```python
def f(n):
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x / 2
        else:
            x = 2 * x - 2
        x odd
        x → 2x - 2
        2x - 2 → x - 1
```

- Unpredictable loop counter change
- Use “patterns” (analysis) to find upper & lower bounds on # of iterations

After $\frac{2^3}{2}$ iteration, $x$ decreases by at least 1.

There will be at most $2(n-1)$ iterations.
leads to $O(n)$ runtime.

Then $x \leq 10^{10}$

... Then $x \leq 100$

Worst-case and best-case runtimes (code)

For previous functions, runtime is a function of input size.

BUT not all programs are like this.
def all_even(L):  # L is a list of ints
    for i in L:
        if i % 2 == 1:
            return False
    return True

# For lists, treat size as the length of the list, i.e., n = len(L).

Problem runtime doesn't just depend on n—it depends on the contents of L.

the worst-case running time (max)
\[
W_{C_p}(n) = \max \left\{ \text{runtime of } f(x) \mid x \text{ has size } n \right\}
\]

\[
B_{C_p}(n) = \min \left\{ \text{subjects, functions being described} \right\}
\]
Possible: $WC_f \in O(n)$ and $WC_f \in \Omega(n) \rightarrow WC_f \in \Theta(n)$

$BC_f \in \Theta(n)$

1. Proving upper bounds on $WC_f(n)$
   [e.g., $WC_f(n) \in O(n)$].

   This is a universal proof.
   We’ll show the runtime for any input of size $n$ is $\leq n$ steps.

   This argument is the same as what we’ve done already.

   For all events, the loops runs at most $n$ times (where $n = \text{len}(L)$).
   So the total # of steps is $\leq n$, which is $O(n)$. 
2. Prove that \( WC_p \in \Omega(n) \) [lower bound on \( WC_p \)].

We show there exists a family of inputs (one per input size) whose runtime is \( \Omega(n) \).

Define the input family as follows: \( \forall n \in \mathbb{N} \), pick the list of length \( n \) that contains all 0's.

In this case, the loop doesn't stop early (because the variable \( i \) is never odd), so it takes \( n \) steps, which is \( \Omega(n) \).

Harder

\[ \text{def palindrome-prefix(s)} \]
def palindrome_prefix(s):
    """Return the length of the longest prefix palindrome."""
    n = len(s)
    for k in range(n, 0, -1):
        # check if s[0:k] is a palindrome
        is_pal = True
        for j in range(k):
            if s[j] != s[k-j-1]:
                is_pal = False
                break
        if is_pal:
            return k

stop early if is palindrome
stop early if is not palindrome