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 * constant

execute n times * constant

```
for (int i=0; i<n, i++){
    int y = a - i;
    int z = a + i;
}
```

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

Inside inner loop is $\Theta(1)$

Inner loop itself total is $\Theta(n)$

Outer loop executes n times

$$\begin{array}{ll} i=0 & 0<3 \\ i=1 & 1<3 \\ i=2 & 2<3 \\ i=3 & 3<3 \end{array}$$

```
int sum = 0;
for (int i=0; i<n; i++) {
    sum += i;
}
```

executes 1 time

comparison executes n+1 times

executes n times

$$1+(n+1)+n = 2+2n$$

$$\Rightarrow \Theta(n)$$

in loop
3 times

$i = 1$
$i = 2$
$i = 4$
$i = 8$

$1 < 8$
$2 < 8$
$4 < 8$
$8 < 8$

4 comparisons

NO

$3 = \log_2(8)$

```
public void foo(int n) {
    int i = 1;
    int sum = 0;
    while (i < n) {
        sum += i;
        System.out.println(sum);
        i *= 2;
    }
}
```

int i = 1; → executes 1 time
int sum = 0; → executes 1 time
while (i < n) → executes $\log_2(n) + 1$ times (comparison)
sum += i; → executes $\log_2(n)$ times
System.out.println(sum); → executes $\log_2(n)$ times
i *= 2; → executes $\log_2(n)$ times

$$1 + 1 + \log_2(n) + 1 + \log_2(n) + \log_2(n) + \log_2(n)$$

$$3 + 4\log_2(n) \Rightarrow \Theta(\log_2(n))$$

$n = 3$

$j = 1$ $\quad j < 3$

$j = 2$ $\quad j < 3$

$j = 3$ $\quad j < 3$ NO

```
public void bar(int n) {
    int sum = 0;                   --> executes 1 time
    for (int i=0; i<n; i++) {      --> comparison executes n+1 times
        for (int j=1; j<n; j++) {  --> comparison executes n(n) times
            sum += i*j;            --> executes n(n-1) times
        }
    }
    System.out.println(sum);       --> executes 1 time
}
```

i-loop inside executes n times

j-loop inside executes n-1 times

$$1 + n+1 + n^2 + n(n-1)$$

$$= 1 + \cancel{n} + 1 + n^2 + n^2 - \cancel{n} = 2 + 2n^2 = \Theta(n^2)$$