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

Approach:

- Identify blocks of code with runtime independent of input
- Identify loops and determine exact iteration count
- Assemble terms to get an expression for runtime
- Use an asymptotic expression (using $\Theta, O, \Omega$) to describe the runtime

- In earlier examples - nested loops - inner loop runtime a simple $\Theta(n)$
```python
def nested_2(n: int) -> None:
    for i in range(n):
        for j in range(i):
            print(i+j)
```

- analyze as before
- discuss

- total # of basic operations:

\[
\sum_{i=0}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2} = \frac{1}{2} n^2 - \frac{1}{2} n \quad \text{basic operations}
\]
\[ \varepsilon \in \Theta(n^2) \]

**Proof:**

\[ \frac{1}{2} n^2 - \frac{1}{2} n \in \Omega(n^2) \]

\[ \wedge \]

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

\[ \Omega(n^2): \quad \frac{1}{2} n^2 - \frac{1}{2} n \geq c_1 n^2 \]

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

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

\[ > c_1 n^2 \quad \text{for } n \geq 2 \quad c_1 = \frac{1}{4} \quad n_1 = 2 \]

\[ O(n^2): \quad \frac{1}{2} n^2 - \frac{1}{2} n \leq c_2 n^2 \]

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

\[ \leq \frac{1}{2} n \cdot n \]

\[ \leq c_2 n^2 \quad c_2 = \frac{1}{2} n_2 = 1 \]
def even_or_odd(n: int) -> None:
    if n % 2 == 0:  # n is even
        for i in range(n * n):
            print(i)  # n^2 iterations
    else:  # n is odd
        for i in range(n):
            print(i)  # n iterations

    when n is even, the runtime is $\Theta(n^2)$

    when n is odd, the runtime is $\Theta(n)$

But for an arbitrary $n \in \mathbb{N}$

There is no $h$ s.t. runtime $\in \Theta(h)$

runtime $\in \Omega(n)$

runtime $\in O(n^2)$
A practical example:

```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
```

Alternate:

- d < n/2  won't change conclusion
- d < \(\sqrt{n}\)

Basic operation:

- \(\mathcal{O}(n)\) \(\mathcal{O}(n^{1/2})\)

When \(n\) is even: # iterations is 1

(at least \(n=2\))

runtime is constant

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

runtime varies with \(n\)

When \(n\) is odd and not prime:

# iterations between 1 and \(n-2\)

runtime of algorithm is \(\mathcal{O}(n)\) and \(\mathcal{O}(n^{1/2})\)
It turns out that there is no elementary function $g(n)$ such that the runtime is $\Theta(g(n))$. A simple function without cases.
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 all n? 

A: This is an open question!

\[ \text{instead consider:} \]
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 = 2 * x - 2
    return steps
```

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

i.e. $RT_f(n) \in O(h(n))$

\[ RT_f(n) \in \Omega(h(n)) \]

1. Find an $h_1(n)$ s.t. $RT_f(n) \in O(h_1(n))$

2. Find an $h_2(n)$ s.t. $RT_f(n) \in \Omega(h_2(n))$

Wish: $h_1(n) = h_2(n)$ to conclude $RT_f(n) \in \Theta(h(n))$
Consider:  n = 10  \# iterations = 5

13  
16  

?  
4