Recall our goal: given an algorithm (code, pseudocode, python) give back an approximate number of steps, as a function of input size, for large inputs.

→ we report this as a $\Theta$ expression (by $\Omega, O, o$).

**Input size:** for lists + strings
  → length
  for integers (all numbers)
  → number.

**Steps:** one step (constant #)
  *arithmetic* $4+5$, $2*3$, $(\land, \lor)$
* assignment \( x = 5 \) (loops)
* printing
* returning
* indexing lists (slicing)
  \( \text{let} \; x = 5 \), \( \text{let} \; z = 10 \)

**Code that runs in non-constant time**

* helper functions \( \equiv 165 \)
* loops
* Data structure operation
  \( 148, 263, 373 \)
* Recursive calls
  \( 236 \)

**Example:**

```python
def foo(n):
    #size = n
    if n > 0:
        ...
```
for i in range(n):  
    for j in range(n*n):  
        print(i+j) * 1 step

i^2 step 1

inner loop: #1

\[ 1 + 1 + 1 + \ldots + 1 = n^2 \]

\[ n^2 \text{ times} \]

outer loop #1

\[ n^2 + n^2 + n^2 + \ldots + n^2 = n \cdot n^2 = n^3 \]

inner loop #2

\[ i = 0 \rightarrow \text{zero times} \]

\[ \frac{n(n+1)(n+2)}{6} = \sum_{i=1}^{n} i^2 \]
\[
\frac{1^2 + 2^2 + \ldots + n^2}{i^2 \ \text{times}} = \sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(n+1)}{6} \in \Theta(n^3)
\]

Thus:

\[\forall f, g : \mathbb{N} \to \mathbb{R}^+, \quad \text{if } g \in O(f) \implies g + f \in O(f)\]

Exact count: \[
2 + n^3 + \frac{n(n-1)(n-2)}{6} \geq 1
\]
\( C \in \Theta(n^3) \)

Strategy: count exact number of steps, apply them to get \( \Theta \) bound.
Recap: basic strategy
- get an exact step count based on the program code
- use the theorem (the one about addition) to get a Θ expression

→ e.g. $RT_f \in \Theta(n^3)$

```python
def is_primes(n):
    for d in range(2, n):
        if n % d == 0:
            return False
    return True
```
for d in range(2, h): x1
    if n % d == 0: x1
        return False x1
    # returns early
Return True

exact step count? NO.
get big-Oh and big-Omega
Ω: lower bound
on # steps: 2 \in \Omega(1)
Ο: upper bound
2(n-2)+1 steps
2 lines iter. time
4n - 3 ∈ O(n)

RT_{is} : runtime of

is_prime.

→ no Θ expression

New strategy:

1. find a lower bound
2. get Ω(g)
(3) find an upper bound
(4) get $O(f)$
optional: (3) if they match
get $O$, otherwise
↓
try to improve
$O \sqrt{\log}$

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

Lower bound:

$n-1$ iterations

$RTip_{prime} \in \Omega(1)$

$2, 3, \ldots, n$

$\Omega(n-1)$ steps

$\Downarrow$

$\Omega(n)$ steps

Upper bound:

Still $n-1$ steps

u.b. on $RTip$

$EO(n)$

Total $RTip_{prime} \in \Omega(n)$
Improving the bounds on print-primes:

\[ \sum_{k=2}^n RT_{ip}(k) \]
\[ \sum_{2 \leq k \leq n} R_{t, p}(k) = \sum_{\text{k is prime}} K \]

How many primes are there?

Number theory

\[ \frac{n}{\log n} \text{ primes} \leq n \]

\[ \in \mathcal{O}\left(\frac{n^2}{\log n}\right) \]

Not on the test! \( \Box \)

Improving the upper bound:

\[ n^2 \times \frac{n}{\log n} \times \frac{n}{10^7} \times \]