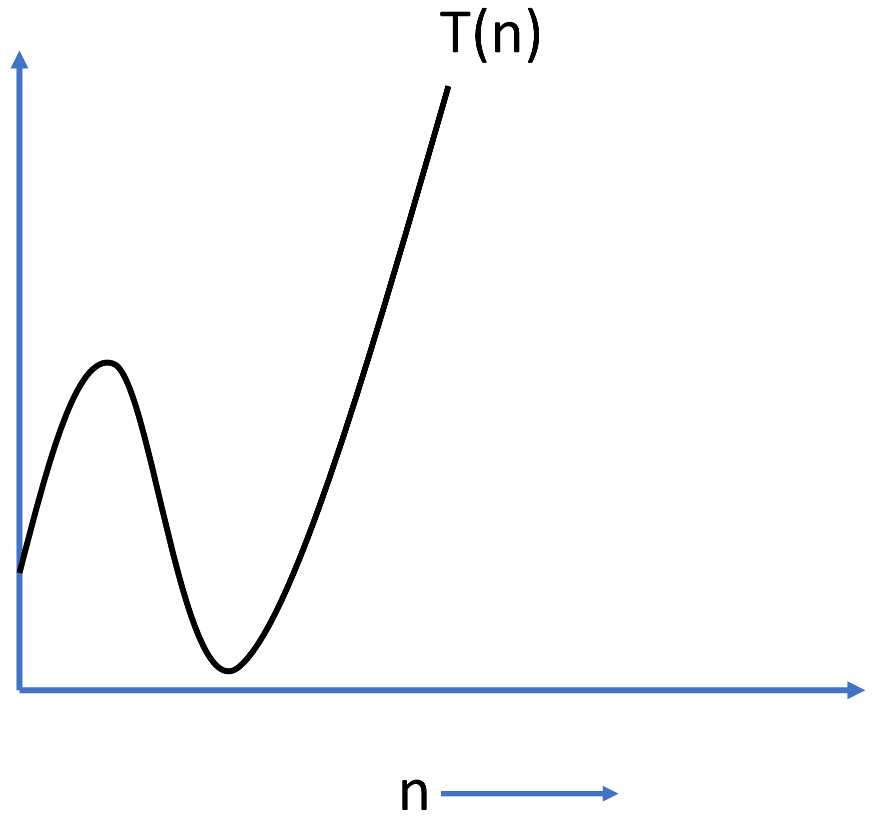
$\Leftrightarrow$ $\exists\ c > 0, \exists\ n_0 > 0,$ such that $\forall\ n >= n_0$:
For **some** input of size n,
the algorithm takes
at least c. g(n) steps

Big O notation

T(n) is Θ(g(n)) $\Leftrightarrow$ T(n) is O(g(n)) AND T(n) is Ω(g(n))

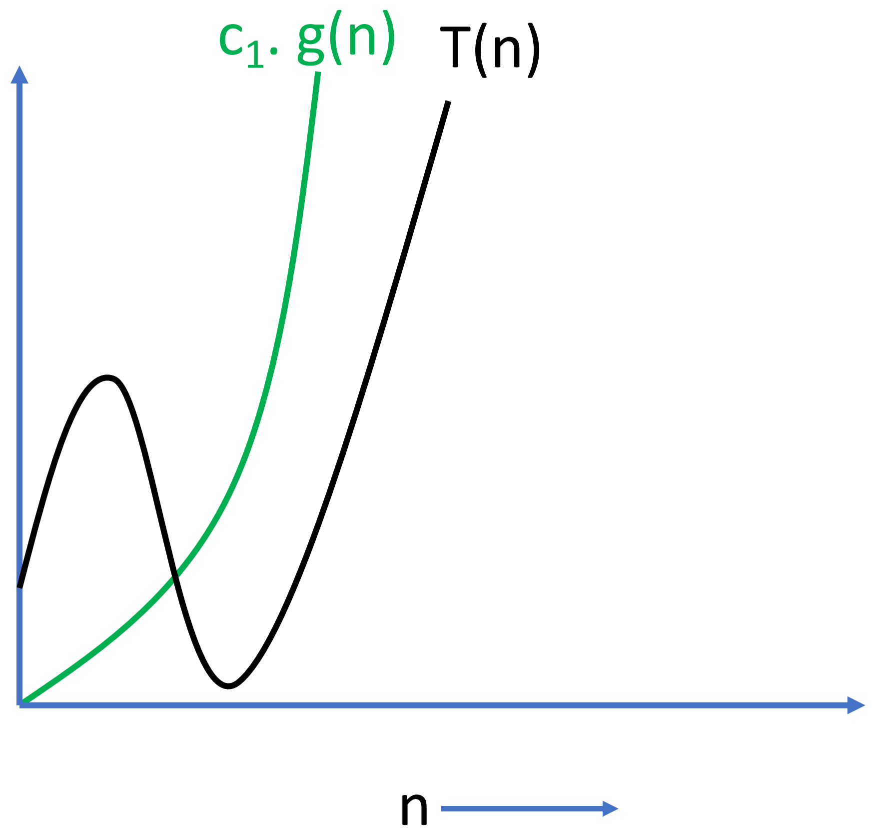
Big O notation

T(n) is Θ(g(n)) ⇔ T(n) is O(g(n)) AND T(n) is Ω(g(n))



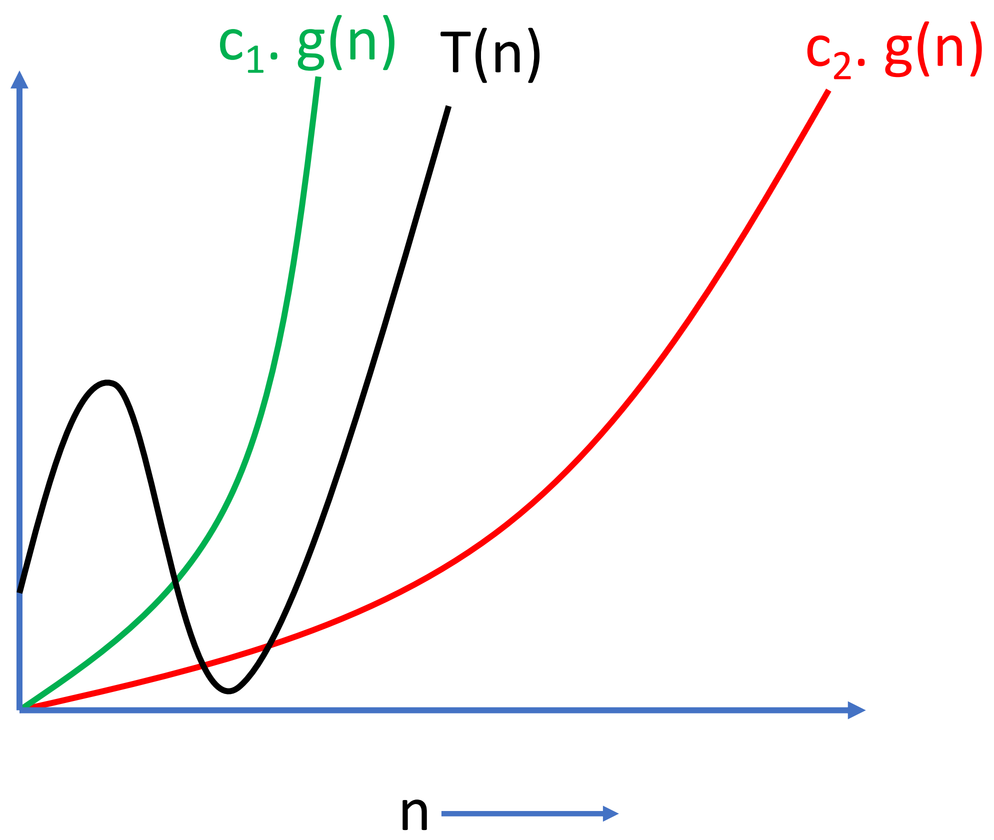
Big O notation

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



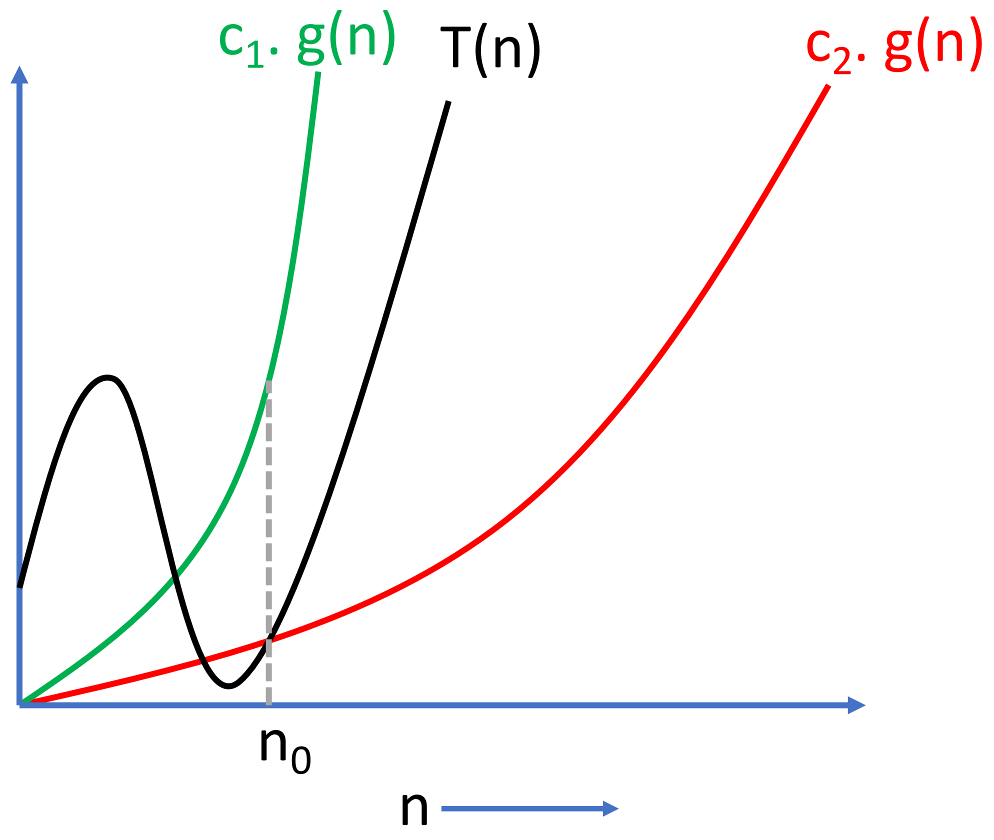
Big O notation

T(n) is Θ(g(n)) $\Leftrightarrow$ T(n) is O(g(n)) AND T(n) is Ω(g(n))



Big O notation

T(n) is Θ(g(n)) $\Leftrightarrow$ T(n) is O(g(n)) AND T(n) is Ω(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?

$$\Theta(n^2) \quad \xrightarrow{\text{BOTH}} \quad O(n^2),\ \Omega(n^2)$$

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

$O(n^2)$

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

$\Theta(n^2)$

BOTH

$\Omega\ (n^2)$

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