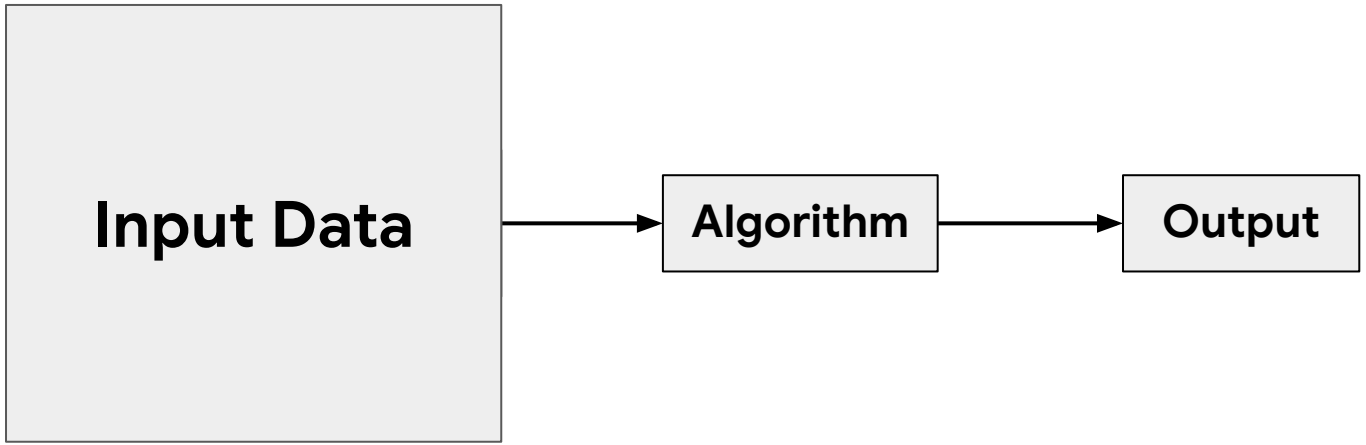
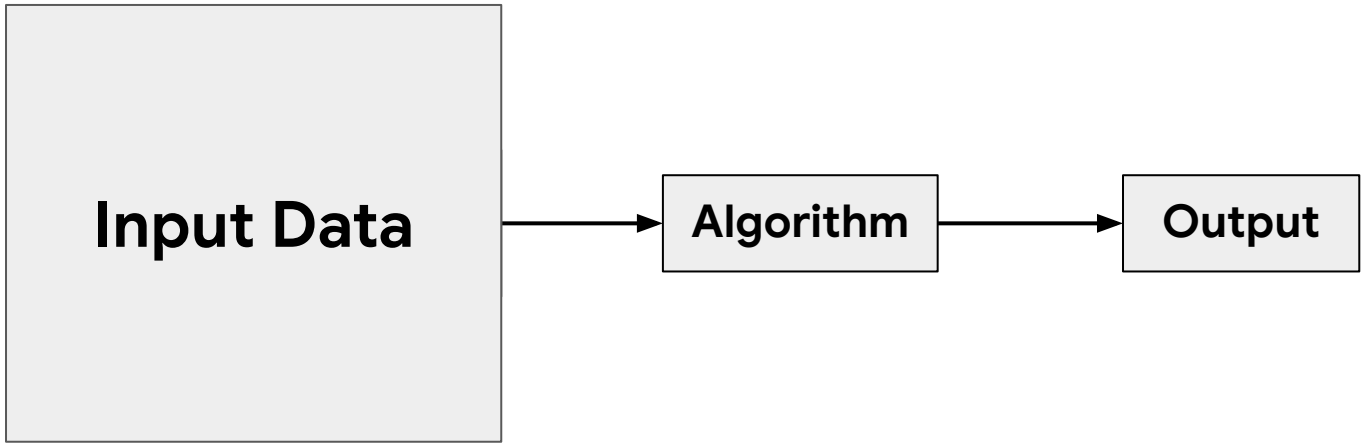
Algorithm Efficiency & Asymptotics Analysis

What this course is about...



- How do we store data?
  - Easily accessible for algorithmic computations
  - Efficient algorithms when data is large
- Implementation in C

What this course is about...



- How do we store data?
  - Easily accessible for algorithmic computations
  - ***Efficient*** algorithms when data is large
- Implementation in C

- Running time
- Memory

Running time of An Algorithm

How do we measure the following algorithm's running time?

```
ExchangeSort(A):
    n = A.length
    For i = 0, …, n-1:
        For j = i,...,n-1:
            If A[i] > A[j]:
                Swap A[i] and A[j]
```

# of times this line is run:
$(n-1)+(n-2)+\ldots+1$
$= 0.5n^2 - 0.5n$ times

May not be run based on the if-else condition

Implement, Run & Time it

- ✔ Captures the Running Time Precisely
- ❌ Depends on environments we run on
- ❌ Depends on the input (size) we run on

Count # operations in terms of input size

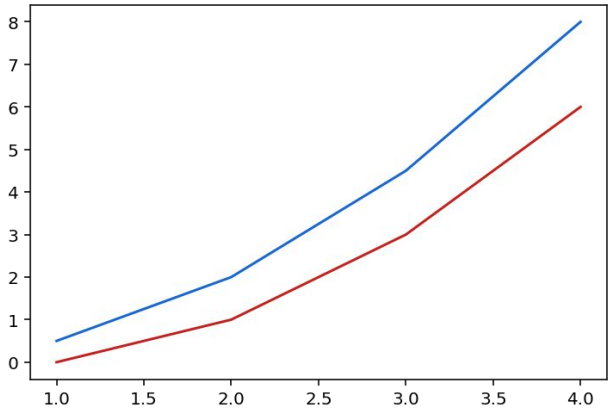
worst case

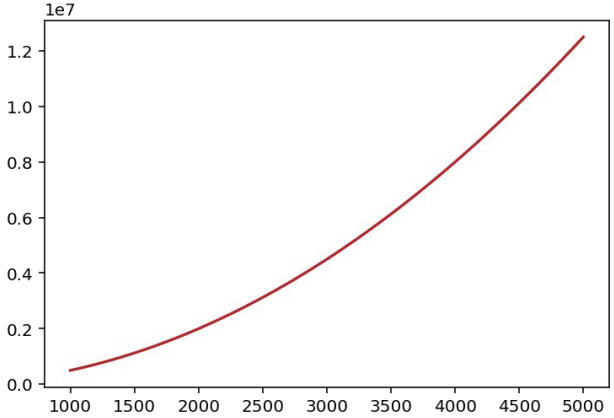
- ✔ Agnostic to environments
- ✔ Shows dependency on input size
- ❌ Doesn't give precise running time
- ❌ Expression can be a little bit complicated

Asymptotic Analysis: Intuition

$0.5n^2 - 0.5n$

$0.5n^2$



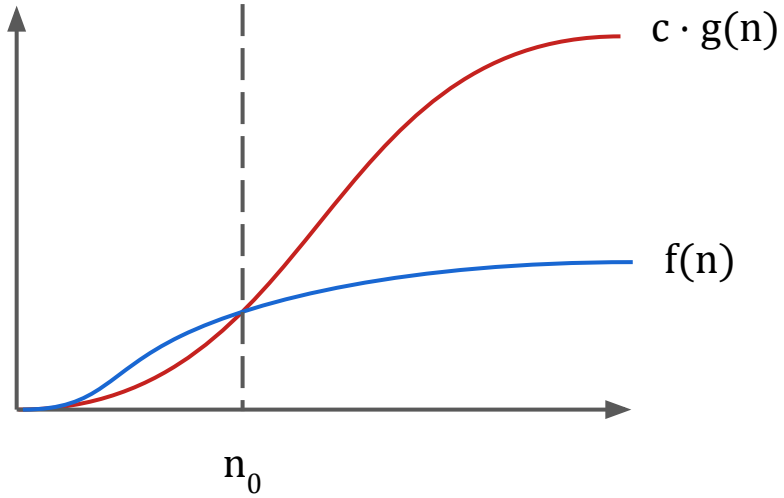


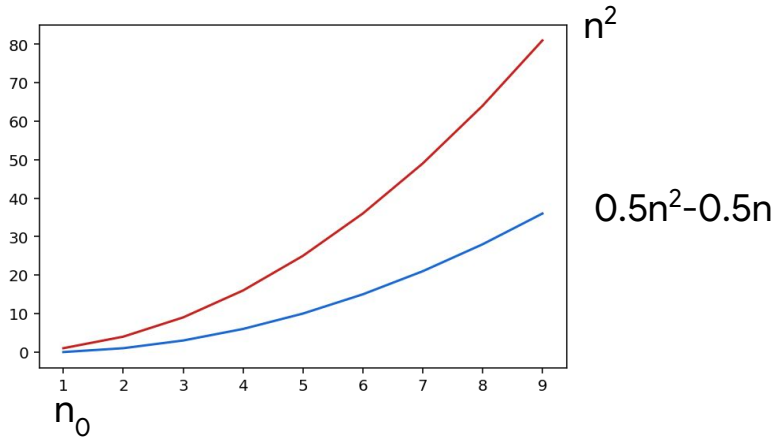
Asymptotics: *"Let's ignore the lower order term & ignore the constants!"*

- ✔ Agnostic to environments
- ✔ Shows dependency on input size
- ✔ Expression can be a little bit complicated
- ❌ Doesn't give precise running time

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"





Example: $0.5n^2-0.5n = O(n^2)$

Running time of An Algorithm

How do we measure the following algorithm's running time?

Running time: $O(n^2)$

```
ExchangeSort(A):
    n = A.length
    For i = 0, …, n-1:
        For j = i,...,n-1:
            If A[i] > A[j]:
                Swap A[i] and A[j]
```

# of times this line is run:
$(n-1)+(n-2)+\ldots+1$
$= 0.5n^2 - 0.5n$ times

May not be run based on the if-else condition

Implement, Run & Time it

- ✔ Captures the Running Time Precisely
- ❌ Depends on environments we run on
- ❌ Depends on the input (size) we run on

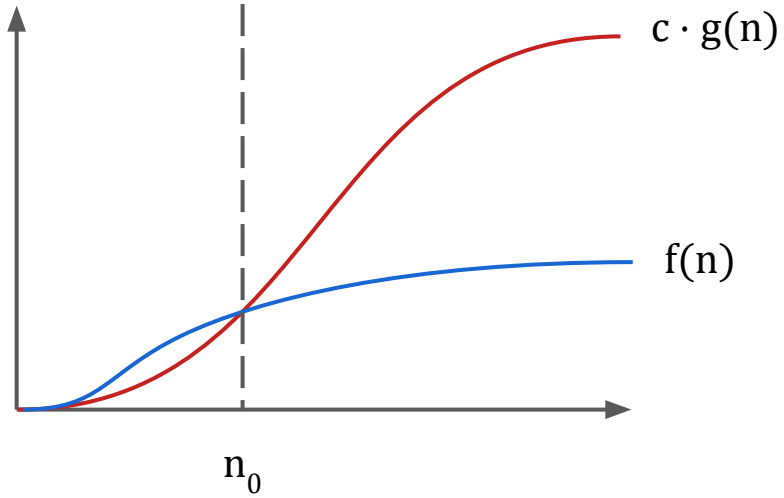
worst case

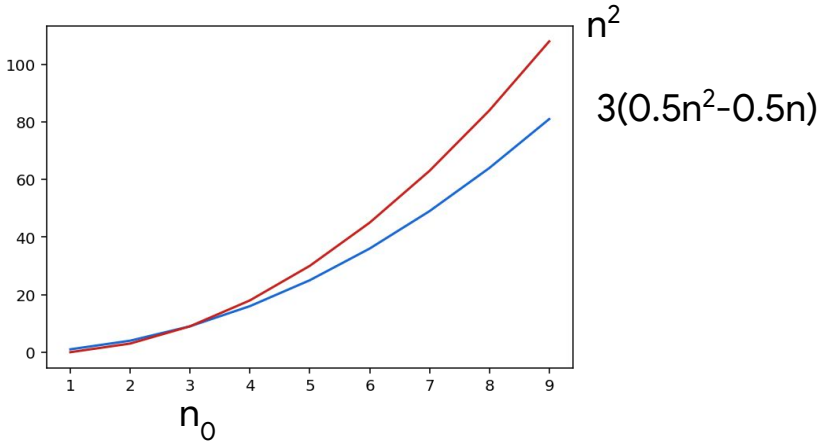
Count # operations in terms of input size

- ✔ Agnostic to environments
- ✔ Shows dependency on input size
- ❌ Doesn't give precise running time
- ❌ Expression can be a little bit complicated

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"





Example: $n^2 = O(0.5n^2-0.5n)$

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"

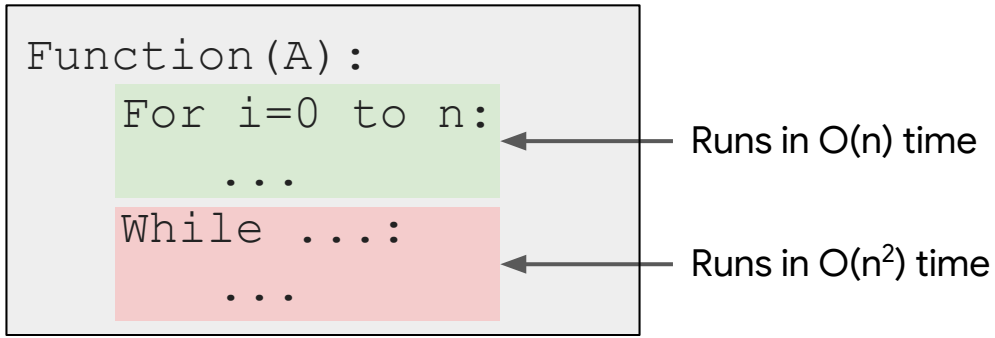
Useful Properties:

1. $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
2. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) + g(n) = O(h(n) + l(n))$
3. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) * g(n) = O(h(n) * l(n))$

Asymptotic Analysis: Algorithmic Example

Total running time
$O(n + n^2)$

$= O(n^2)$



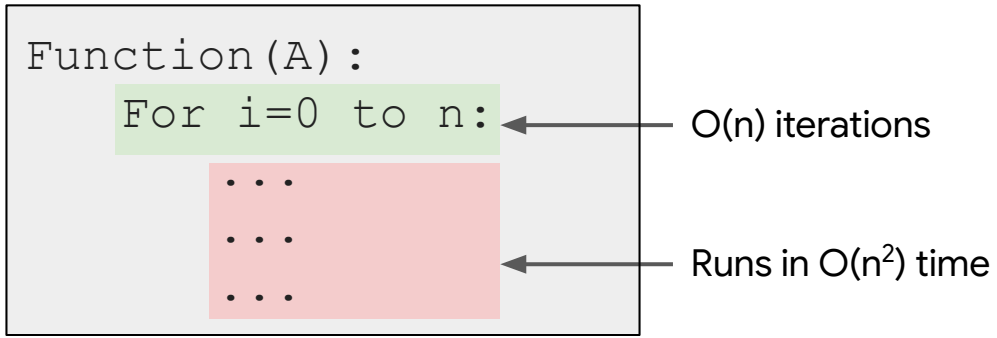
Useful Properties:

1. $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
2. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) + g(n) = O(h(n) + l(n))$
3. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) * g(n) = O(h(n) * l(n))$

Asymptotic Analysis: Algorithmic Example

Total running time
$O(n * n^2)$

$= O(n^3)$



Useful Properties:

1. $f(n) = O(g(n))$ and $g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$
2. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) + g(n) = O(h(n) + l(n))$
3. $f(n) = O(h(n))$ and $g(n) = O(l(n)) \Rightarrow f(n) * g(n) = O(h(n) * l(n))$

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"
$f(n) = \Omega(g(n))$	For some constant $c > 0$ and n_0, $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"

Equivalent to $g(n) = O(f(n))$

Asymptotic Analysis: Formalization

Notations	Definition	Informal description
$f(n) = O(g(n))$	For some constant $c > 0$ and n_0, $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"
$f(n) = \Omega(g(n))$ Equivalent to $g(n) = O(f(n))$	For some constant $c > 0$ and n_0, $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$	"$f(n)$ is less than or equal to $g(n)$ when constants are ignored"
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$	"$f(n)$ is equal to $g(n)$ when constants are ignored"

Examples:

- $0.5n^2 - 0.5n$ vs n^2:
  - $0.5n^2 - 0.5n = O(n^2)$
  - $n^2 = O(0.5n^2 - 0.5n)$
  - $n^2 = \Theta(0.5n^2 - 0.5n)$
- n vs n^2:
  - $n = O(n^2)$
- More generally, for all $a \leq b$
  - $n^a = O(n^b)$
- $\log n$ vs n for any $a > 0$:
  - $\log n = O(n^a)$
- $O(1)$ contains all constants
  - E.g., $35 = O(1)$

Asymptotic: Exercises for Self-Study

Exercises are adapted from CLRS* chapter 3

1. Is $2^{n+1} = O(2^n)$?
2. Is $2^{2n} = O(2^n)$?
3. Is $\log(n^2) = O(\log n)$?
4. Is $(\log n)^2 = O(\log n)$?
5. Prove that, if $f(n) = O(h(n))$ and $g(n) = O(l(n))$, then $f(n) + g(n) = O(h(n) + l(n))$
6. Prove that $\max(f(n), g(n)) = \Theta(f(n) + g(n))$

* Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

Model of Computation: Word RAM

Random Access Machine (RAM)

- Random Access Memory (RAM)
  - Can be viewed as a very long array of *words* each of b bits
    - For SEN-107, not important what b is
  - Can read / write word from / to memory at any location i in time $O(1)$
  - Arithmetic over words takes $O(1)$ time
  - Can allocate array of length n in time $O(n)$

`Allocate arr[5]`

- `arr` can be thought of as "pointer" to address 3
- `arr[i]` is simply `memory[address(arr) + i]`