Quick Review (but go to tutorial!!)

\[ g \in O(f) \quad "g \leq f" \]
\[ g \in \Omega(f) \quad "g \geq f" \]
\[ g \in \Theta(f) \quad "g = f" \]

\[ g(n) \leq cf(n) \]
\[ g(n) \geq cf(n) \]
\[ c_1 f(n) \leq g(n) \leq c_2 f(n) \]

\[ g \in O(n^{1000}) \]
\[ g \in \Theta(n^{1000}) \]
\[ g \neq n^{0.99} \]
\[ g \neq n^{0.5} \]
\[ g \neq \log n \]

Analysing the runtime of algorithms

Goal: Given a program, find an
Goal: Given a program, find an approximate # of steps, in terms of input size, as the input gets large.

\[ \Theta(?) \]

The answer is usually a \( \Theta \) expression.

input size: formally, the # of bits required to represent the input.

informally, use approximations like:
- for a natural number, its value
- for a list, its length

step: a block of code whose runtime doesn't depend on the input size.

Examples:
1. arithmetic \( +, -, *, /, \% \)
2. comparison of numbers \( \leq, \geq, = \)
3) Variable assignment and lookup

4) print

5) function call and return

Example

Find the asymptotic (Theta) runtime of the following function:

```python
def f(n:int) -> int:
    x = n + 1
    print(x * 2)
    return x * n
```

We can treat the entire body as a single step. It doesn't depend on how large n is!

The number of steps is 3 (for all n).
And \( I \in \Theta(1) \).
So the runtime is \( \Theta(1) \).

How does runtime grow?

1. Loops! \( \rightarrow \) CSC165!!
2. Recursive functions \( \rightarrow \) CSC236
3. Dealing with compound data structures \( \rightarrow \) CSC148, CSC263, 373, ...

When given a loop, do two things:

1. Identify \# of iterations
2. Identify \# of steps per iteration.

Ex: def f2(n):

for i in range(n):
    print(i)

1. This loop runs n times.
2. Each iteration takes 1 step (for print(i)).
   The total # of steps is n.
   So the runtime is \( \Theta(n) \).
1. There are $\left\lceil \frac{n}{2} \right\rceil$ iterations.
2. One iteration takes 1 step.

There are $\left\lceil \frac{n}{2} \right\rceil \times 1 = \left\lceil \frac{n}{2} \right\rceil$ steps total for the loop.

The initial $i=0$ counts as 1 step, for a total of $\left\lceil \frac{n}{2} \right\rceil + 1 \in \Theta(n)$ steps.

prove this!

( generalize:
$g \in O(f) \Rightarrow f + g \in O(f)$
)

Generalize:

$\text{def } g(n):$

$\left[ A \right] \left\lceil \frac{n}{2} \right\rceil$

$\left[ B \right] 1$

Total cost of $g$ is $\text{cost}(A) + \text{cost}(B) + \text{cost}(C)$
\[ C \cdot n^2 \left\lceil \frac{n}{2} \right\rceil + 1 + n^2 \in \Theta(n^2) \]

**Nested loops**

```python
def f4(n):
    for i in range(n):
        for j in range(n):
            print(i+j)
```

**Strategy:** Same loop technique, but start with innermost loop and work outwards.

**Loop 2:** For one iteration of the outer loop:

1. \( n \) iterations
2. 1 step per iteration

So, total cost is \( n \times 1 = n \) steps.

So the cost of one iteration of loop 1 is \( n \) steps.
So the cost of one iteration of loop 1 is \( n \) steps.

**Loop 1:**

1. \( n \) iterations
2. \( n \) steps per iteration

So the total cost is \( n \times n = n^2 \) steps, which is \( \Theta(n^2) \).

---

Ex:

```python
def f(n):
    for i in range(n):
        for j in range(i):
            print(i+j)
```

**Loop 1**

**Loop 2**

---

**Loop 2**

For the \( i \)-th iteration of the outer loop:

1. Loop 2 takes \( i \) iterations
2. 1 step per iteration
So the cost is $i \times 1 = i$ steps.

**Loop 1:**

1. Take $n$ iterations, $i = 0, 1, 2, ..., n-1$.
2. Cost for iteration $i$ is $i$ steps.

Total cost:

$$\sum_{i=0}^{n-1} i \text{ steps} = \frac{(n-1)n}{2} \text{ steps}$$

And finally, the cost is

$$\frac{(n-1)n}{2} \subseteq \Theta(n^2).$$
Ex (This challenges the way we think about analysing runtime)

```python
def is_prime(n):  # pre: n ≥ 2
    for d in range(2, n):
        if n % d == 0:
            return False
    return True
```

Current strategy: For the loop,

1. Between 1 and n-2
2. cost per iteration is 1 step