CSCI 204: Data Structures & Algorithms

Algorithm Analysis

Pretend we had a list with x different names.
We build the following method to see if a name is in our list:

```
def is_name_in_list(my_list, search_name):
        for item in my_list:
                if item == search_name:
                        return True
        return False
```

How do we measure the "speed" of a program?
What do we need to know to determine how fast this will run?

Why does this matter?

- Computers are so fast! But…
- Large Scale Data
  - Google, Twitter, Facebook.. Big Data
- Limited Resources
  - phones, watches, wearable computing
- High Performance Environments
  - milliseconds matter

Measure the work instead of timing

- If we actually measure time, e.g., using the Linux `time` command, we can't account for the speed differences among different computers.
- Rather, we'd measure the steps an algorithm or a program will take when comparing them.
- Try a few examples with the `time` command …

Big-O Notation

- No need to count precise number of steps
- Classify algorithms by order of magnitude
  - Execution time
  - Space requirements
- Big O gives us a rough **upper bound**
- Goal is to give you intuition

How do we know what matters in code?

```
def sumOfN(n):
    theSum = 0                      1
    for i in range(1,n+1):          1 * (n)
        theSum = theSum + i         1 * (n)

    return theSum                   1

print(sumOfN(10))                   1
```

2n+3

How do we know what matters in code?

```
def sumOfN(n):
    theSum = 0
    for i in range(1,n+1):
        theSum = theSum + i

    return theSum

print(sumOfN(10))
```

2n+3

```
def sumOfN(n):
    theSum = 0
    for i in range(1,n+1):
        theSum = theSum + i

    doubleIt = theSum * 2
    halveIt = theSum/2
    theSum = halveIt

    return theSum

print(sumOfN(10))
```

2n+6

O(n)

Pay attention to what changes as the variables increases

Describing Growth

f(n)	Name
1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Log Linear
n^2	Quadratic
n^3	Cubic
2^n	Exponential

Let's Visualize It
See for example:
http://science.slc.edu/jmarshall/courses/2002/spring/cs50/BigO/index.html

Does it REALLY matter?

- Try out two examples
  - time python bubblesort.py
  - time python quicksort.py
- Try out a few more examples from mainRun.py which calls various operations in bigO.py

Definition

- Given a function *T(n)*
  - # of steps required for an input of size *n*.
  - Ex: $T_2(n) = n^2 + n$
- Suppose there exist a function *f(n)* for all integers $n \geq 0$ such that

$$T(n) \leq c\ f(n)$$

for some constant *c* and for all large values of $n \geq m$ (a constant).

We say function T(n) is on the order of f(n). In our above example, T(n) is on the order of n^2.

Code Examples

What is the Big O?

3 n^2 + 10 n log n	O(n^2)
n log n + n/2	O(n log n)
0.01n + 100n^2	O(n^2)
100n + 0.1n^2	O(n^2)
5 + 0.001n^3 + 0.025n	O(n^3)

Code Evaluation #1

```
def ex1( n ):
  count = 0
  for i in range( n ):
    count += i
  return count
```

Code Evaluation #2

```
def ex2( n ):
  count = 0
  for i in range( n ):
    count += 1
  for j in range( n ):
    count += 1
  return count
```

Code Evaluation #3

```
def ex3( n ):
  count = 0
  for i in range( n ):
    for j in range( n ):
      count += 1
  return count

def ex3b( n ):
  count = 0
  for i in range( n ):
    count += ex3( n )
  return count
```

Code Evaluation #4

```
def ex4( n ):
  count = 0
  for i in range( n ):
    for j in range( 25 ):
      count += 1
  return count
```

Code Evaluation #6

```
def ex6( n ):
  count = 0
  i = n
  while i >= 1 :
    count += 1
    i = i // 2
  return count
```

17

Code Evaluation #7

```
def ex6( n ):
  count = 0
  i = n
  while i >= 1 :
    count += 1
    i = i // 2
  return count

def ex7( n ):
  count = 0
  for i in range( n ) :
    count += ex6( n )
  return count
```

18