simple.

\[ f(n) = n^3 - 148n^4 + 165n^2 \]

\[ f \in \Theta(n^0) \quad \text{not say} \quad f \in \Theta(165n^0) \]

\[ f \subseteq O(n^4) \quad \text{simpler decay.} \]

Lecture: \[ g \subseteq O(f) \quad \text{e.g.} \quad 100n \subseteq O(n^2) \]

Eventually \[ g(n) \leq cf(n) \]

\[
\begin{array}{c}
g \subseteq \Omega(f) \\
\end{array}
\]

Eventually \[ g(n) \geq cf(n) \]

\[ g \subseteq \Theta(f) \]

Eventually \[ c_1 f(n) \leq g(n) \leq c_2 f(n) \]

\[ \Theta(n) : \text{a set of functions that eventually stay between two lines through origin} \]
```python
def print_items(lst: List[bool]) -> None:
    for item in lst:
        print(item)
```

def n:
    count the "basic operations"

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>prints</td>
</tr>
<tr>
<td>2n</td>
<td>print, item assignments</td>
</tr>
<tr>
<td>1.5n</td>
<td>print costs 10x, costs to assign</td>
</tr>
<tr>
<td>n + 1.5</td>
<td>incl. costs of call + return</td>
</tr>
</tbody>
</table>

↑ use Θ(n) to describe running time.
↑ allows ignoring scaling factors.
↑ describe the growth of the running time as Θ(n).

**def**: A "basic operation" is any block of code whose running time does not depend on the size of input data.
(Also referred to as "a step").
Examples of basic operations:

- Comparisons: ==, <, >
- Arithmetic: +, -, *, /
- Using a variable: \[ x = y \]
- print, return

Can prove:

"The running time of print items is \( \Theta(n) \) where \( n \) is the length of the list."

Proof: For this algorithm, each iteration of the loop can be counted as a single basic operation, because nothing in it depends on the size of the list.

- The running time depends on the number of loop iterations. Since this is a for loop over list argument, we know that the loop runs \( n \) times.

- Thus the total number of basic ops performed is \( n \times 1 \) and so the running time is \( \Theta(n) \).
Summary of approach:

1. Identify your measure of input/data size (e.g. length of list, # bits in int, ...)

2. Identify the blocks of code that can be counted as a single basic operation, since do not depend on input size (usually inside loops)

3. Identify any loops in the code that cause the basic operations to repeat. Figure out exactly how many times these loops run, based on the size of the input.

4. Combine these observations to get an expression for the number of basic operations e.g. $3n^2 + 10n + 7$

5. Convert this expression to big O notation. The running time is $\Theta(n^2)$ basic operations.
Consider

def print_sums(lst: List[float]) -> None:
    for item1 in lst:
        for item2 in lst:
            print(item1 + item2)

Prove that the function `print_sums` runs in time $\Theta(n^2)$, where $n$ is the length of the list.

**Proof:** Let $n$ be the length of the list.

- The inner loop (lines 3-4) runs $n$ times (once per item in `lst`) and each operation in the loop is a basic operation.
- The outer loop (lines 2-4) runs $n$ times, and each iteration takes $n$ basic ops.

So the total number of basic operations is: $\text{cost of inner loop} \times \text{number of repeats of inner loop}$

$= n \times n$ basic operations

$= n^2$
So the running time of this algorithm is $\Theta(n^2)$.

Consider

```python
1  def f(lst: List[int]) -> None:
2    for item in lst:
3      for i in range(10):
4        print(item + i)
```

Determine a description of the running time.

Proof: Let $n$ be the length of the input list `lst`.

- The inner loop (lines 3-4) repeats a basic operation 10 times for a total of 10 basic operations.
- The outer loop (lines 2-4) repeats $n$ times, and each iteration requires 10 steps, for a
total of 10 steps.
So the running time of this algorithm is $\Theta(n)$.

**alt:** label the body of outer loop as a single basic operation since runtime does not depend on n.

**lesson:** don't just look at loop nesting level
Consider

```python
def g(lst: List[int]) -> None:
    for item in lst:
        i = 0
        while i < len(lst):
            print(item + i)
            i = i + 2
```

determine a description of running time:

\[
    n \left( \lceil \frac{n}{2} \rceil + 1 \right) = n + n \left( \left\lfloor \frac{n}{2} \right\rfloor \right)
\]

Fact: \( \left\lfloor \frac{n}{2} \right\rfloor \in \Theta(n) \land \frac{1}{14} \ln n \in \Theta(n) \)

so running time is \( \Theta(n^2) \).

\( n \left\lfloor n^{1/2} \right\rfloor \in O(n^2) \)

\( n \left\lfloor n^{1/2} \right\rfloor \in \Omega(n) \)

\( n \left\lfloor n^{1/2} \right\rfloor \in \Theta(n^{1/2}) \)
\[ \forall c_0, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n > n_0 \Rightarrow n \lfloor \frac{n}{2} \rfloor \leq c_0 n^2. \]

Let \( c_0 = 1 \), \( n_0 = 1 \). Assume \( n \geq n_0 \).
\[
n \cdot \lfloor \frac{n}{2} \rfloor \leq n \cdot \lfloor \frac{2n}{2} \rfloor = n^2 = c_0 n^2
\]

```python
def c(lst: List[float]) -> None:
    print("Here is the list:")

    for item in lst:
        print(item)

for item1 in lst:
    for item2 in lst:
        print(item1 + item2)
```

1

ntimes

ntimes

ntimes

ntimes
Determine a description of the running time.

**Proof:** Let $n$ be the length of the list $L_0$.

- First print counts as 1 basic operation.
- First loop runs in time $\Theta(n)$.
- Second \( \Theta(n^2) \)

need to combine these:

**Theorem:** \( f \in \Theta(h) \land g \in o(h) \Rightarrow (f + g) \in \Theta(h) \)

**Example:**\( \quad n^2 \in \Theta(n^2) \)
\( n \in O(n^2) \)
\( 1 \in o(n^2) \)

Apply Theorem 2 times: \( 1 + n + n^2 \in \Theta(n^2) \)

This algorithm runs in time $\Theta(n^2)$.