DATA STRUCTURE AND ALGORITHMS

Algorithmic Complexity

Overview

Computational complexity

- How much time will it take a program to run?
- How much memory will it need to run?
- Need to balance minimizing computational complexity with conceptual complexity

Measuring complexity

- Goals in designing programs
  1. It returns the correct answer on all legal inputs
  2. It performs the computation efficiently

- Typically (1) is most important, but sometimes (2) is also critical
- Even when (1) is most important, it is valuable to understand and optimize (2)

How do we measure complexity?

- Given a function, would like to answer: “How long will this take to run?”
- Problem is that this depends on:
  1. Speed of compute.
  2. Specifics of Programming Language implementation
  3. Value of input
- Avoid (1) and (2) by measuring time in terms of number of basic steps executed
- For point (3), measure time in terms of size of input

Cases for measuring complexity

- **Best case:** minimum running time over all possible inputs of a given size
  - For linearSearch – constant, i.e. independent of size of inputs
- **Worst case:** maximum running time over all possible inputs of a given size
  - For linearSearch – linear in size of list
- **Average (or expected) case:** average running time over all possible inputs of a given size
- We will focus on worst case – a kind of **upper bound** on running time

Example

```
def fact(n):
    answer = 1
    while n > 0:
        answer *= n
        n -= 1
    return answer
```

- Number of steps:

- 1 (for assignment)

- 5*n (1 for test, plus 2 for first assignment, plus 2 for second assignment in while; repeated n times through while)

- 1 (for return)

- 5*n+2steps
- But as n gets large, 2 is irrelevant, so basically 5*n steps

Big-Oh notation!

- Gives us a meaningful way to talk about the running time of an algorithm, independent of programming language, computing platform, etc., without having to count all the operations.
- Focus on how the runtime **scales** with n (the input size).

Big-Oh Example

Number of operations	Asymptotic Running Time
$\frac{1}{10} \cdot n^2 + 100$	$O(n^2)$
$0.063 \cdot n^2 - .5\,n + 12.7$	$O(n^2)$
$100 \cdot n^{1.5} - 10^{10000}\sqrt{n}$	$O(n^{1.5})$
$11 \cdot n\log(n) + 1$	$O(n\log(n))$

We say this algorithm is "asymptotically faster" than the others.

Why is this a good idea?

- Suppose the running time of an algorithm is:

$$T(n) = 10n^2 + 3n + 7 \quad \text{ms}$$

This constant factor of 10 depends a lot on my computing platform...

These lower-order terms don't really matter as n gets large.

We're just left with the n^2 term! That's what's meaningful.

Formal definition of O(…)

- Let $T(n)$, $g(n)$ be functions of positive integers.
  - Think of $T(n)$ as a runtime: positive and increasing in n.

- Formally,

$$T(n) = O\big(g(n)\big)$$

"If and only if"

$$\Longleftrightarrow$$

"For all"

$$\exists c > 0, n_0 \;\; s.t. \;\; \forall n \geq n_0,$$

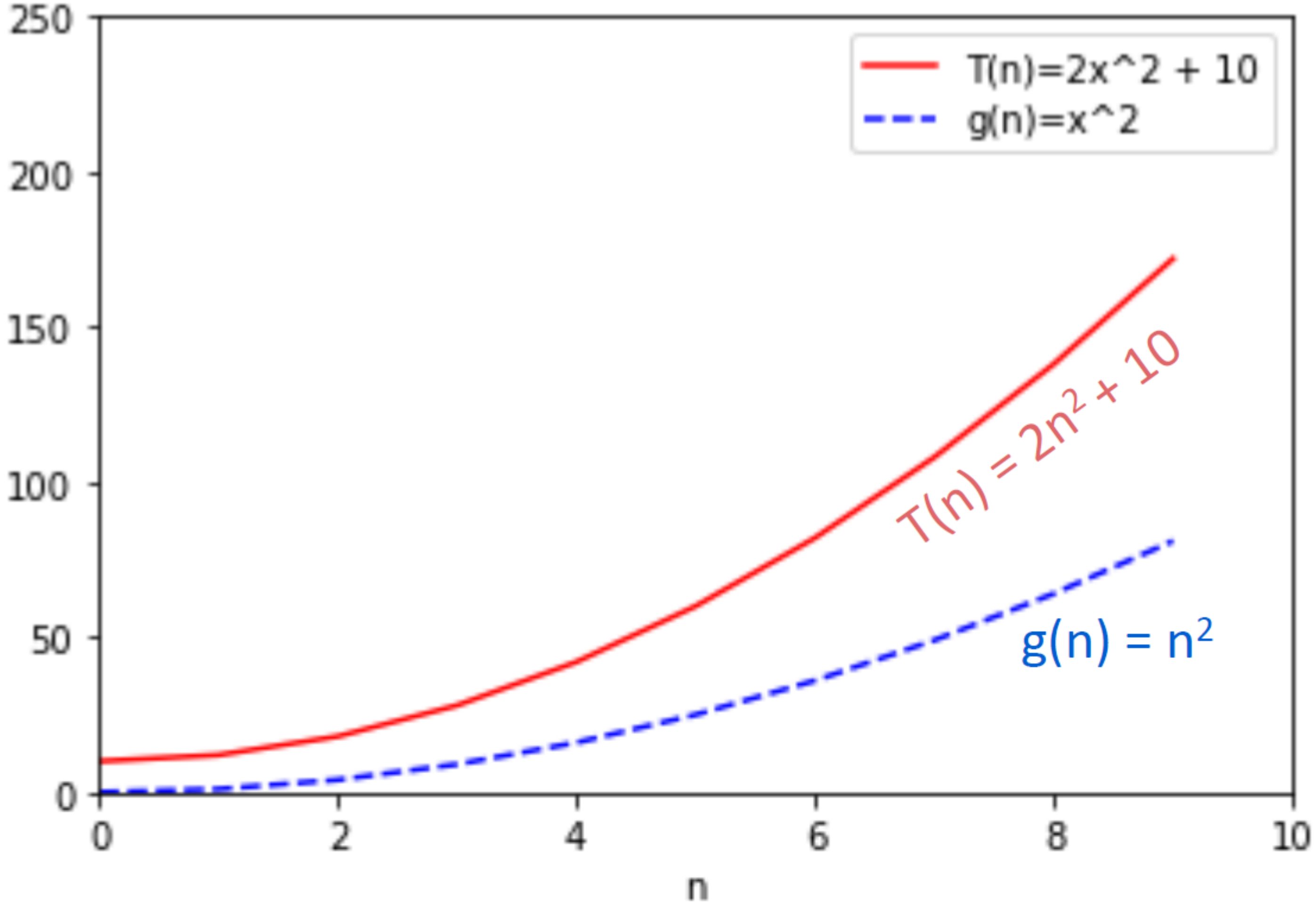
$$T(n) \leq c \cdot g(n)$$

"There exists"

"such that"

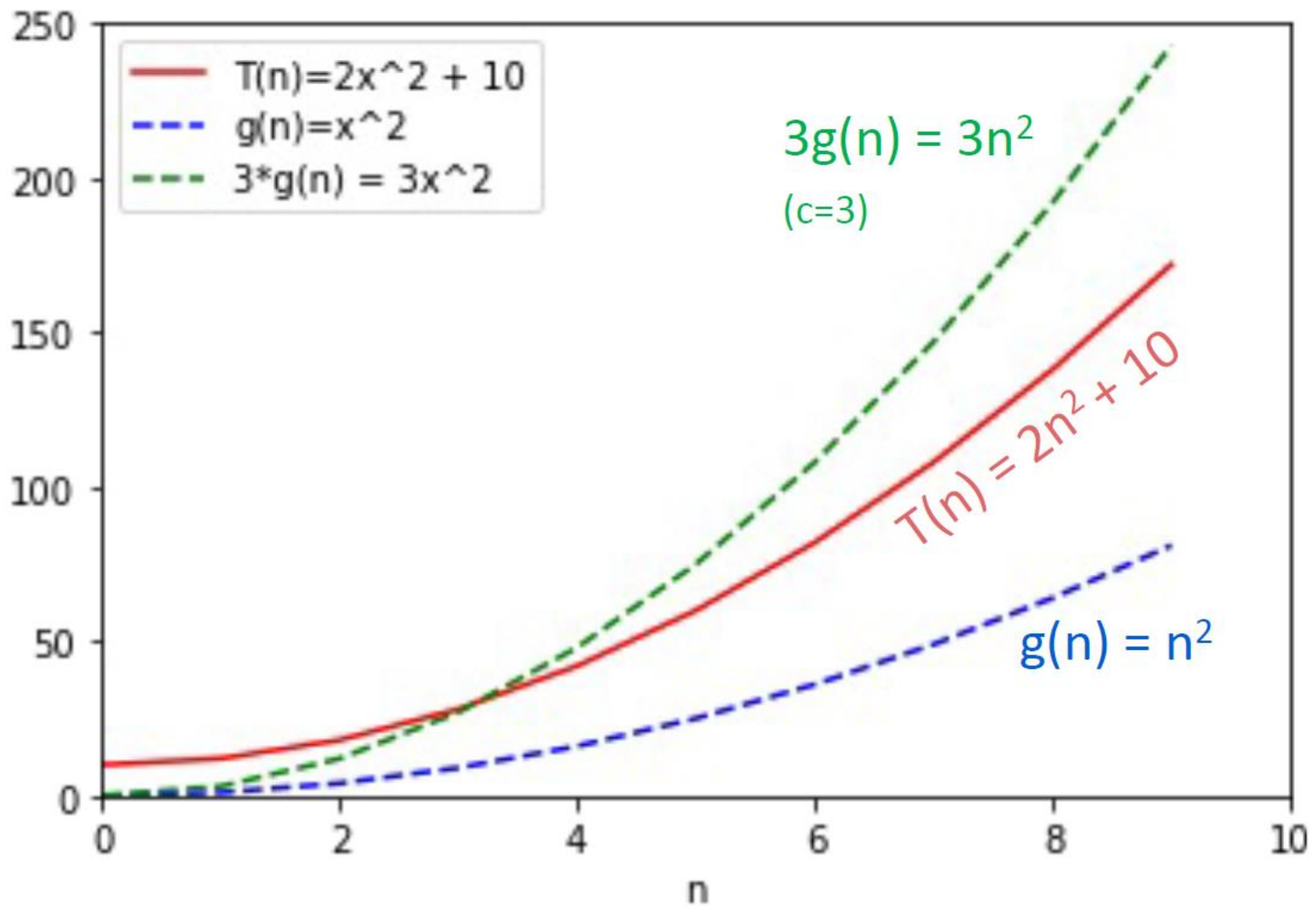
Example

- $2n^2 + 10 = O(n^2)$



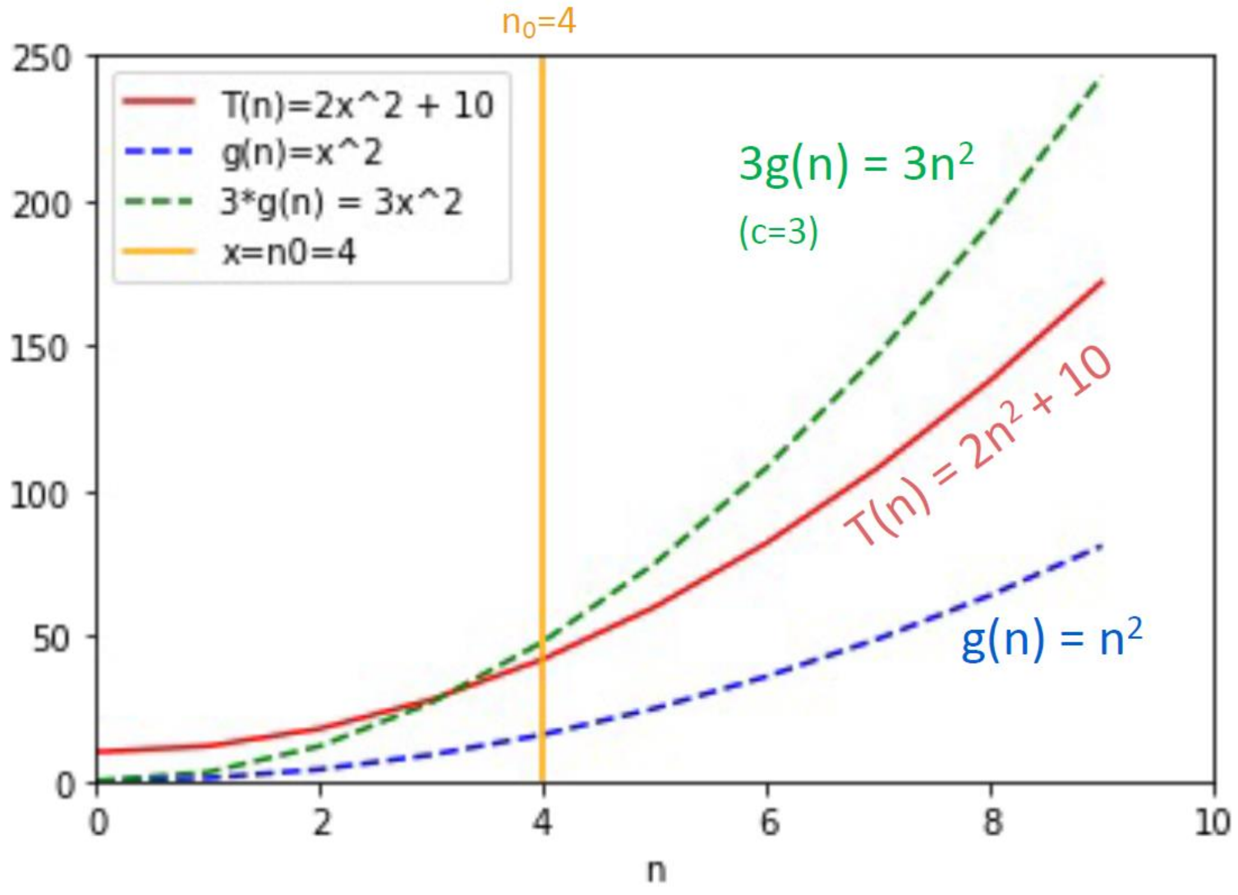
Example

- $2n^2 + 10 = O(n^2)$



Example

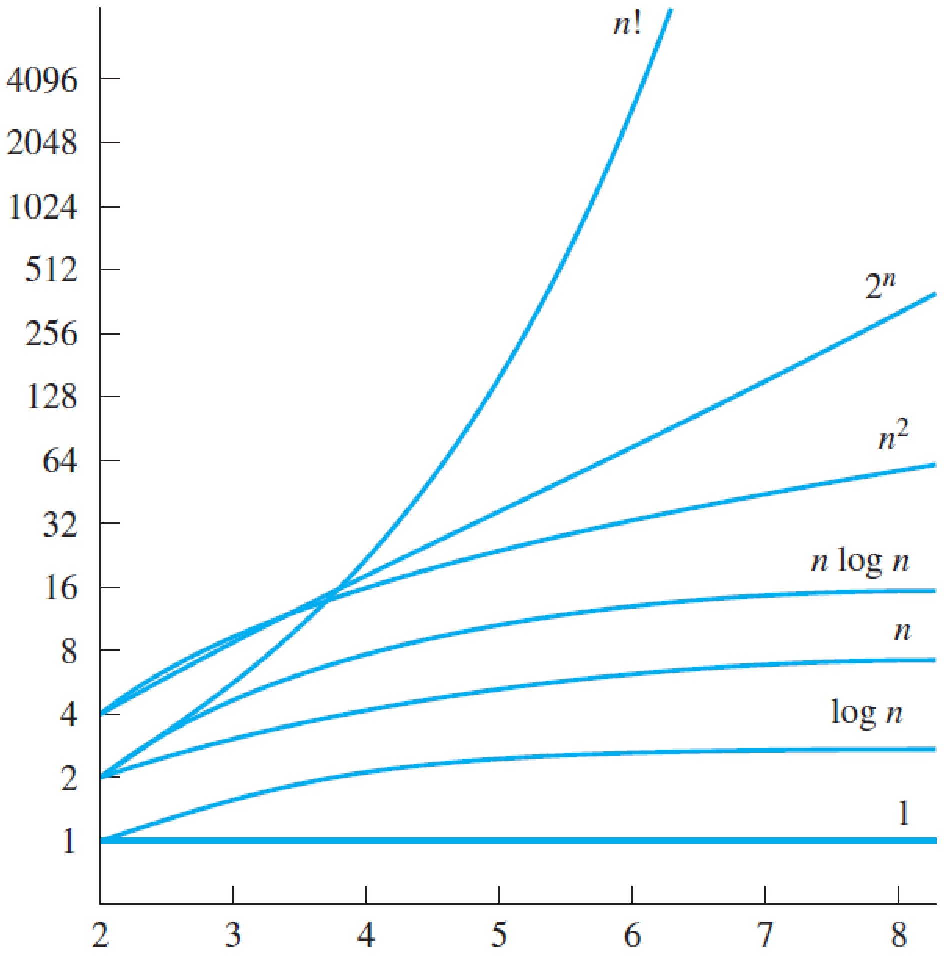
- $2n^2$ + 10 = $O(n^2)$



Choose c = 3
Choose n_0 = 4
Then:

$$\forall n \geq 4, 2n^2 + 10 \leq 3n^2$$

Growth of functions commonly used in big-O



Exercise

- Determine whether each of these functions is O(x).

$a)\ f(x) = 10$

$b)\ f(x) = 3x + 7$

$c)\ f(x) = x^2 + x + 1$

$d)\ f(x) = 5\log(x)$

Ω(…) means a lower bound

- Let $T(n)$, $g(n)$ be functions of positive integers.
  - Think of $T(n)$ as a runtime: positive and increasing in n.

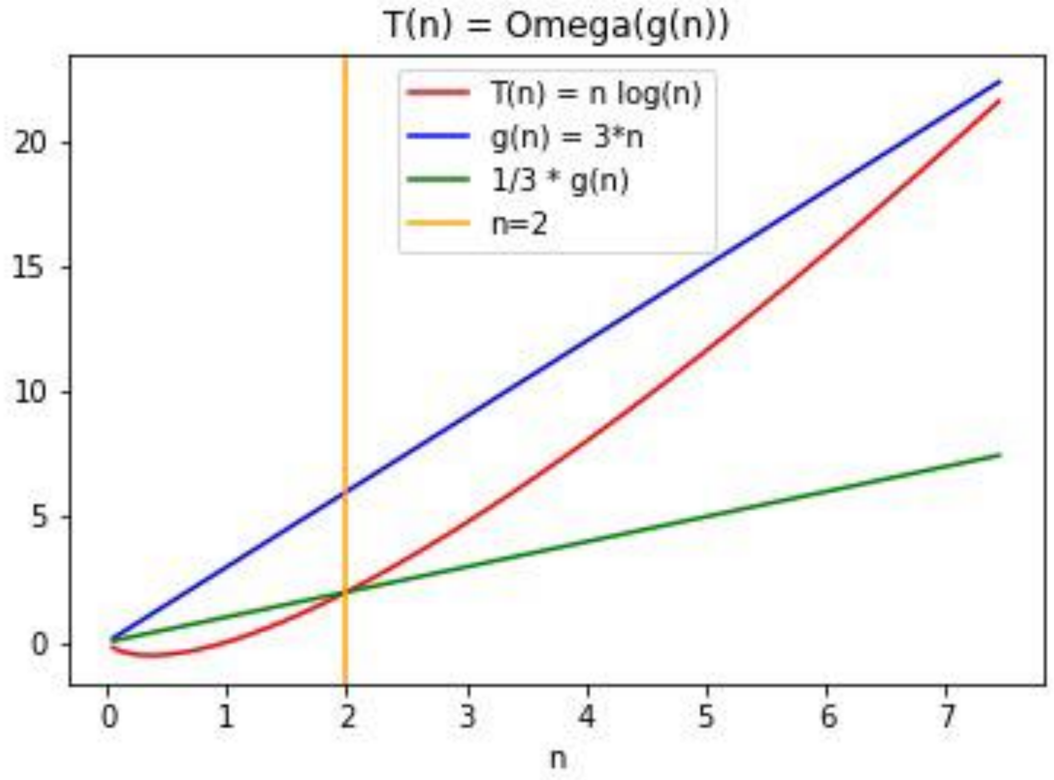
- Formally,

$$T(n) = \Omega\big(g(n)\big)$$
$$\Longleftrightarrow$$
$$\exists c > 0\,, n_0 \;\; s.t. \;\; \forall n \geq n_0,$$
$$c \cdot g(n) \leq T(n)$$

Switched these!!

Example

- $n log_2(n) = \Omega(3n)$



- Choose c = 1/3
- Choose $n_0 = 2$
- Then $\forall n \geq 2, \frac{3n}{3} \leq n log_2(n)$

Θ(…) means both!

- $T(n)\ is\ \Theta\big(g(n)\big)$ iff both:

$$T(n) = O\big(g(n)\big)$$

and

$$T(n) = \Omega\big(g(n)\big)$$

Exercise: Show that

- $3x^2 + 8xlogx = \Theta(x^2)$

Exercise

- Arrange the functions $logn$, n, $\mathrm{n}logn$, $n^m, m > 1$,2^n,n! in a list so that each function is big-*O* of the next function.