

# Algorithms

## Part 2: Time Complexity

Notes for CSC 100 - The Beauty and Joy of Computing  
The University of North Carolina at Greensboro

---

---

---

---

---

---

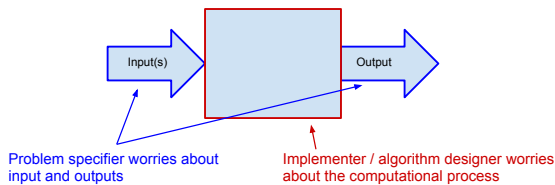
---

---

## Last Time We Saw...

Problems are defined by input/output relation, with no reference to how they are solved (*focus is on what*)

Algorithms are well-defined computational procedures that solve problems (*focus is on how*)



---

---

---

---

---

---

---

---

## In BYOB

### Problem Focus

GCD of  $px$  and  $py$

With a well-chosen name, that may define the problem well enough for the user!

### Algorithm Focus

```
GCD of px and py
script variables xCounter
set xCounter to px
if py < px
  set xCounter to py
repeat until xCounter divides evenly into px and py
  change xCounter by 1
input xCounter
```

This is an over-simplification: Sometimes the user wants to know some properties of the block implementation.

Question: What kinds of properties?

---

---

---

---

---

---

---

---

## Algorithm Characteristics

- Does the algorithm work correctly (does it solve the problem)?
- Is the answer provided precise?
- How confident are you in the correctness of the algorithm and implementation (simpler algorithms are easier to verify)?
- How much memory does the algorithm require?
- How fast is the algorithm?

---

---

---

---

---

---

---

---

## Algorithm Characteristics

- Does the algorithm work correctly (does it solve the problem)?
- Is the answer provided precise?
- How confident are you in the correctness of the algorithm and implementation (simpler algorithms are easier to verify)?
- How much memory does the algorithm require?
- How fast is the algorithm?

Assume no problems with correctness or precision for now.

Memory is a problem for some algorithms, but not as common a limiting factor as...

Time is usually the most interesting and limiting characteristic, whether talking about running a big computation for a week, or calculating a new graphics frame in 1/30 of a second.

---

---

---

---

---

---

---

---

## What is "time" for an Algorithm?

Time is time, right?

But...

- Does time depend on things other than the algorithm?
- If run many times (on the same input), is time always the same?
- If QuickSort runs in 20 seconds on my old IBM PC, and SelectionSort runs in 0.5 seconds on my current computer, is SelectionSort a faster algorithm?
- Can we give clock time without implementing the algorithm?



---

---

---

---

---

---

---

---

## Correcting for vagueness of timing

Wall-clock times depend on:

- Speed of computer that it's run on
- What else is happening on the computer
- ... and a few other things we'll address later

But... these are not differences in algorithms!

Solution: Algorithms are sequences of steps, so count steps!

Question: What's a step?

---

---

---

---

---






---

---

---

## BYOB blocks and "steps"

Which of these should not be treated as "one step"?

- a) 
- b) 
- c) 
- d) 
- e) 

---

---

---

---

---

---

---

---

## Experimenting with timing BYOB scripts

Timer is available to help test things out

- Reset timer to start it at zero



- Save current timer value into a variable for "lap timer"



- Watch variable shows limited precision - for more use "say"



- Tip: surround only what you're interested in timing with reset/set blocks (not initializations)



---

---

---

---

---

---

---

---

## Constant time

We say a script (or part of a script or block definition) takes *constant time* if it is a constant (usually small) number of basic steps, regardless of input.

*Question:* Are all of these constant time?




---

---

---

---

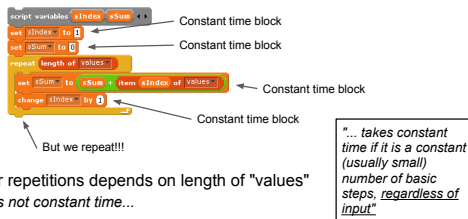
---

---

---

---

## What about loops?



The number repetitions depends on length of "values"

- So this is not constant time...

Constant time operations, repeated "length of input" times is *linear time*

Mathematically: Constant time loop body is time "c"  
Repeated "n" times where n is length of list  
Total time is then c\*n (that's a linear function!)

---

---

---

---

---

---

---

---

## General list index iterator pattern

On previous slide:

- Time was expressed as a function of input size
- Could write time as  $T(n) = c \cdot n$

Very important "Big Idea"!!!

In general:



We know how many times it repeats, and all basic blocks are constant time except perhaps our "do something..." block

- In general, if time for "do something..." block is  $T(n)$ , then time for complete script with loop is  $n \cdot T(n)$
- If "do something" is constant time, total time is  $c \cdot n$  (linear)
- If "do something" is linear time, total time is  $c \cdot n^2$  (quadratic)

---

---

---

---

---

---

---

---

## Two challenges

```
What's the time complexity?  
max pos in pList  
script variables answer index **  
set answer to 0  
set index to 0  
repeat length of pList 1  
if  
  item index of pList >  
  item answer of pList  
  set answer to index  
change index by 1  
report answer
```

```
What's the time complexity?  
sort pList  
script variables sIndex maxPos **  
set sIndex to length of pList  
set sIndex to length of pList - 1  
repeat length of pList 1  
set maxPos to max pos in first sIndex of pList  
swap positions maxPos and sIndex of pList  
change sIndex by 1
```

---

---

---

---

---

---

---

---

## Another challenge

The following predicate tests whether a list has any duplicates:

```
pList has duplicates  
script variables sIndex1 sIndex2 **  
set sIndex1 to 0  
repeat length of pList 1  
set sIndex2 to sIndex1 + 1  
repeat length of pList - sIndex1  
if  
  item sIndex1 of pList =  
  item sIndex2 of pList  
  report true  
change sIndex2 by 1  
change sIndex1 by 1  
report false
```

**Question:** What's the time complexity?

---

---

---

---

---

---

---

---

## Predicting Program Times - Linear

Basic idea: Given time complexity and sample time(s) can estimate time on larger inputs

Linear time: When input size doubles, time doubles  
When input size triples, time triples  
When input size goes up by a factor of 10, so does time

Example: A linear time algorithm runs in 10 sec on input size 10,000  
How long to run on input size 1,000,000?

Answer:  $1,000,000 / 10,000 = 100$  times larger input  
Therefore 100 times larger time, or  $10 * 100 = 1,000$  sec  
Or  $1,000 / 60 = 16.667$  minutes

---

---

---

---

---

---

---

---

## Predicting Program Times - Quadratic

Basic idea: Given time complexity and sample time(s) can estimate time on larger inputs

Quadratic time: When input size doubles (2x), time quadruples (4x)  
Input size goes up by a factor of 10, time goes up  $10^2=100$  times  
Input size goes up  $k$  times, time goes up  $k^2$  times

Example: A quadratic time algorithm runs in 10 sec on input size 10,000  
How long to run on input size 1,000,000?

Answer:  $1,000,000 / 10,000 = 100$  times larger input  
Therefore  $100^2 = 10,000$  times larger time, or 100,000 sec  
Or  $100,000 / 60 = 1666.7$  minutes (or 27.8 hours)

---

---

---

---

---

---

---

---

---

---

## Predicting Program Times - Your Turn

Joe and Mary have created programs to analyze crime statistics, where the input is some data on each resident of a town

- Joe's algorithm is quadratic time
- Mary's algorithm is linear time
- Both algorithms take about 1 minute for a town of size 1000

Both would like to sell their program to the City of Greensboro (population 275,000)

**Problem:** Estimate how long each program would take to run for Greensboro

---

---

---

---

---

---

---

---

---

---

## Faster than linear list operations

Think about how you find a word in a dictionary:

- From the Webster's web site: "*Webster's Third New International Dictionary, Unabridged*, together with its 1993 Addenda Section, includes some *470,000 entries*."
- If you checked every possible entry to see if it was the one you wanted, it would take way too long.
- How is a dictionary organized in order to make this easier?

**Challenge:** Describe precisely how to quickly look up a word.

---

---

---

---

---

---

---

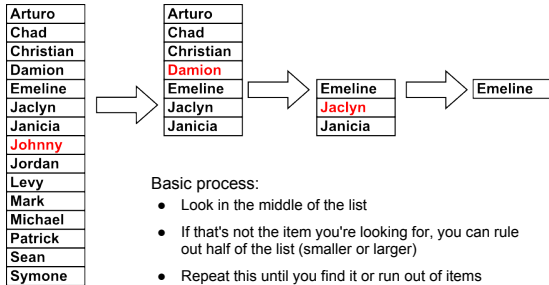
---

---

---

## Illustration for a list of students

*Problem:* Where's Emeline? (Like "Where's Waldo?" but without the goofy hat)



Basic process:

- Look in the middle of the list
- If that's not the item you're looking for, you can rule out half of the list (smaller or larger)
- Repeat this until you find it or run out of items

---

---

---

---

---

---

---

---

---

---

## How long does this take?

At beginning: Could be any of  $n$  items  
 After 1 step: Could be any of  $n/2$  items  
 After 2 steps: Could be any of  $n/4$  items  
 After 3 steps: Could be any of  $n/8$  items  
 ...  
 After  $k$  steps: Could be any of  $n/2^k$  items

To get to one item, need  $n=2^k$  - so  $k = \log_2 n$

This is called *logarithmic time*, and gives very fast algorithms!

$n$	$\log_2 n$
1000	10
1,000,000	20
1,000,000,000	30

← So can find one item out of a billion in just 30 comparisons!!!

*While you're not responsible for knowing or being able to do this derivation, you do need to know about binary search and logarithmic time.*

*This analysis doesn't require anything beyond high school algebra to understand - so try to understand it!*

---

---

---

---

---

---

---

---

---

---

## Something worse...

*Problem:* I have 60 items, each with a value, and want to find a subset with total value as close to some target  $T$  as possible.

(The Price is Right on steroids...)



Algorithm: List all possible subsets of items  
 Add up total value of each subset  
 Find which one is closest

Question: If I have  $n$  items, how many subsets of  $n$  items are there?

Answer: There are  $2^n$  subsets - this is *exponential time* (and very bad!)

---

---

---

---

---

---

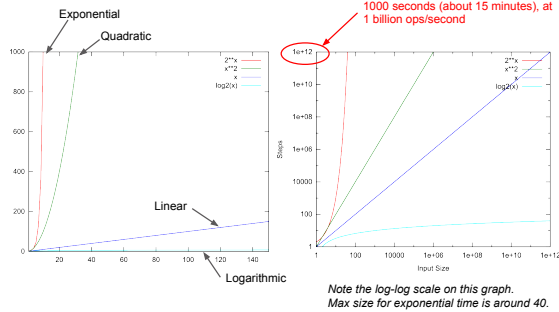
---

---

---

---

## Graphically comparing time complexities




---

---

---

---

---

---

---

---

---

---

## Comparing with numbers

Different time complexities, by the numbers...

	Time in seconds at 1 billion ops/sec		Largest problem in 1 min at 1 billion ops/sec
	$n=1,000$	$n=1,000,000$	
$\log_2 n$	0.00000001	0.00000002	Huge*
$n$	0.000001	0.001	60,000,000,000
$n^2$	0.001	1000	244,949
$2^n$	$10^{292}$	$10^{301029}$	35

\* Huge means a problem far larger than the number of atoms in the universe

There is a lot more to this than what we have covered - but this gives a pretty accurate picture of basic algorithm time complexity!

---

---

---

---

---

---

---

---

---

---

## Summary

- Algorithm "time complexity" is in basic steps
- Common complexities, from fastest to slowest are logarithmic, linear, quadratic, and exponential
  - A simple loop with constant time operations repeated is linear time
  - A loop containing a linear time loop is quadratic
  - A loop halving the problem size every iteration is logarithmic time
  - A program considering all subsets is exponential time
- Speed depends on algorithm time complexity
  - Logarithmic time is fantastic
  - Linear time is very good
  - Quadratic time is OK
  - Exponential time is awful
- Given time complexity and one actual time, can estimate time for larger inputs

---

---

---

---

---

---

---

---

---

---