Fri 2 Mar 2018

- PS3
- midterm1
- about remarks

- a simple description for the long term behaviour \((n \to \infty)\) of these hces:

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

- \( f \in O(n^{10}) \) not \( f \in O(165n^{10}) \)
- say \( f \in O(165n^{10} - 148n^4) \)

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

\[ 100n \in O(n^2) \]

tutorial: 
- \( g \in \Omega(f) \) eventually \( g(n) \geq Cf(n) \)
- \( g \in \Theta(f) \) eventually \( C_1 f(n) \leq g(n) \leq C_2 f(n) \)
$\Theta(n)$ is the set of all functions that eventually stay between two lines that go through origin.
```python
def print_items(lst):
    for item in lst:
        print(item)
```

Assume: `lst` contains simple items (int, floats) not sublists so `print(item)` has constant time.

**basic operations**

- \( n \)
- \( 2n \)
- \( n \)
- \( n \log n + 1.5 \)

**prints**

- print, item assign
- print costs 10 times with assign
- add cost to call + return

\( \Theta(n) \)

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

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

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

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 $\Theta$ of basic operations is $n \times 1$.

So the running time is $\Theta(n)$. \[\square\]
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. \(\Theta(n^2)\)
```
def print_sums(lst):
    for item1 in lst:
        for item2 in lst:
            print(item1 + item2)
```

(Start an interior of nested loop and work outwards)

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

\[ n + n + n + \ldots + n = n^2. \]
```python
def f(lst):
    n = len(lst)
    for item in lst:
        for i in range(10):
            print(item+i)
```

Determine a description of the runtime in terms of \( n \), the length of the input list \( \text{lst} \).

So the runtime of this algorithm is \( \Theta(n) \).

**Alt:** as a single basic op.

Lesson: don't just look at nested level

```
for i in range(n**2):
    print(i)
```

\( \Theta(n^2) \) despite being single loop.
def g(lst):
    for item in lst:
        i = 0
        # $i/2$ times
        while $i < \text{len(lst)}$:
            print(item + i)
            i = i + 2

Description of runtime:

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

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

Fact: \[\left\lceil n \right\rceil \in \Theta(n) \land \lceil n \rceil \in \Theta(n)\]

So running time: \( \Theta(n^2) \)
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

# basic operations:
1 + n + n^2

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

\( n^2 \in \Theta(n^2) \land n \in O(n^2) \land 1 \in O(n^2) \)

\( \Rightarrow 1 + n + n^2 \in O(n^2) \)

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