

And More Algorithm Analysis



Computer Science Department
University of Central Florida

COP 3502 – Computer Science I



Announcements

- Quiz 3 is today
 - Purpose: make you study the current material TODAY
 - Hopefully (ideally) giving you less material to study on the weekend
 - “Why am I studying on the weekend?”
 - Ooooooh, yeah, Exam 1

- Exam 1 is on Tuesday!



And More Algorithm Analysis

- Examples of Analyzing Code:
 - Last time we went over examples of analyzing code
 - We did this in a somewhat naïve manner
 - Just analyzed the code and tried to “trace” what was going on
 - This Lecture:
 - We will do this in a more structured fashion
 - We mentioned that summations are a tool for you to help coming up with a running time of iterative algorithms
 - Today we will look at some of those same code fragments, as well as others, and show you how to use summations to find the Big-O running time



More Algorithm Analysis

■ Example 1:

- Determine the Big O running time of the following code fragment:
 - We have two for loops
 - They are NOT nested
 - The first runs from $k = 1$ up to (and including) $n/2$
 - The second runs from $j = 1$ up to (and including) n^2

```
for (k = 1; k <= n/2; k++) {  
    sum = sum + 5;  
}  
for (j = 1; j <= n*n; j++) {  
    delta = delta + 1;  
}
```



More Algorithm Analysis

■ Example 1:

- Determine the Big O running time of the following code fragment:
 - Here's how we can express the number of operations in the form of a summation:

$$\sum_{k=1}^{n/2} 1 + \sum_{j=1}^{n^2} 1$$

The constant value, 1, inside each summation refers to the one, and only, operation in each for loop.

```
for (k = 1; k <= n/2; k++) {  
    sum = sum + 5;  
}  
for (j = 1; j <= n*n; j++) {  
    delta = delta + 1;  
}
```

Now you simply solve the summation!



More Algorithm Analysis

■ Example 1:

- Determine the Big O running time of the following code fragment:
 - Here's how we can express the number of operations in the form of a summation:

$$\sum_{k=1}^{n/2} 1 + \sum_{j=1}^{n^2} 1$$

You use the formula: $\sum_{i=1}^n k = k * n$

$$\sum_{k=1}^{n/2} 1 + \sum_{j=1}^{n^2} 1 = \frac{n}{2} + n^2$$

- This is a **CLOSED FORM** solution of the summation
- So we approximate the running time as $O(n^2)$



More Algorithm Analysis

■ Example 2:

- Determine the Big O running time of the following code fragment:
 - Here we again have two for loops
 - But this time they are nested

```
int func2(int n) {
    int i, j, x = 0;
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            x++;
        }
    }
    return x;
}
```



More Algorithm Analysis

- Example 2:
 - Determine the Big O running time of the following code fragment:
 - Here we again have two for loops
 - But this time they are nested
 - The outer loop runs from $i = 1$ up to (and including) n
 - The inner loop runs from $j = 1$ up to (and including) n
 - The sole (only) operation is a “ $x++$ ” within the inner loop



More Algorithm Analysis

■ Example 2:

- Determine the Big O running time of the following code fragment:

- We express the number of operations in the form of a summation and then we solve that summation:

$$\sum_{i=1}^n \sum_{j=1}^n 1$$

You use the formula: $\sum_{i=1}^n k = k * n$

$$\sum_{i=1}^n \sum_{j=1}^n 1 = \sum_{i=1}^n n = n^2$$

All we did is apply the above formula twice.

- This is a **CLOSED FORM** solution of the summation
- So we approximate the running time as $O(n^2)$



More Algorithm Analysis

- Example 3:
 - Determine the Big O running time of the following code fragment:
 - Here we again have two for loops
 - And they are nested. So is this $O(n^2)$?

```
int func3(int n) {
    sum = 0;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n * n; j++) {
            sum++;
        }
    }
}
```



More Algorithm Analysis

- Example 3:
 - Determine the Big O running time of the following code fragment:
 - Here we again have two for loops
 - And they are nested. So is this $O(n^2)$?
 - The outer loop runs from $i = 0$ up to (and not including) n
 - The inner loop runs from $j = 0$ up to (and not including) n^2
 - The sole (only) operation is a “sum++” within the inner loop



More Algorithm Analysis

■ Example 3:

- Determine the Big O running time of the following code fragment:

- We express the number of operations in the form of a summation and then we solve that summation:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n^2-1} 1$$

You use the formula: $\sum_{i=1}^n k = k * n$

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n^2-1} 1 = \sum_{i=0}^{n-1} n^2 = n^2 \sum_{i=0}^{n-1} 1 = n^3$$

All we did is apply the above formula twice.

- This is a **CLOSED FORM** solution of the summation
- So we approximate the running time as $O(n^3)$



More Algorithm Analysis

■ Example 4:

- Write a summation that describes the number of multiplication operations in this code fragment:
 - Here we again have two for loops
 - Pay attention to the limits (bounds) of the for loop

```
int func3(int n) {
    bigNumber = 0;
    for (i = 100; i <= 2n; i++) {
        for (j = 1; j < n * n; j++) {
            bigNumber += i*n + j*n;
        }
    }
}
```



More Algorithm Analysis

- Example 4:
 - Write a summation that describes the number of multiplication operations in this code fragment:
 - Here we again have two for loops
 - Pay attention to the limits (bounds) of the for loop
 - The outer loop runs from $i = 100$ up to (and including) $2n$
 - The inner loop runs from $j = 1$ up to (and not including) n^2
 - Now examine the number of multiplications
 - Because this problem specifically said to “describe the number of multiplication operations, we do not care about ANY of the other operations
 - `bigNumber += i*n + j*n;`
 - There are TWO multiplication operations in this statement



More Algorithm Analysis

■ Example 4:

- Write a summation that describes the number of multiplication operations in this code fragment:
 - We express the number of multiplications in the form of a summation and then we solve that summation:

$$\sum_{i=100}^{2n} \sum_{j=1}^{n^2-1} 2$$

$$\sum_{i=100}^{2n} \sum_{j=1}^{n^2-1} 2 = \sum_{i=100}^{2n} 2(n^2 - 1) = 2(n^2 - 1) \sum_{i=100}^{2n} 1 = 2(n^2 - 1)(2n - 99)$$

- This is a **CLOSED FORM** solution of the summation
- Shows the number of multiplications

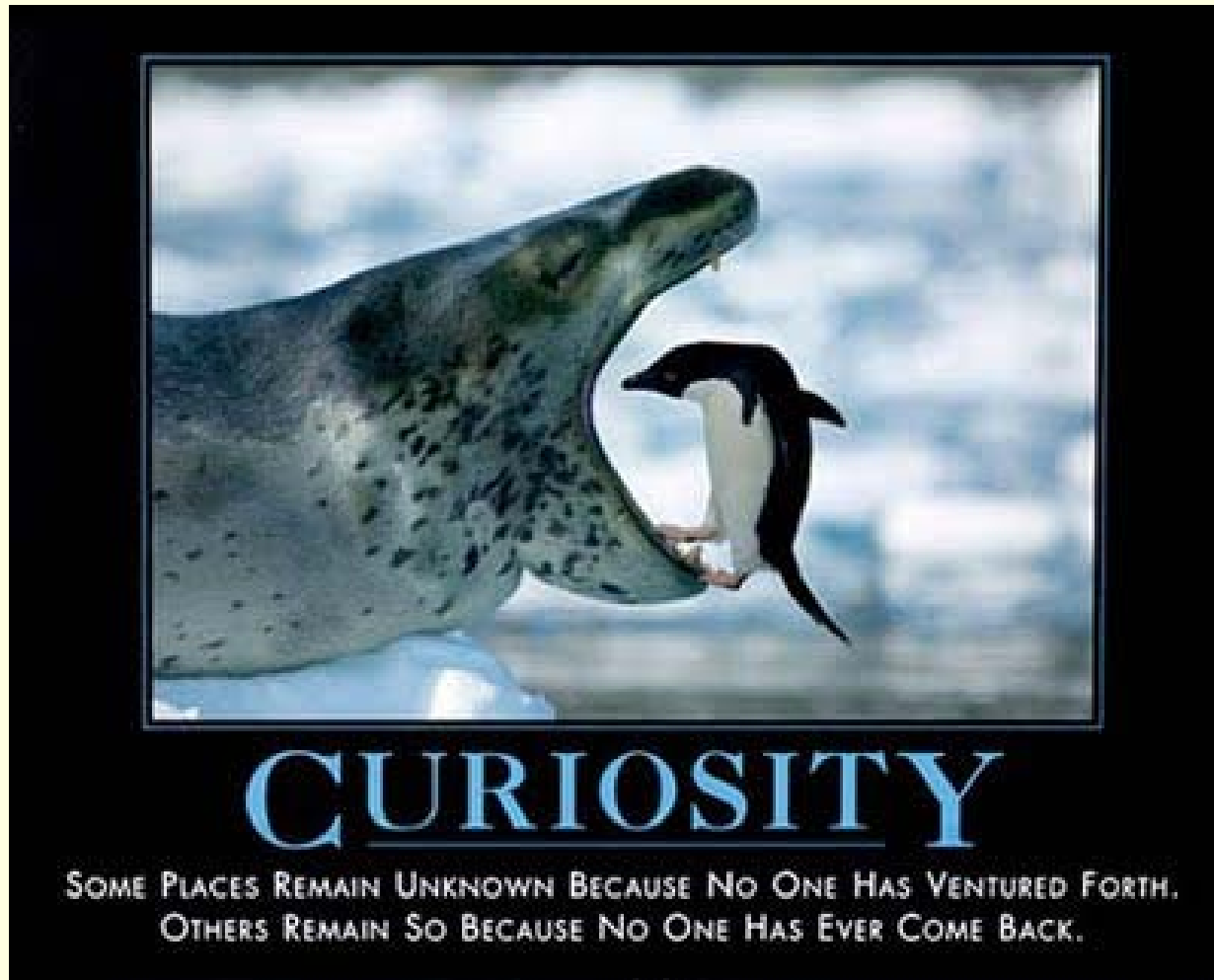


More Algorithm Analysis

**WASN'T
THAT
THE COOLEST!**



Daily Demotivator



And More Algorithm Analysis



Computer Science Department
University of Central Florida

COP 3502 – Computer Science I