CSE 12 – Basic Data Structures and Object-Oriented Design
Lecture 9

Greg Miranda & Paul Cao, Winter 2021

This lecture is being recorded

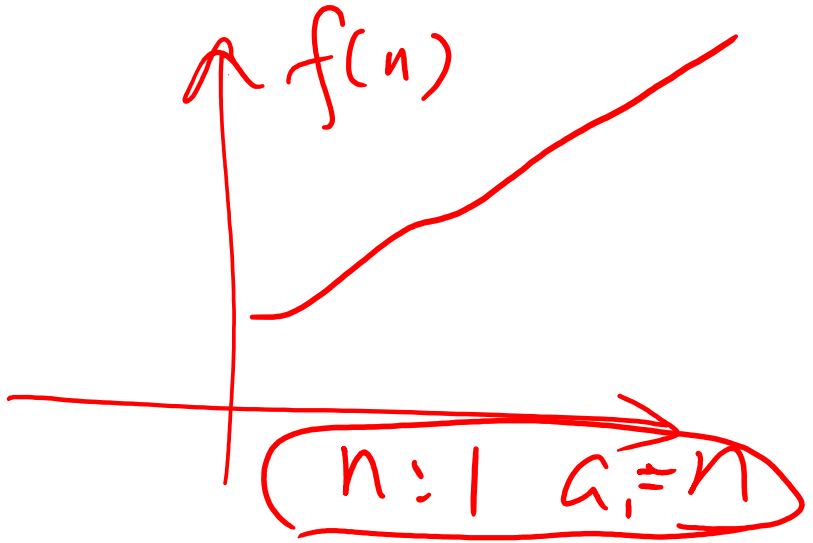
Announcements

- Quiz 9 due Wednesday @ 8am
- PA3 due Wednesday @ 11:59pm
- Survey 4 due Friday @ 11:59pm
- Exam 1 released Friday @ 8am, due Saturday before 8am

Topics

- Questions on Lecture 9?
- Counting Steps

Questions on Lecture 9?



$$\begin{cases} a_n = a_{n-1} + 7 \\ a_{n+1} = a_n + 7 \end{cases}$$

$n = 0, a_0 = 0$

Analyzing the worst case

```
boolean find( String[] theList, String toFind ) {
  for ( int i = 0; i < theList.length; i++ ) {
    if ( theList[i].equals( toFind ))
      return true;
  }
  return false;
}
```

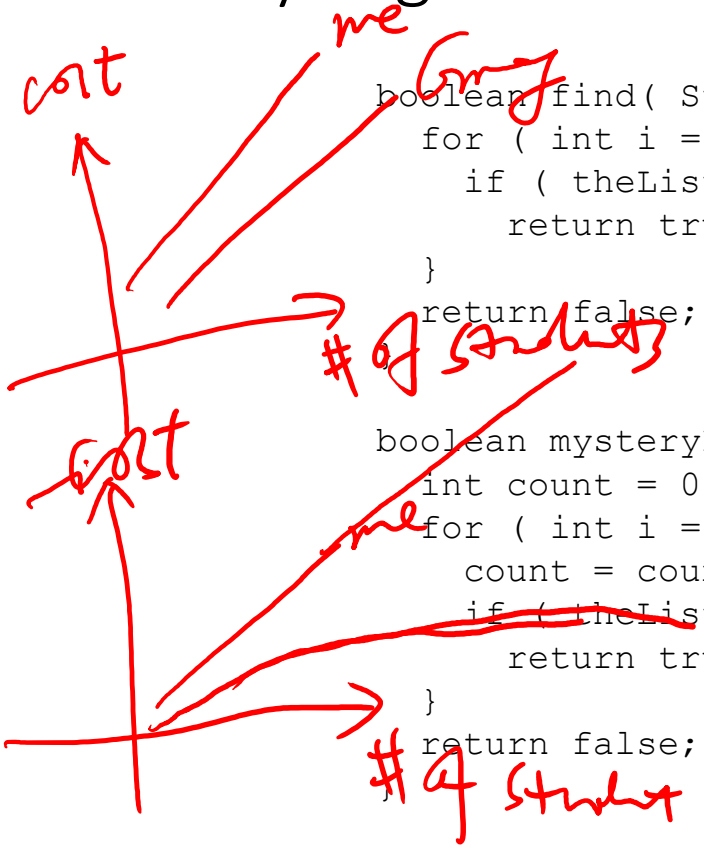
How many instructions do you have to execute to find out if the element is in the list in the worst case, if n represents the length of the list?

Analyzing the worst case

Code	# of instr.
`boolean find( String[] theList, String toFind ) {`	N/A
`    for ( int i = 0; i < theList.length; i++ ) {`	
`        if ( theList[i].equals( toFind )) `	
`            return true;`	
`    }`	
`    return false;`	
`}`	

How many instructions do you have to execute to find out if the element is in the list in the worst case, if n represents the length of the list?

Analyzing the worst case




```
boolean find( String[] theList, String toFind ) {
  for ( int i = 0; i < theList.length; i++ ) {
    if ( theList[i].equals(toFind) )
      return true;
  }
  return false;
}
```

```
boolean mysteryFind( String[] theList, String toFind ) {
  int count = 0;
  for ( int i = 0; i < theList.length; i++ ) {
    count = count + 1;
    if ( theList[i].equals(toFind) )
      return true;
  }
  return false;
}
```

n is Large

fundamental difference

3n + 3

4n + 4

Which method is faster?

A. find
B. mysteryFind
C. They are about the same

Analyzing the worst case : n large

```
boolean find( String[] theList, String toFind ) {
  for ( int i = 0; i < theList.length; i++ ) {
    if ( theList[i].equals(toFind) )
      return true;
  }
  return false;
}

boolean fastFind( String[] theList, String toFind ) {
  return false;
}
```

n

for (i=0 ; i<1e6 ; i++){
} }
return false

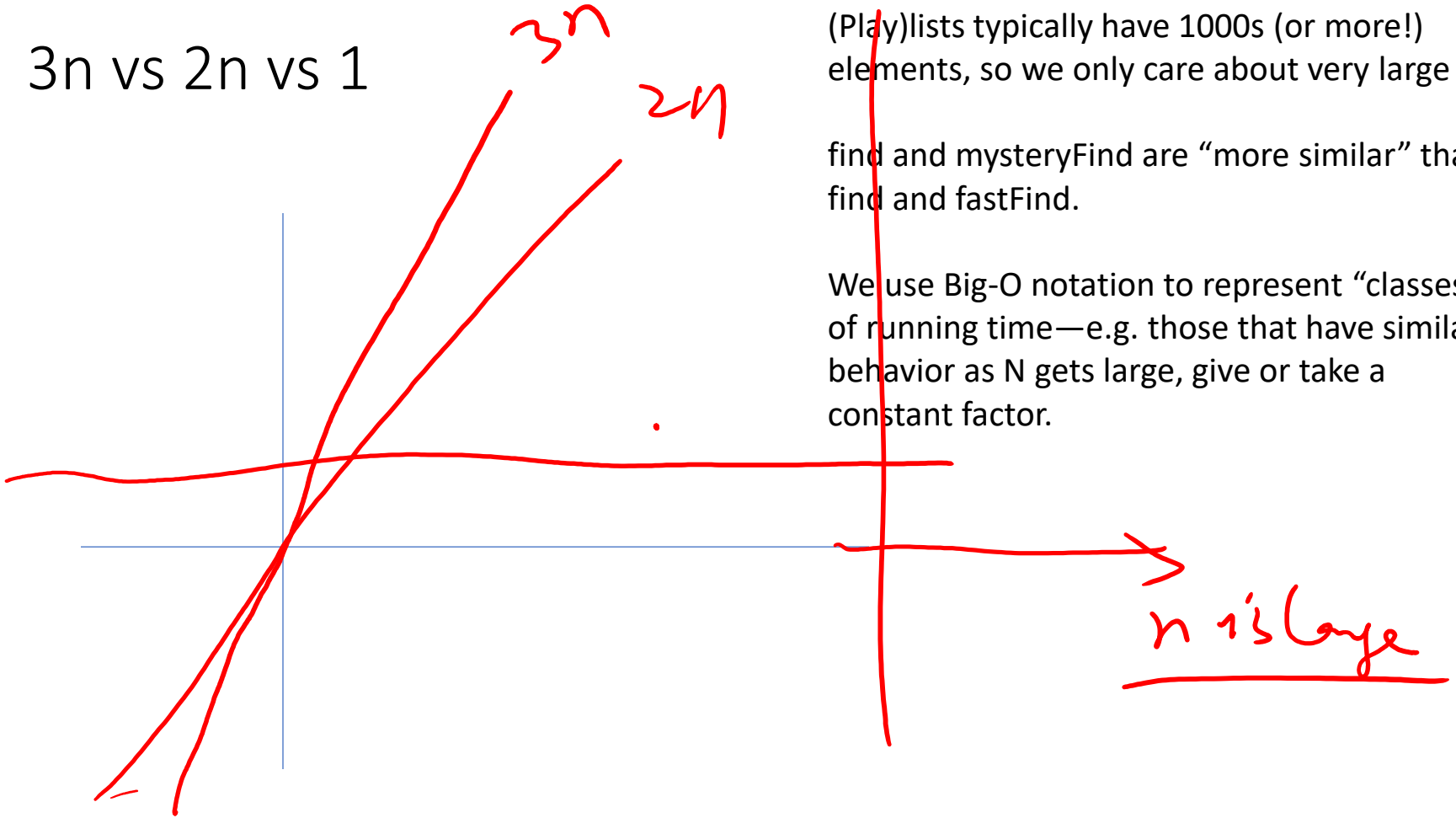
10^6

constant

Which method is faster?

A. find
B. fastFind
C. They are about the same

3n vs 2n vs 1



(Play)lists typically have 1000s (or more!) elements, so we only care about very large n.

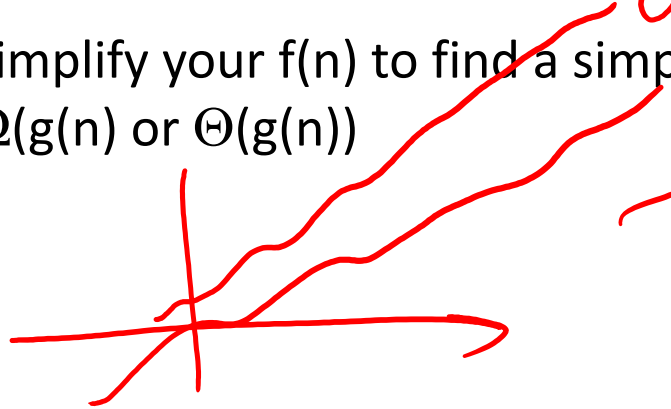
find and mysteryFind are “more similar” than find and fastFind.

We use Big-O notation to represent “classes” of running time—e.g. those that have similar behavior as N gets large, give or take a constant factor.

Steps for calculating the Big O (Theta, Omega) bound on code or algorithms

$f(n) = 3n^2 + 7n - 6$

1. Count the number of instructions in your code (or algorithm) as precisely as possible as a function of n, which represents the size of your input (e.g. the length of the array). This is your f(n).
   - Make sure you know if you are counting best case, worst case or average case – could be any of these!
2. Simplify your f(n) to find a simple g(n) such that f(n) = O(g(n)) (or Ω(g(n) or Θ(g(n))



$f(n) \in O(n^2)$

$\in O(n^3)$

Counting Steps – ArrayList Insert – ignore EC

n: size of ArrayList

```
public void insert(int index, String s) {
        expandCapacity();          //ignore
        for (int i = size - 1; i >= index ; i--) {
                this.elements[i+1] = this.elements[i];
        }
        this.elements[index] = s;
        this.size += 1;
}
```

linear

n

2

A: about n

B: n^2

C: n^3

Counting Steps – ArrayList Expand Capacity

```
private void expandCapacity() {
    int currentCapacity = this.elements.length;
    if(this.size < currentCapacity) { return; }
    String[] expanded = new String[currentCapacity * 2];
    for(int i = 0; i < this.size; i += 1) {
        expanded[i] = this.elements[i];
    }
    this.elements = expanded;
}
```

about

A : n

B : n^2

C : n lg n

D : 2dk

n

series of actions: worst part dominates about

Counting Steps – ArrayList Insert w/ EC

```
public void insert(int index, String s) {
    expandCapacity();
    for (int i = size - 1; i >= index ; i--) {
        this.elements[i+1] = this.elements[i];
    }
    this.elements[index] = s;
    this.size += 1;
}
```

n → expandCapacity();

n → for (int i = size - 1; i >= index ; i--) {

n → this.elements[i+1] = this.elements[i];

n → }

n → this.elements[index] = s;

n → this.size += 1;

n + n

A: n

B: n^2

C: n^3

D: n lg n

E: 2dk

Challenge

```
void tricky(int n) {
    int operations = 0;
    while(n > 0) {
        for(int i = 0; i < n; i++) {
            cout << "Operations: " << operations++ << endl;
        }
        n /= 2;
    }
}
```

What is the *step counting result for the algorithm above?*

A : n

B : $n \lg n$

C : n^2

D : n^3

E : 2dk