PS3
midterm1
about remarks

- A simple description for the long term behaviour \((n \to \infty)\) of \(f(n)\): 

\[ f(n) = n^3 - 148n^4 + 165n^{10} \]

- \(f \in O(n^{10})\) not \(f \in O(165n^{10})\) or 
  - best "simple" function

lecture: big Oh \(g \in O(f)\)
  - eventually \(g(n) \leq Cf(n)\)

- \(100n \in O(n^{2})\)

tutorial: \(g \in \Omega(f)\) eventually \(g(n) \geq Cf(n)\)

check: and \(g \in \Theta(f)\) eventually
  - \(C_1f(n) \leq g(n) \leq C_2f(n)\)
Θ(n) is the set of all functions that eventually stay between two lines that go through origin.
def print_items(lst):
    for item in lst:
        print(item)

Assume: lst contains simple items (int, floats) not sub lists.
So print(item) has constant time.

```text
# basic operations
\[ \Theta(n) \]
```

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>( n )</td>
</tr>
<tr>
<td>2n</td>
<td>( 2n )</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>( n^2 )</td>
</tr>
<tr>
<td>( n + 1.5 )</td>
<td>( n + 1.5 )</td>
</tr>
</tbody>
</table>

\( \Theta(n) \) is the description of runtime.

Use \( \Theta(n) \) as an operation (allows us to ignore relative costs of operations).

def \n: A "basic operation" is any block of code whose running time does...
not depend on the size of input.

e.g. \[
\begin{align*}
\text{comparisons} & \quad =,!,<,\leq, \ldots \\
\text{arithmetic} & \quad +,\times, -, /, \ldots \\
\text{using variable} & \quad x = y, x = y + z \\
\text{print, return.} & \\
\end{align*}
\]

Can then prove results like:

"The running time of print-items in $\Theta(n)$, where $n$ is the length of the list of simple items."

**Proof:**

- Each iteration of the loop can be counted as a single basic operation, because nothing in loop body depends on size of the list.
- The loop runs $n$ times.
- Total no of basic operations is $n \times 1$.
- So the running time is $\Theta(n)$. \qed
summary of the approach

1. Identify your measure of the input size (your "n")
   - eg list length,
   - int: # bits reg'd to store the int

2. Identify the blocks of code that be counted as a single basic operation since they don't depend on the input size

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

4. Combine these observations to get an expression for the # of basic ops
   - eg. $3n^2 + 10n + 7$

5. Convert this expression to $\Theta$ notation.
   - eg. $\Theta(n^2)$

   e.g.
def print_sums(lst):
    for item1 in lst:
        for item2 in lst:
            print(item1 + item2)

(Start an interior of nested loop + work outwards)

Let \( n \) be the length of the list. The running time of this algorithm is \( \Theta(n^2) \).

\[
\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \frac{n^2 + n}{2} = \frac{n^2}{2} + \frac{n}{2} = \Theta(n^2).
\]
def f(lst):
    n_iterations = 10
    for item in lst:
        for i in range(10):
            print(item + i)

Determine a descrip. of the runtime in terms of $n$, the length of the input list $lst$.

so the running time of this algorithm is $\Theta(n)$

Alt: $\Theta(n)$ as a single basic op.

lesson: don't just look at nested level
for i in range(n*n):
    print(i)

$\Theta(n^2)$ despite being single loop
def g(lst):
    for item in lst:
        i = 0
        while i < len(lst):
            print(item + i)
            i = i + 2

description of runtime:

\[
\sum_{i=0}^{n/2} \left( 1 + \left\lceil \frac{n}{2} \right\rceil \cdot 1 \right)
\]

\(4\) outer loop \(4\) inner loop

\[= n + n \left\lceil \frac{n}{2} \right\rceil\]

fact: \(\left\lceil n \right\rceil \in \Theta(n) \land \left\lfloor n \right\rfloor \in \Theta(n)\)

so running time: \(\Theta(n^2)\)
```python
def c(lst):
    print("Here is the list:")
    # loop 1
    for item in lst:
        print(item)
    # loop 2
    for item1 in lst:
        for item2 in lst:
            print(item1 + item2)
```

Let $n$ be the length of the list

```plaintext
# basic operations:
1 + n + n^2
```

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

\( n^2 \in \Theta(n^2) \land n \in \Theta(n^2) \land 1 \in \Theta(n^2) \Rightarrow 1 + n + n^2 \in \Theta(n^2) \)

running time is \( \Theta(n^2) \)
Wed March 7, 2018

Problem: describe the runtime of an algorithm as a function of its input

Approach:

- Identify blocks of code with runtime independent of input
- Identify loops + exact iteration count for each loop
- Assemble terms to get an expression for runtime
- Use asymptotic expression ($\Theta$) to summarize runtime

Example last time got: $n^{\lceil n/2 \rceil}$

Wrote as $\Theta(n^2)$

To prove this summary correct, need to prove $n^{\lceil n/2 \rceil} \in O(n^2)$ \( \Theta \)

and $n^{\lceil n/2 \rceil} \in \Omega(n^2)$ \( \Theta \)
(1) \( \exists c_0, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0 \implies n \left\lfloor \frac{n}{2} \right\rfloor \leq c_0 n^2 \)

Let \( c_0 = \frac{1}{2} \)
\[ n_0 = \frac{1}{2} \]
Assume \( n \geq n_0 \)
\[ n \left\lfloor \frac{n}{2} \right\rfloor \leq n \left\lfloor \frac{2n}{2} \right\rfloor \]
\[ = n \left\lfloor n \right\rfloor \]
\[ = n \cdot n \quad \text{since} \left\lfloor n \right\rfloor = n \]
\[ = c_0 n^2 \quad \text{for} \ n \in \mathbb{N} \]
\[ \implies n \left\lfloor \frac{n}{2} \right\rfloor \in \Theta(n^2) \]

(2) \( \exists c_1, n_1 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n > n_1 \implies n \left\lfloor \frac{n}{2} \right\rfloor > c_1 n^2 \)

Let \( c_1 = \frac{1}{2} \) and \( n_1 = 1 \) Assume \( n > n_1 \)
The \( n \left\lfloor \frac{n}{2} \right\rfloor > n \cdot \frac{n}{2} \]
\[ = \frac{1}{2} n^2 \]

---

nested loop example:
- last time: inner loop independent
- now: not ( )
```python
def nested_2(n):
    for i in range(n):
        for j in range(i):
            print(i+j)
```

Total # basic operations is:

\[0 + 1 + 2 + \ldots + n-1 = \sum_{i=0}^{n-1} i = \sum_{i=0}^{n-1} i = \frac{(n-1)(n)}{2}
\]

\[= \frac{(n-1)(n+1)}{2}
\]

\[= \frac{1}{2} n^2 - \frac{1}{2} n
\]

\[\in \Theta(n^2)
\]

Proof:

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

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

\[\Omega(n^2)
\]
\[
\frac{n(n-1)}{2} \geq \frac{n}{a} \cdot \frac{n}{2} \quad \text{for } n-1 \geq \frac{n}{2}
\]
\[
= \frac{1}{4} n^2 
\]

\[
C_0 = 1/4
\]

\[
O(n^2) : \quad \frac{n(n-1)}{2} \leq \frac{n(n)}{a} = \frac{1}{2} n^2 \quad C_0 = 1/2
\]

def even_or_odd(n: int) -> bool:
    if n % 2 == 0:
        # n is even
        for i in range(n*n):
            print(i)
        return True
    else:
        # n is odd
        for i in range(n):
            print(i)
        return False

- When n is even, the runtime grows like \(n^2\).
- When n is odd, the runtime grows like \(n\).
runtime $\in O(n^2)$  runtime $\not\in \mathcal{O}(n)$

A more practical example:

```python
def smallest_factor(n: int) -> int:
    """Return the smallest non-trivial factor of n or -1 if there are none.
    Precondition: n $\geq$ 2, so loop body entered
    """
    d = 2
    while d < $\lceil \sqrt{n} \rceil$:
        if n % d == 0:
            return d
    return -1
```

not $\frac{n}{2}$ or $\frac{n}{3}$
if \( n \mod d == 0 \):
    return d

    d = d+1

    return -1

when \( n \) is even:
    \# iterations: 1
    runtime is constant

when \( n \) is prime:
    \# iterations: \( n-2 \)
    runtime grows like \( n \)
    improvement \( n^{1/2} \)

when \( n \) is odd and not prime:
    \# iterations: between 1 and \( n-2 \)

runtime is \( \Omega(1) \) \( \forall n \)
and \( O(n) \) \( O(n^{1/2}) \)

Simple, without cases

There is no elementary function \( g(n) \)
such that the runtime is \( \Theta(g(n)) \)

next: how to deal with this
meets precondition