CS 2230
CS II: Data structures

Meeting 13: Asymptotic analysis 3
Brandon Myers
University of Iowa
Today’s big ideas

• Try using big-Oh/Theta/Omega to describe the running time of some real algorithms!

• When counting running time, what should you count? If the goal is big-Oh/theta/omega, then usually count one representative operation.

• Even if an individual operation is expensive, with the right algorithm, many operations can be efficient when looked at together
Binary recursion to compute Fibonacci numbers

```java
public int fibBad(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fibBad(n - 1) + fibBad(n - 2);
}
```

For fibBad, what is the best way to describe R(n)?

a) $O(n^2)$  
b) $O(n \log n)$  
c) $O(\log n)$  
d) $O(2^n)$  
e) $O(n)$

note: normally, we do analysis based on input size n, but in this case the input is n

https://b.socrative.com/login/student/  
CS2230A ids 1000-4999  
CS2230B ids 5000+
Binary recursion to compute Fibonacci numbers

```java
public int fibBad(int n) {
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return fibBad(n - 1) + fibBad(n - 2);
}
```

<table>
<thead>
<tr>
<th>Tfib(N)</th>
<th>1+Tfib(N-1)+Tfib(N-2)</th>
<th>time</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tfib(2)</td>
<td>1+Tfib(1)+Tfib(0)</td>
<td>1+1+1</td>
<td>3</td>
</tr>
<tr>
<td>Tfib(3)</td>
<td>1+Tfib(2)+Tfib(1)</td>
<td>1+3+1</td>
<td>5</td>
</tr>
<tr>
<td>Tfib(4)</td>
<td>1+Tfib(3)+Tfib(2)</td>
<td>1+5+3</td>
<td>9</td>
</tr>
<tr>
<td>Tfib(5)</td>
<td>1+Tfib(4)+Tfib(3)</td>
<td>1+9+5</td>
<td>15</td>
</tr>
<tr>
<td>Tfib(6)</td>
<td>1+Tfib(5)+Tfib(4)</td>
<td>1+15+9</td>
<td>25</td>
</tr>
<tr>
<td>Tfib(7)</td>
<td>1+Tfib(6)+Tfib(5)</td>
<td>1+25+15</td>
<td>41</td>
</tr>
<tr>
<td>Tfib(8)</td>
<td>1+Tfib(7)+Tfib(6)</td>
<td>1+41+25</td>
<td>67</td>
</tr>
<tr>
<td>Tfib(9)</td>
<td>1+Tfib(8)+Tfib(7)</td>
<td>1+67+41</td>
<td>109</td>
</tr>
</tbody>
</table>
Binary recursion to compute Fibonacci numbers

```java
public int fibBad(int n) {
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return fibBad(n - 1) + fibBad(n - 2);
}
```

For `fibBad`, what is the best way to describe R(n)?

- **a)** $O(n^2)$
- **b)** $O(n \log n)$
- **c)** $O(\log n)$
- **d)** $O(2^n)$
- **e)** $O(n)$

**note:** normally, we do analysis based on input size `n`, but in this case the `input` is `n`
Linear recursion to computer Fibonacci numbers

```java
public static long[] fibGood(int n) {
    // Computes fib(n), fib(n-1) together
    if (n <= 1) {
        long[] answer = {n, 0};
        return answer;
    } else {
        long[] tmp = fibGood(n-1);
        long[] answer = {tmp[0] + tmp[1],
                         tmp[0]};
        return answer;
    }
}
```

a) $O(n^2)$  
b) $O(n \log n)$  
c) $O(\log n)$  
d) $O(2^n)$  
e) $O(n)$
public static long[] fibGood(int n) {
    // Computes fib(n), fib(n-1) together
    if (n <= 1) {
        long[] answer = {n, 0};
        return answer;
    } else {
        long[] tmp = fibGood(n-1);
        long[] answer = {tmp[0] + tmp[1],
                          tmp[0]};
        return answer;
    }
}

<table>
<thead>
<tr>
<th>Tfib(N)</th>
<th>1+Tfib(N-1)</th>
<th>time</th>
<th>difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tfib(2)</td>
<td>1+Tfib(1)</td>
<td>1+1</td>
<td></td>
</tr>
<tr>
<td>Tfib(3)</td>
<td>1+Tfib(2)</td>
<td>1+2</td>
<td>+1</td>
</tr>
<tr>
<td>Tfib(4)</td>
<td>1+Tfib(3)</td>
<td>1+3</td>
<td>+1</td>
</tr>
<tr>
<td>Tfib(5)</td>
<td>1+Tfib(4)</td>
<td>1+4</td>
<td>+1</td>
</tr>
<tr>
<td>Tfib(6)</td>
<td>1+Tfib(5)</td>
<td>1+5</td>
<td>+1</td>
</tr>
</tbody>
</table>
Return of the ArrayList: an analysis of resizing

/*
 A List that is implemented using an array
 */

public class ArrayList implements List {
    private Object[] elements;
    private int numElements;

    @Override
    public void append(Object ele) {
        // copy existing elements to a bigger array if necessary
        if (elements.length == numElements) {
            Object[] n = new Object[elements.length+1];
            for (int i=0; i<elements.length; i++) {
                n[i] = elements[i];
            }
            elements = n;
        }
        // insert ele
        elements[numElements] = ele;
        numElements++;
    }
}

Return of the ArrayList: an analysis of resizing
Return of the ArrayList: an analysis of resizing

If it takes one “step” to copy one element, about how many total steps will be taken to call append 1000 times when the initial size was 4?

a) 1000  
b) 2000  
c) 1,000,000  
d) 1,000,000,000

https://b.socrative.com/login/student/CS2230A ids 1000-4999
CS2230B ids 5000+
<table>
<thead>
<tr>
<th>Nth element</th>
<th>number of steps for append</th>
<th>total steps for appends to this point</th>
</tr>
</thead>
<tbody>
<tr>
<td>5&lt;sup&gt;th&lt;/sup&gt;</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6&lt;sup&gt;th&lt;/sup&gt;</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7&lt;sup&gt;th&lt;/sup&gt;</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>8&lt;sup&gt;th&lt;/sup&gt;</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>9&lt;sup&gt;th&lt;/sup&gt;</td>
<td>9</td>
<td>35</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>999</td>
<td>499490</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>500490</td>
</tr>
</tbody>
</table>
Simulation

Calculate the total time by a simple simulation!

```python
sum = 0
for i in range(5, 1001):
    sum+=i
print str(i)+","+str(sum)
```

(not the same as an experiment, where we actually time the ArrayList insertions)
append time is accelerating!

- **# calls to append**
- **total time to insert**

The graph shows the relationship between the number of calls to append and the total time to insert, indicating that the append time is accelerating.
Another approach: Geometric resizing

(double the size when we run out of space)
Simulate this new scenario

Calculate the total time by a simple simulation!

```python
sum = 0
arraysize = 4
for i in range(5, 1001):
    if i > arraysize:
        sum += i
        arraysize = 2*arraysize
    else:
        sum+=1

print str(i)+","+str(sum)
```

(not the same as an experiment, where we actually time the ArrayList insertions)
some insertions take a long time, but the total time is growing linearly!
Today’s big ideas

• Try using big-Oh/Theta/Omega to describe the running time of some real algorithms!

• When counting running time, what should you count? If the goal is big-Oh/theta/omega, then usually count one representative operation.

• Even if an individual operation is expensive, with the right algorithm, many operations can be efficient when looked at together
  • called “amortized analysis”