Reminders:

HW3 due Mar 14
Midterm 2: Mar 28/29

Review: from Week 7
O, Ω, Θ
Quick Review

1. $g \in O(f)$ "$g$ is eventually dominated by $f$, ignoring constant factors"
   $$\exists \eta_0, c \in \mathbb{R}^+ \quad \forall n \geq \eta_0 \quad g(n) \leq c \cdot f(n)$$

2. $g \in \Omega(f)$ "$g$ eventually dominates $f$, ignoring constant factors"
   $$\exists \eta_0, c \in \mathbb{R}^+ \quad \forall n \geq \eta_0 \quad g(n) \geq c \cdot f(n)$$

3. $g \in \Theta(f)$: $g \in O(f)$ and $g \in \Omega(f)$

*② : ③ from tutorial*
Growth rates

\[ \log_2 n < (\log n)^2 \leq n < n^2 < n^3 \leq 2^n \]

\[ n \log n ? \]

Examples

1. \( 100n + 5000 \leq O(n^2) \)

   Pick \( n_0 = 5000, c = 101 \) (\( \forall n \geq 5000 \quad 100n + 5000 \leq 101 n^2 \))

   Then \( \forall n \geq 5000 \quad 100n + 5000 \leq 100n \)

   \[ 5000 \leq n \leq n^2 \quad \text{since} \quad n \geq 5000 \]

   \[ 100n \leq 100n^2 \]

   \[ \therefore 5000 + 100n \leq 100n^2 + n^2 = 101 n^2 \]

2. \( 2^n + 50n \leq O(3^n) \)

   Pick \( n_0 = 1, c = 51 \)

   \[ n \leq 3^n, 2^n \leq 3^n \]

   \[ 50n \leq 50 \cdot 3^n, 2^n \leq 3^n \quad \Rightarrow \quad 2^n + 50n \leq 51 \cdot 3^n \]
\[ \log_2 n \in \Theta(\log_{10} n) \]

\[ 2n^2 + \sqrt{n} \in \Theta(?) \]

\[ \Theta(n^2) \]

1. \[ 2n^2 + \sqrt{n} \in O(n^2) \]

2. \[ 2n^2 + \sqrt{n} \in \Omega(n^2) \]

\[ n_0 = 10, \quad c_0 = 3 \]

\[ 2n^2 + \sqrt{n} \leq 2n^2 + n \leq 3n^2 \]

\[ n_0 = 1, \quad c = 1 \]

\[ 2n^2 + \sqrt{n} \geq n^2 \quad \forall n > n_0 \]
$$100n + 5000 \in \Theta(n)$$

$$\exists n_0, c \quad \forall n \geq n_0 \quad 100n + 5000 \geq c \cdot n$$

$$c = 1$$

$$n_0 = 5000$$

can check

$$\forall n \geq n_0 \quad 100n + 5000 > n$$

$$\therefore \quad 100n + 5000 \in \Theta(n)$$
Theorem
\( f \in O(h), \ g \in O(h) \Rightarrow f + g \in O(h) \)

Theorem
\( f_1 \in O(g_1), \ f_2 \in O(g_2) \Rightarrow f_1 \cdot f_2 \in O(g_1 \cdot g_2) \)

Other useful Facts \( a, b \in \mathbb{N} \)

1. \( a > 1, \ b > 0 \quad \log_a n = \Theta(\log_b n) \)
   \( a > 1, \ b > 0 \quad \log_a n = O(n^b) \)

2. \( a \leq b \quad n^a \in O(n^b) \)

3. \( 1 \leq a \leq b \quad a^n \in O(b^n) \)

4. \( \forall n \in \mathbb{N} \quad f(n) \geq 1 \Rightarrow \exists \ f \in \Theta(f) \land \exists \ f \in \Theta(f) \)
Analyzing Runtime of Code

Goal: Find the approximate number of steps that a program takes as a function of input size in the long term.

Runtime $f : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$

We will be interested in asymptotic behavior of $f$.

$i.e. \ f \in \Theta(n^2)$

Why is $f \in \Theta(n^2)$ better than knowing $f \in O(n^2)$ and $f \in \Omega(\log n)$?
How to decide what is a basic step?

- It should not depend on the input size
- Identify blocks of code which can be counted as a single basic step
- Identify loops that cause basic operations to repeat. Count number of repetitions exactly
- Come up with an expression for runtime $f$
- Use asymptotic notation to find an elementary function $g$ s.t. $f(n) = \Theta(g(n))$

theta is ideal but may not always be possible (more on this later)
Example 1 (Warmup)

def print_sums (list):
    for item1 in list:
        for item2 in list:
            print (item1 + item2)

Let n = length of list
outer loop executes n times
inner loop executes n times

# A_{n^2} time * executes \( n^2 \) <= \( \Theta(n^2) \)
The total # A time all line execute 1+n+n^2+n^2 \in \Theta(n^2)
Example 2 (Warmup)

def f(list):
    for item in list:
        for i in range(10):
            print(item + i)

n = size of the list

How many times does \* execute
10 \* n \( \in \Theta(n) \)

The more careful calculation (total \# of times all lines execute) = 1 + n + 10n + 10n
\( \in \Theta(n) \)
Example 3 Nested Loop

```python
def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print(i+j)
            j = j + 2
        i = i + 1
```

We will estimate the number of times the inner loop block executes. Work from inside out.

Argue the # of times it executes is $\Theta(\text{total # of steps of algorithm})$.
Example 3

def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print (i + j)
            j = j + 2
        i = i + 1

When \( i = 0 \)

\((\ast)\) executes \( \left\lceil \frac{n}{2} \right\rceil \) times

\( \hat{c} = 1 \)

\((\ast)\) executes \( \left\lceil \frac{n}{2} \right\rceil \) times

\( i = n - 1 \)

\((\ast)\) executes \( \left\lfloor \frac{n}{2} \right\rfloor \) time

then \((\ast)\) doesn't execute anymore

So total \# \( \# \) executions of \((\ast)\)

is \( n \cdot \left\lceil \frac{n}{2} \right\rceil \)

\( \in \Theta(n^2) \)
Example 3

def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print (i+j)
            j = j + 2
        i = i + 1

Summary:
1. Every time outer loop executes, inner loop executes \( \frac{n^2}{2} \) time
   (Each execution of inner loop doesn't depend on i)
2. So overall \# executions of inner loop =
   (\# executions per outer loop) \times \frac{n^2}{2}
Example 3

```python
def Nested1(n):
    i = 0
    while i < n:
        j = 0
        while j < n:
            print (i+j)
            j = j + 2
        i = i + 1
```

We could have done a perfect analysis and gotten an exact expression for the steps but it would still be \( \Theta(n^2) \)
So now we want to give a theta expression

\[ n \cdot \left\lfloor \frac{n}{2} \right\rfloor \in \Theta(n^2) \]

To prove \( n \cdot \left\lfloor \frac{n}{2} \right\rfloor \in \Theta(n^2) \), need to show

1. \( n \cdot \left\lfloor \frac{n}{2} \right\rfloor \in \mathcal{O}(n^2) \).

   \[ \exists n_0 \in \mathbb{N} \exists c \in \mathbb{R}^+ \quad (\forall n \geq n_0 \quad n \cdot \left\lfloor \frac{n}{2} \right\rfloor \leq c \cdot n^2) \]

   Let \( n_0 = 1 \), \( c_0 = 1 \). Check \( \forall n \in \mathbb{N} \) \( n \geq 1 \Rightarrow n \cdot \left\lfloor \frac{n}{2} \right\rfloor \leq n \cdot n = n^2 \)

2. \( n \cdot \left\lfloor \frac{n}{2} \right\rfloor \in \Omega(n^2) \)

   Show \( \exists n_0 \in \mathbb{N} \exists c \in \mathbb{R}^+ \quad \forall n \geq n_0 \quad n \cdot \left\lfloor \frac{n}{2} \right\rfloor \geq c \cdot n^2 \)

   Pick \( c = \frac{1}{2} \), \( n_0 = 1 \)

   \[ n \cdot \left\lfloor \frac{n}{2} \right\rfloor \geq n \cdot \frac{n}{2} = \frac{1}{2} \cdot n^2 \]
So now we want to give a theta expression

\[ n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in \Theta(n^2) \]

To prove \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in \Theta(n^2) \), need to show

1. \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in O(n^2) \)

2. \( n \cdot \left\lfloor \frac{n^2}{2} \right\rfloor \in \Omega(n^2) \)
Example 4: Nested loop where loop cost changes

```python
def Nested2(n):
    i = 0
    while i < n:
        j = 0
        while j < i:
            print(i + j)
            j = j + 1
        i = i + 1
```

The outer loop runs from `i = 0` to `i = n-1`. The inner loop runs `j` times, where `j` is equal to `i` at the start of the inner loop.

We will analyze in the same way, but now # q times (*) executes every time outer loop executes depends on `i`
Example 4  Nested loop where loop cost changes

def Nested2(n):
    i=0
    while i<n:
        j=0
        while j<i:
            print (i+j)  
            j = j+1
        i=i+1

When outer loop has i set to i, inner loop \(\sum j \) executes \( i \) times.
So overall \# of times \(\sum j \) executes is
\[
0 + 1 + 2 + 3 + \ldots + n-1 \leq \sum \frac{n(n-1)}{2} \]
\[
i=0 \quad i=1 \quad i=2 \quad i=3 \quad \ldots \quad i=n-1 \quad i=0
\]
\[
= \Theta(n^2) \]
Example 4  Nested loop where loop cost changes

def Nested2(n):
    i=0
    while i<n:
        j=0
        while j<i:
            print (i+j)
            j = j+1
        i = i+1

Want to write this in $\Theta$ form:

$\sum_{i=0}^{n-1} i = \frac{n \cdot (n-1)}{2} \in \Theta(n^2)$

$\Theta(n^2-n) \in \Theta(n^2)$
Show \[ \frac{n(n-1)}{2} \in \Theta(n^2) \]

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

\( \exists n_0, c \in \mathbb{R}^+ \forall n \geq n_0 \quad \frac{n(n-1)}{2} \geq c(n^2) \quad n_6 = 3 \)

\[ \frac{n(n-1)}{2} \geq \frac{n \cdot (\frac{n}{2})}{2} = \frac{n^2}{4} \]

\[ \therefore \forall n \geq n_6, \quad \frac{n(n-1)}{2} \geq \frac{1}{4}n^2 \]

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

\[ \frac{n(n-1)}{2} \leq \frac{n \cdot n}{2} = n^2 \quad \therefore \forall n \geq 1 \quad \frac{n(n-1)}{2} \leq 1 \cdot n^2 \]
Example 4.1A Nested loop where loop cost changes

def Nested2(n):
    i = 0
    while i < n:
        j = 1
        while j < i
            print(i+j)
            j = 2*j
        i = i+1

We will analyze in the same way, but now # q times (*) executes every time outer loop executes depends on i

\begin{align*}
i = 0 & \quad i = 1 \quad i = 2 \quad \cdots \quad i = k \\
0 & \quad 0 & \quad 1 \quad \vdots \quad \leq \log k
\end{align*}
\[ \log 1 + \log 2 + \log 3 + \ldots + \log n \]

\[ \log a + \log b = \log (a \cdot b) \]

\[ = \log (1 \cdot 2 \cdot 3 \ldots \cdot n) \]
\[ = \log (n!) \sim \log (n^n) \]
\[ = \log (2^{\log n}) = n \log n \]
Sometimes it isn't possible to get runtime $= \Theta(f)$, for a closed-form expression $f$.

Ex

Even-or-odd $(n)$

If $n$ is even

For $i = 1$ to $n^2$
    print $i$

If $n$ is odd

For $i = 1$ to $n$
    print $n$

For all $n$ Runtime $\in O(n^2)$, $\Omega(n)$
Example 5: Factoring - find a nontrivial factor of $n$

def factor(n):  
    (n ≥ 2)  
    d = 2  
    while d < n  
        if n % d == 0:  
            return d + quit  
        d = d + 1  
    return -1

If $n$ is even $\Rightarrow$ runtime is constant  
other extreme $n$ is prime $\Rightarrow$ runtime is $n$  
in between $\Rightarrow$ runtime depends on the smallest prime divides $n$  
so Runtime is $\Omega(1)$, and $O(n)$
It turns out that there is no elementary function $f$ s.t. the runtime of factor is $\Theta(g)$.

But factoring is supposed to be really hard!

The input size is $\log n$, not $n$.

So to factor a number with 100 digits takes time $2^{100}$ in worst case, which is terrible.