Big-O

Theoretical measure of how fast a program is

Benefits:

- independent of platform (e.g. speed of hardware)
- can compare performance of different algorithms BEFORE coding them up!

A function of the input size

Could also think about $T_{avg}(N)$ or $T_{best}(N)$, but Big-O focuses on $T_{worst}(N)$

Rules of Big-O

Only care about dominant terms (constants don’t matter)

returns statements, conditional statements, assignments, arithmetic

-> all count as 1 step (a constant)

Estimate functions such as print and input as some K amount of steps

Example

def sum(N):	Analysis	
total = 0	assignment -> 1 step	
for i in range(N):		repeated N times
total += i*i*i	1 add, 3 mults, updating i -> K steps	
return total	return -> 1 step	

K*N + 2 total steps => O(N) function

Big-O analysis: Consecutive statements add

```
def printSimple():
    i = 10
    turtle = True
    print("hello")
```

Analysis

assignment -> 1 step

assignment -> 1 step

print -> K steps

K + 2 total steps => O(1) function

constant time

doesn't change based on input!

Big-O analysis: for/while loops

#steps = statements inside the loop multiplied by the #iterations

def foo(N):	Analysis	
for i in range(N):		
print(i)	print -> K steps	repeated N times

K*N steps => O(N)

Big-O analysis: Nested for loops

Analyze them inside-out, #steps = product of the sizes of the for loops

```
def MultTable(N):
    for i in range(N):
        for j in range(N):
            print(i*j)
```

Analysis

print -> K steps

repeated N times

repeated N times

N*N steps => O(N^2)

Big-O analysis: If/Else

#steps is the larger of either the first or second case

if a > b: Analysis

```
for i in range(N):
    print(i*j)
```

print -> K1 steps

repeated N times

else:

print(“No!”) print -> K2 steps

N*K1 or K2 steps => O(N)

Example

Analysis

```
total = []
```

assignment -> 1 step

```
for i in range(N):
    total.append(0)
```

append,etc -> K1 steps

repeated N times

```
for i in range(N):
    for j in range(10):
        total[i] += j*j
```

assignment, arithmetic, etc -> K2 steps

repeated 10 times

repeated N times

---

1 + N*K1 + N*10*K2 steps => O(N)