CSC165 fall 2017

counting steps...

Danny Heap
csc16517f@cs.toronto.edu
BA4270 (behind elevators)
Web page:
http://www.teach.cs.toronto.edu/~heap/165/F17/
416-978-5899

Using Course notes: more Induction

Outline

notes

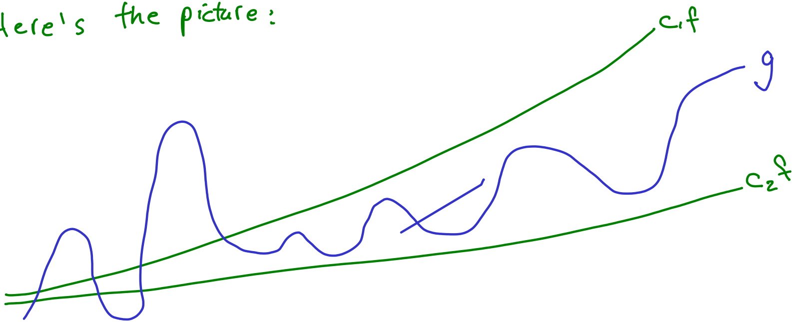
big-Theta means...

To describe running time we most often use Θ — it says two functions grow at the same rate.

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

$$\Rightarrow g(n) \leq c_1 f(n) \wedge g(n) \geq c_2 f(n)$$

Here's the picture:



products and sums redux

$(f+g)(n) \equiv f(n) + g(n)$

extremely useful for simplifying expressions for runtime

- $f + g$ $\quad g \in O(f) \wedge f \in O(h) \Rightarrow f+g \in O(h)$

  $f+g \in \Omega(f)$

  $f+g \in \Theta(f)$

- $af \in \Theta(f)$

- $f \cdot g$ $\quad f \in O(h_1) \wedge g \in O(h_2) \Rightarrow f \cdot g \in O(h_1 \cdot h_2)$

$(f \cdot g)(n) \equiv f(n) \cdot g(n)$

back to code...

"steps" and input "size"

operations whose runtime does not depend on the input size. It is valid to lump an entire block of such code into 1 "step"

strictly speaking, size of input in bits (0s and 1s). In practice we initially treat all integers as though they require the same # of bits, and sometimes ignore the size of list elements and focus on size of list. We often label the size of input n

counting loops...

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

all of these have run-time independent of n, so this block can count as 1 "step"

$\Theta(1)$

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

1 step

iterates 10 times $i = 0, 1, 2, \ldots, 9$

so 1×10 steps

$\Theta(1)$

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

1 step

iterates n times, $i = 0, 1, \ldots, n-1$

$1 \times n$ steps $\Theta(n)$

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

1 step

1 step

iterates for i (at end) $= 1, 2, \ldots, i^2 \geq n$

$i \geq \sqrt{n}$

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

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

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

1 step

1 step

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

iterates for i (at end) $= 1, 2, \ldots, \sqrt{i} \geq n$

$i \geq n^2$

$i = n^2$

nested loops

$1 \times n \times \lceil \frac{n}{2} \rceil \in \Theta(n^2)$

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

1 step

n steps

$\lceil \frac{n}{2} \rceil$

i (at end) $= 2, 4, \dots, 2k \geq n \rightarrow k \geq \frac{n}{2}$

$k = \lceil n/2 \rceil$

so $\Theta(n^2)$

```
def f6(n):
    for i in range(n):
        for j in range(i):
            print(i - j)
```

1 step

i steps

n steps

Cannot multiply $1 \times i \times n$ since $i \in \{0, 1, \dots, n-1\}$

must add individual iterations of inner loop:

$0 + 1 + 2 + \dots + n-1 = \frac{(n-1)(n)}{2}$

$\frac{n^2 - n}{2} \in \Omega(n^2)$

$c = 1/4$

$n_0 = 2$

$\frac{n^2 - n}{2} \in O(n^2)$ [choose $c = 1$]

composition, combination

```
def f7(n):
    for k in range(n):
        f6(n)
        f5(n)
```

f2

$f6(n)$ — $\Theta(n^2)$

$f5(n)$ — $\Theta(n)$

$\Theta(n^2)$

iterates n times

$\Theta(n^2 \times n) = \Theta(n^3)$

clumsy is_prime

```
def is_prime(n):
    if n < 2:
        return False
    else:
        for d in range(2,n):
            if n % d == 0:
                return False
        return True
```

n	$RT_{is\_prime}(n)$
35	5
36	2
37	37
⋮	⋮

$\left.\begin{array}{l} O(n) \\ \Omega(1) \end{array}\right\}$?#!

You cannot assume RT will be a "nice" function defined in terms of elementary functions

Notes

```
def twisty3(n):
    while n > 0:
        if n % 3 == 0:
            n = n//3
        elif n % 3 == 1:
            n = 3n-3
        else:
            n = 3n-6
```

n	$RT_{twisty3}(n)$
0	
1	
2	
3	
4	
5	
6	
7	
⋮	

fill these in