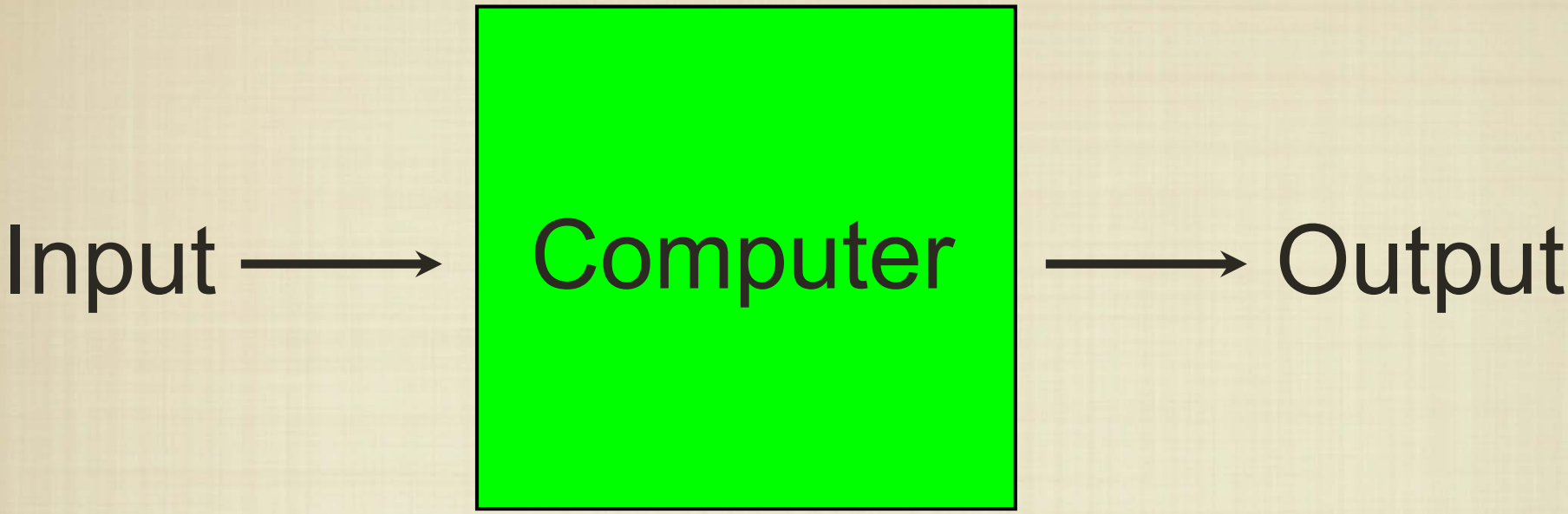
Algorithm Analysis

Fall 2013

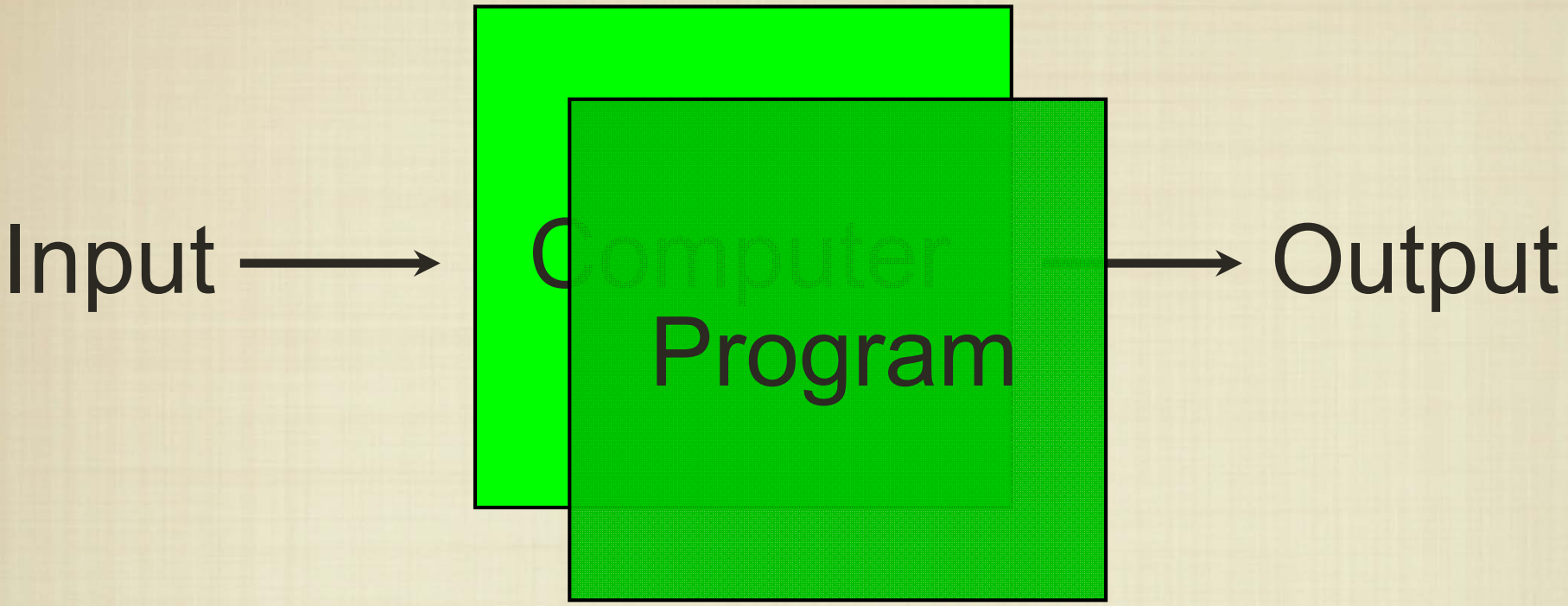
Carola Wenk

Computer – Program – Algorithm



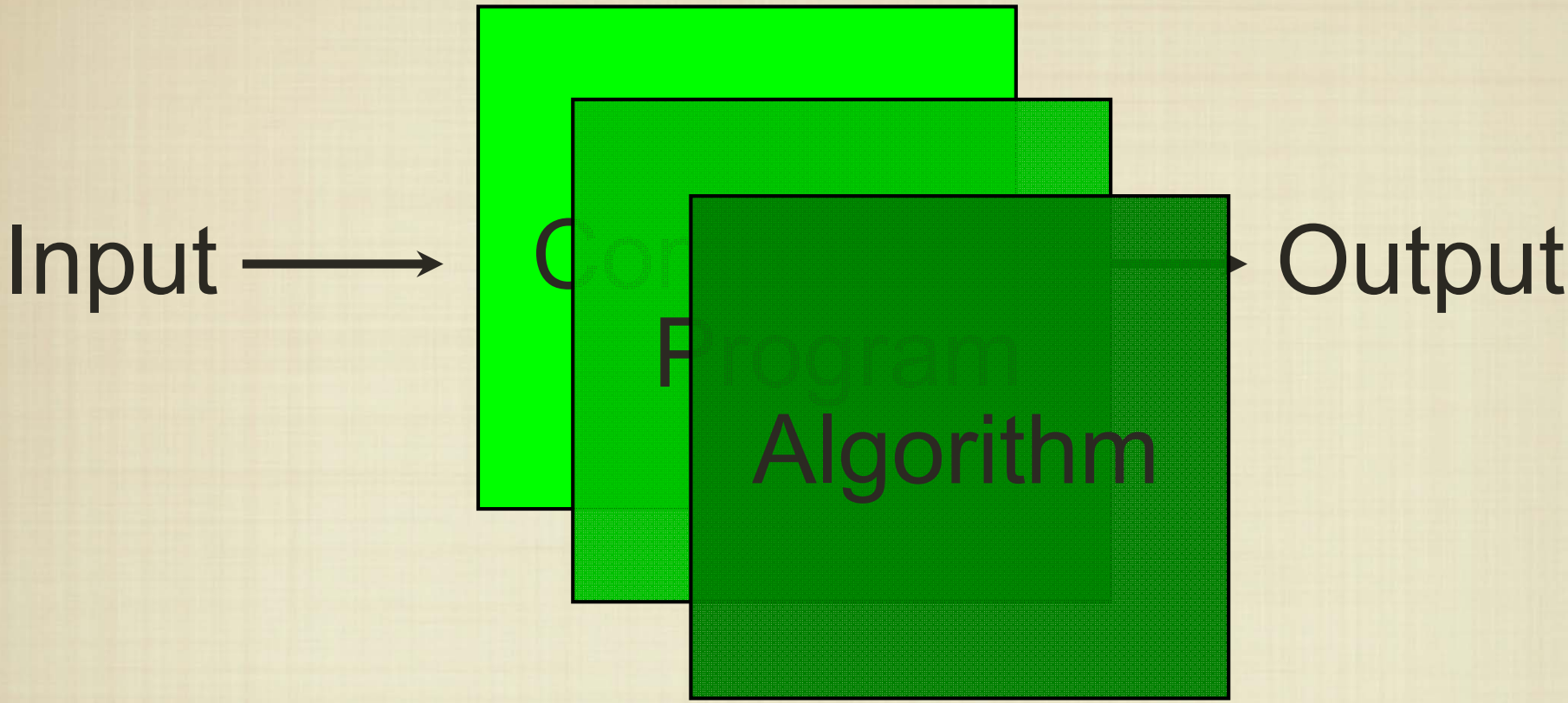
Nearly every modern electronic device can be thought of as a computer that transforms input to the desired output.

Computer – Program – Algorithm



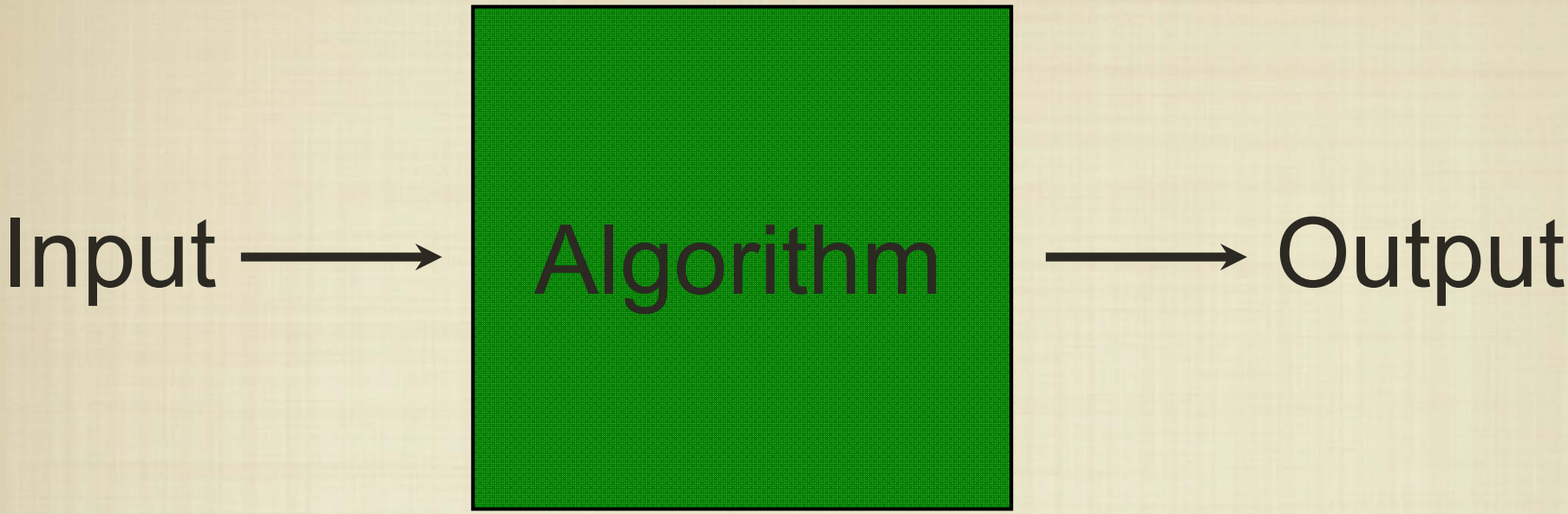
Most computers are general-purpose: we can accomplish a number of different tasks on a single piece of hardware by changing the program being executed.

Computer – Program – Algorithm



A program is just a realization (= implementation) of an abstract procedure, or algorithm, on a particular hardware platform (= computer). One algorithm can be used for a variety of applications.

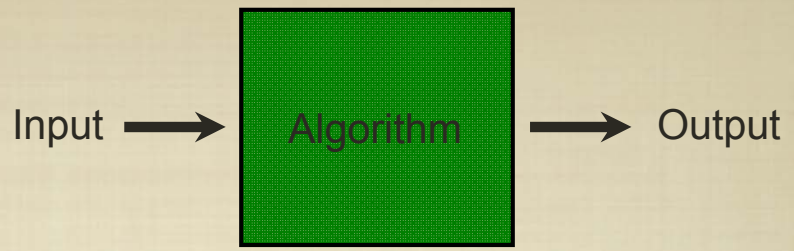
What is an Algorithm?



We think of an algorithm as a sequence of actions to take input and produce output.

We assume a model of computation, usually an abstract instruction set (arithmetic operations, if-then, loops) and assign unit cost (= time) to them.

Describe an Algorithm



1. Define the problem (input, output)
2. Describe the algorithm (in words or in pseudo-code)
3. Proof of correctness (convince the reader of correctness)
4. Analysis (runtime, space)

Describe an Algorithm: Computing the minimum of *n* numbers

1. Define the problem (input, output)

Input: A list of *n* numbers.

Output: The value of the minimum number.
[Other option: The index of the minimum number.]

Describe an Algorithm: Computing the minimum of *n* numbers

2. Describe the algorithm (in words or in pseudo-code)

- Loop through all numbers.
- Keep track of the minimum number seen so far.
- If current number is less than the minimum seen so far, update the minimum.

Describe an Algorithm: Computing the minimum of *n* numbers

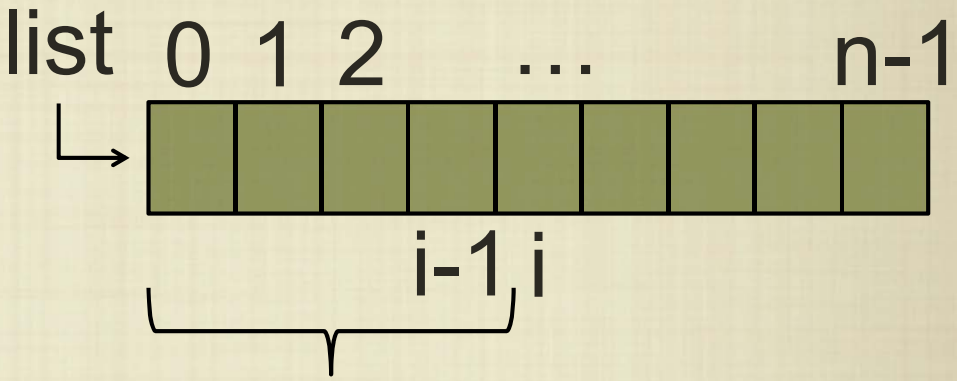
2. Describe the algorithm (in words or in pseudo-code)

```
def my_min(list):
    min_so_far = list[0]
    for i in range(1,len(list)):
        if list[i] < min_so_far:
            min_so_far = list[i]
    return min_so_far
```

Describe an Algorithm: Computing the minimum of *n* numbers

3. Proof of correctness (convince the reader of correctness)

```
def my_min(list):
    min_so_far = list[0]
    for i in range(1,len(list)):
        if list[i] < min_so_far:
            min_so_far = list[i]
    return min_so_far
```



- At beginning of loop body:
  min_so_far = minimum of list[0]…list[i-1]
- Loop body updates min_so_far

Describe an Algorithm: Computing the minimum of *n* numbers

4. Analysis (runtime, space)

- Let *n* = len(list)

```
def my_min(list):
    min_so_far = list[0]
    for i in range(1,len(list)):
        if list[i] < min_so_far:
            min_so_far = list[i]
    return min_so_far
```

Instructions:

1

≤ 3(*n*-1)

1

- Runtime: ≤ 2 + 3*n* – 3 = 3*n* -1

Runtime linear in *n*

Runtime O(*n*)

Slow algorithm to compute the minimum of *n* numbers

- Let *n* = len(list)

```
def my_min_slow(list):
    min_so_far = list[0]
    for j in range(1,len(list)+1):
        for i in range(0,j):
            if list[i] < min_so_far:
                min_so_far = list[i]
    return min_so_far
```

Instructions:

1

≤ 3j $\sum_{j=1}^{n} (3j + 1)$

$= 3\,\frac{n(n+1)}{2} + n$

1

- Runtime: ≤ $2 + \frac{3}{2}n^2 + \frac{5}{2}n$

Runtime quadratic in *n*

Runtime $O(n^2)$

Runtimes: Functions in input size *n*

Runtime for my_min: f(*n*)= 3*n*-1

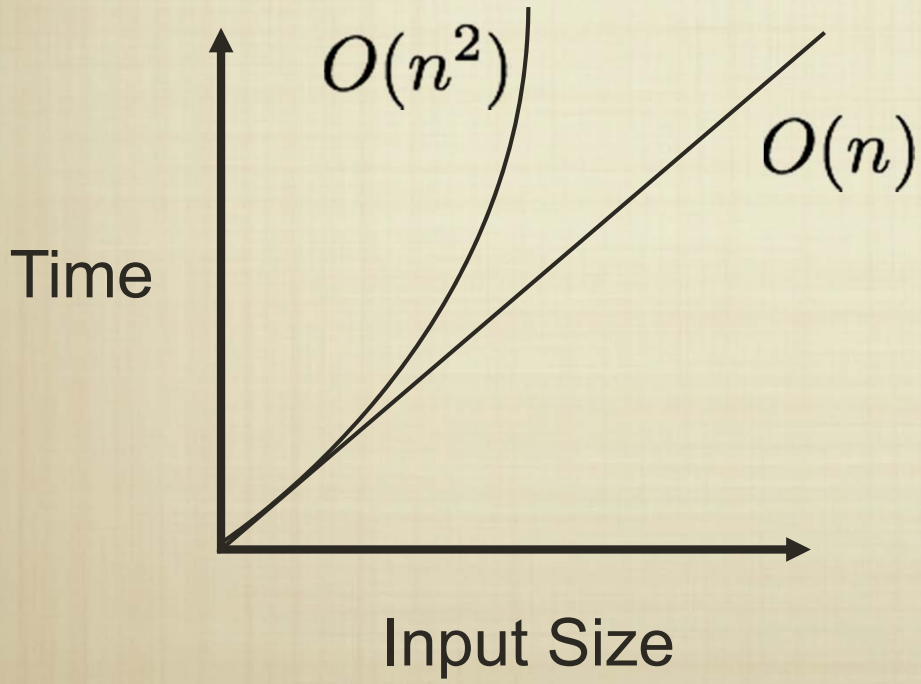
Runtime for my_min_slow: g(*n*) = $2 + \frac{3}{2}\, n^2 + \frac{5}{2} n$

Which one is better? And for what values of *n*?

Asymptotic Runtime Analysis

To evaluate the abstract runtime of an algorithm, we want to know its asymptotic behavior as a function of the input size.

$\rightarrow$ How does the algorithm perform if the input grows larger and larger?

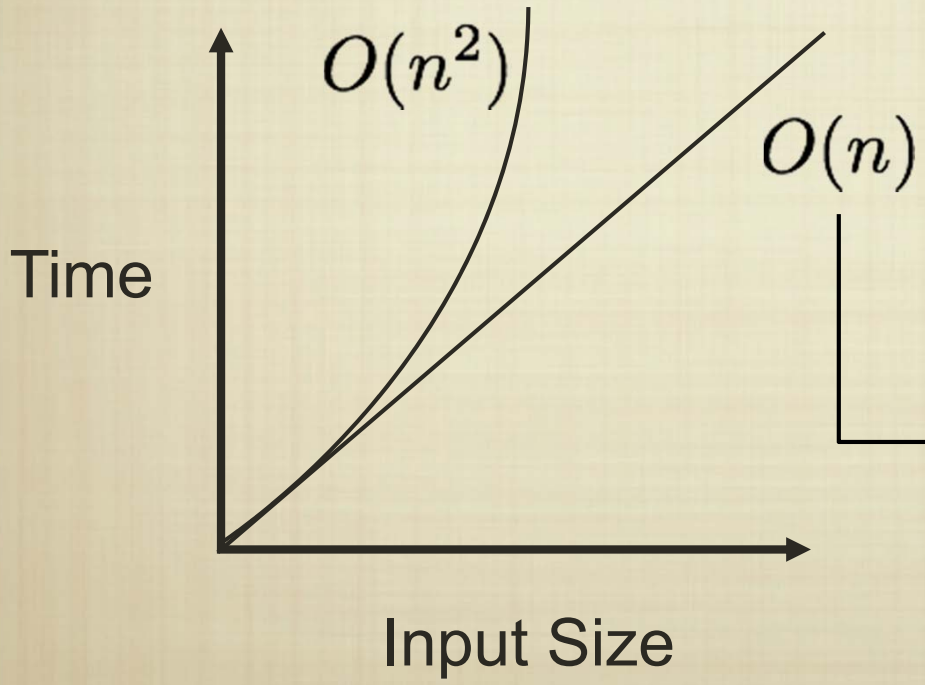


The growth rate of the running time allows us to compare and contrast two potential algorithms before implementing them.

(Worst-Case) Asymptotic Runtime Analysis

Usually, the abstract performance of an algorithm depends on the actual input for any particular size n.

Which inputs should we use to characterize runtime?



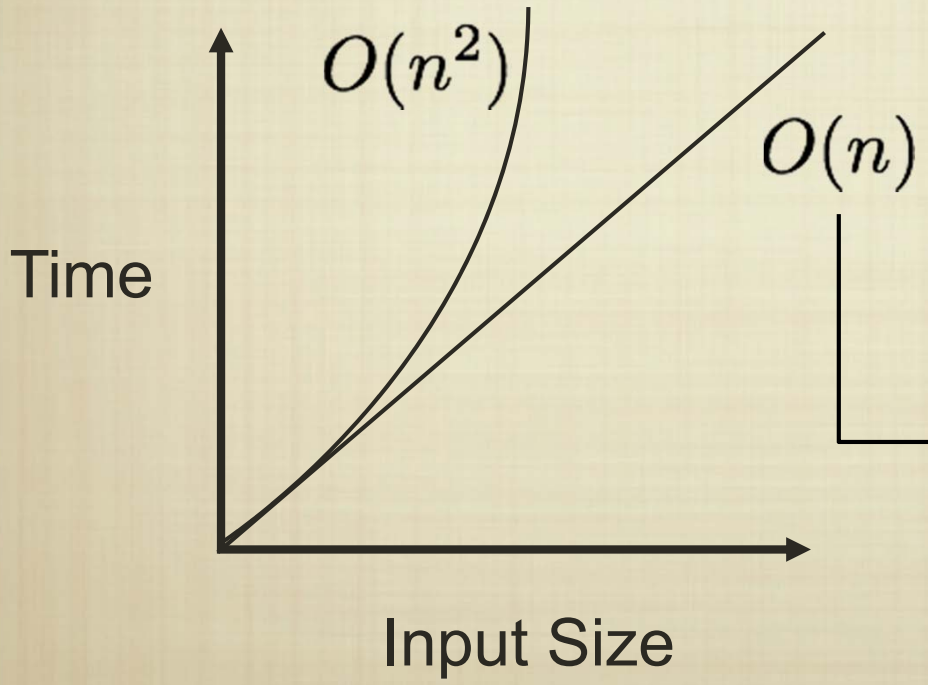
We define algorithm performance as conservatively as possible, on the worst-case inputs.

"No matter what, my algorithm takes at most c·*n* steps for an input size of *n*."

(Worst-Case) Asymptotic Runtime Analysis

Usually, the abstract performance of an algorithm depends on the actual input for any particular size n.

Which inputs should we use to characterize runtime?



We define algorithm performance as conservatively as possible, on the worst-case inputs.

Definition: f(n)∈O(g(n)) iff

$\exists c, n_0 > 0 \forall n \geq n_0 : f(n) \leq c\ g(n)$