CS 61A
DISCUSSION 4

September 29, 2016

```
ld_rational( [1, 2], [1, 4] )

ef divide_rational(x, y):
    return [ x[0] * y[1], x[1] * y[0] ]
```

TOPICS FOR TODAY

- List mutation
- Growth
- Nonlocality

ATTENDANCE

Link: http://tiny.cc/gammafish

```
(lambda t: root(t))(tree(52, [tree(52)]))
```

LIST MUTATION

List mutation

means that you change a list *in-place (i.e. you modify the same list in memory)*

instead of, say, creating a list and just changing a variable name to point to that new list.

Some list methods for mutation

- append(elt) - *appends element to end of list*
- insert(i, elt) - *inserts element at index i*
- remove(elt) - *removes first-seen element with given value (otherwise errors)*
- pop(i) - *removes and returns element at index i*

GROWTH (ASYMPTOTIC ANALYSIS)

Growth explained

Growth: how much of a resource (TIME or SPACE) our program consumes as the input size gets bigger and bigger

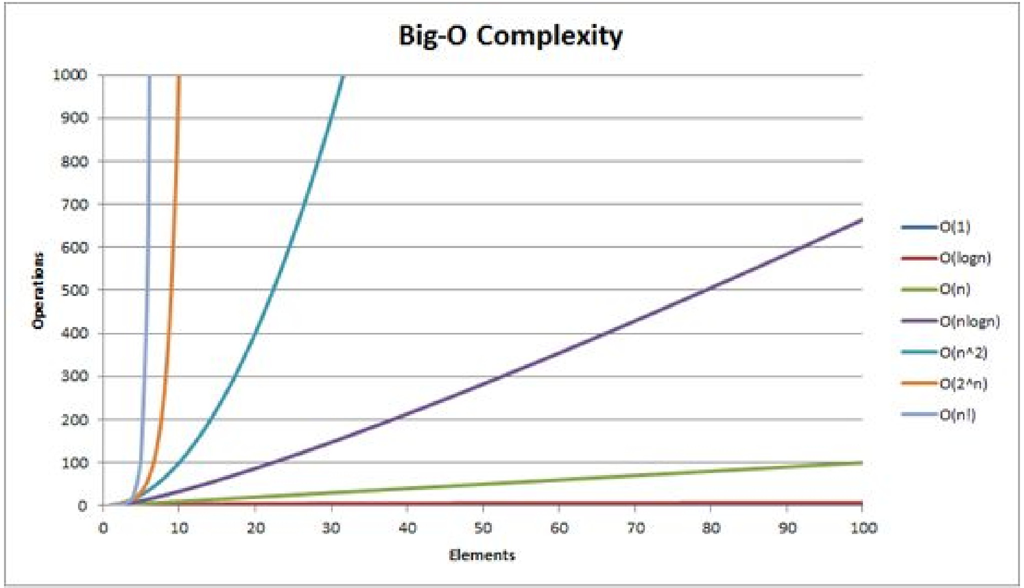
Order: how we quantify that growth. Also known as *time/space complexity*.

- Example orders: O(1), O(log(n)), O(n), O(nlog(n)), O(n^2), O(n^3), and O(x^n)
- Drop constants and lower-order terms! Growth-wise, they're not important

Order of growth is extremely important to take into account when designing algorithms!

To visualize

Just plot your algorithm runtime (which you can look at as the number of operations it needs to execute) against n, where n is the input size. The shape of the resulting plot will be the order of growth.



It may also help

to draw the **call tree**.

Basic example

```
def mystery(n):
    total = 0
    for i in range(n):
        total += constant(i)
    return total
```

What is the order of growth of `mystery` as a function of n?

Basic example

```
def mystery(n):
    total = 0
    for i in range(n): # loop n times
        total += constant(i) # each iteration, do constant work
    return total
```

What is the order of growth of `mystery` as a function of n?

O(n)

Trickier example

```
def mystery(n):
    total = 0
    for i in range(1, n):
        total *= 2
        if i % n == 0:
            total *= mystery(n - 1) * mystery(n - 2)
        elif i == n // 2:
            for j in range(1, n):
                total *= j
    return total
```

Answer: O(n) [linear work + linear work!]

```
def mystery(n):
    total = 0
    for i in range(1, n):
        total *= 2
        if i % n == 0: # this will never happen
            total *= mystery(n - 1) * mystery(n - 2)
        elif i == n // 2: # this will only ever happen ONCE
            for j in range(1, n):
                total *= j
    return total
```

An even trickier example

```
def f(n):
    i = 2
    while i < n:
        print(i)
        i = i * i
```

An even trickier example

```
def f(n):
    i = 2
    while i < n:
        print(i)
        i = i * i
```

Answer: O(log(logn)).

NONLOCAL

Nonlocality explained

When you say `nonlocal x`:

You're saying that in this function, `x` *refers to a variable that was defined in some parent frame. When you make assignments to* `x`*, it will change the* `x` *in the parent frame.*

Notes:

- x must be in a parent frame that ISN'T the global frame.

- If you have a nonlocal `x`*, you can't have a local* `x`*. Any time you refer to* `x` *within the function, you're talking about the* `x` *in the parent frame.*