Running times

1/22/16

Green slides taken (w/ minor modifications) from Cynthia Lee’s CS 2 slides on http://www.peerinstruction4cs.org, licensed under Creative Commons Attribution-NonCommercial 4.0 International License.
Announcements

• Reading for Monday: Read Chapter 5
• HW out soon
• Quiz today!
Running times
(basic version)

• Constant time
  – Method takes the same amount of time no matter how much is in the Bag

• Linear time
  – Method takes time proportional to the number of items in the Bag

(For both, assume code goes through longest path)
What is the running time of the linked-memory implementation of add?

A. Constant time
B. Linear time
What is the running time of the linked-memory implementation of add?

A. Constant time
B. Linear time
What is the running time of the array-based implementation of add?

A. Constant time
B. Linear time
What is the running time of the array-based implementation of add?

A. Constant time  
B. Linear time
What is the running time of the array-based implementation of add?

A. Constant time
B. Linear time

If you double the array in a resize operation, resizes are rare enough that any sequence of operations takes constant time per operation on average. We call this constant amortized time.
More running times:
Big-O notation
Running times from lab
(courtesy of Lola Renteria and Vlad Papancea; Winter ‘15)

<table>
<thead>
<tr>
<th>size</th>
<th>ArrayList</th>
<th></th>
<th>LinkedList</th>
<th></th>
<th>TreeSet</th>
<th></th>
<th>HashSet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add</td>
<td>get</td>
<td>remove</td>
<td>add</td>
<td>get</td>
<td>remove</td>
<td>add</td>
<td>contains</td>
</tr>
<tr>
<td>25K</td>
<td>211</td>
<td>0</td>
<td>172</td>
<td>235</td>
<td>1,804</td>
<td>5</td>
<td>5,624</td>
<td>4,050</td>
</tr>
<tr>
<td>50K</td>
<td>399</td>
<td>8</td>
<td>391</td>
<td>389</td>
<td>3,039</td>
<td>12</td>
<td>12,134</td>
<td>3,837</td>
</tr>
<tr>
<td>100K</td>
<td>649</td>
<td>3</td>
<td>1,242</td>
<td>779</td>
<td>7,157</td>
<td>15</td>
<td>25,135</td>
<td>4,174</td>
</tr>
<tr>
<td>200K</td>
<td>1,391</td>
<td>3</td>
<td>2,727</td>
<td>1,565</td>
<td>13,667</td>
<td>5</td>
<td>58,807</td>
<td>4,974</td>
</tr>
<tr>
<td>400K</td>
<td>2,829</td>
<td>12</td>
<td>4,649</td>
<td>2,979</td>
<td>24,492</td>
<td>2</td>
<td>112,334</td>
<td>5,063</td>
</tr>
<tr>
<td>800K</td>
<td>3,922</td>
<td>13</td>
<td>8,696</td>
<td>4,764</td>
<td>48,448</td>
<td>5</td>
<td>233,346</td>
<td>5,236</td>
</tr>
</tbody>
</table>
Adding a tail pointer

“Tail pointer” (tail reference) refers to last Node in the list
Helped adding to the end (now constant time)
Didn’t help removing from the end (still linear)
Doubly-linked lists

• Each node also stores a reference to the previous Node

• Allows insertion of new nodes before or after any node to which you have a reference
A student has counted how many times we perform each line of code

<table>
<thead>
<tr>
<th>Statements</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> float findAvg ( int []grades )</td>
<td></td>
</tr>
<tr>
<td><strong>2</strong> float sum = 0;</td>
<td>1</td>
</tr>
<tr>
<td><strong>3</strong> int count = 0;</td>
<td>1</td>
</tr>
<tr>
<td><strong>4</strong> while ( count &lt; grades.length ) {</td>
<td>n + 1</td>
</tr>
<tr>
<td></td>
<td>sum += grades[count];</td>
</tr>
<tr>
<td></td>
<td>count++;</td>
</tr>
<tr>
<td><strong>5</strong> }</td>
<td></td>
</tr>
<tr>
<td><strong>7</strong> if ( grades.length &gt; 0 )</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>return sum / grades.length;</td>
</tr>
<tr>
<td><strong>9</strong> else</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>return 0.0f;</td>
</tr>
<tr>
<td><strong>11</strong> }</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>3n+5</td>
</tr>
</tbody>
</table>
Is $10n$ or $20n$ bigger?
Is $n^2$ or $10n+11$ faster?
$n^2$ vs. $10n+11$ revisited

Cross at $n=11$
10n vs. 20n revisited (with $n^2$)
Numbers of operations for different rates of growth

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>n²</th>
<th>n³</th>
<th>2ⁿ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>1,024</td>
</tr>
</tbody>
</table>
Numbers of operations for different rates of growth

<table>
<thead>
<tr>
<th>1,000,000</th>
<th>n</th>
<th>n²</th>
<th>n³</th>
<th>2^n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>1,024</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100</td>
<td>10,000</td>
<td>1,000,000</td>
<td>~10^{30}</td>
</tr>
</tbody>
</table>
Numbers of operations for different rates of growth

<table>
<thead>
<tr>
<th>1,000,000</th>
<th>n</th>
<th>n²</th>
<th>n³</th>
<th>2ⁿ</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000,000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10</td>
<td>100</td>
<td>1,000</td>
<td>1,024</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100</td>
<td>10,000</td>
<td>1,000,000</td>
<td>~10³⁰</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1,000</td>
<td>1,000,000</td>
<td>1,000,000,000</td>
<td>~10³⁰⁰</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10,000</td>
<td>100,000,000</td>
<td></td>
<td>10¹²</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100,000</td>
<td>10,000,000,000</td>
<td></td>
<td>10¹⁵</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1,000,000</td>
<td>10¹²</td>
<td></td>
<td>10¹⁸</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10,000,000</td>
<td>10¹⁴</td>
<td></td>
<td>10²¹</td>
</tr>
<tr>
<td>1,000,000</td>
<td>100,000,000</td>
<td>10¹⁶</td>
<td></td>
<td>10²⁴</td>
</tr>
<tr>
<td>1,000,000</td>
<td>1,000,000,000</td>
<td>10¹⁸</td>
<td></td>
<td>10²⁷</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10,000,000,000</td>
<td>10²⁰</td>
<td></td>
<td>10³⁰</td>
</tr>
</tbody>
</table>

Not really worth calculating...
Want to ignore
a) multiplicative constants
b) behavior at “small” n
Want to ignore

a) multiplicative constants
b) behavior at “small” n

Big-O notation:
f(n) = O(g(n)) if there exist constants $n_0$ and $c$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$
Count how many times each line executes, then which $O( )$ most accurately characterizes the growth?

```java
int maxDifference(int[] arr){
    max = 0;
    for (int i=0; i<arr.length; i++) {
        for (int j=0; j<arr.length; j++) {
            if (arr[i] - arr[j] > max)
                max = arr[i] - arr[j];
        }
    }
    return max;
}
```

A. $f(n) = O(2^n)$   D. $f(n) = O(n^3)$
B. $f(n) = O(n^2)$   E. Not exactly one
C. $f(n) = O(n)$   (assume $n = arr.length$)
Count how many times each line executes, then which $O(\ )$ most accurately characterizes the growth?

```java
int maxDifference(int[] arr) {
    max = 0;
    for (int i=0; i<arr.length; i++) {
        for (int j=0; j<arr.length; j++) {
            if (arr[i] - arr[j] > max)
                max = arr[i] - arr[j];
        }
    }
    return max;
}
```

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^3)$
E. Not exactly one (assume $n = arr.length$)
Count how many times each line executes, then which $O(\ )$ most accurately characterizes the growth?

```java
public int within10dups(int[] arr) {
    int cnt = 0;
    for(int i=0; i < arr.length; i++)
        for(int j=1; j <= 10; j++)
            if((j+i < arr.length) && (arr[i] == arr[i+j]))
                cnt++;
    return cnt;
}
```

A. $f(n) = O(2^n)$  
B. $f(n) = O(n^2)$  
C. $f(n) = O(n)$  
D. $f(n) = O(n^3)$  
E. Not exactly one of the above  

(assume $n = arr.length$)
Count how many times each line executes, then which \( O(\ ) \) most accurately characterizes the growth?

```java
public int within10dups(int[] arr) {
    int cnt = 0;
    for(int i=0; i < arr.length; i++)
        for(int j=1; j <= 10; j++)
            if((j+i < arr.length) && (arr[i] == arr[i+j]))
                cnt++;
    return cnt;
}
```

A. \( f(n) = O(2^n) \) 
B. \( f(n) = O(n^2) \) 
C. \( f(n) = O(n) \) 
D. \( f(n) = O(n^3) \) 
E. Not exactly one of the above (assume \( n = arr.length \))
Let \( f(n) = 2^n + 14n^2 + 4n^3 \)

- Which of the following is true?

A. \( f(n) = O(2^n) \)
B. \( f(n) = O(n^2) \)
C. \( f(n) = O(n) \)
D. \( f(n) = O(n^3) \)
E. Not exactly one of the above
Let $f(n) = 2^n + 14n^2 + 4n^3$

• Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^3)$
E. Not exactly one of the above
Let $f(n) = 100$

• Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^{100})$
E. Not exactly one of the above
Let $f(n) = 100$

• Which of the following is true?

A. $f(n) = O(2^n)$
B. $f(n) = O(n^2)$
C. $f(n) = O(n)$
D. $f(n) = O(n^{100})$
E. Not exactly one of the above (all of them)