Old problem sets & work sheets: Sometimes I look back for questions that can be extended, e.g., ps#1 q 4d → ps#2 2b

CSC165 fall 2019
counting steps...

Danny Heap
csc165-2019-09f@cs.toronto.edu
BA4270 (behind elevators)
Web page:
http://www.teach.cs.toronto.edu/~heap/165/F19/

Using Course notes:
from pizza \( a > 0, \ b > 1 \) \( n^a \in \Omega(b^n) \)

e.g. \( a = 1000, \ b = 2 \)

want to show \( n^{1000} < 2^n \) for any \( c \), some \( n \)

try \( \lg (\log_2) \)

\[ 1000 \lg n < \lg c + n \]

\[ -\lg c < 2^k - 1000k \]

more feasible now
big-Theta means...\[ g \in \Theta(f) : \exists c_1, c_2, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0 \Rightarrow g(n) \leq c_1 f(n) \land g(n) \geq c_2 f(n) \]g grows at same rate as f w.r.t. input — rough idea of behaviour asymptotically
Why care about $\Theta$? e.g. my $\Theta(n^2)$ program sorts 100 entries in 1 second,
your $\Theta(n \log n)$ program sorts 100 entries in 5 seconds...

which is better?

if we increase input to 1,000,000 entries, my $\Theta(n^2)$ program increases $10^4 \times 10^4$ times, so now $10^8$ seconds... about 3 years!
your $\Theta(n \log n)$ program increases $10^4 \times 4$ times so now 200,000 seconds... about 2.3 days...
back to code...

“steps” and input “size”

operations with runtimes that do not depend on input size. It is valid to lump entire blocks of such code into one “step”

strictly speaking, size of input is # of bits (0s + 1s) to represent input. But, in first year we don’t speak strictly... e.g. we pretend all integers same # of bits to do arithmetic and sometimes ignore size of list elements and focus on # of elements.

we usually label/name input size $n$
counting loops... \( n \) is a natural number

\[
\text{def } f0(n): \\
\quad x = n \\
\quad \text{print}(x * 2) \\
\quad \text{return } x + 3
\]

\[\text{1 step}\]

\[
\text{def } f1(n): \\
\quad \text{for } i \text{ in range}(10): \\
\quad \quad \text{print}(n) \\
\quad \text{imagine } i \text{ increments here to help count loop iterations...}
\]

\[
\text{10 iterations } \times \text{1 step} = 10 \text{ steps} \\
\text{\( \Theta(1) \)}\]

\[\text{\( \Theta(1) \) or \( \Theta(3) \)} \\
\text{(but \( \Theta(1) \) is simpler...)}\]
def f2(n):
    for i in range(n):
        print(n) # 1 step
        i = 1, 2, ..., n = n steps $\Theta(n)$

def f3(n):
    i = 0  # 1 step
    while i*i < n:
        print(i)
        i = i + 1
        i = 1, 2, 3, ..., i^2 >= n
        i >= $\sqrt{n}$
        $i = \sqrt{n}$ # $\sqrt{n}$ steps
        $\Theta(\sqrt{n}) + 1$ = $\Theta(\sqrt{n})$

def f4(n):
    i = 0
    while i**(1/2) < n:
        print(2*i) # 1 step
        i = i + 1
        i = 1, 2, 3, ..., $\sqrt{n}$ >= n ...
        $\sqrt{n}$ steps $\sqrt{n}$ iterations $\Theta(n^2 + 1) = \Theta(n^2)$
def f5(n):
    for i in range(0, n, 2):
        for j in range(n):
            print(i - j)
    j = 1, 2, ..., n
    l = 2, 4, ..., ≥ n
    s = 2s, 4s, ..., s ≥ \frac{n}{2}
    s = \lceil \frac{n}{2} \rceil

\exists \in [0, 1)

\lceil \frac{n}{2} \rceil \text{iterations} \times n \text{ steps} = \left( \frac{n}{2} + \varepsilon \right) \cdot n = \frac{n^2}{2} + \varepsilon n \ldots \Theta(n^2)

\lceil \frac{n}{2} \rceil \text{iterations} \times 1 \text{ step} = n \text{ steps}