PS3

- next week
- exam schedule

Problem: describe the runtime of an algorithm as a function of its input.

Approach:

1. Identify blocks of code with runtime independent of input.
2. Identify loops and determine exact iteration counts.
3. Assemble terms to get an expression for runtime.
4. Use asymptotic expressions (e.g. \( \Theta(n) \), \( O(n) \), \( \Omega(n) \)) to describe runtime.

Before (last example): \# iterations a simple function of \( n \)
```python
def nested_2(n: int) -> None:
    for i in range(n):
        for j in range(i):
            print(i+j)
```

- The total number of basic operations is:
  
  \[
  0 + 1 + 2 + \ldots + (n-1) \\
  i=0 \quad i=1 \quad i=2 \quad i=n-1 \\
  \sum_{i=0}^{n-1} i \\
  = \frac{(n-1)(n-1+1)}{2} \\
  = \frac{n(n-1)}{2} \\
  = \frac{1}{2}n^2 - \frac{1}{2}n \quad \text{basic operations}
  \]

\[\in \Theta(n^2) \quad \text{... how to justify conclusion}\]
Need to prove

\[ \frac{1}{2} n^2 - \frac{1}{2} n \in \mathcal{O}(n^2) \land \frac{1}{2} n^2 - \frac{1}{2} n \in \mathcal{O}(n^2) \]

\[ \text{want } \frac{n(n-1)}{2} \geq C_1 n^2 \]

\[ \frac{n(n-1)}{2} = \frac{n}{2} \cdot (n-1) \]

\[ n-1 = \frac{1}{2} n + \left( \frac{1}{2} n - 1 \right) \]

\[ \geq \frac{1}{2} n \text{ when } n \geq 2 \]

\[ \geq \frac{n}{2} \cdot \frac{n}{2} \]

\[ = C_1 n^2 \text{ when } C_1 = \frac{1}{4}, \ n \geq 2, \ n_1 = 2 \]

\[ \text{want } \frac{n(n-1)}{2} \leq C_2 n^2 \]

\[ \frac{n(n-1)}{2} \leq \frac{n}{2} \cdot n \]

\[ = \frac{1}{2} n^2 \text{ when } C_2 = \frac{1}{2}, \text{ when } n_2 = 1 \]
```python
def even_or_odd(n: int) -> None:
    if n % 2 == 0:  # n is even
        for i in range(n * n):
            print(i)
    else:  # n is odd
        for i in range(n):
            print(i)
```

- When n is even, the runtime is $\Theta(n^2)$.
- When n is odd, the runtime is $\Theta(n)$.
- For arbitrary $n \in \mathbb{N}$.

Conclusion: runtime $\in O(n^2) \lor$ runtime $\in \Omega(n)$
```python
def smallest_factor(n: int) -> int:
    
    """Return the smallest non-trivial factor of n or -1 if none.
    Precondition: n >= 2""

    d = 2
    while d < n:
        if n % d == 0:
            return d
        d = d + 1
    return -1
```

When \( n \) is even:
- \# iterations: 1
- Runtime is constant

When \( n \) is prime:
- \# iterations is \( n - 2 \)
- Runtime varies with \( n \)

When \( n \) is odd and not prime:
- \# iterations varies between 1 and \( n - 2 \)

So runtime is \( \Omega(1) \) and \( O(n) \)
In this case, there is no elementary function $g(n)$ s.t. the runtime is $\Theta(g)$.
def collatz(n: int) -> int:
    """ Precondition: n > 0 """

    steps = 0
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x // 2
        else:
            x = 3 * x + 1
        steps += 1
    return steps

Q: Does this algorithm terminate for every n ∈ N?

An open question!

Consider this variant?
A variant that always terminates:

```python
def f(n: int) -> int:
    """Precondition: n > 0"""
    steps = 0
    x = n
    while x > 1:
        if x % 2 == 0:
            x = x // 2
        else:
            x = 3 * x - 2
    return steps
```

Goal: Find a function \( h(n) \) s.t.
the runtime of \( f(\RT_f(n)) \) \( \in \Theta(h(n)) \)

i.e. \( RT_f(n) \in O(h(n)) \land RT_f(n) \in \Omega(h(n)) \)

unwrap:

1. Find an \( h_1(n) \) s.t. \( RT_f(n) \in O(h_1(n)) \)
2. Find an \( h_2(n) \) s.t. \( RT_f(n) \in \Omega(h_2(n)) \)

Desire \( h_1(n) = h_2(n) \) so \( RT_f(n) \in \Theta(h_1(n)) \)
<table>
<thead>
<tr>
<th>Try values: n=10</th>
<th># if</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>4</td>
</tr>
</tbody>
</table>