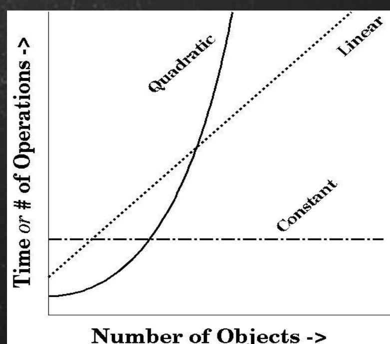
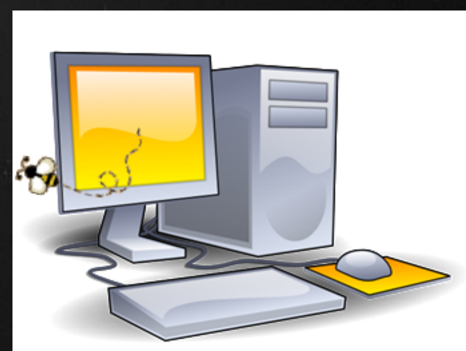
Algorithmic Complexity and Concurrency





Algorithmic Complexity

"Algorithmic Complexity", also called "Running Time" or "Order of Growth", refers to the number of steps a program takes as a function of the size of its inputs. In this class, we will assume the function only has one input, which we will say has length n.

Algorithmic Complexity

Notes on Notation:

Algorithmic complexity is usually expressed in 1 of 2 ways. The first is the way used in lecture - "logarithmic", "linear", etc. The other is called Big-O notation. This is a more mathematical way of expressing running time, and looks more like a function. For example, a "linear" running time can also be expressed as O(n). Similarly, a "logarithmic" running time can be expressed as O(log n).

Algorithmic Complexity

Here is a list of some common running times:

Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$

We will talk about each briefly.

Constant-Time Algorithms - O(1)

A constant-time algorithm is one that takes the same amount of time, regardless of its input. Here are some examples:

- Given two numbers*, report the sum
- Given a list, report the first element
- Given a list of numbers*, report the result of adding the first element to itself 1,000,000 times

Why is the last example still constant time?

*Here, we are referring to numbers of a set maximum size (i.e. 32-bit numbers, 64-bit numbers, etc.)

Logarithmic-Time Algorithm - O(log n)

A logarithmic-time algorithm is one that requires a number of steps proportional to the log(n). In most cases, we use 2 as the base of the log, but it doesn't matter which base because we ignore constants. Because we use the base 2, we can rephrase this in the following way: *every time the size of the input doubles, our algorithm performs one more step*. Examples:

- Binary search
- Searching a tree data structure (we'll see what this is later)

Linear-Time Algorithms - O(n)

A linear-time algorithm is one that takes a number of steps directly proportional to the size of the input. In other words, if the size of the input doubles, the number of steps doubles. Examples:

- Given a list of words, say each item of a list
- Given a list of numbers, add each pair of numbers together (item 1 + item 2, item 3 + item 4, etc.)
- Given a list of numbers, multiply every 3rd number by 2

Again, why is the last algorithm still linear?

Quadratic-Time Algorithms - $O(n^2)$

A quadratic-time algorithm is one takes a number of steps proportional to n^2. That is, if the size of the input doubles, the number of steps quadruples. A typical pattern of quadratic-time algorithms is performing a linear-time operation on each item of the input (n steps per item * n items = n^2 steps). Examples:

- Compare each item of a list against all the other items in the list
- Fill in a n-by-n game board

Cubic-Time Algorithms - $O(n^3)$

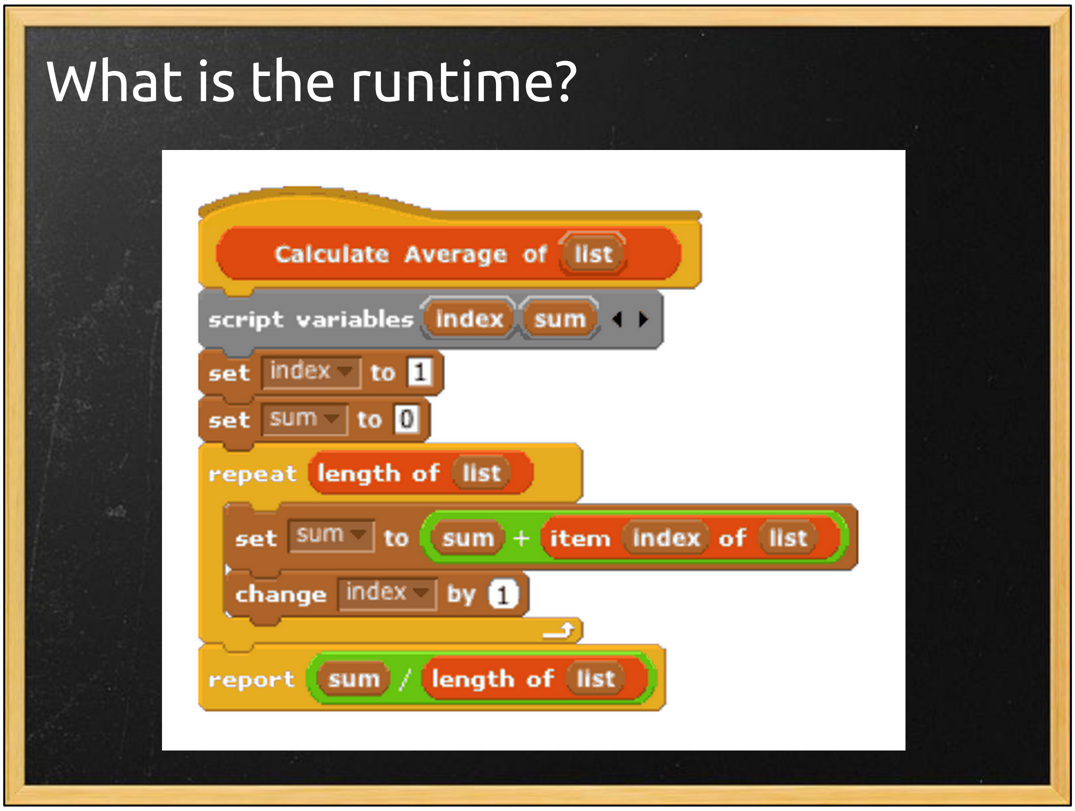
A cubic-time algorithm is one that takes a number of steps proportional to n^3. In other words, if the input doubles, the number of steps is multiplied by 8. Similarly to the quadratic case, this could be the result of applying an n^2 algorithm to n items, or applying a linear algorithm to n^2 items. Examples:

- Fill in a 3D board (or environment)
- For each object in a list, construct an n-by-n bitmap drawing of the object

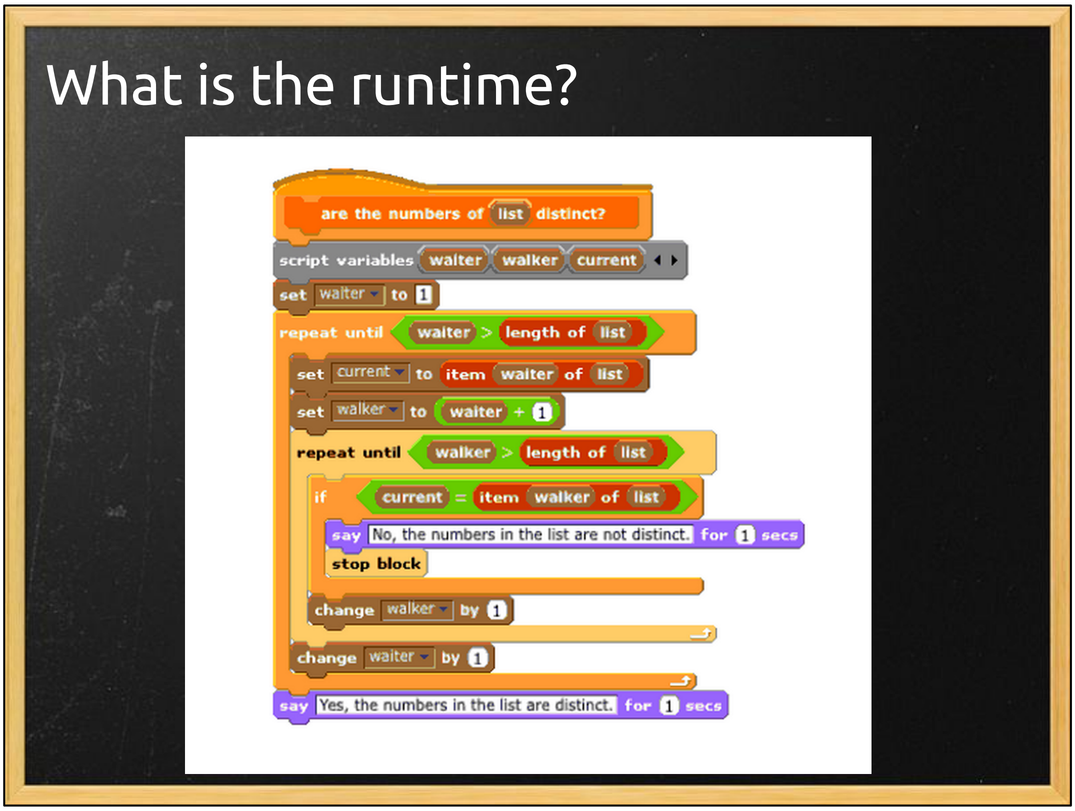
Exponential-Time Algorithms - $O(2^n)$

An exponential-time algorithm is one that takes time proportional to 2^n. In other words, if the size of the input increases by one, the number of steps doubles. Note that logarithms and exponents are inverses of each other. Algorithms in this category are often considered too slow to be practical, especially if the input is typically large. Examples:

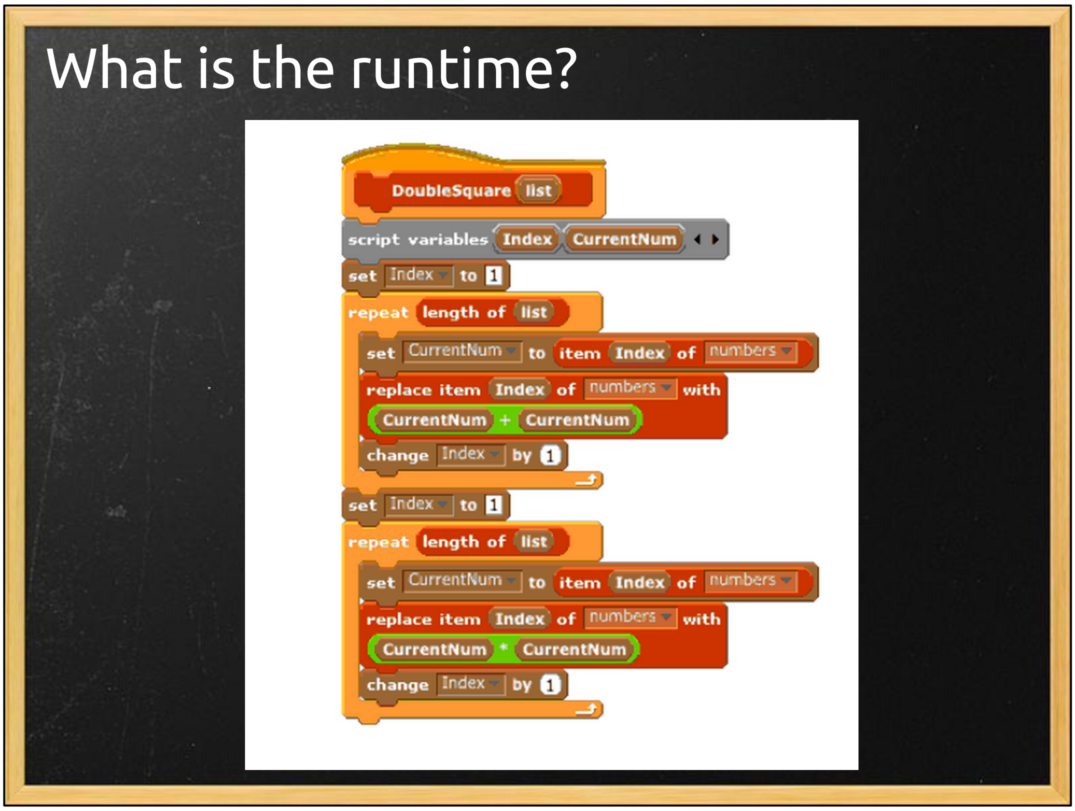
- Given a number n, generate a list of every n-bit binary number


What is the runtime?
Calculate Average of list
script variables index sum
set index to 1
set sum to 0
repeat length of list
set sum to sum + item index of list
change index by 1
report sum / length of list

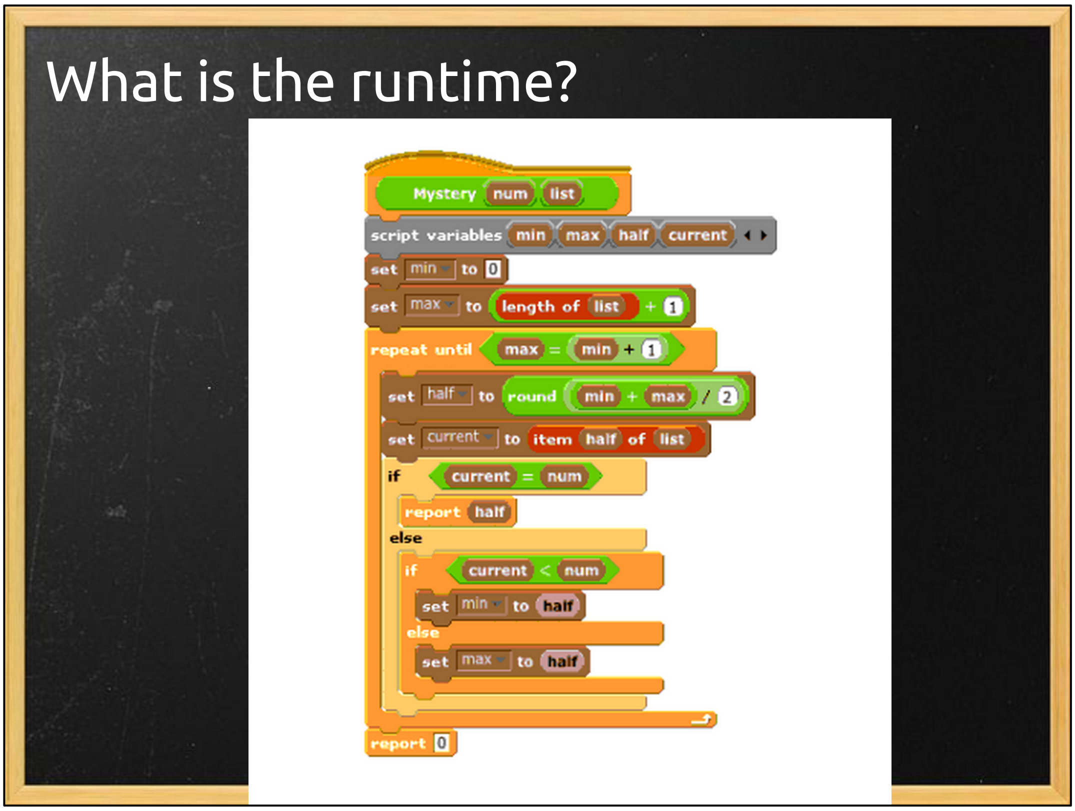
Linear


What is the runtime?
are the numbers of list distinct?
script variables walter walker current
set walter to 1
repeat until walter > length of list
set current to item walter of list
set walker to walter + 1
repeat until walker > length of list
if current = item walker of list
say No, the numbers in the list are not distinct. for 1 secs
stop block
change walker by 1
change walter by 1
say Yes, the numbers in the list are distinct. for 1 secs

quadratic


What is the runtime?
DoubleSquare list
script variables Index CurrentNum
set Index to 1
repeat length of list
set CurrentNum to item Index of numbers
replace item Index of numbers with
CurrentNum + CurrentNum
change Index by 1
set Index to 1
repeat length of list
set CurrentNum to item Index of numbers
replace item Index of numbers with
CurrentNum * CurrentNum
change Index by 1

linear

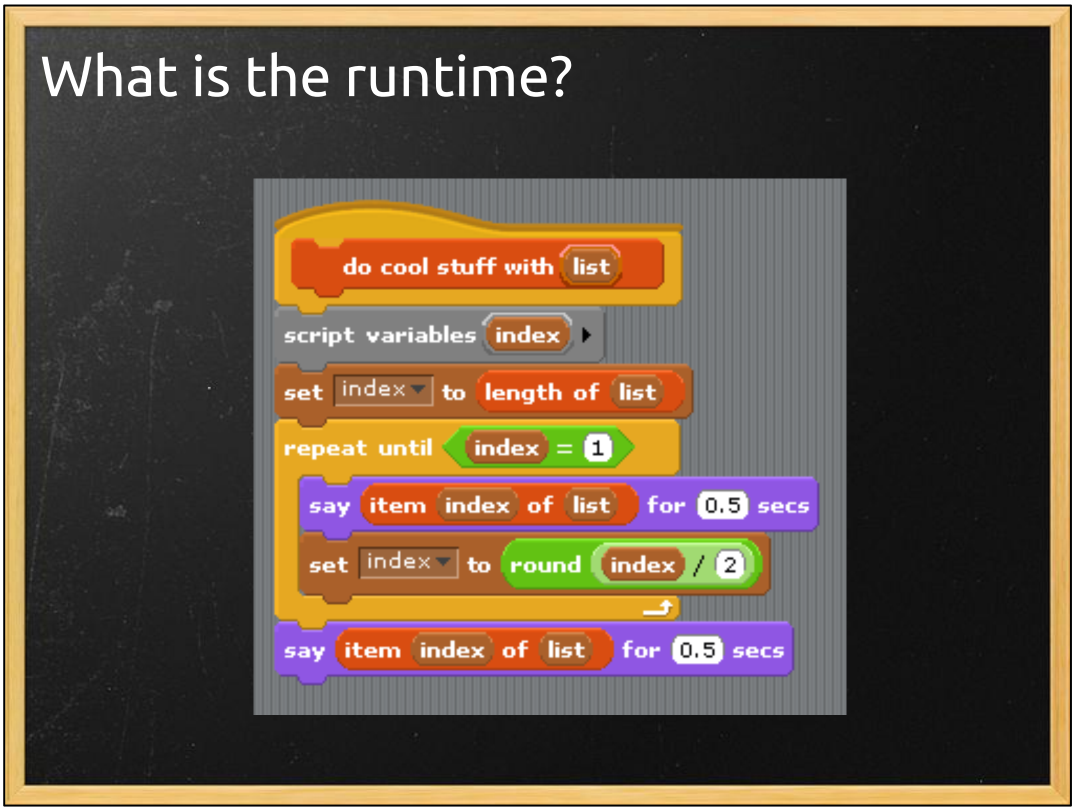


logarithmic

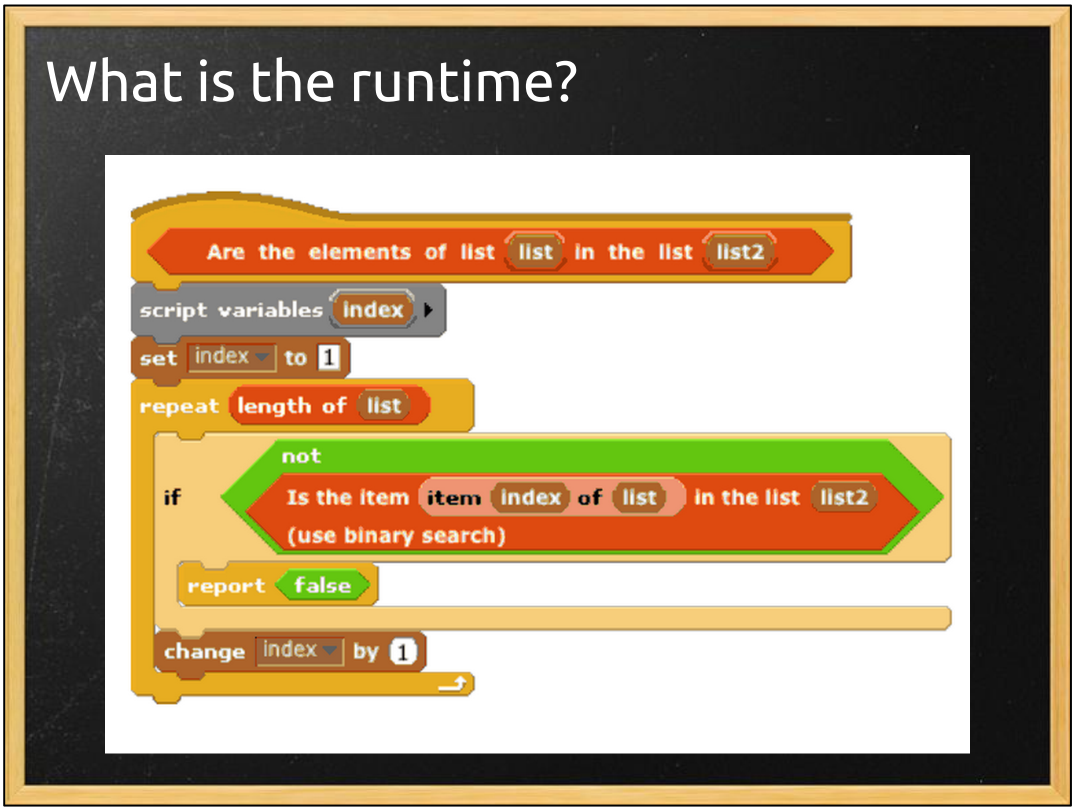
What is the runtime?

Take a look at the code to the right. What is it doing? What is its running time? Hint: it drew the picture below.

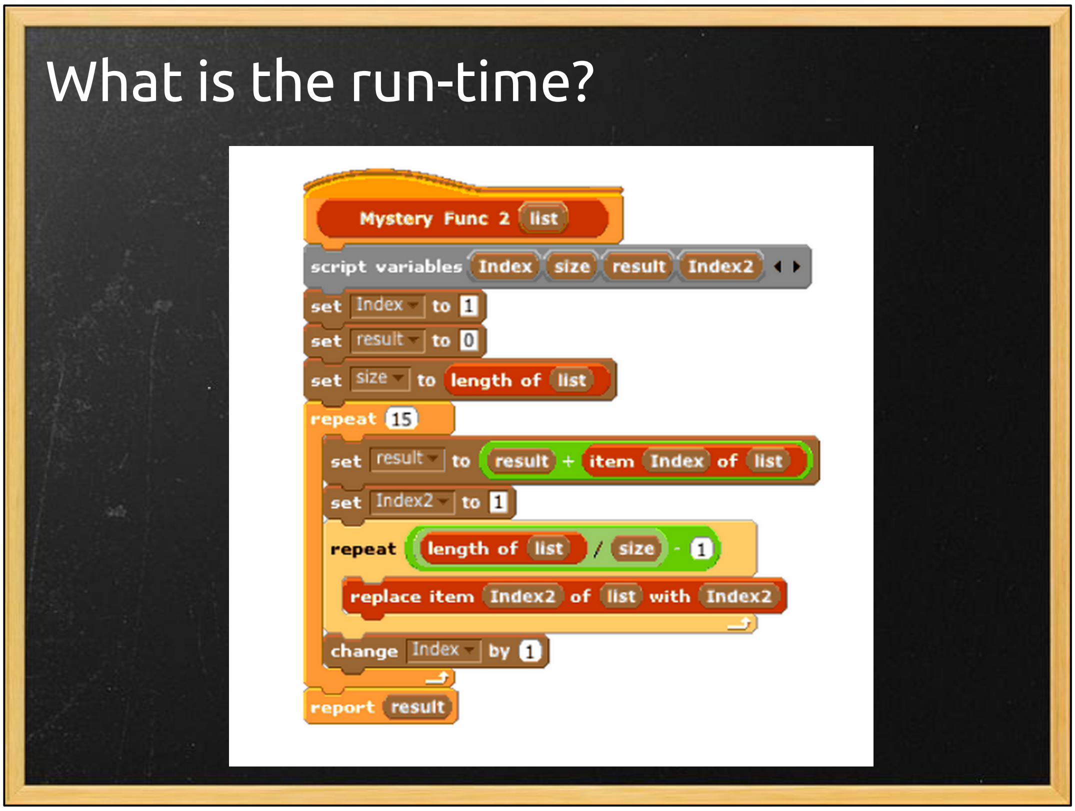
Answer: quadratic



Answer: Logarithmic

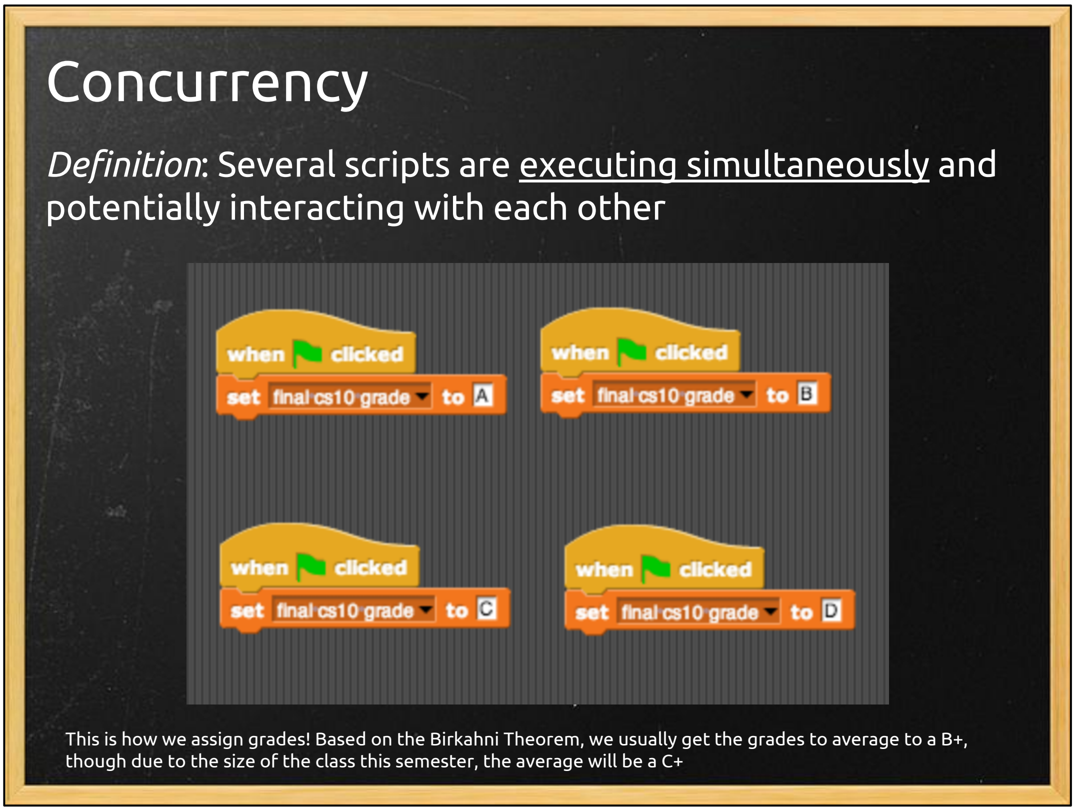


linear * logarithmic


What is the run-time?
Mystery Func 2 list
script variables Index size result Index2
set Index to 1
set result to 0
set size to length of list
repeat 15
set result to result + item Index of list
set Index2 to 1
repeat length of list / size - 1
replace item Index2 of list with Index2
change Index by 1
report result

constant

Concurrency



another definition: processes each take turns working toward accomplishing their goals.

Race Condition

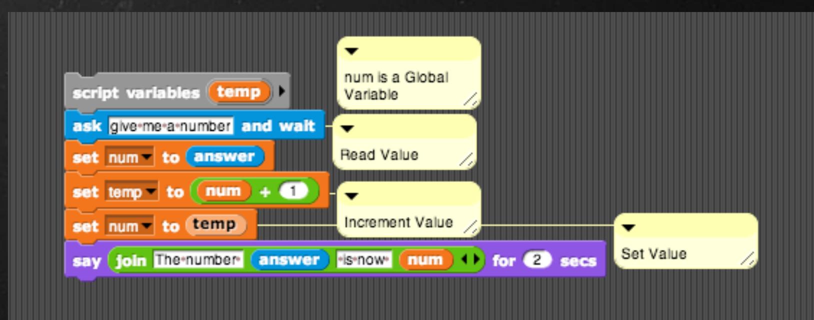
Concurrency Issue

Race Condition

Definition: when events of a program don't happen in the order that the programmer intended.

Function Definitions

- read value: reads in a value from user input
- increments value: increments the value, but does not set it
- sets value: sets the value to the incremented version of it.



Serial - Example

Program 1	Program 2	Global Integer Value
		0

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0
sets value		1

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0
sets value		1
	read value	1

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0
sets value		1
	read value	1
	increments value	1

Serial - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0
sets value		1
	read value	1
	increments value	1
	sets value	2

Serial - Example

This is the expected output. We're good here!

Program 1	Program 2	Global Integer Value
		0
read value		0
increments value		0
sets value		1
	read value	1
	increments value	1
	sets value	2

What if we interleaved the commands?

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0
increments value		0

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0
increments value		0
	increments value	0

Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0
increments value		0
	increments value	0
sets value		1

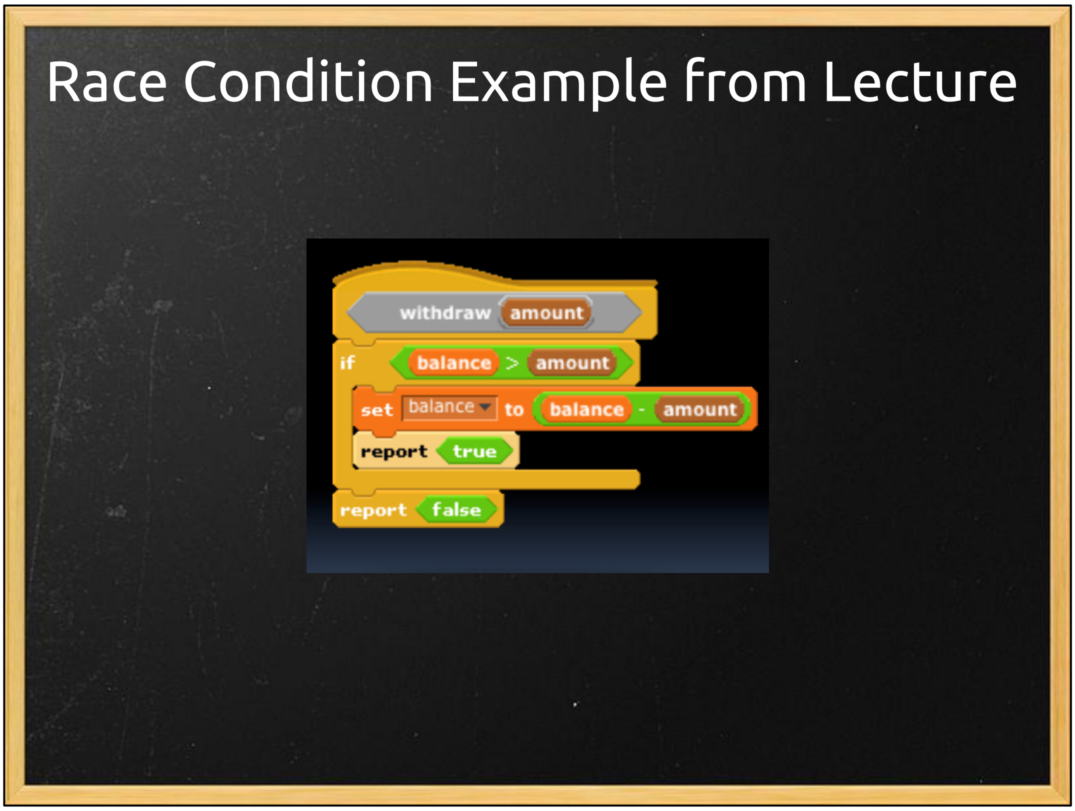
Race Condition - Example

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0
increments value		0
	increments value	0
sets value		1
	sets value	1

Race Condition - Example

This is the NOT the expected output. The integer is only 1!

Program 1	Program 2	Global Integer Value
		0
read value		0
	read value	0
increments value		0
	increments value	0
sets value		1
	sets value	1



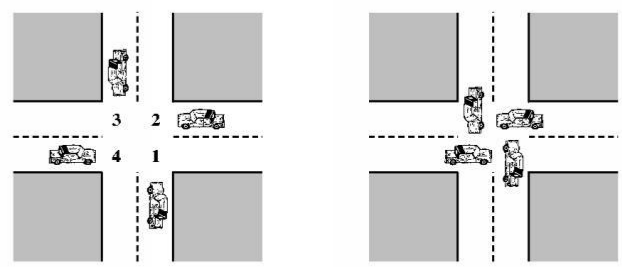
Go over, or have students give an example of how this can go wrong.

Deadlock

Concurrency Issue

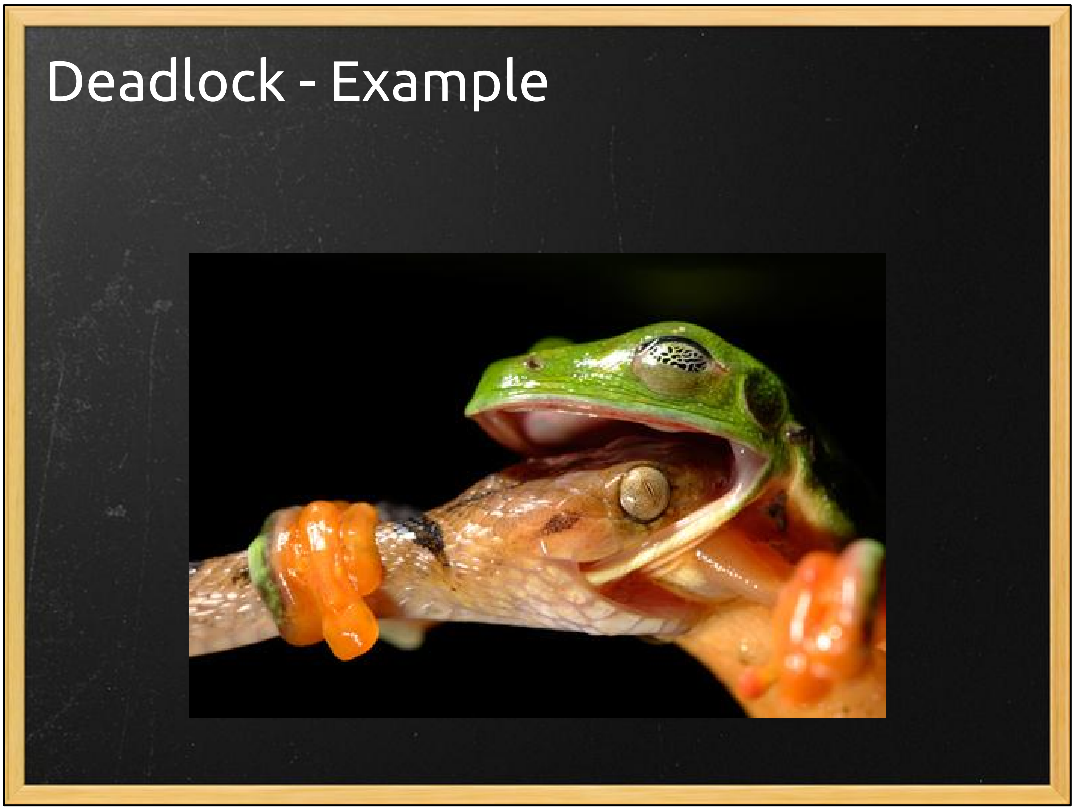
Deadlock

Definition: a situation in which two or more competing actions are each waiting for the other(s) to finish, and thus no one ever finishes.



(a) Deadlock possible (b) Deadlock

Bring up lecture example with pencil and ruler



According to photographer, locked like this for 3 hours. he didn’t stick around to see who won…
Article can be found here

Concurrency Practice Problems

Question 13: Your faaaaace… *(5 pts)*

You want to draw a face, so you write this serial script that produces the "winking" face right beside it:

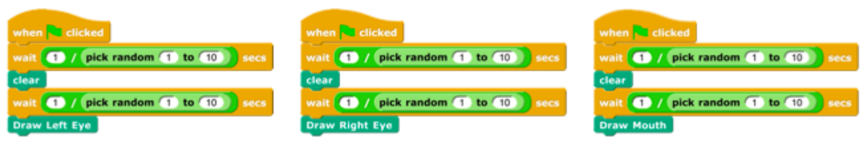
But then you want to simulate what it would be like to parallelize the code and run it on three separate "cores", so you change the serial script above into the following parallel scripts, which all run at the same time:

Draw all the faces that could result from running this new parallel code. You may not need all the blanks.

Question 13: Your faaaaace… *(5 pts)*

You want to draw a face, so you write this serial script that produces the "winking" face right beside it:

But then you want to simulate what it would be like to parallelize the code and run it on three separate "cores", so you change the serial script above into the following parallel scripts, which all run at the same time:



Question 12/13: *Draw all the faces* that could result from running this new parallel code. You may not need all blanks. These result from interlacing 3 LeftEye/RightEye/Mouth Clear (LC,RC,MC), LeftEye(L), RightEye(R), & Mouth(M).

RC,R,MC,M,LC,L	LC,L,MC,M,RC,R	LC,L,RC,R,MC,M	RC,R,MC,LC,M,L	LC,L,MC,RC,M,R	MC,M,LC,RC,L,R	RC,LC,MC,R,L,M	

SID: ________________

Question 12: ***Dining Philosophers*** (5 pts)

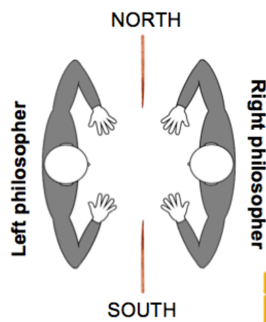
Two philosophers (left and right) are having dinner, sitting across from each other. There is a NORTH and a SOUTH chopstick on the table. Each philosopher continually looks down to see if a chopstick is on the table, and tries to grab it; if both are ever grabbed by one person, that person eats, updates HISTORY (a record of what happened) and puts the chopsticks down. Ten seconds after the green flag is clicked, what could HISTORY be?
(all the boxes are not necessarily needed)

```
when green flag clicked
set NORTH chopstick to table
set SOUTH chopstick to table
set HISTORY to Started...
broadcast Eat!
```


```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <(NORTH chopstick) = table>
set NORTH chopstick to left
```

```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <(SOUTH chopstick) = table>
set SOUTH chopstick to left
```

```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <<(NORTH chopstick) = left> and <(SOUTH chopstick) = left>>
set HISTORY to (join (HISTORY) left ate...)
set NORTH chopstick to table
set SOUTH chopstick to table
```




```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <(SOUTH chopstick) = table>
set SOUTH chopstick to right
```

```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <(NORTH chopstick) = table>
set NORTH chopstick to right
```

```
when I receive Eat!
wait (1 / (pick random 1 to 10)) secs
wait until <<(NORTH chopstick) = right> and <(SOUTH chopstick) = right>>
set HISTORY to (join (HISTORY) right ate...)
set NORTH chopstick to table
set SOUTH chopstick to table
```

SID: _______________

Question 12: *Dining Philosophers* (5 pts)

Two philosophers (left and right) are having dinner, sitting across from each other. There is a NORTH and a SOUTH chopstick on the table. Each philosopher continually looks down to see if a chopstick is on the table, and tries to grab it; if both are ever grabbed by one person, that person eats, updates HISTORY (a record of what happened) and puts the chopsticks down. Ten seconds after the green flag is clicked, what could HISTORY be?
(all the boxes are not necessarily needed)

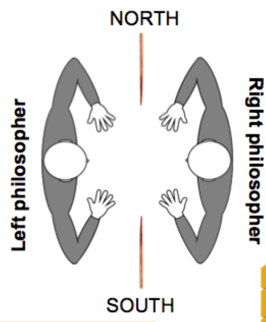
```
when green flag clicked
set NORTH chopstick to table
set SOUTH chopstick to table
set HISTORY to Started...
broadcast Eat!
```

Started… Left ate… Right ate…	Started… Right ate… Left ate…	Started…		

```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <(NORTH chopstick) = table>
set NORTH chopstick to left
```

```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <(SOUTH chopstick) = table>
set SOUTH chopstick to left
```

```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <<(NORTH chopstick) = left> and <(SOUTH chopstick) = left>>
set HISTORY to (join HISTORY left-ate...)
set NORTH chopstick to table
set SOUTH chopstick to table
```




```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <(SOUTH chopstick) = table>
set SOUTH chopstick to right
```

```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <(NORTH chopstick) = table>
set NORTH chopstick to right
```

```
when I receive Eat!
wait (1 / pick random 1 to 10) secs
wait until <<(NORTH chopstick) = right> and <(SOUTH chopstick) = right>>
set HISTORY to (join HISTORY right-ate...)
set NORTH chopstick to table
set SOUTH chopstick to table
```