COMPLEXITY FROM CODE

REMEMBER

- Think about complexity at algorithm stage
- Should know complexity of approach before you start coding

CODE COMPLEXITY

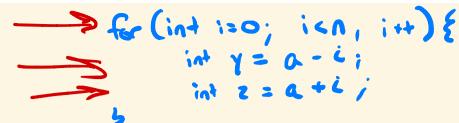
- Important to know how to determine in code
- Some code lines execute only a constant number of times, some dependent on n, etc.
- Typically this focuses on analyzing loop execution
 - How many times does loop execute
 - What is complexity of operations inside the loop?
- Most detailed level: count how many times each line executes
- Becomes natural enough to skip exact counts

EXAMPLES OF DIFFERENT LEVELS

y=a+b z=a-b

- Constant O(1)
 - A single operation (initialization, addition, comparison)
 - Multiple (constant number) of constant amount of work is constant
- Methods
 - Overhead of method doesn't affect complexity
 - Have to go into method to know method complexity

execute n times & constart



EXAMPLES OF DIFFERENT LEVELS

- O(n)
 - Typically single (not nested loops)
 - Inner operations combined are O(1)
- O(n^2)
 - Typically doubly nested loops
 - Inner loop is O(n) and outer loop executes approximately n times

```
int sum = 0; \Rightarrow executes 1 time

for (int i=0; i<n; i++) { \Rightarrow comparison executes n+1 times

sum += i; \Rightarrow executes n times

}

\Rightarrow (n+1) + n = 2 + 2n
\Rightarrow (n+1) + n = 2 + 2n
```

```
\frac{1}{3} + \frac{1}{1} = \frac{1}{2}
\frac{1}{3} + \frac{1}{1} = \frac{1}{3}
\frac{1}{4} = \frac{1}{3}
\frac{1}
```

```
public void foo(int n) {

int i = 1;

int sum = 0;

while (i < n) {

sum += i;

System.out.println(sum);

i *= 2;

executes log (n) + ines

executes log (n) + ines

i *= 2;

executes log (n) + ines

i *= 2;

executes log (n) + ines

i *= 2;
```

$$1 + 1 + \log_3(n) + 1 + \log_3(n) + \log_3(n)$$

+ $\log_3(n)$
 $3 + 4 \log_3(n) \Rightarrow O(\log_3(n))$

```
y=3
j=1
j=3
j=3
j=3
j=3
j=3
j=3
```

```
public void bar(int n) {

revecutes | time
       for (int i=0; i<n; i++) { -> composition executes nt } ->
           for (int j=1; j<n; j++) { \rightarrow composion executes n(n) +thres
               sum += i*j; >> executes n (n-1) times
       System.out.println(sum); >> executes 1 +we
      > i-loop inside executes n times
> j-loop inside executes n-1 times
    1+ n+1 + n2 + n6n-1.
=1+x(+1+u_3+u_3)=x=5+3u_3=0(v_3)
```