Section 1 Overview

Agenda

1. Review Introduction - More than 3 unexcused absence to section = NC, mini-assignments.
2. Analysis of algorithms (Big O)
    a. Movie_night
    b. sumList
3. Big-O Proof
4. Python tips and tricks

Analysis of Algorithms

Movie Night

```
movie_night(x,y,z):
      for i in range 0 to x: // O(x)
            for j in range 0 to y: //O(y)
                  print "LIGHTS CAMERA ACTION" //O(1)
      return 3*z //O(1)
answer: O(x*y)
```

Sum List

```
sumList(list): //of length n
s = 0
for element in list:
      s += element
return s
answer: O(n)
```

Big-O Proof

- **Prove that** $f(n) = n^2 + 5n + 7$ **is** $O(n^2)$
- Solution:

From Slide 44 we have the definition of Big-O:
"If there exist c, n_0 such that for all $n >= n_0$, $T_A(n) <= c * T_b(n)$ then $T_A(n)$ is $O(T_b(n))$."

Our problem is now to find c, n_0 such that for all $n >= n_0$, $n^2 + 5n + 7 <= c * n^2$

We can choose our n_0, so let's choose 1 (0 wouldn't work if we plugged it in). Now we have $n >= 1$. Now our goal is to show that for some constants c the right side will always overtake

the left side.

To deal with the n^2 term: pretty simple because by definition $n^2 >= n^2$

To deal with the + 5n term: We have that $n >= 1$, so it follows that $n^2 >= n$ (by multiplying both sides by n). We then expand this to

$n^2 >= n >= 1$
$5n^2 >= 5n >= 5$ (multiply by 5)

Similarly, when dealing with the + 7 term, we get if $n >= 1$, then
$7n^2 >= 7n >= 7$

Combining the three inequalities from above, we get:
$n^2 + 5n + 7 <= n^2 + 5n^2 + 7n^2$, and then simplifying we get
$n^2 + 5n + 7 <= 13n^2$ ⇒ so $c = 13$ and $n = 1$ and we have shown that $f(n)$ is $O(n^2)$!

They can plug in some values for n to prove the point.
n = 1: $f(1) = 1^2 + 5 * 1 + 7 = 13$ **<=** $13 * 1^2 = 13$
n = 2: $f(2) = 2^2 + 5 * 2 + 7 = 21$ **<=** $13 * 2^2 = 52$
n = 11: $f(11) = 11^2 + 5 * 11 + 7 = 183$ **<=** $13 * 11^2 = 1573$

Python Tips and Tricks (yay):

- Consider typing some stuff into python in your terminal and projecting that if possible!!
- Initializing lists:
    - Empty list of length 0:
        - `L = [ ]`
    - List with some elements:
        - `L = [1,2,3,"snuggie"] or L = [4]`
    - List of a certain initial size:
        - `L = [0]*100 (List of 100 0's)`
- List slicing:
    - `L[1:5]`
    - elements 1 through 5, (inclusive of 1, exclusive of 5)g
- Indexing the last element of a list:
    - `L[-1]`
- Reverse a list:
    - `L[::-1]`
- Enumerate: for i, element in enumerate(L):
    ….. ex. :

```
L = [3,4,5]
for i, number in enumerate(L):
     i = 0, number = 3 # first iteration
     i = 1, number = 4 # second iteration
     i = 2, number = 5 # third iteration
     # end loop
```

- Iterating through items in sequence: loop variable type self-defines
    - for book in library (library is defined, creates loop variable book that is whatever class library is full of)
    - For i in range(x)
- Stress importance of **indentation** in python
- Assert statements for testing.
    - assert func(x) == <expected value>, “String printed if condition isn’t met”

Bonus Python Problem

```
def get_odds(nums):
     """Given a list of positive ints, returns another list of any of
        the odd values in the given list
        Input: list of positive ints
        Output: list of ints, only odd values
     """
     odds = []
     for n in nums:
          if n % 2 != 0:
               odds.append(n)
     return odds
```