Recall our goal:

Given an algorithm, find an approximately # of steps, as a function of input size, for large inputs.

When we perform a runtime analysis of a algorithm, we almost always communicate a \( \Theta(\cdot) \) as the result.

**Examples of "step" - constant time operations**

1. Arithmetic +, -, *
2. Comparison (of numbers) ==, <=, >=
3. Assignment statements and variable lookup
   \( y = x + 3 \)
4. Return statements and passing arguments to a function call.
Examples of code that can take non-constant time (i.e., take longer as input size grows)

1. Loops
2. Helper functions
3. Operations on compound data structures
4. Recursive functions

Analysing loop running time

Key idea: add up the total runtime from each loop iteration.

Example

```python
def f(n: int): # n ≥ 0
    for i in range(n):
        for j in range(n * n):
            print(i + j)
```

Analysis

Start with innermost loop, and work our
way out.

For Loop 2:
1) The loop body (print(i+j)) takes constant time, so it counts as 1 step per iteration.
2) There are \( n^2 \) iterations in total (\( j=0...n-1 \)).

So the total time is

\[
\frac{1+1+1+1+\cdots+1}{\text{n}^2} = \frac{n^2 \times 1}{n^2} = n.
\]

But this is only for one iteration at Loop 1.

So for Loop 1:
1) Each iteration takes \( n^2 \) steps (the cost of Loop 2)
2) There's \( n \) iterations at Loop 1. (\( i=0...n-1 \))

\[
\frac{n^2 + n^2 + n^2 + \cdots + n^2}{n} = \frac{n^2 \times n}{n} = n^3.
\]

So the total runtime is \( \Theta(n^3) \).

Example
Example
def f1(n: int):
    for i in range(n):
        for j in range(i * i):
            Loop 1
            print(i+j)

We start with Loop 2, assuming we're on iteration i of Loop 2.

[Similar to Loop 2 analysis in previous example]

Loop 2 takes $i^2$ steps.

So for Loop 1:
1. Iteration $i$ takes $i^2$ steps (the cost of Loop 2)
2. $i$ takes on the values 0, 1, ..., $n-1$.

Cost: $0^2 + 1^2 + 2^2 + \ldots + (n-1)^2$

$$
\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6}
$$
So the runtime of $f_1$ is $\Theta(n^3)$.

Recap Basic strategy for analysing runtime:

① Determine an "exact" step count
   (e.g., by analysing loops)

② Go from the exact step count to
   a $\Theta$ expression.

Example

```python
def is_prime(n: int) -> bool:
    # n ≥ 2 (precondition)
    for d in range(2, n):
        if n % d == 0:
            return False
    return True
```

Analysis For the loop:

① Each iteration takes 1 step
   (constant time)

② The number of iterations is:
   - at least 1 iteration
- at most \( n-2 \) iterations

\underline{Part 1: Lower bound}

Since the loop takes at least 1 iteration, the total number of steps is

\[
\begin{align*}
1 & = 1 \\
\end{align*}
\]

So the total runtime of is_prime is at least 1... So the runtime is \( \Omega(1) \).

\underline{Part 2: Upper bound}

Since the loop takes at most \( n-2 \) iterations, the total cost is

\[
\left(1 + 1 + 1 + \ldots + 1\right) = n-2.
\]

So the runtime of is_prime is at most \( n-2 \).

So the runtime is \( O(n) \).

\underline{Alternate strategy}

\( \Omega \) is \( \Omega \) lower bound on the
1. Find a lower bound on the # of steps.
2. Go from lower bound to $\Omega$.
3. Find an upper bound on the # of steps.
4. Go from upper bound to $O$.

**Example**

```python
def print_primes(n):
    for k in range(2, n+1):
        if is_prime(k):
            print(k)
```

**Analysis**

- For the loop:
  1. $k$ goes from 2 to $n$.
  2. Let $RT_{ip}(x)$ be the runtime function of `is_prime`.

  The cost of an iteration of the loop is $RT_{ip}(k)$.

  So the total cost for the loop is $RT_{ip}(2) + RT_{ip}(3) + RT_{ip}(4) + \ldots + RT_{ip}(n)$.
\[ RT_{i,p}(2) + RT_{i,p}(3) + RT_{i,p}(4) + \cdots + RT_{i,p}(n) = \sum_{k=2}^{n} RT_{i,p}(k) \]

**Part 1: Upper bound**

We know that \( RT_{i,p}(k) \leq k-2 \) (for all \( k \geq 2 \)).

So then:

\[ \sum_{k=2}^{n} RT_{i,p}(k) \leq \sum_{k=2}^{n} (k-2) = \mathcal{O}(n^2) \]

**Part 2: Lower bound**

We also know \( RT_{i,p}(k) \geq 1 \).

\[ \sum_{k=2}^{n} RT_{i,p}(k) \geq \sum_{k=2}^{n} 1 = n-1 \in \Omega(n) \]

**Part 3: A better lower bound**

Key idea: when \( k \) is prime, \( RT_{i,p}(k) = k-2 \).
\[
\sum_{k=2}^{n \leq \text{k prime}} RT_p(k) \geq \sum_{2 \leq k \leq n, \text{k is prime}} RT_p(k) = \sum_{2 \leq k \leq n, \text{k is prime}} (k-2)
\]

\[
\Omega\left(\frac{n^2}{\log n}\right)
\]

Number Theory

Part 4: Outsmart David

(Is this even possible?)