Week 9: Worst-Case Running Time Analysis

Before: runtime depends only on input size

![Graph showing runtime vs. input size]

Now: runtime depends on input size and value

![Graph showing runtime vs. input size]

Definitions

Let \( \text{func} \) be an algorithm.

For each \( n \in \mathbb{N} \), \( I_{\text{func}, n} \) is the set of inputs of size \( n \).

\[
\text{WC}_{\text{func}}(n) = \max \{ \text{runtime of func}(x) \mid x \in I_{\text{func}, n} \}
\]

\( \Rightarrow \text{worst-case running time} \)

\( f_1 \) is an upper bound on \( \text{WC}_{\text{func}} \):

\[
\forall n \in \mathbb{N}, \forall x \in I_{\text{func}, n}, \text{runtime of func}(x) \leq f_1(n)
\]

\( f_2 \) is a lower bound on \( \text{WC}_{\text{func}} \):

\[
\forall n \in \mathbb{N}, \exists x \in I_{\text{func}, n}, \text{runtime of func}(x) \geq f_2(n)
\]

Example

Analyse the worst-case running time of:

```python
1 def is_palindrome(s: str) -> bool:  
2     abba  
3     ccc  
4     d
5     for i in range(len(s)):  
6         if s[i] != s[len(s)-1-i]:  
7             return False  
8     return True  
9
10 def reverse(s: str) -> str:  
11     if len(s) == 0:  
12         return s  
13     return reverse(s[1:]) + s[0]
```

Analysis

Part 1: analysing an upper bound on \( \text{WC} \)

Let \( n \in \mathbb{N} \). Let \( s \) be an arbitrary string.
of length n. We'll analyse the runtime for s.

The loop runs at most n iterations
\( i = 0, 1, \ldots, n-1 \).
Each loop iteration takes 1 step (constant time), so the loop takes at most n steps total.
Line 5 takes at most 1 step.
So the total runtime is at most n+1, which is \( O(n) \).

Part 2: analysing a lower bound on WC

Let \( n \in \mathbb{N} \). Let \( s = \) the string of \( n \) 'a' characters.
We'll analyse the runtime for this \( s \).

In this case, the if condition
\[
\text{\( s[i] \neq s[\text{len}(s) - 1 - i] \)}
\]
is never true, so the loop doesn't stop early.
So the loop runs (exactly) \( n \) times.

So the runtime for this input \( s \) is \( n+1 \) steps, which is \( \Omega(n) \).

Conclusion
So the worst-case runtime is \( \Theta(n) \) and \( \Omega(n) \), and so it is \( \Theta(n) \).

Example Analysing the worst-case runtime of...
```python
def palindrome_prefix(s: str) -> int:
    n = len(s)
    for pre_len in range(n, 0, -1):  # Loop 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
```

Analysis

**Part 1: upper bound on worst-case**

Let $n \in \mathbb{N}$ and let $s$ be an arbitrary string of length $n$.

**Informal**

It's still true that

1. Loop 2 iterates at most $\text{pre-len}$ times.
2. Loop 1 iterates at most $n$ times.

We still get an upper bound of $O(n^2)$.

**Part 2: lower bound on worst-case**

Let $n \in \mathbb{N}$. Let $s =$ ____________________.
### Rough work

\[ s = \overbrace{aaa \ldots aaa}^{n \text{ times}} \rightarrow \Theta(n) \]

\[ s = \overbrace{aaaabaaa}^{(n=8)} \]

### Loop 1:

<table>
<thead>
<tr>
<th>pre-len</th>
<th># of iterations Loop 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4 (is palindrome, so stops after)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{pre-len} & \quad \text{# iterations Loop 2} \\
\frac{n}{2} & \quad \frac{n}{2} - 1 \\
\vdots & \quad \vdots \\
\frac{n}{2} + 1 & \quad 1 \\
\frac{n}{2} & \quad \frac{n}{2}
\end{align*}
\]

Sum is \( \Theta(n^2) \)

### Last example!

Analyse the runtime of:

```python
def twisty(n: int) -> None:
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x // 2
        else:
            x = x * 2
```

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n=9 )</th>
<th>( n=7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( x=9 )</td>
<td>( x=7 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( x = 16 )</td>
<td>( x = 12 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( x = 8 )</td>
<td>( x = 6 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( x = 4 )</td>
<td>( x = 3 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( x = 2 )</td>
<td>( x = 4 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( x = 1 )</td>
<td>( x = 1 )</td>
</tr>
</tbody>
</table>
\[ x = \frac{x}{2} \]

else:
\[ x = 2 \times x - 2 \]

**Helper Statement** \( \forall x \in \mathbb{N}, \text{ if } x > 2 : \)

Let \( x_0, x_1, x_2 \) be the current value of \( x \), the value after 1 iteration, and the value after 2 iterations, respectively.

Then \( x_2 \leq x_0 - 1 \).

**Proof (slightly informal)**

**Case 1: assume \( x_0 \) is odd**

Then the else branch executes, and we know
\[ x_1 = 2x_0 - 2. \]

So by the definition of divisibility, \( x_1 \) is even, and so at the next iteration the if branch executes.

So
\[ x_2 = \frac{x_1}{2} \]
\[ = \frac{2x_0 - 2}{2} \]
\[ = x_0 - 1 \]

So \( x_2 \leq x_0 - 1 \) in this case.