Time complexity

CS 146 - Spring 2017

What does this compute?

```
int foo(int n) {
    if (n == 1) return 2;
    int a = foo(n / 2);
    return a * a;
}
```

How many steps?

Today

- O, Theta, Omega notation
- Time complexity
- Examples
- Where do logarithms come up in time complexity?

True or false?

$$n^3 + 1 \text{ is } O(n^2)$$

$$n \text{ is } O(n/10 + 1)$$

$$n \text{ is } O(n/\log n)$$

O-notation is not a simplification operation!

- true that: often used to simplify functions (like rounding)

- by definition: used to compare functions (think <=)

Recall: we use O-notation to

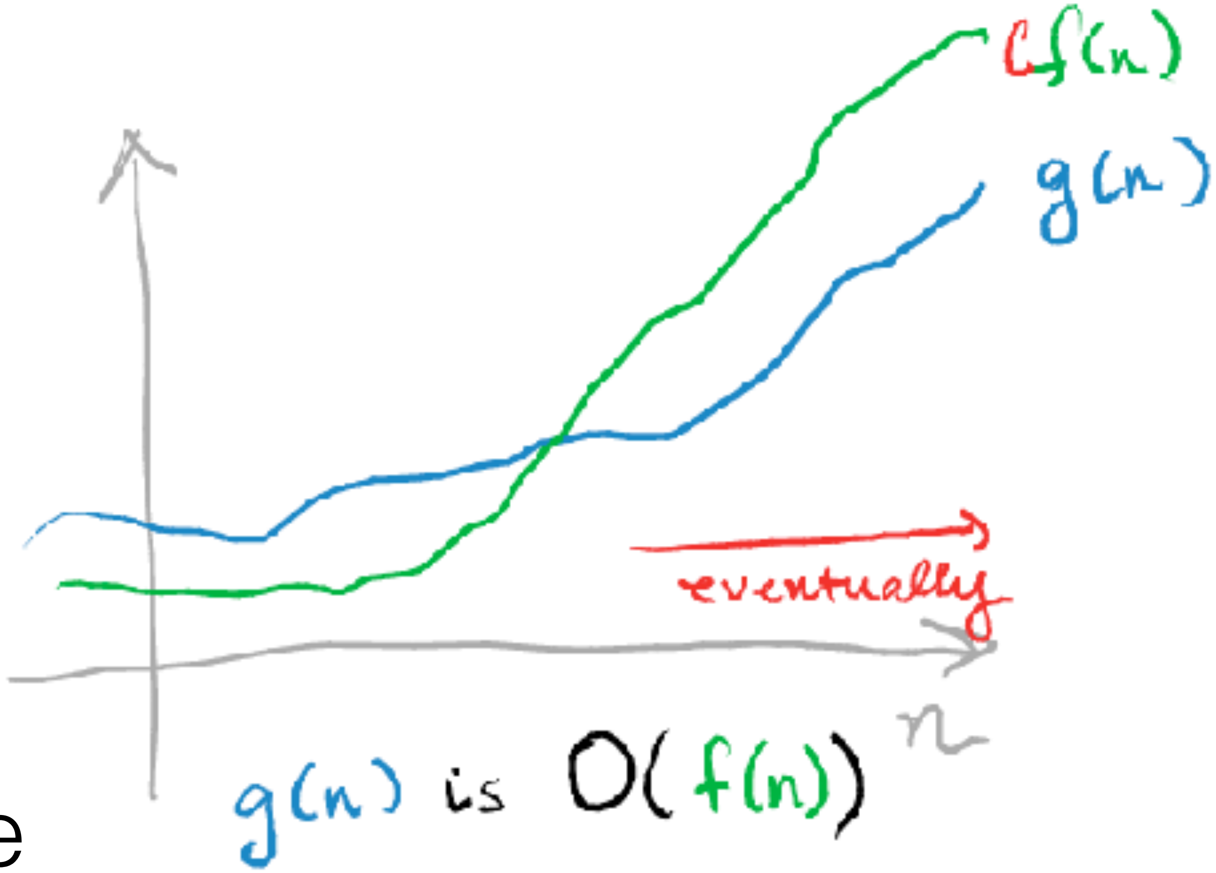
- (roughly) compare functions

$$n^2 \text{ is } O(n^3)$$

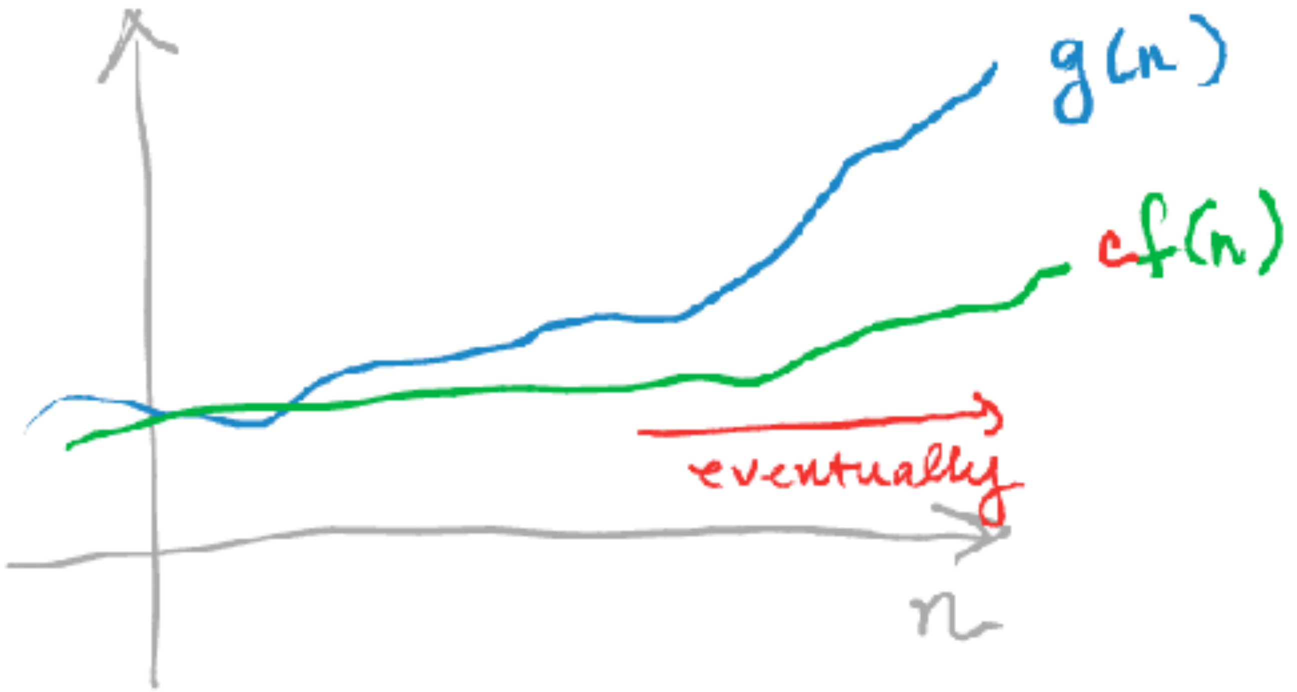
$$n^3 \text{ is } O(n^3 + 2n)$$

- simplify, (roughly) categorize

$$3n^3 + 2n \text{ is } O(n^3)$$



O(micron) and Omega



$g(n) \geqslant cf(n)$

$\Rightarrow$ we say

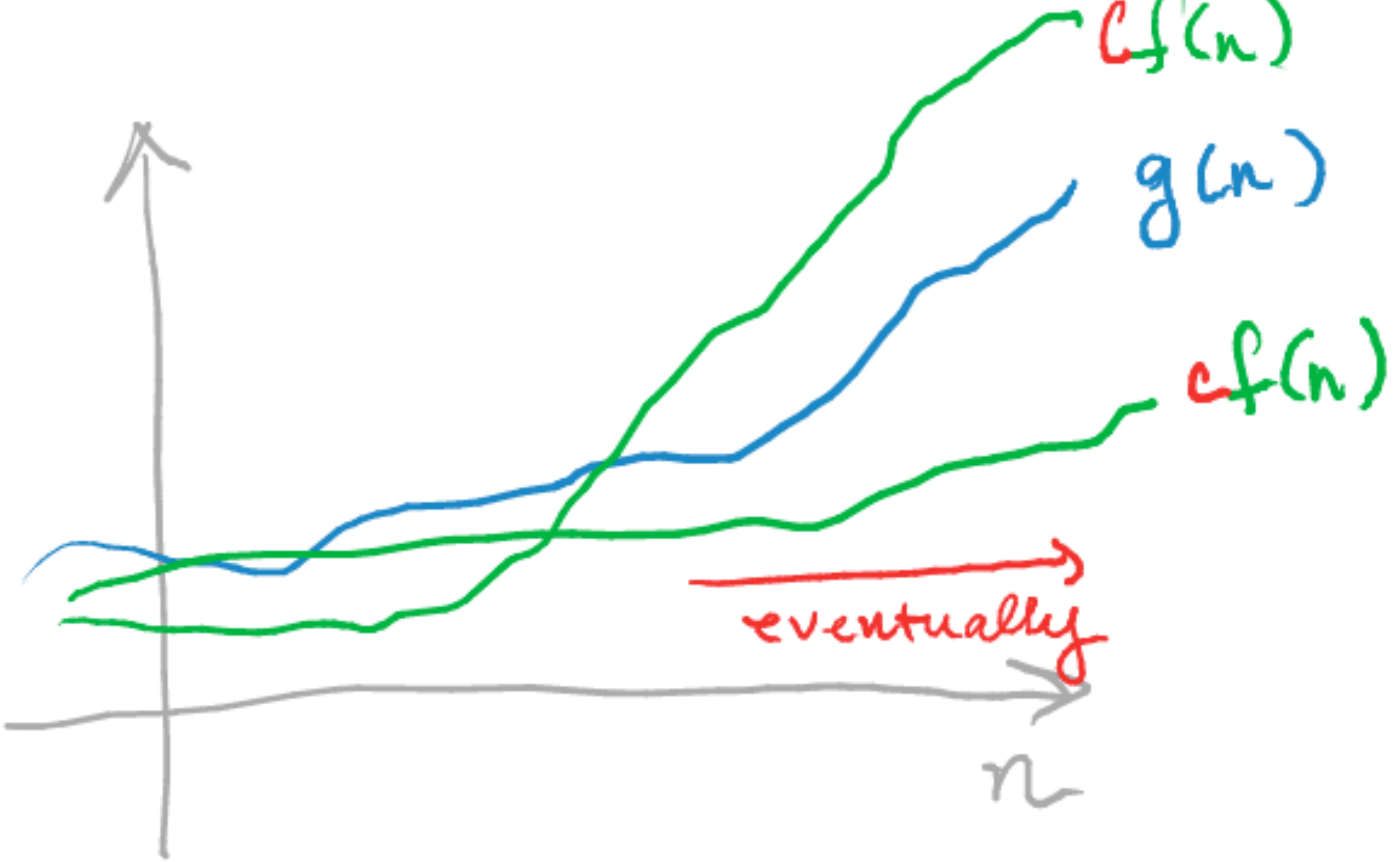
$g(n)$ is $\Omega(f(n))$

n^3+1 is $O(2^n)$

$\Updownarrow$

2^n is $\Omega(n^3+1)$

Fun with O, Theta, Omega

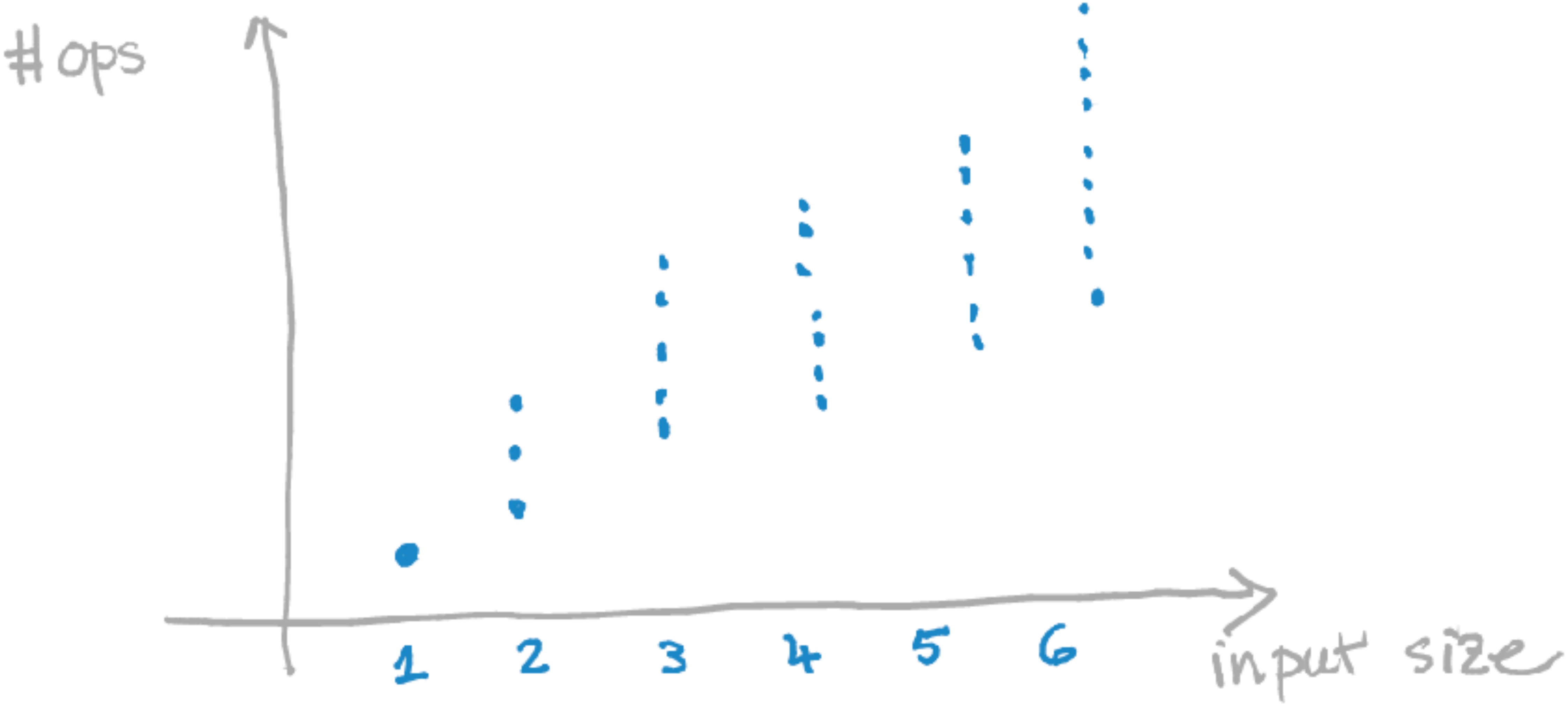


$g(n)$ is $O(f(n))$

$g(n)$ is $\Omega(f(n))$ ⟷ $g(n)$ is $\Theta(f(n))$

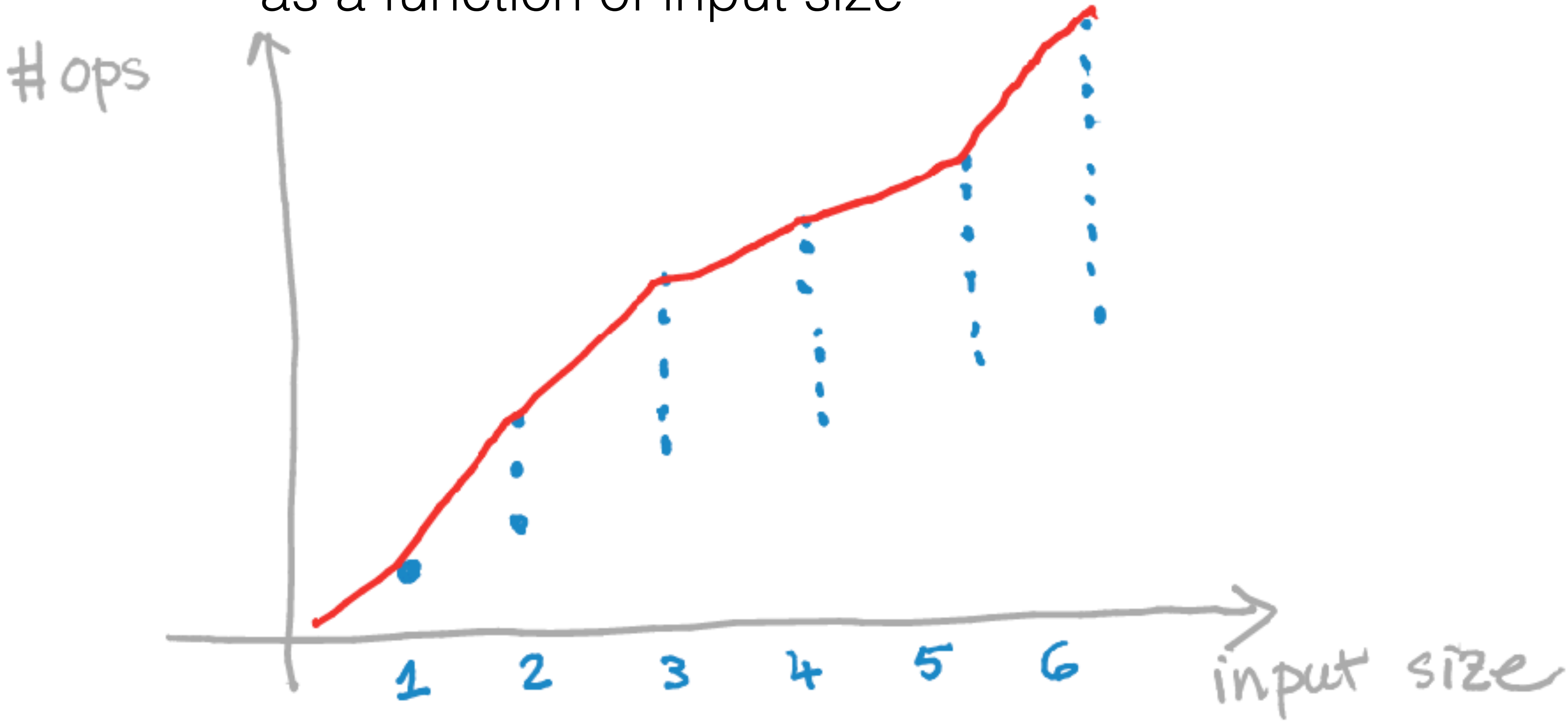
Time complexity (take 1)

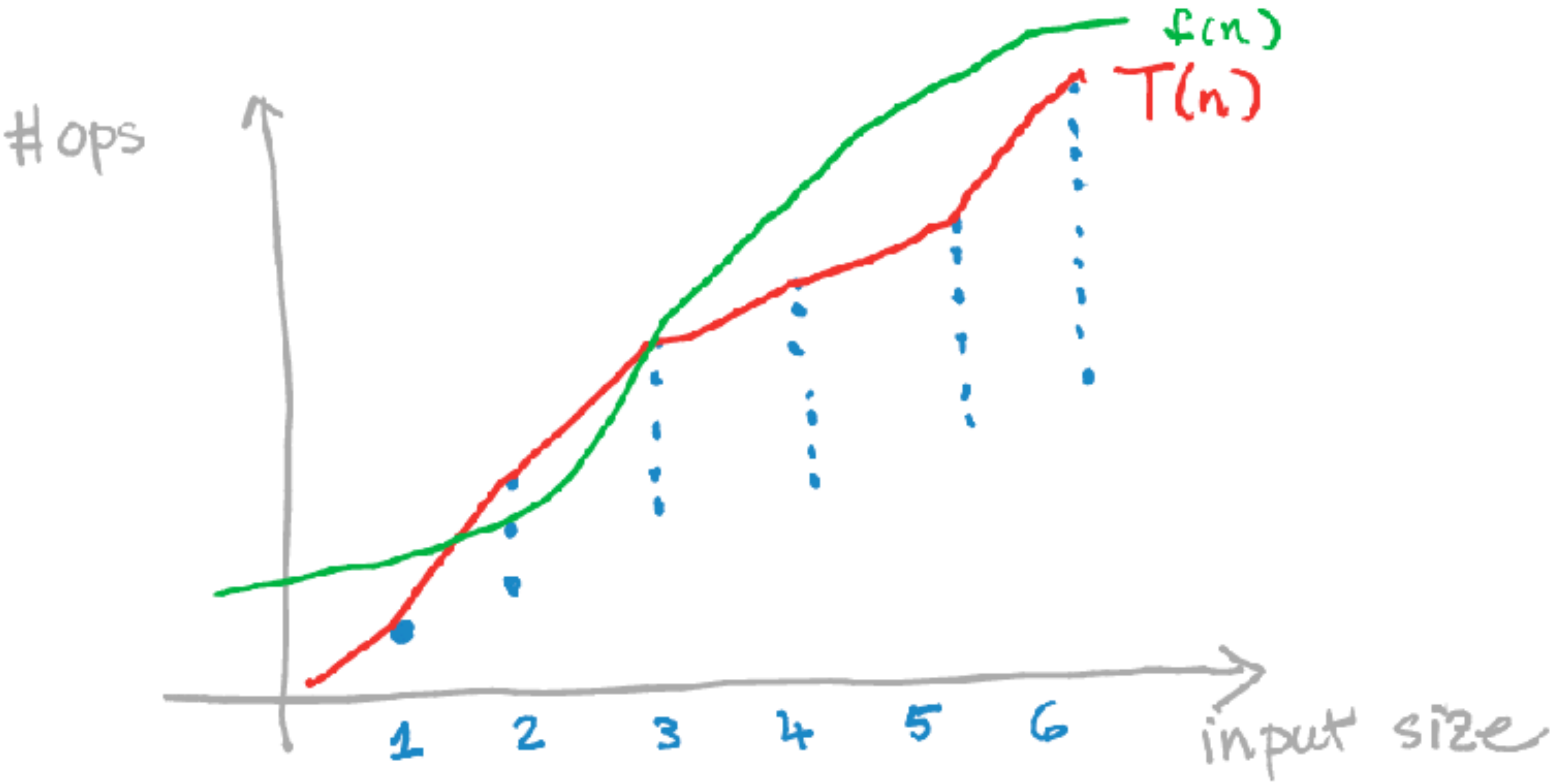
- The time complexity of an algorithm is
  - the number of basic operations
  - as a function of input size



Time complexity (take 2)

- The **worst-case time complexity** of an algorithm is
  - the **most** number of basic operations
  - as a function of input size



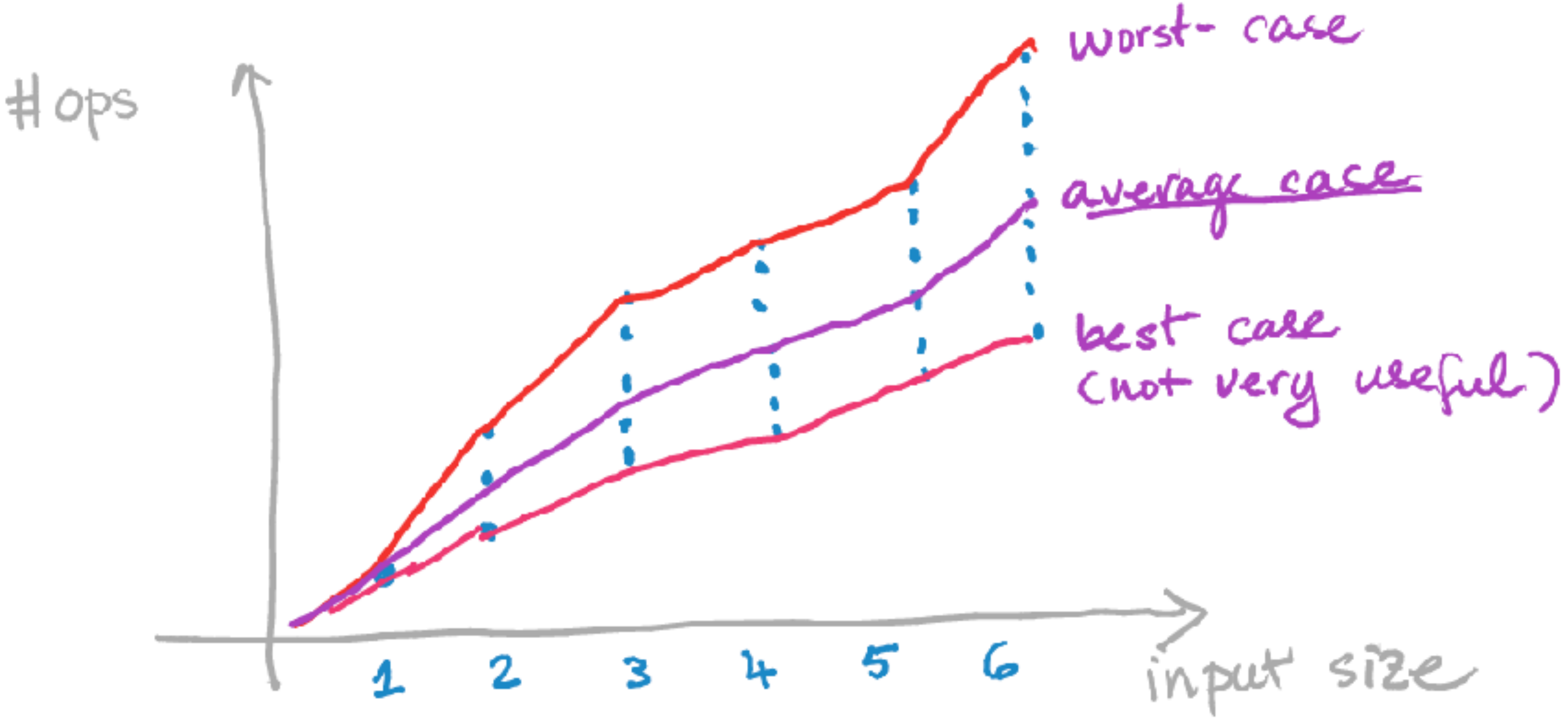


Actual worst-case time complexity = $T(n)$ ← typically complicated

if $T(n)$ is $O(f(n))$ ← something simple

we say "the worst-case time complexity of alg A is $O(f(n))$"

More time complexity

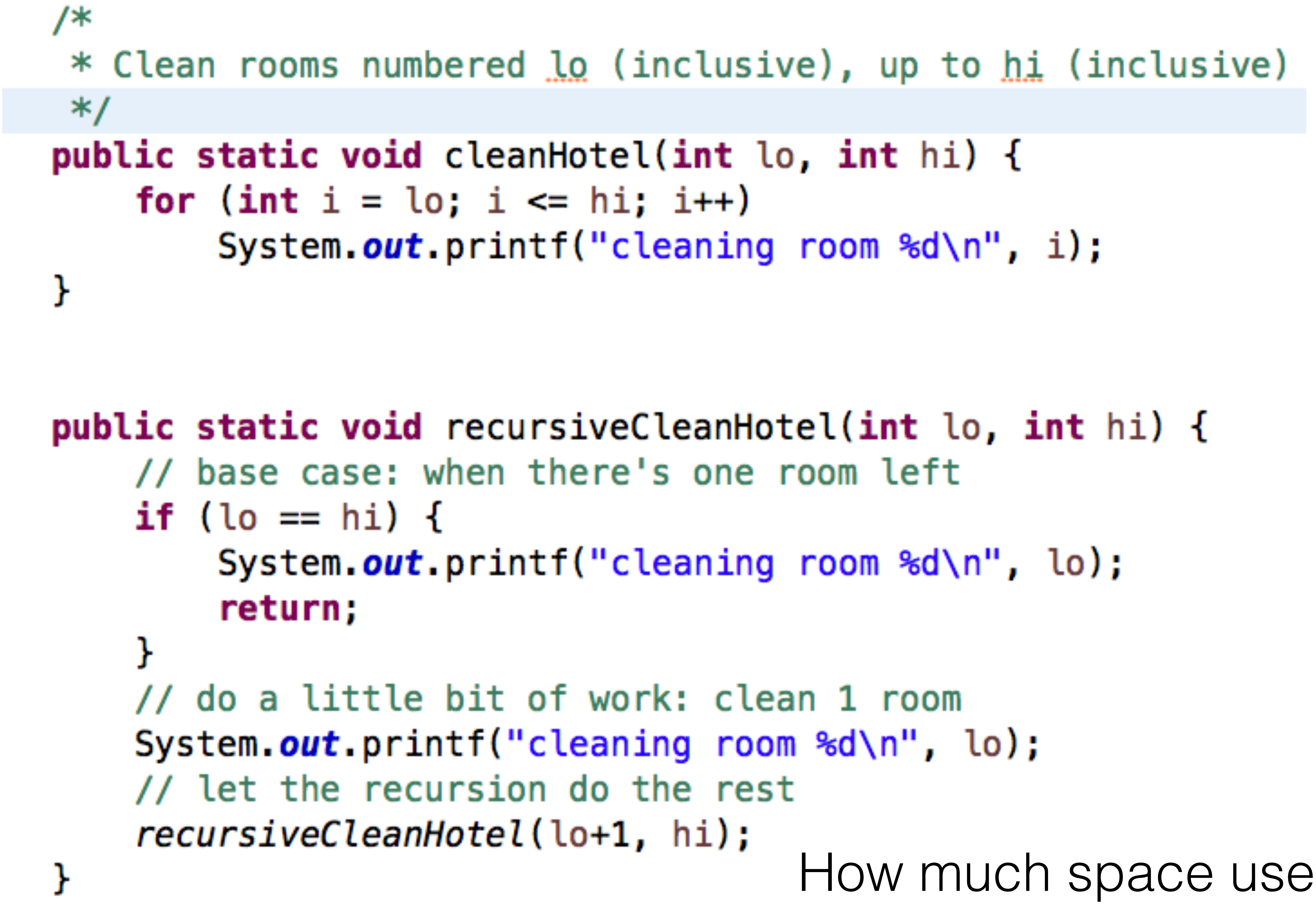


O-notation can be used to talk about **any** time complexity
not just worst-case time complexity

What’s the time complexity?

```
boolean isPrime(int n) {
    for (int i = 0; i*i <= n; i++)
        if (n % i == 0)
            return false;
    return true;
}
```

How many steps?



```
/*
 * Clean rooms numbered lo (inclusive), up to hi (inclusive)
 */
public static void cleanHotel(int lo, int hi) {
    for (int i = lo; i <= hi; i++)
        System.out.printf("cleaning room %d\n", i);
}


public static void recursiveCleanHotel(int lo, int hi) {
    // base case: when there's one room left
    if (lo == hi) {
        System.out.printf("cleaning room %d\n", lo);
        return;
    }
    // do a little bit of work: clean 1 room
    System.out.printf("cleaning room %d\n", lo);
    // let the recursion do the rest
    recursiveCleanHotel(lo+1, hi);
}
```

How much space used?

How many meows?

```
void talk(int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            for (int k = 0; k < 100000000; k++)
                meow()
}
```

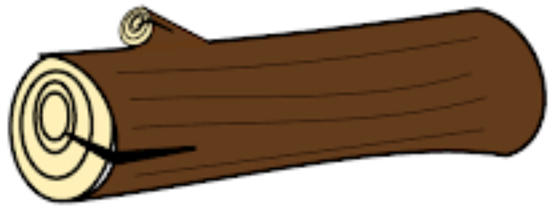
How many barks?

```
void talk(int n, int m) {

     for (int i = 0; i < n; i++)

          for (int j = 0; j < m; j++)

               meow()

     for (int k = 0; k < n+m; k++)

          meow()

}
```

How many tweets?

```
void sing(String[] song) {
    int n = song.length() - 1;
    while (n >= 1) {
        tweet(song[n]);
        n = n/2;
    }
}
```

What does log mean?



- It's the inverse of the exponential function of the same base

$$\log_3(3^x) = x$$

$$10^{\log_{10} B} = B$$

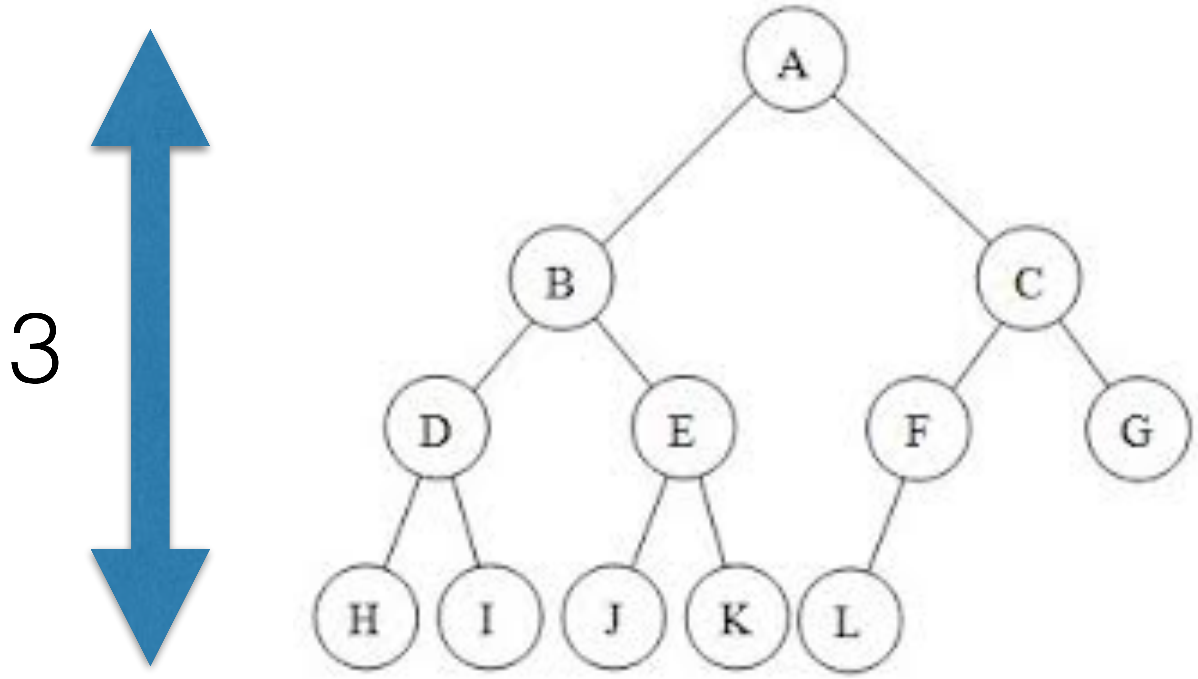
What dos log mean?

- (roughly) Number of times one needs to divide a number by to go down to 1.

How many times do you need to subdivide this array of n elements to get a subarray of size 1?

What if you divide into 3 equal parts each time?

- What is the length of the number 1,000,000 represented in base-16?



- What is the maximum height of a binary tree with n leaves?

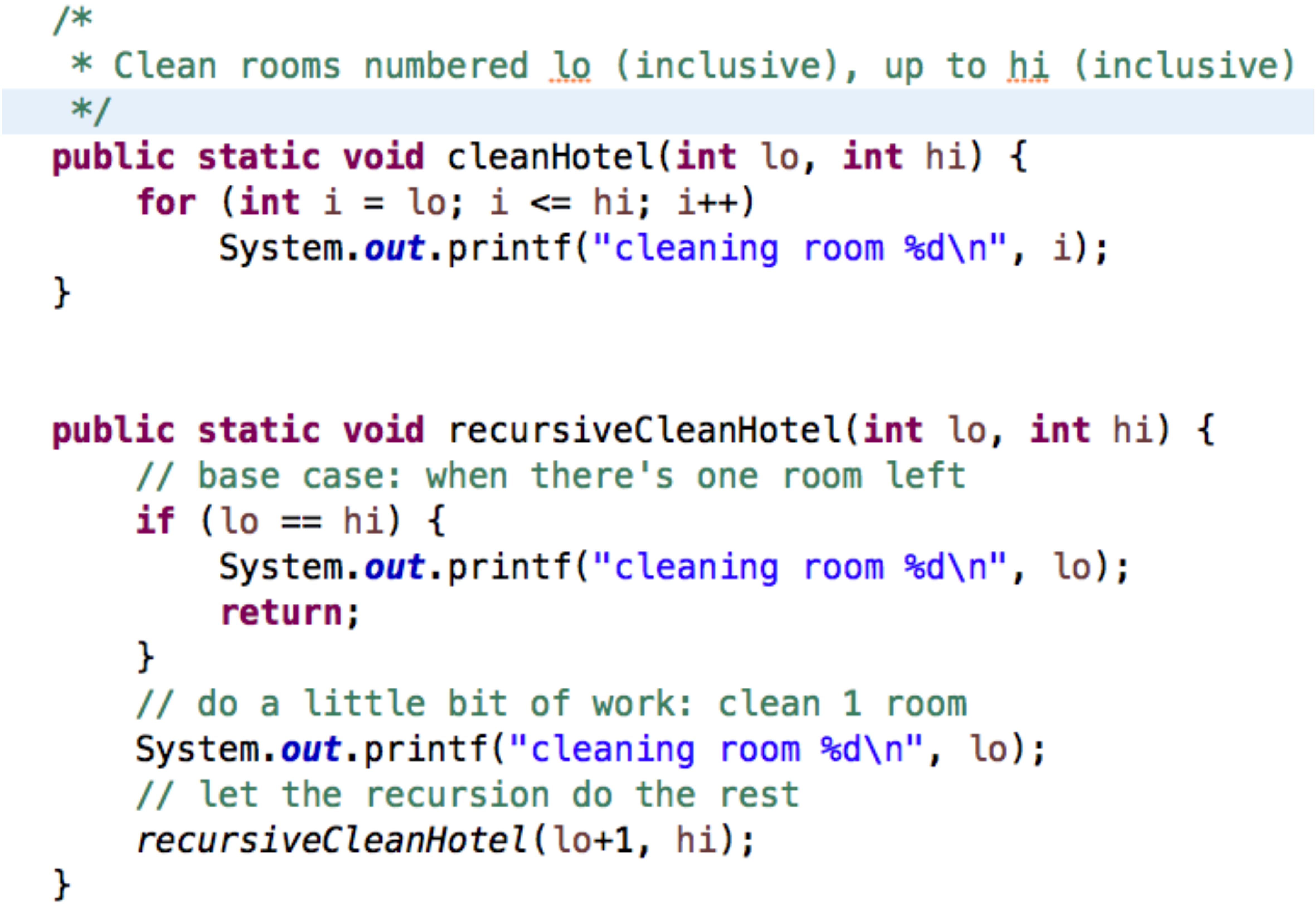
- What about minimum height?

How many steps?

```
int foo(int n) {
    if (n == 1) return 2;
    int a = foo(n/2);
    return a * a;
}
```

- Parting thoughts

Recall: How many steps?



```
/*
 * Clean rooms numbered lo (inclusive), up to hi (inclusive)
 */
public static void cleanHotel(int lo, int hi) {
    for (int i = lo; i <= hi; i++)
        System.out.printf("cleaning room %d\n", i);
}


public static void recursiveCleanHotel(int lo, int hi) {
    // base case: when there's one room left
    if (lo == hi) {
        System.out.printf("cleaning room %d\n", lo);
        return;
    }
    // do a little bit of work: clean 1 room
    System.out.printf("cleaning room %d\n", lo);
    // let the recursion do the rest
    recursiveCleanHotel(lo+1, hi);
}
```

Analyzing by counting steps - the fine print

- when deciding what is a step, we are assuming a model of computation

- a model of computation is a simplification of reality

- the model may not be perfect, but we can gain insight from it.