Tue 30 Oct 2018

simple

description of long term behaviour of function

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

\[ f \in O(n^{10}) \quad \text{not say } f \in O(165n^{10}) \]

Simplest description

\[ f \in O(n^{10}) \quad \text{or } f \in O(148n^{10}) \]

\[ g \in O(f) \quad \text{eventually } g(n) \leq C f(n) \]

e.g. \( 100n \in O(n^2) \)

\[ g \in \Omega(f) \quad \text{eventually } g(n) \geq C f(n) \]

e.g. \( n^2 \in \Omega(n) \)

\[ g \in \Theta(f) \quad \text{eventually } C_1 f(n) \leq g(n) \leq C_2 f(n) \]

Last worksheet:

\[ g \in \Omega(f) \quad \text{eventually } g(n) \geq C f(n) \]

e.g. \( n^2 \in \Omega(n) \)

and \( g \in \Theta(f) \)

\[ \text{eventually } C_1 f(n) \leq g(n) \leq C_2 f(n) \]
$\Theta(n)$ is the set of all functions that eventually stay between two lines through the origin.
Consider

```python
def print_items (list: List[bool]) -> None:
    for item in list:
        print(item)
```

<table>
<thead>
<tr>
<th>Operation Count</th>
<th>Method for Counting</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td># prints</td>
</tr>
<tr>
<td>( 2n )</td>
<td># prints, assign. to item</td>
</tr>
<tr>
<td>( \ll n )</td>
<td>relative cost of print to assign.</td>
</tr>
<tr>
<td>( 11n + 1.5 )</td>
<td>includes cell &amp; return cost</td>
</tr>
</tbody>
</table>

- Describe the growth of the routine
  - increasing scaling factors

All are \( \Theta(n) \)

**Def:** A "basic operation" is any block or code whose running time does not depend on size of input/data (sometimes called a step)
Example: Simple objects:

- Comparison: $=, <, >$
- Arithmetic: $+, -, *, /$
- Assign to variable: $x = y$
- $x = y + z$
- print, return

These can prove results like:

"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 the list argument, we know the loop runs $n$ times.

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

1. Identify your measure of the input/data size
   (e.g. \texttt{len(lst)}, tree height)

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

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 size of input.

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

5. Convert this expression to \( \Theta \) notation. e.g. \( \Theta(n^2) \)
Consider the function

1. \[ \text{def print sums (lst : List[Float]) -> None:} \]

2. \[ \text{for item1 in lst:} \]

3. \[ \text{for item2 in lst:} \]

4. \[ \text{print(item1 + item2)} \]

Prove 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 list `lst`.

- The inner loop (lines 3-4) runs \(n\) times and each iteration has one basic operation.

- The outer loop (lines 2-4) runs \(n\) times, and each iteration takes \(n\) basic operations.

- So the total number of basic operations is \(\text{cost for inner loop } \times \text{ number of times inner loop is repeated} = n \times n\).
\[ n^2 \]

\[ \therefore \text{The running time of this algorithm is } \Theta(n^2). \]
Consider this function:

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

Determine a description of the running time.

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

- The inner loop 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 $10n$ steps.

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

Lesson: Don't just look at nested level. Think about the number of iterations.

Alt: Label the body of outer loop as a basic op. Since runtime does not depend on $n$.
Consider the function:

```python
def g(lot: List[int]) -> None:
    for item in lot:
        print(item + i)
        i = i + 2
```

Determine a description of the running time:

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

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

Fact: \( \left\lceil \frac{n}{2}\right\rceil \in \Theta(n) \) and \( \left\lfloor \frac{n}{2}\right\rfloor \in \Theta(n) \)

So running time \( \Theta(n^2) \) [informal]

formal: \( n \cdot \left\lceil \frac{n}{2}\right\rceil \in O(n^2) \) or \( n \cdot \left\lceil \frac{n}{2}\right\rceil \in \Omega(n^2) \)
\[ \forall c_0, n_0 \in \mathbb{R}^+ \text{, } \forall n \in \mathbb{N}, n \geq n_0 \implies n \cdot \lceil \frac{n}{2} \rceil \leq c_0 n^2 \]

Let \( c_0 = \frac{1}{2} \) and \( n_0 = 1 \) and assume \( n \geq n_0 \).

\[
\begin{align*}
n \cdot \left\lceil \frac{n}{2} \right\rceil & \leq n \cdot \left\lceil \frac{2 \cdot n}{2} \right\rceil \\
& = n \cdot n \\
& = c_0 n^2
\end{align*}
\]

(2) Similar: \( n_0 = 1 \), \( c_0 = \frac{1}{2} \).
Consider the function:

```python
def c(lst: List[int]) -> None:
    print("Here is the list:")
    for item in lst:  # exactly n times
        print(item)
    for item1 in lst:
        for item2 in lst:
            print(item1 + item2)
```

Determine a description of the running time.

**Proof:** Let n be the length of the list.
- the first print counts as one basic operation.
- we have seen that the first loop runs in time \(\Theta(n)\), while the second runs in time \(\Theta(n^2)\)

we need to combine the terms:
\[ \text{Th}^n: \quad f \in \Theta(h) \land g \in o(h) \quad \Rightarrow \quad (f \cdot g) \in \Theta(h) \]

\[
\begin{align*}
1 + n + n^2 & \quad \overset{f}{\to} \quad \frac{n^2}{n} \in \Theta(n^2) \\
& \quad \overset{g}{\to} \quad n \in o(n^2) \\
g' & \quad \overset{g}{\to} \quad 1 \in o(n^2)
\end{align*}
\]

Apply Th\textsuperscript{2}: \quad 1 + n + n^2 \in \Theta(n^2)

So the running time of our algorithm is \( \Theta(n^2) \). \qed