# <section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><section-header><text>

In	+i	tion
	Lai	CION

• Are the following operations "fast" or "slow"?

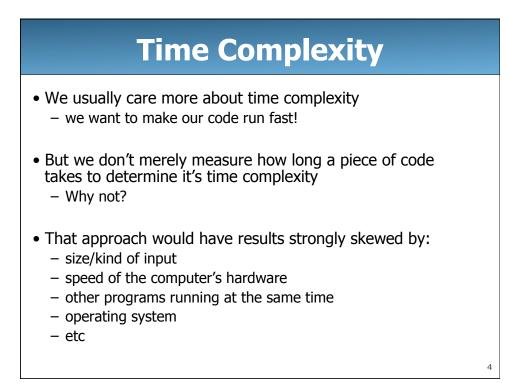
array			
behavior	fast/slow		
add at front	slow		
add at back	fast		
get at index	fast		
resizing	slow		
binary search	(pretty) fast		

### linked list

behavior	fast/slow
add at front	fast
add at back	slow
get at index	slow
resizing	N/A (fast!)
binary search	(really) slow

# Complexity

- "Complexity" is a word that has a special meaning in computer science
- **complexity**: the amount of computational resources a block of code requires in order to run
- main computational resources:
  - **time**: how long the code takes to execute
  - **space**: how much computer memory the code consumes
- Often, one of these resources can be traded for the other:
  - e.g.: we can make some code use less memory if we don't
    - mind that it will need more time to finish (and vice-versa)



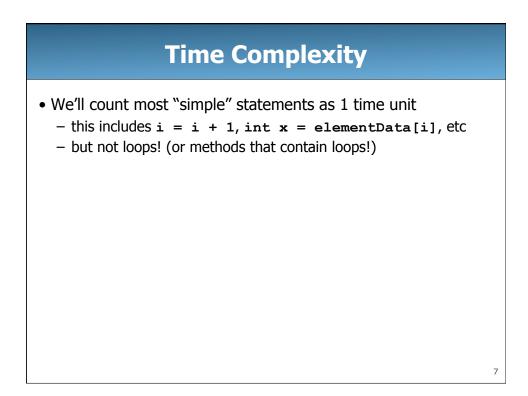
# **Time Complexity**

- Instead, we care about the growth rate as the input size increase
- First, we have to be able to measure the input size
  - the number of names to sort
  - the number of nodes in a linked list
  - the number of students in the IPL queue
- We usually call the input size "n"
- What happens if we double the input size  $(n \rightarrow 2n)$ ?
  - Will the running time double? quadruple? take forever?

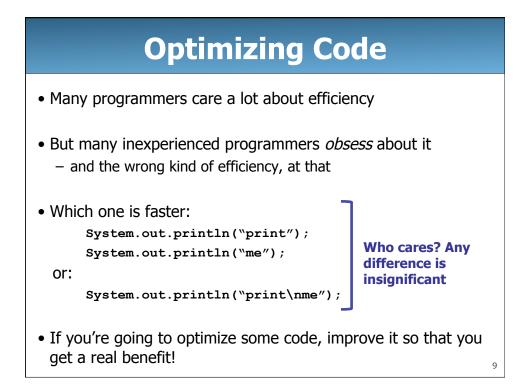
## **Time Complexity**

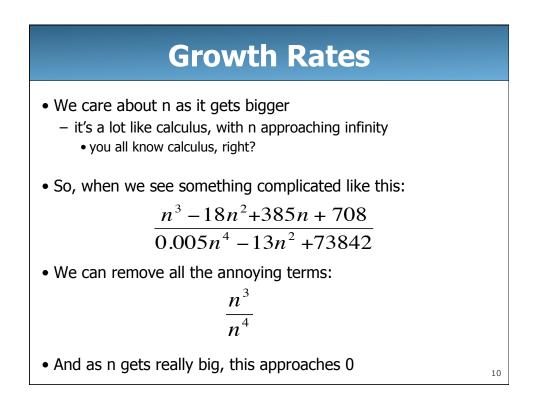
- We can learn about this growth rate in two ways:
  - by examining code
  - by running the same code over different input sizes
- Measuring the growth rate by is one of the few places where computer science is like the other sciences
  - here, we actually collect data
- But this data can be misleading
  - modern computers are very complex
  - some features (code optimizations) interfere with our data

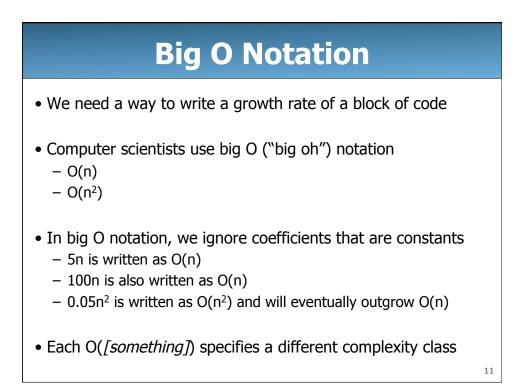
5



Time Complexity				
• Exan	nples: int $x = 4 * 10 / 3 + 2 - 10 * 42;$			
100	<pre>for (int i = 0; i &lt; 100; i++) {     x += i; }</pre>	n <sup>2</sup> + 100 + 1		
n²	<pre>for (int i = 0; i &lt; n; i++) {     for (int j = 0; j &lt; n; j++) {         x += i + j;     } }</pre>			
		-	8	







<b>Complexity Classes</b>			
<u>Complexity</u> <u>Class</u>	<u>Name</u>	Example	
0(1)	constant time	accessing an array element	
O(log n)	logarithmic time	binary search on an array	
O(n)	linear time	scanning all elements of an array	
O(n log n)	log-linear time	binary search on a linked list and good sorting algorithms	
O(n²)	quadratic time	poor sorting algorithms (like inserting n items into SortedIntList)	
O(n <sup>3</sup> )	cubic time	(example later today)	
O(2 <sup>n</sup> )	exponential time	Really hard problems. These grow so fast that they're impractical	

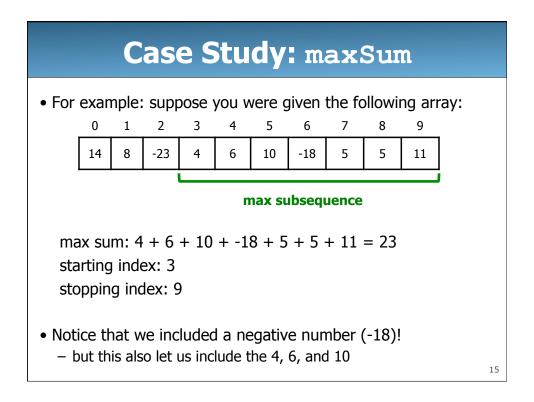
<b>Examples of Each Complexity</b>	7
Class's Growth Rate	

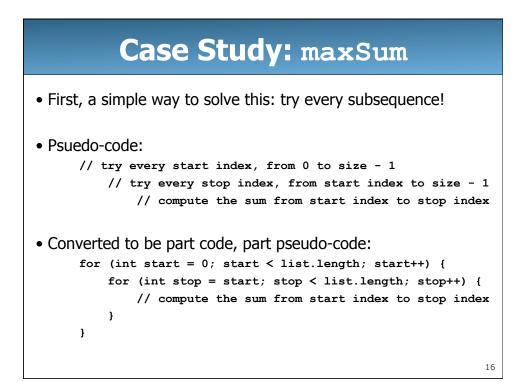
• Assume that all complexity classes can process an input of size 100 in 100ms

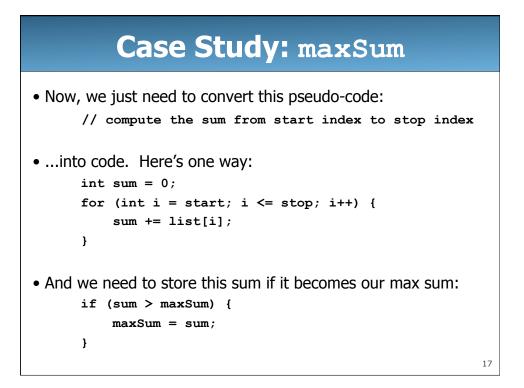
<u>Input</u> <u>Size (n)</u>	<u>O(1)</u>	<u>O(log n)</u>	<u>O(n)</u>	<u>O(n log n)</u>	<u>O(n²)</u>	<u>O(n³)</u>	<u>O(2<sup>n</sup>)</u>
100	100ms	100ms	100ms	100ms	100ms	100ms	100ms
200	100ms	115ms	200ms	240ms	400ms	800ms	32.7 sec
400	100ms	130ms	400ms	550ms	1.6 sec	6.4 sec	12.4 days
800	100ms	145ms	800ms	1.2 sec	6.4 sec	51.2 sec	36.5 million years
1600	100ms	160ms	1.6 sec	2.7 sec	25.6 sec	6 min 49.6 sec	4.21 * 10 <sup>24</sup> years
3200	100ms	175ms	3.2 sec	6 sec	1 min 42.4 sec	54 min 36 sec	5.6 * 10 <sup>61</sup> years

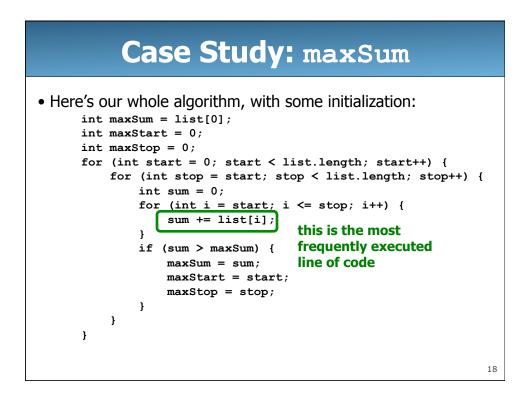
### Case Study: maxSum

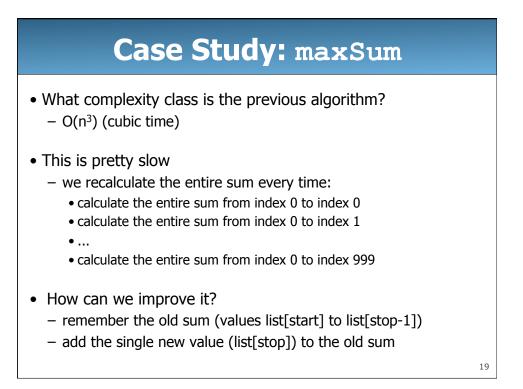
- Given an array of ints, find the subsequence with the maximum sum
- Additional information:
  - values in the array can be negative, positive, or zero
  - the subsequence must be contiguous (can't skip elements)
  - you must compute:
    - the value of the sum of this subsequence
    - the starting index (inclusive) of this subsequence
    - the stopping index (inclusive) of this subsequence
- This has been used as a Microsoft interview question!

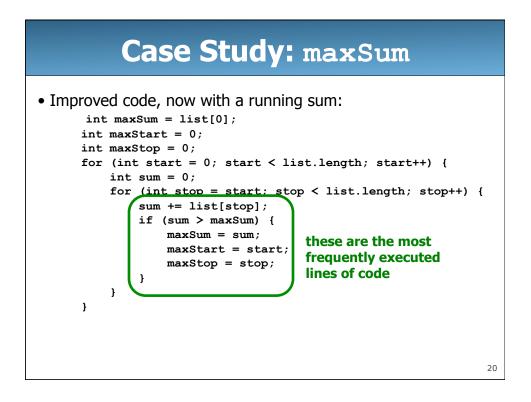


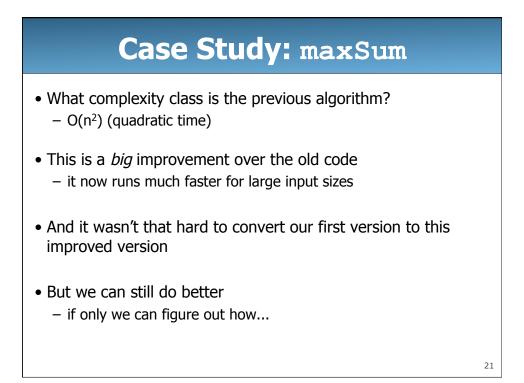


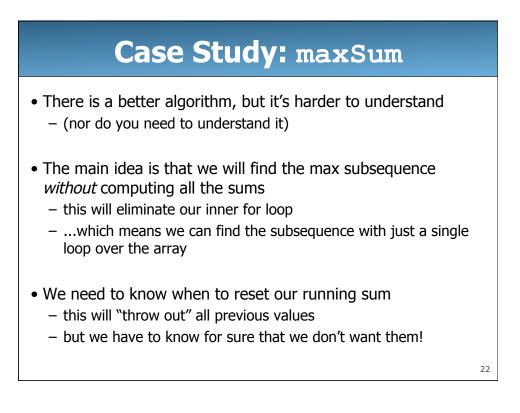


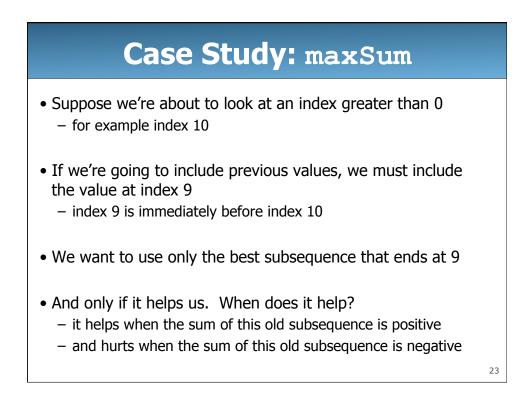


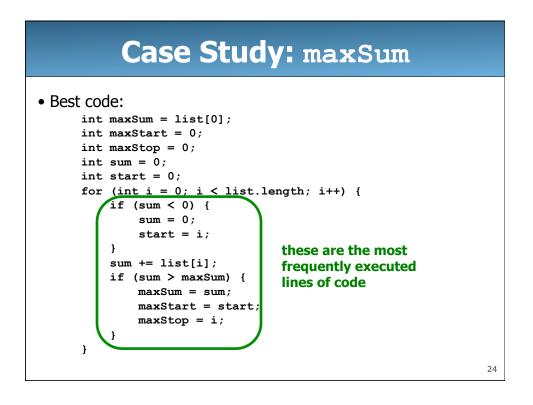






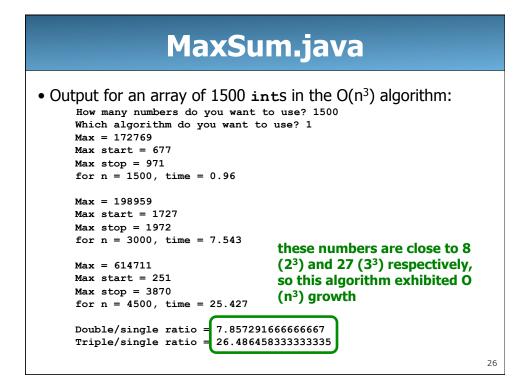






### Case Study: maxSum

- What complexity class is our best algorithm?
   O(n) (linear time)
- This is again a big improvement over both other versions
- But let's not just take my word for it
- Let's conduct an experiment (in MaxSum.java -- available on the website)
  - we'll give an array of ints of some size to each algorithm
  - ...and then give the algorithm an array of twice that size
  - ...and then give the algorithm an array of triple that size
  - ...and see how long it takes



### MaxSum.java

