CS 62, Lecture 6
Complexity

Guest Lecturer: Nate Derbinsky

Research Talk @ 4:00PM, Tuesday
Shanahan 1430

Disney Research

Today

Reading: JS 5.1

Agenda

1. This week's programming assignment: N-Grams!
2. Complexity & Big-O notation
3. Complexity analysis of ArrayList
   - This week's lab!
4. Induction: tool for complexity analysis

This Week's Assignment: N-Grams

Continuous sequence of *n* items from a given sequence:

"blast of a century leaves thousands holmesless" –Mufti (1987)

- **Unigram** (1-gram): blast, of, a, century, leaves, thousands, holmesless
- **Bigram** (2-gram): blast of, of a, a century, century leaves, leaves thousands, thousands holmesless
- **Trigram** (3-gram): blast of a, of a century, a century leaves, century leaves thousands, leaves thousands holmesless

…

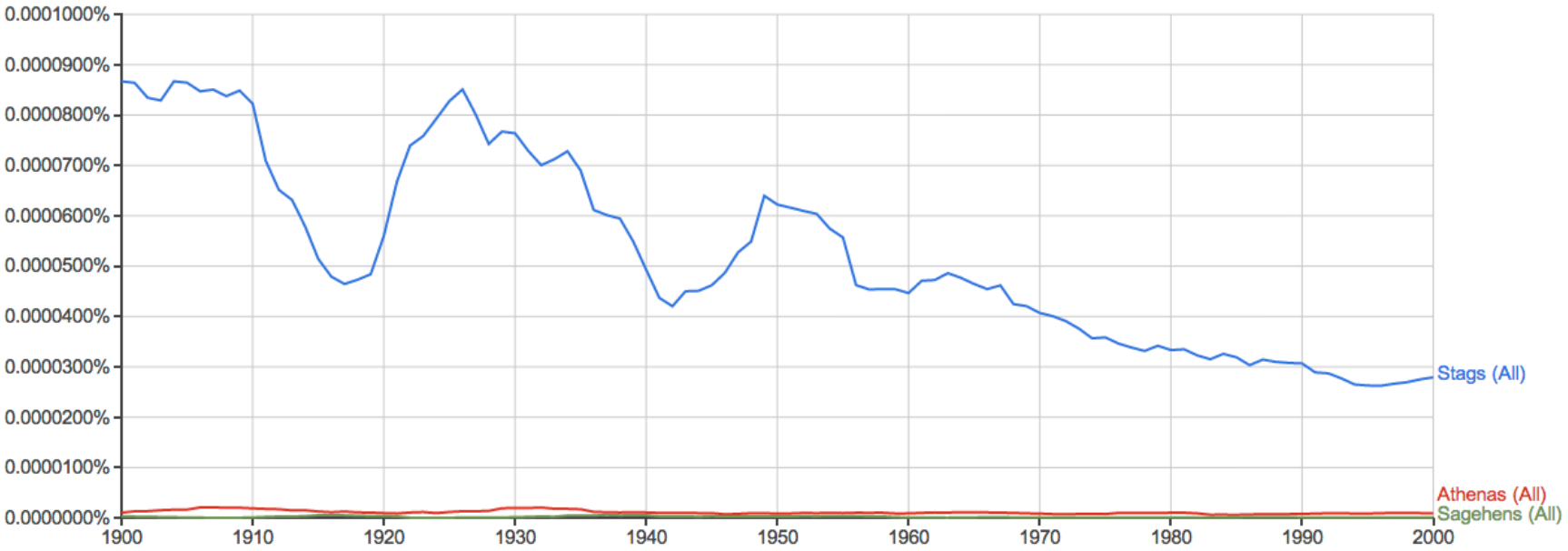
Highly scalable, used for…

- Protein/DNA sequencing (e.g. homology)
- Information retrieval (e.g. query completion)
- Natural language processing (e.g. translation)

…

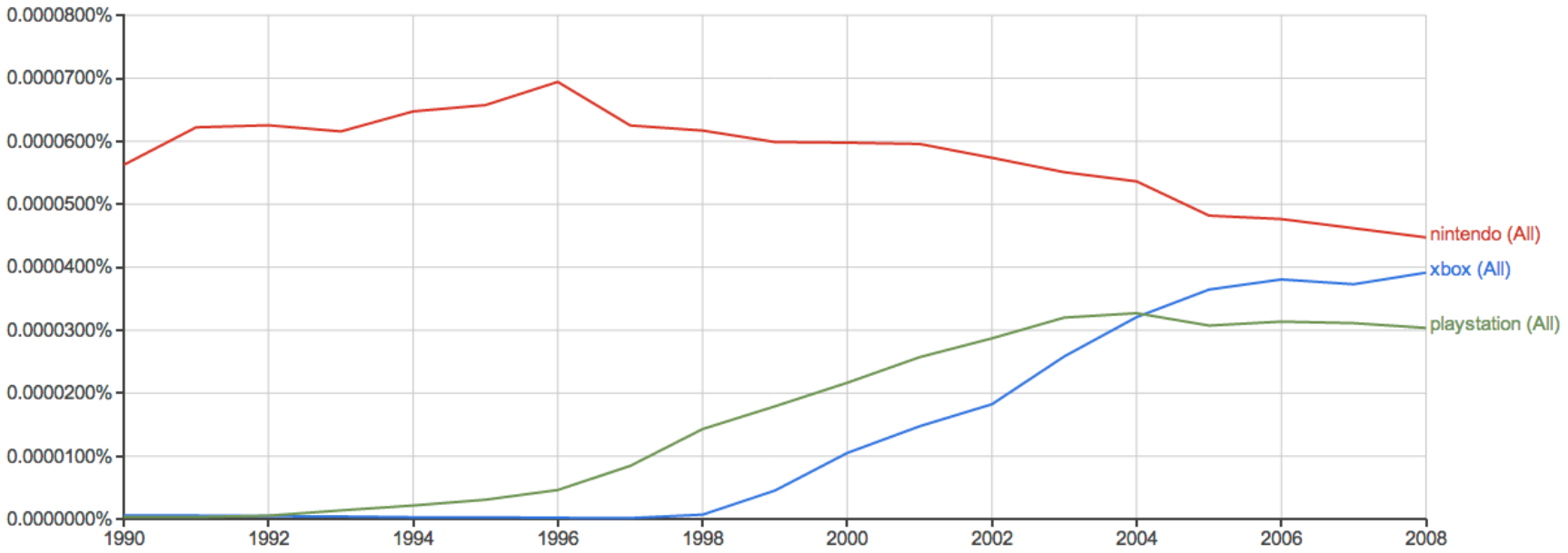
Google N-Gram Viewer

https://books.google.com/ngrams



Google N-Gram Viewer

https://books.google.com/ngrams



Statistical Text Generation

- Read **trigrams** (3-grams) from a text
- Design and implement a data structure to aggregate statistics, building a distribution over word sequence
  - P(“can” | “yes we”) vs. P(“coupon” | “yes we”)
- Generate new text from prompt

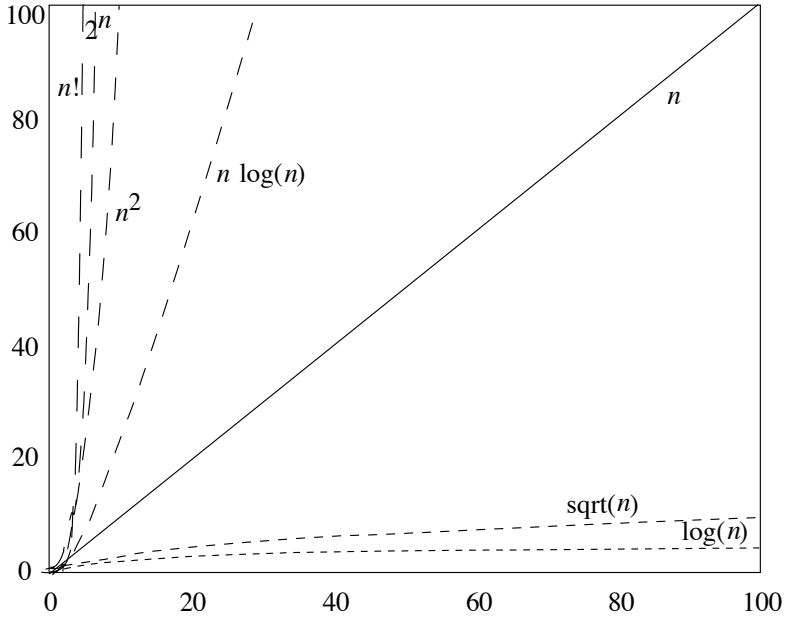
Tips

- Start early (not as easy as assignment #1)
- Think before sitting down to a computer

Asymptotic Analysis

Rather than counting individual operations, characterize how algorithm memory/time increases as a function of input size (n)

What happens at Google scale?



	10 n	100n
log n	3 + t	7 + t
n	10 t	100 t
n log n	> 10 t	> 100 t
n^2	100 t	10,000 t
2^n	~ t^{10}	~ t^{100}

Consider: an algorithm ***a*** requires ***t*** units of time/ space for a problem of size ***n***

Big-O Notation

Asymptotic Upper Bound

Independent of CPU

From some time point on

$f(n)$ is O($g(n)$) iff…

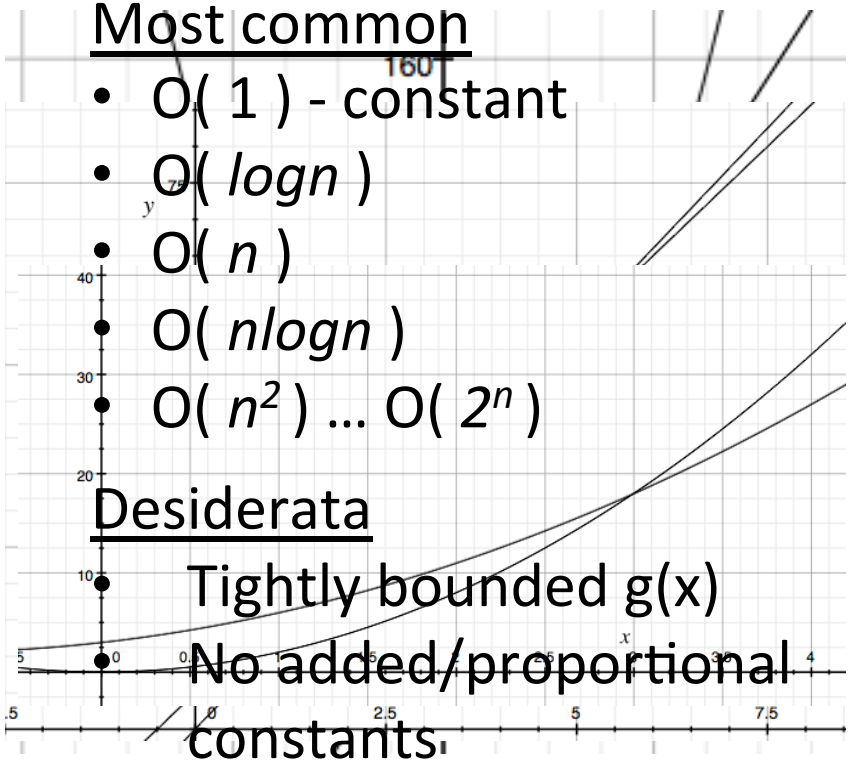
there exist positive constants: c, n_0

such that for all $n \geq n_0$ …

$| f(n) | \leq c \bullet | g(n) |$

Show:

- $10n$ is O(n^2)
- $10n + 5$ is O(n)
- $n^2 + 2n + 3$ is O (n^2)



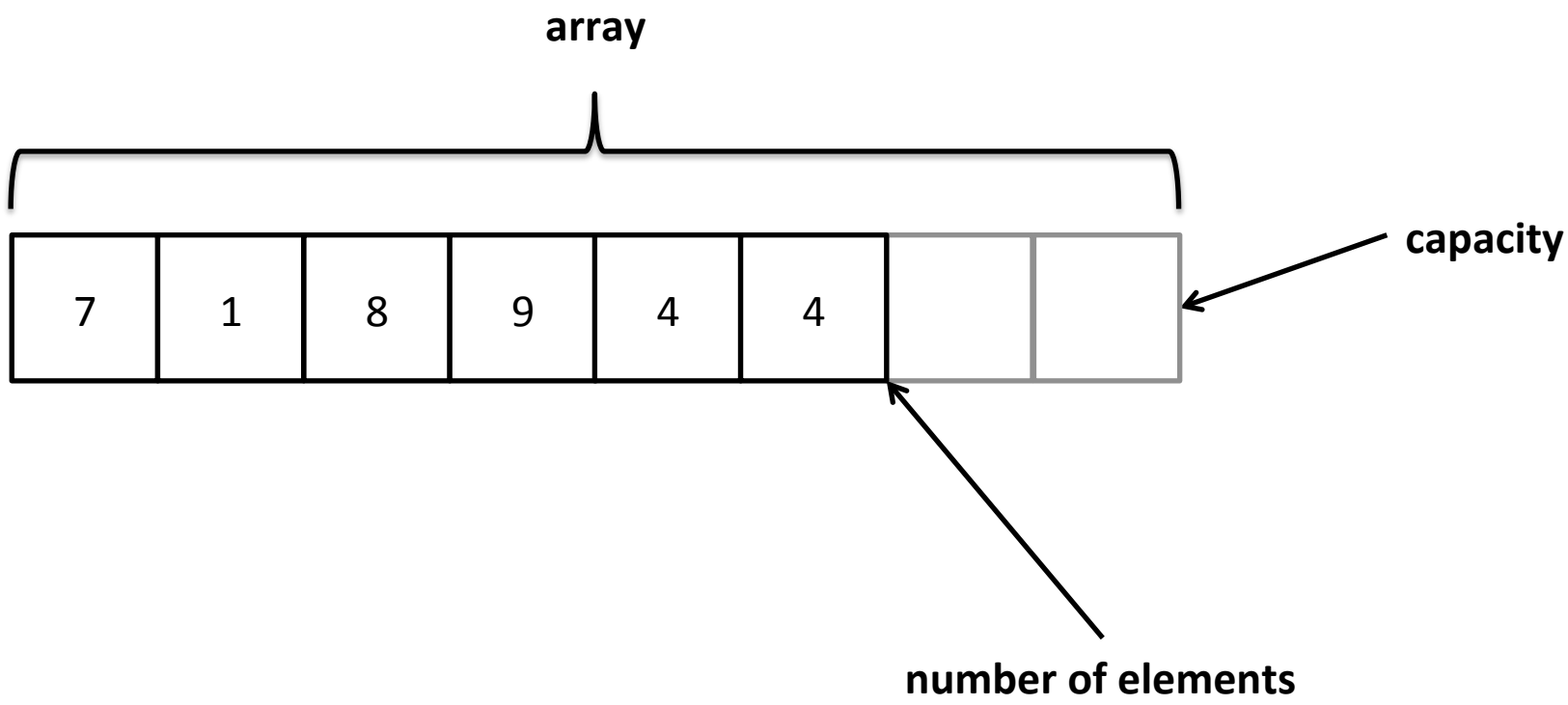
Most common

- O(1) - constant
- O(*logn*)
- O(*n*)
- O(*nlogn*)
- O(n^2) … O(2^n)

Desiderata

- Tightly bounded g(x)
- No added/proportional constants

Analyzing ArrayList



Analyzing ArrayList Operations

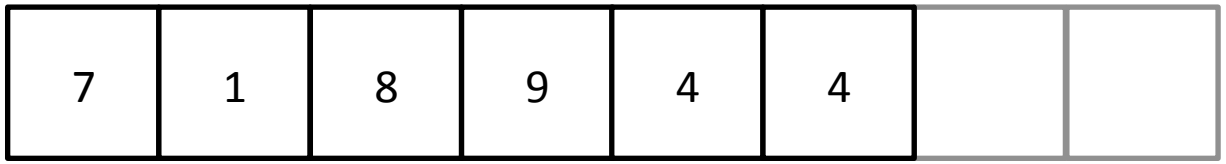
Worst Case

size	O(1)
isEmpty	O(1)
get(index)	O(1)
set(index, value)	O(1)

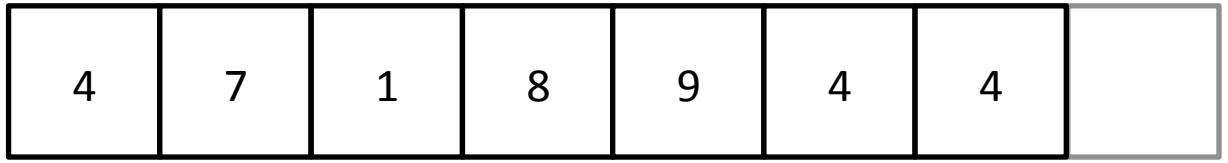
Adding to an ArrayList

With Extra Space

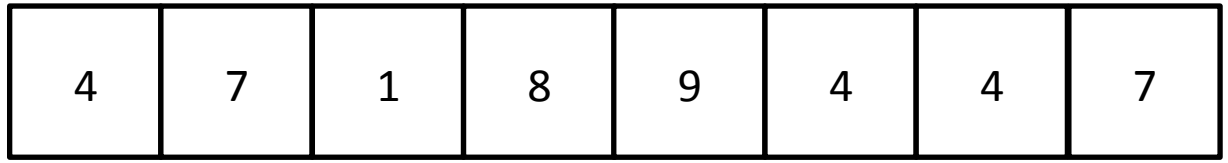
Assume existing contents, space to spare



- add(0, 4) is O(?)



- add(7) is O(?)



O(*n*)

remove(index)

O(1)

Adding to an ArrayList

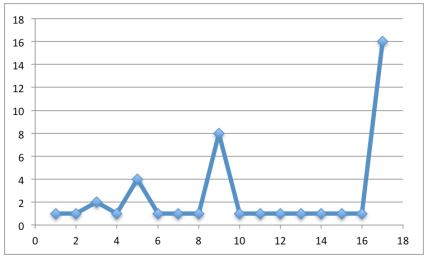
Without Extra Space

4	7	1	8	9	4	4	7

add(4), add(7), …

Depends upon implementation policy…

- Policy #1: increment size by 1
  - 1 copy of (*n-1*) data per 1 add = O(*n*)
- Policy #2: double
  - 1 copy of (*n*) data per (*n*) adds = O(1)



This Week's Lab – Empirically Test!

- Using StopWatch to time operations
  - start(), stop(), getTime(), reset()
- Using Vector from Bailey
  - Allows modification of add-increment policy
- Java has Just-In-Time compilation
  - Need to "warm up" to get accurate timing

Complexity Analysis Tool: Induction

Method of proving…

- Mathematical statements over natural numbers
- Statements about data structures (e.g. big-o)
- Correctness of programs
- Recursive properties
- …

<u>Recipe</u>

1. Prove "base" case ($n=n_0$)
2. Assume true for some $k \geq n_0$
3. Prove for $n = k + 1$

THUS! Must be true for base+1, base+2, …

Example from ArrayList

add(v), increment size by 1

- Copies: 1 + 2 + 3 + … (n-1)

Prove

The total number of copies is [*n*(*n*-1) / 2]

Follow-up

[*n*(*n*-1) / 2] is O(?)

Another Common Example

Prove

$$1 + 2 + 3 + \ldots n = [\, n\,(\, n+1\,)\,/\,2\,]$$

Follow-up

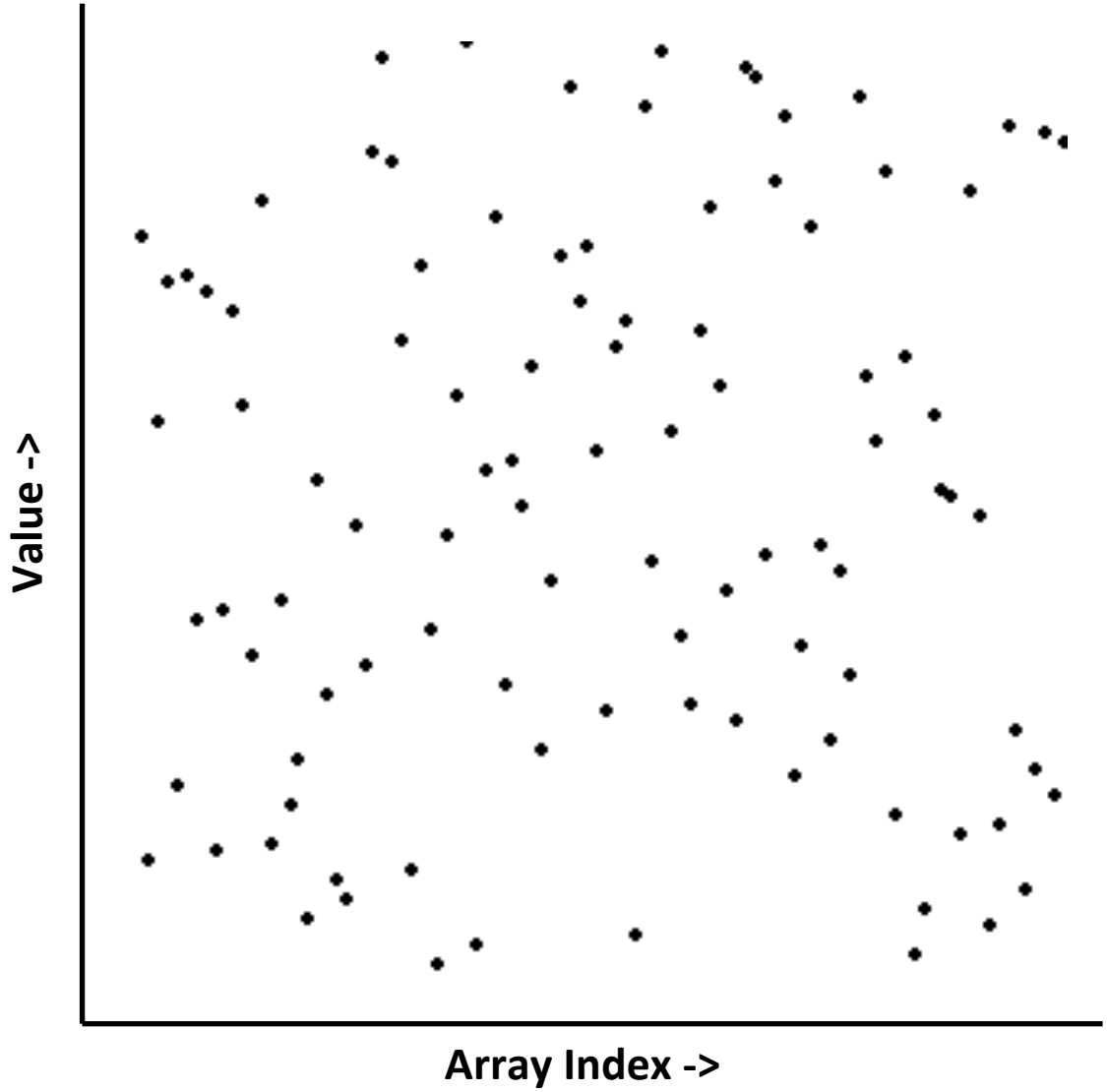
$$[\, n(\, n+1\,)\,/\,2\,] \text{ is } O(\,?\,)$$

Example: Orders of Magnitude

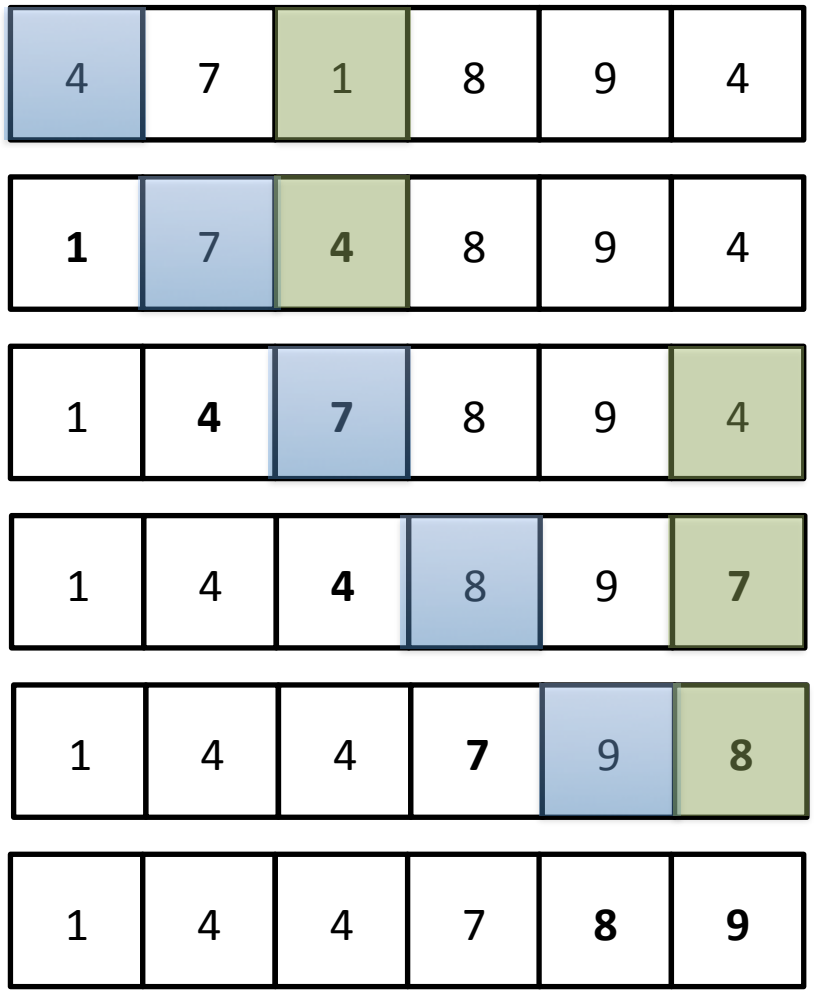
Prove

For $n \geq 4$, $2^n < n!$

Example: Selection Sort



Algorithm: Selection Sort



1. Swap "start" and "smallest"
2. Recurse(start+1)

Code: Selection Sort (1)

```
/**
 * @param array array of integers
 * @param startIndex a valid index into array
 */
public static void selectionSort(int[] array, int startIndex) {
     if(startIndex < array.length) {

          // find smallest element in array[startIndex...n]
          int smallest = indexOfSmallest(array, startIndex);

          // move smallest element to position startIndex
          swap(array, smallest, startIndex);

          // recurse on everything to the right of startIndex
          selectionSort(array, startIndex+1);

     }
}
```

Code: Selection Sort (2)

```
/**
 * @param array array of integers
 * @param startIndex valid index into array
 * @return index of smallest value in array[startIndex...n]
 */
public static int indexOfSmallest(int[] array, int startIndex) {

     int smallest = startIndex;
     for(int i = startIndex; i < array.length; ++i) {
          if(array[i] < array[smallest]) {
               smallest = i;
          }
     }
     return smallest;
}
```

Next Class: Induction & Sorting

Use induction to prove correctness & complexity of sorting algorithms

Thank You :)

Questions?