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 piazza $a>0$, $b>1$ $n^a \notin \Omega(b^n)$

e.g. $a=1000$, $b=2$

want to show $n^{1000} < c2^n$ for any $c$, some $n$

try lg ($\log_2$) $\iff$ $1000 \lg n < \lg c + n$

try $n = 2^k$, some $k$, so... $1000k < \lg c + 2^k$

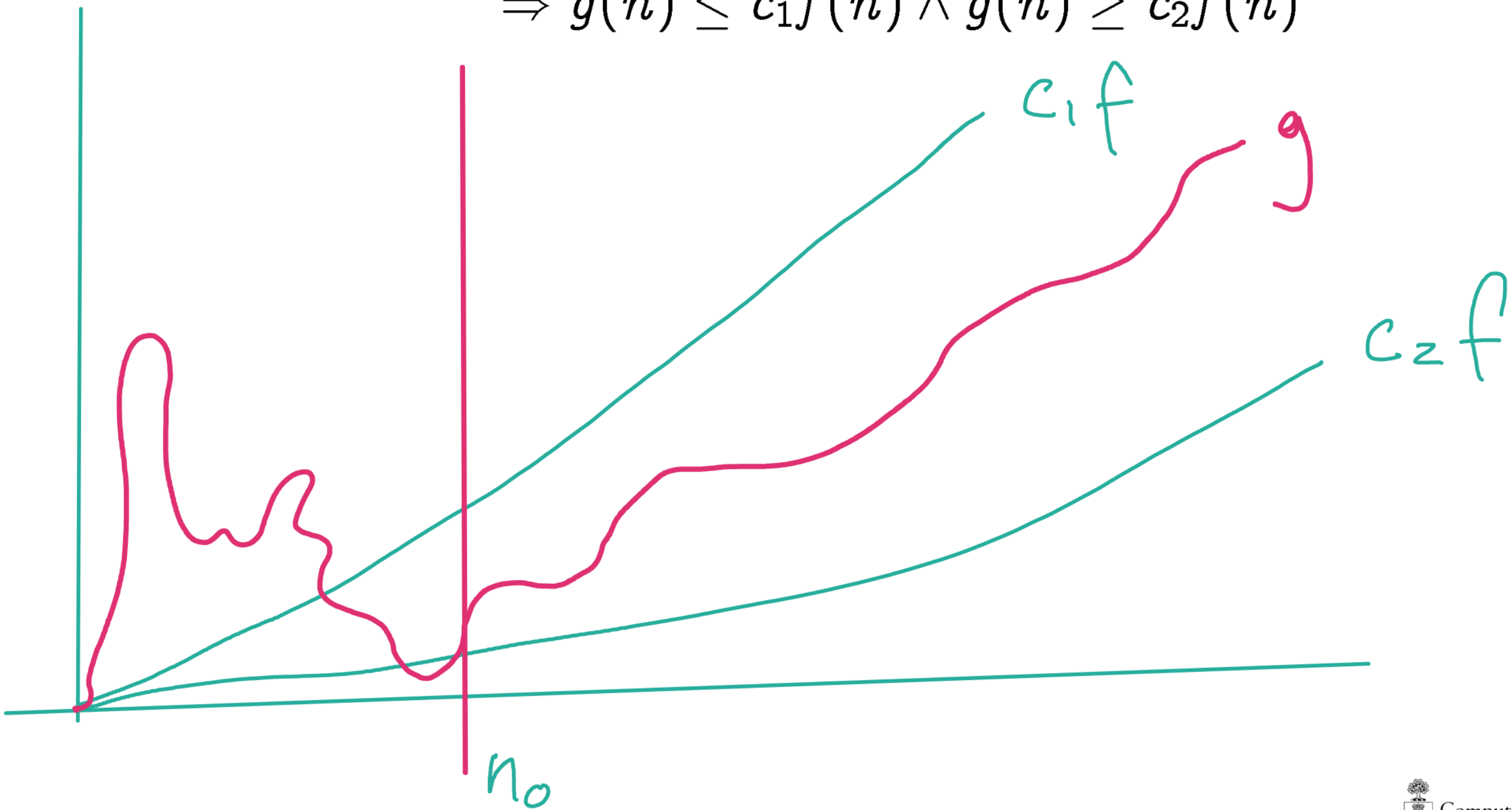
$-\lg c < 2^k - 1000k$

more feasible now

# big-Theta means...

g grows at same rate as f w.r.t. input — rough idea of behaviour asymptotically

$$g \in \Theta(f) : \exists c_1, c_2, n_0 \in \mathbb{R}^+, \forall n \in \mathbb{N}, n \geq n_0$$
$$\Rightarrow g(n) \leq c_1 f(n) \wedge g(n) \geq c_2 f(n)$$



Why care about $\theta$? big-Theta means...

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)

```
def f0(n):
    x = n
    print(x * 2)
    return x + 3
```

1 step

$\Theta(1)$ OR $\Theta(3)$

(but $\Theta(1)$ is simpler...)

```
def f1(n):
    for i in range(10):
        print(n)
```

1 step

imagine i increments here to help count loop iterations...

$i = 1, 2, \ldots, 10$

10 iterations × 1 step = 10 steps

$\Theta(1)$

```
def f2(n):
    for i in range(n):
        print(n)
```

1 step

$i = 1, 2, \ldots, n$

$n$ iterations $\times$ 1 step $= n$ steps

$\Theta(n)$

```
def f3(n):
    i = 0
    while i*i < n:
        print(i)
        i = i + 1
```

1 step

1 step

$i = 1, 2, 3, \ldots, i^2 \geq n$

$i \geq \sqrt{n}$

$i = \lceil \sqrt{n} \rceil$

$\lceil \sqrt{n} \rceil$ iterations $\times$ 1 step... $\lceil \sqrt{n} \rceil$ steps

$\Theta(\lceil \sqrt{n} \rceil + 1)$

$= \Theta(\lceil \sqrt{n} \rceil) = \Theta(\sqrt{n})$

simplest

```
def f4(n):
    i = 0
    while i**(1/2) < n:
        print(2*i)
        i = i + 1
```

1 step

$i = 1, 2, 3, \ldots, \sqrt{i} \geq n \ldots i = n^2$

$n^2$ iterations $\times$ 1 step...

$\Theta(n^2 + 1) = \Theta(n^2)$

# nested loops

$\varepsilon \in [0,1)$

$\lceil n/2 \rceil$ iterations $\times$ $n$ steps

$= (n/2 + \varepsilon) \cdot n = \frac{n^2}{2} + \varepsilon n \ldots \quad \Theta(n^2)$

```
def f5(n):
    for i in range(0, n, 2):
        for j in range(n):
            print(i - j)
```

1 step

$n$ iterations $\times$ 1 step $= n$ steps

$j = 1, 2, \ldots, n$

$i = 2, 4, \ldots, \geq n$

$i_s = 2s, \quad 2s, 4s, \ldots, s \geq \frac{n}{2}$

$s = \lceil n/2 \rceil$