Recall "Simple" approach

Exact step count $\Rightarrow \Theta$

Alternate approach

Upper bound $\Rightarrow \Omega$  $\Rightarrow$ If same $\downarrow \Theta$

Lower bound $\Rightarrow \Omega$

Example

```python
1. def twisty(n: int) $\Rightarrow$ None:
2.     # Pre: $n \geq 0$
3.     x = n
4.     while $x > 1$:
5.         if $x \% 2 == 0$:
6.             $x = x / 2$
7.         else:
8.             $x = 2 * x - 2$
```

Helper Statement

For all values of $x$ greater than 2,
after two iterations the value decreases by at least 1.

**Proof**

Let \( x_0 \) be the current value of \( x \). Assume \( x_0 > 2 \).

Let \( x_1 \) be the value of \( x \) after 1 iteration, and \( x_2 \) the value after 2 iterations.

We want to prove that \( x_0 - x_2 \geq 1 \) or \( x_2 \leq x_0 - 1 \).

We'll use proof by cases for the remainder when \( x_0 \) is divided by 4.

\[
\begin{align*}
x_0 &= 14 \\
x_1 &= 7 \\
x_2 &= 12
\end{align*}
\]

Case 1: Assume \( \exists k \in \mathbb{Z}, x_0 = 4k + 2 \).

Because \( x_0 \) is even, Line 6 executes, and so

\[
x_1 = \frac{x_0}{2} = 2k + 1.
\]

Then at the next iteration, since \( x_1 \) is odd, Line 8 executes, and
\[ x_2 = 2x_1 - 2 \\
  = 2(2k+1) - 2 \\
  = 4k + 2 - 2 \\
  = 4k = x_0 - 2 \]

So then \( x_k \leq x_0 - 1 \).

(See Course Notes)

---

**Analysis**

Using the previous statement, the value of \( x \) after 2\( k \) iterations is

\[ x_{2k} \leq n - k \]

at most

\[ x_{2(n-1)} \leq n - (n-1) = 1 \]

So then after \( 2(n-1) \) iterations, \( x \) is \( \leq 1 \), so the loop will stop.

\[ \ldots \quad \mathcal{O}(n) \text{ runtime.} \]

---

**Worst-case runtime**

Before: runtime depends only on input size

Now: runtime depends on input size and
Now: runtime depends on input size and the actual input value

Example

```python
def is_palindrome(s: str) -> bool:
    # Return whether s is a palindrome.
    for i in range(len(s)):
        if s[i] != s[len(s) - 1 - i]:
            return False
    return True
```

(review pp.104-105) Let `func` be an algorithm. 
`f(n)` is an upper bound on the worst-case runtime of `func`:

\[\forall n \in \mathbb{N}, \forall x \in \mathcal{P}_n, \text{runtime of } \text{func}(x) \leq f(n)\]
\( \forall n \in \mathbb{N}, \forall x \in I, \text{ runtime of } \text{func}(x) \leq f(n) \)

\( f(n) \) is a lower bound on the w.c. runtime of \( \text{func} \):

\( \forall n \in \mathbb{N}, \exists x \in I, \text{ runtime of } \text{func}(x) \geq f(n) \)

(Warmup, informal) Analysis of the worst-case runtime of \( \text{is-palindrome} \).

**Part 1: upper bound**

Let \( n \in \mathbb{N} \). Let \( s \) be an arbitrary string of size \( n \).

We'll find an upper bound on the runtime of \( \text{is-palindrome}(s) \).

Ignoring the early return, we know there are at most \( n \) iterations of the loop.

So the runtime is \( O(n) \), and so the worst-case runtime is \( O(n) \).
Part 2: lower bound

Let $n \in \mathbb{N}$. Let $s =$ the string consisting of $n$ `a` letters. Or, $s = a^m \ldots a^n$ input family.

So for this value of $s$, the "if" condition is always false, so the loop runs $n$ times.

So the runtime of is_palindrome on this $s$ is $\Omega(n)$, so the worst-case runtime is $\Omega(n)$.

Example

```python
def palindrome_prefix(s: str) -> int:
    n = len(s)
    for pre_len in range(n, 0, -1):  # n, n-1, ..., 2, 1
        is_pal = True
        for i in range(pre_len):  # Loop 2
            if s[i] != s[pre_len-1-i]:
                is_pal = False
                break
        if is_pal:
            return pre_len
```

Check if $s[:pre_len]$ is palindrome.
return pre_len

Analysis at worst-case runtime

Part 1 Upper bound

Let $n \in \mathbb{N}$. Let $s$ be a string of length $n$.

Ignoring early “loop stops”,

Loop 2 iterates at most $pre_len$ times,

and Loop 1 iterates at most $n$ times, for $pre_len = n, n-1, \ldots, 2, 1$.

[Same calculations as previous nested loops] for $s$

So total runtime is $O(n^2)$, so worst-case runtime is $O(n^2)$.

Part 2 Lower bound

Let $n \in \mathbb{N}$. Let $s = \underline{\text{Rough work}}$.
\[
\begin{align*}
\text{Rough work} \\
S &= \underbrace{aa \ldots a}_n \Rightarrow \Omega(n) \\
S &= \underbrace{a \underbrace{bb \ldots b}_{n-1}}_n \Rightarrow \Omega(n) \\
S &= \underbrace{aa \ldots a}_{n^c} \ \text{index} \ \frac{n}{2} + 1 \\
\frac{n}{2} + \left(\frac{n}{2}-1\right) + \left(\frac{n}{2}-2\right) + \cdots
\end{align*}
\]