طراحی الگوریتم ها

مبحث دوم:
تحلیل زمانی الگوریتمها

سجاد شیرعلی شهرضا

بهار 1402

سه شنبه، 18 بهمن 1401

Based on Stanford CS161 Slides from Summer 2020 by Karey Shi

اطلاع رسانی

- نظرسنجی اول
  - مهلت ارسال آن: 8 صبح یکشنبه آینده، 23 بهمن 1401

بخشهای مرتبط در کتاب

- بخش مرتبط در کتاب CLRS برای مباحث این جلسه (تحلیل زمانی): 3
- واژه نامه ی انگلیسی به فارسی و فارسی به انگلیسی (پیوستهای 3 و 4 کتاب دکتر قدسی):
http://sharif.edu/~ghodsi/books/ds-algf-dics-both.pdf

FROM DATA STRUCTURE COURSE

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent *irrelevant for large inputs*

- **Some guiding principles:** we care about how the running time/number of operations *scales* with the size of the input (i.e. the runtime's *rate of growth*), and we want some measure of runtime that's independent of hardware, programming language, memory layout, etc.
    - We want to reason about high-level algorithmic approaches rather than lower-level details

A NOTE ON RUNTIME ANALYSIS

There are a few different ways to analyze the runtime of an algorithm:

Worst-case analysis:
What is the runtime of the algorithm on the *worst* possible input?

Best-case analysis:
What is the runtime of the algorithm on the *best* possible input?

Average-case analysis:
What is the runtime of the algorithm on the *average* input?

A NOTE ON RUNTIME ANALYSIS

There are a few different ways to analyze the runtime of an algorithm:

We'll mainly focus on worst case analysis since it tells us how fast the algorithm is on *any* kind of input

Worst-case analysis:
What is the runtime of the algorithm on the *worst* possible input?

Best-case analysis:
What is the runtime of the algorithm on the *best* possible input?

Average-case analysis:
What is the runtime of the algorithm on the *average* input?

We'll work with this more when we discuss amortized analysis!

BIG-O NOTATION

Let $T(n)$ & $f(n)$ be functions defined on the positive integers.

(In this class, we'll typically write $T(n)$ to denote the worst case runtime of an algorithm)

What do we mean when we say "$T(n)$ is $O(f(n))$"?

English Definition

Pictorial Definition

Mathematical Definition

BIG-O NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is O(f(n))"?

In English

T(n) = O(f(n)) if and only if T(n) is *eventually* **upper bounded** by a constant multiple of f(n)

Pictorial Definition

Mathematical Definition

BIG-O NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

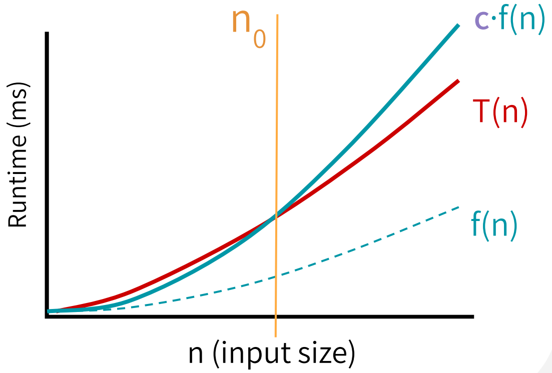
(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is O(f(n))"?

In English

T(n) = O(f(n)) if and only if T(n) is *eventually* **upper bounded** by a constant multiple of f(n)

In Pictures



Mathematical Definition

BIG-O NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

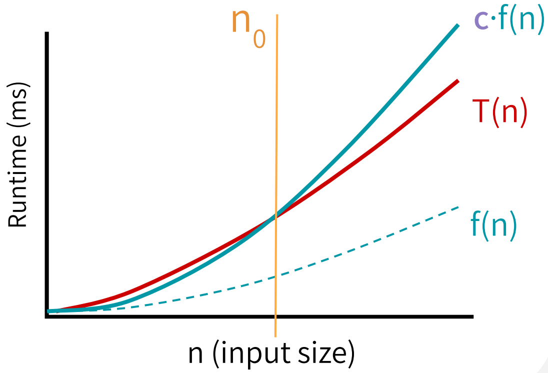
(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is O(f(n))"?

In English

T(n) = O(f(n)) if and only if T(n) is *eventually* **upper bounded** by a constant multiple of f(n)

In Pictures



In Math

T(n) = O(f(n)) if and only if there exists positive **constants** **c** and $\mathbf{n_0}$ such that *for all* $n \geq n_0$

$$T(n) \leq c \cdot f(n)$$

BIG-O NOTATION

Let $T(n)$ & $f(n)$ be functions defined on the positive integers.

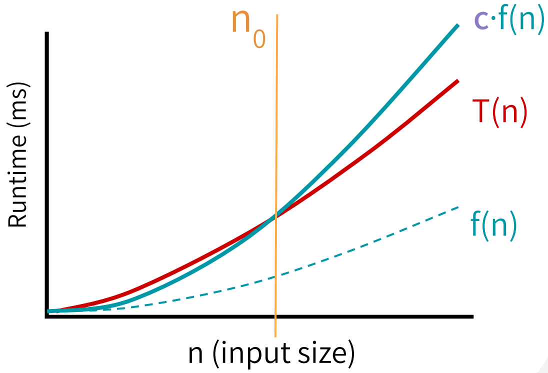
(In this class, we'll typically write $T(n)$ to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is O(f(n))"?

In English

$T(n) = O(f(n))$ if and only if $T(n)$ is *eventually* **upper bounded** by a constant multiple of $f(n)$

In Pictures



In *Math*

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

$$\Leftrightarrow$$

$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$\mathbf{T(n) \leq c \cdot f(n)}$$

BIG-O NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

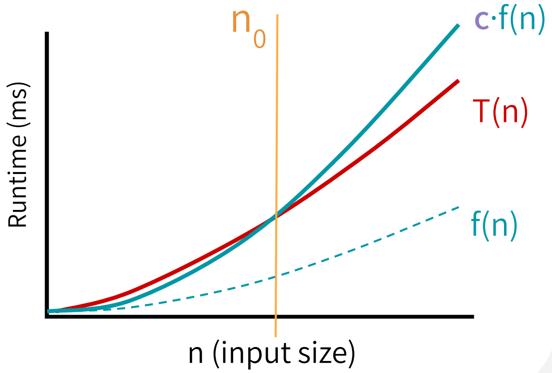
(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is O(f(n))"?

In English

T(n) = O(f(n)) if and only if T(n) is *eventually* **upper bounded** by a constant multiple of f(n)

In Pictures



In *Math*

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

"if and only if" $\Leftrightarrow$

"for all"

$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$T(n) \leq c \cdot f(n)$$

"such that"

"there exists"

PROVING BIG-O BOUNDS

If you're ever asked to formally prove that T(n) is O(f(n)), use the *MATH* definition:

$$T(n) = O(f(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$\mathbf{T(n) \leq c \cdot f(n)}$$

must be constants!
i.e. c & n_0 cannot
depend on n!

- To **prove** T(n) = O(f(n)), you need to announce your c & n_0 up front!
  - Play around with the expressions to find appropriate choices of c & n_0 (positive constants)
  - Then you can write the proof! Here how to structure the start of the proof:

PROVING BIG-O BOUNDS

If you're ever asked to formally prove that T(n) is O(f(n)), use the *MATH* definition:

$$T(n) = O(f(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$\mathbf{T(n) \leq c \cdot f(n)}$$

must be constants!
i.e. c & n_0 cannot
depend on n!

- To **prove** T(n) = O(f(n)), you need to announce your c & n_0 up front!
    - Play around with the expressions to find appropriate choices of c & n_0 (positive constants)
    - Then you can write the proof! Here how to structure the start of the proof:

"Let c = __ and n_0 = __. We will show that T(n) ≤ c·f(n) for all n ≥ n_0."

PROVING BIG-O BOUNDS: EXAMPLE

$$T(n) = O(f(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$T(n) \leq c \cdot f(n)$$

Prove that $3n^2 + 5n = O(n^2)$.

Let $c = 4$ and $n_0 = 5$. We will now show that $3n^2 + 5n \leq c \cdot n^2$ for all $n \geq n_0$. We know that for any $n \geq n_0$, we have:

$$5 \leq n$$
$$5n \leq n^2$$
$$3n^2 + 5n \leq 4n^2$$

Using our choice of c and n_0, we have successfully shown that $3n^2 + 5n \leq c \cdot n^2$ for all $n \geq n_0$. From the definition of Big-O, this proves that $3n^2 + 5n = O(n^2)$. ■

DISPROVING BIG-O BOUNDS

If you're ever asked to formally disprove that T(n) is O(f(n)), use **proof by contradiction!**

DISPROVING BIG-O BOUNDS

If you're ever asked to formally disprove that T(n) is O(f(n)), use **proof by contradiction!**

For sake of contradiction, assume that T(n) is O(f(n)). In other words, assume there does indeed exist a choice of c & n_0 s.t. $\forall\ n \geq n_0$, $\mathbf{T(n) \leq c \cdot f(n)}$

pretend you have a friend that comes up and says "I have a c & n_0 that will prove T(n) = O(f(n))!!!", and you say "ok fine, let's assume your c & n_0 does prove T(n) = O(f(n))"

DISPROVING BIG-O BOUNDS

If you're ever asked to formally disprove that T(n) is O(f(n)), use **proof by contradiction!**

For sake of contradiction, assume that T(n) is O(f(n)). In other words, assume there does indeed exist a choice of c & n_0 s.t. $\forall$ $n \geq n_0$, $\mathbf{T(n) \leq c \cdot f(n)}$

pretend you have a friend that comes up and says "I have a c & n_0 that will prove T(n) = O(f(n))!!!", and you say "ok fine, let's assume your c & n_0 does prove T(n) = O(f(n))"

↓

Treating c & n_0 as variables, derive a contradiction!

although you are skeptical, you'll entertain your friend by saying: "let's see what happens. [some math work... and then...] AHA! regardless of what your constants c & n_0, trusting you has led me to something *impossible!!!*"

DISPROVING BIG-O BOUNDS

If you're ever asked to formally disprove that T(n) is O(f(n)), use **proof by contradiction!**

For sake of contradiction, assume that T(n) is O(f(n)). In other words, assume there does indeed exist a choice of c & n_0 s.t. $\forall\ n \geq n_0$, $\mathbf{T(n) \leq c \cdot f(n)}$

pretend you have a friend that comes up and says "I have a c & n_0 that will prove T(n) = O(f(n))!!!", and you say "ok fine, let's assume your c & n_0 does prove T(n) = O(f(n))"

Treating c & n_0 as variables, derive a contradiction!

although you are skeptical, you'll entertain your friend by saying: "let's see what happens. [some math work... and then...] AHA! regardless of what your constants c & n_0, trusting you has led me to something *impossible!!!*"

Conclude that the original assumption must be false, so **T(n) is *not* O(f(n))**.

you have triumphantly proven your silly (or lying) friend wrong.

DISPROVING BIG-O: EXAMPLE

$$T(n) = O(f(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$T(n) \leq c \cdot f(n)$$

Prove that $3n^2 + 5n$ is *not* $O(n)$.

For sake of contradiction, assume that $3n^2 + 5n$ is $O(n)$. This means that there exists positive constants c & n_0 such that $3n^2 + 5n \leq c \cdot n$ for all $n \geq n_0$. Then, we would have the following:

$$3n^2 + 5n \leq c \cdot n$$
$$3n + 5 \leq c$$
$$n \leq (c - 5)/3$$

However, since $(c - 5)/3$ is a constant, we've arrived at a contradiction since n cannot be bounded above by a constant for all $n \geq n_0$. For instance, consider $n = n_0 + c$: we see that $n \geq n_0$, but $n > (c - 5)/3$. Thus, our original assumption was incorrect, which means that $3n^2 + 5n$ is not $O(n)$. ■

BIG-O EXAMPLES

$\log_2 n + 15 = O(\log_2 n)$

$3^n = O(4^n)$

Polynomials

Say $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ is a polynomial of degree $k \geq 1$.

Then:

i. $p(n) = O(n^k)$
ii. $p(n)$ is **not** $O(n^{k-1})$

$6n^3 + n \log_2 n = O(n^3)$

$25 = O(1)$

[any constant] $= O(1)$

BIG-O EXAMPLES

lower order terms
don't matter!

$$\log_2 n + 15 = O(\log_2 n)$$

remember, big-O
is upper bound!

$$3^n = O(4^n)$$

Polynomials

Say $p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0$ is a polynomial of degree $k \geq 1$.

Then:

i. $p(n) = O(n^k)$
ii. $p(n)$ is **not** $O(n^{k-1})$

constant multipliers & lower
order terms don't matter

$$6n^3 + n \log_2 n = O(n^3)$$

$$25 = O(1)$$

$$[\text{any constant}] = O(1)$$

سوال؟

BIG-Ω NOTATION

Let $T(n)$ & $f(n)$ be functions defined on the positive integers.

(In this class, we'll typically write $T(n)$ to denote the worst case runtime of an algorithm)

What do we mean when we say "$T(n)$ is $\Omega(f(n))$"?

English Definition

Pictorial Definition

Mathematical Definition

BIG-Ω NOTATION

Let $T(n)$ & $f(n)$ be functions defined on the positive integers.

(In this class, we'll typically write $T(n)$ to denote the worst case runtime of an algorithm)

What do we mean when we say "$T(n)$ is $\Omega(f(n))$"?

In English

$T(n) = \Omega(f(n))$ if and only if $T(n)$ is eventually **lower bounded** by a constant multiple of $f(n)$

Pictorial Definition

Mathematical Definition

BIG-Ω NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

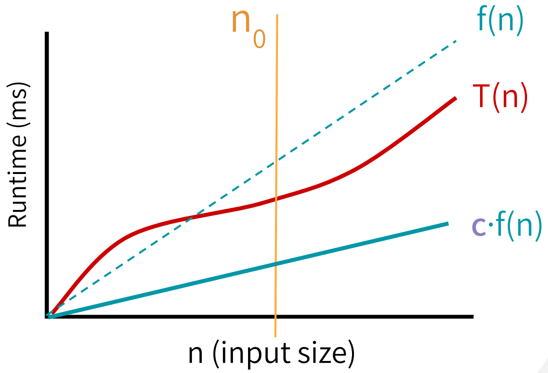
(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is Ω(f(n))"?

In English

T(n) = Ω(f(n)) if and only if T(n) is eventually **lower bounded** by a constant multiple of f(n)

In Pictures



Mathematical Definition

BIG-Ω NOTATION

Let T(n) & f(n) be functions defined on the positive integers.

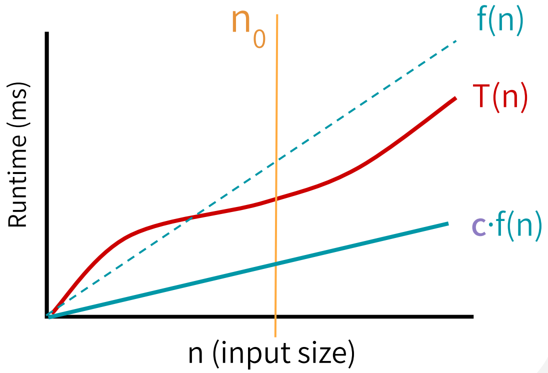
(In this class, we'll typically write T(n) to denote the worst case runtime of an algorithm)

What do we mean when we say "T(n) is Ω(f(n))"?

In English

T(n) = Ω(f(n)) if and only if T(n) is eventually **lower bounded** by a constant multiple of f(n)

In Pictures



In *Math*

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

$$\Leftrightarrow$$

$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$T(n) \geq c \cdot f(n)$$

inequality switched directions!

BIG-Θ NOTATION

We say **"T(n) is Θ(f(n))"** if and only if both

$$\mathbf{T(n) = O(f(n))}$$

and

$$\mathbf{T(n) = \Omega(f(n))}$$

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

$$\Leftrightarrow$$

$$\exists\ c_1, c_2, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

ASYMPTOTIC NOTATION CHEAT SHEET

BOUND	DEFINITION (HOW TO PROVE)	WHAT IT REPRESENTS
$T(n) = O(f(n))$	$\exists\ c > 0,\ \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ T(n) \leq c \cdot f(n)$	upper bound
$T(n) = \Omega(f(n))$	$\exists\ c > 0,\ \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ T(n) \geq c \cdot f(n)$	lower bound
$T(n) = \Theta(f(n))$	$T(n) = O(f(n))$ and $T(n) = \Omega(f(n))$	tight bound

سوال؟