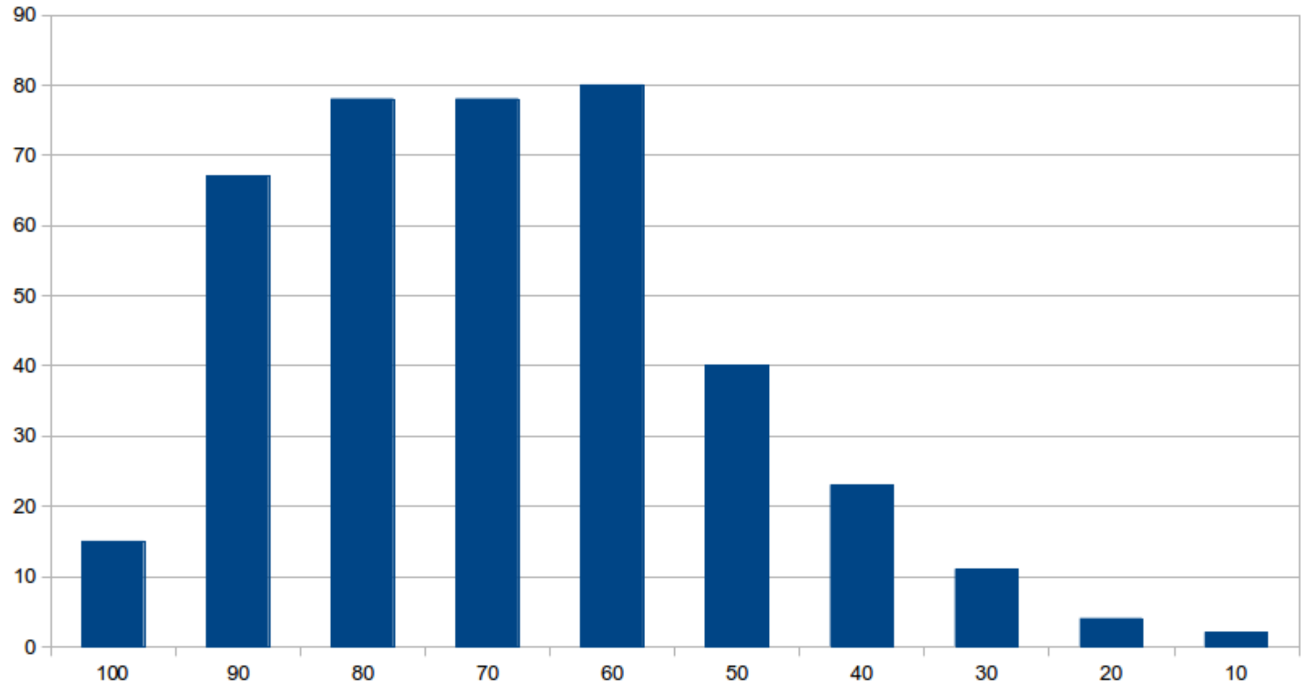
CSC165 Week 10

Larry Zhang, November 11, 2014

Test 2 result

average: 8.9 + 6.2 + 6.6 = 21.7 / 30



fill in this form for re-marking request

http://www.cdf.toronto.edu/~heap/165/F14/re-mark.txt

Next week: no lecture, no tutorial

Assignment 2 marks: ready by next week

Assignment 3 will be out sometime next week. Stay tuned.

today's outline

- ➔ big-Ω proof
- ➔ big-O proofs for general functions

- ➔ introduction to computability

Recap of definitions

upper bound

a function $f(n)$ is in $O(n^2)$ iff

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq cn^2$$

lower bound

a function $f(n)$ is in $\Omega(n^2)$ iff

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \geq cn^2$$

Recap of a proof for big-O

$$7n^6 - 5n^4 + 2n^3 \in \mathcal{O}(6n^8 - 4n^5 + n^2)$$

pick ***B*** *= 1 (magic brk-pt)*

assume n ≥ 1

large

$$6n^8 - 4n^5 + n^2$$

under-estimate

$$6n^8 - 4n^5$$

$$6n^8 - 4n^8 = \underline{2n^8}$$

pick a ***c large*** *enough to make the right side an* ***upper bound***

$$\boxed{9n^6 \leq \frac{9}{2} \cdot 2n^8}$$

$$7n^6 + 2n^6 = \underline{9n^6}$$

over-estimate

$$7n^6 + 2n^3$$

$$7n^6 - 5n^4 + 2n^3$$

small

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \geq cn^2$$

now a new proof

Prove $\quad n^2 + n \in \Omega(15n^2 + 3)$

*pick **B** = 1 (magic brk-pt)*

assume n ≥ 1

under-estimate

pick a **c** ***small*** enough to make the right side an ***lower bound***

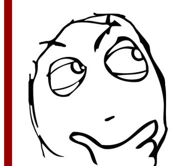
over-estimate

large

$$n^2 + n$$

$$n^2$$

$$\boxed{\frac{1}{18} \cdot (18n^2)}$$



$$c \cdot (18n^2)$$

$$c \cdot (15n^2 + 3n^2)$$

$$c \cdot (15n^2 + 3)$$

small

$$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \geq cn^2$$

Proof: $n^2 + n \in \Omega(15n^2 + 3)$

Pick $c = 1/18$, then $c \in \mathbb{R}^+$

Pick $B = 1$, then $B \in \mathbb{N}$

Assume $n \in \mathbb{N}$ # generic natural number

Assume $n \geq 1$ # n ≥ B, the antecedent

then $n^2 + n \geq n^2 = (1/18) \cdot 18n^2$ # n > 0, 1 = (1/18)18

$= (1/18) \cdot (15n^2 + 3n^2)$ # 18 = 15 + 3

$\geq (1/18) \cdot (15n^2 + 3) = c \cdot (15n^2 + 3)$ # n ≥ 1, c = 1/18

then $n^2 + n \geq c \cdot (15n^2 + 3)$

then $n \geq B \Rightarrow n^2 + n \geq c \cdot (15n^2 + 3)$ # intro =>

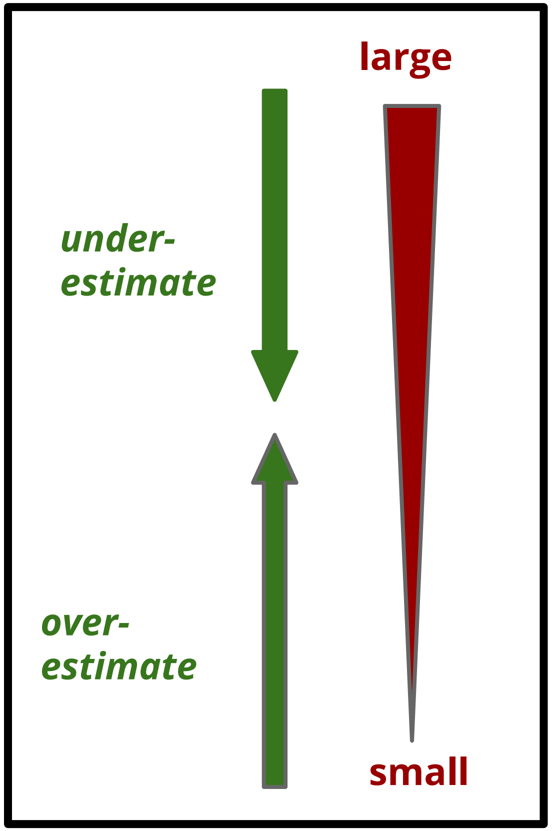
then $\forall n \in \mathbb{N}, n \geq B \Rightarrow n^2 + n \geq c \cdot (15n^2 + 3)$ # intro ∀

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow n^2 + n \geq c \cdot (15n^2 + 3)$ # intro ∃

then $n^2 + n \in \Omega(15n^2 + 3)$ # def of Ω

takeaway

choose Magic Breakpoint **B = 1**, then we can assume **n ≥ 1**



under-estimation tricks

→ **remove** a **positive** term
- $3n^2 + 2n \geq 3n^2$

→ **multiply** a **negative** term
- $5n^2 - n \geq 5n^2 - n \times n = 4n^2$

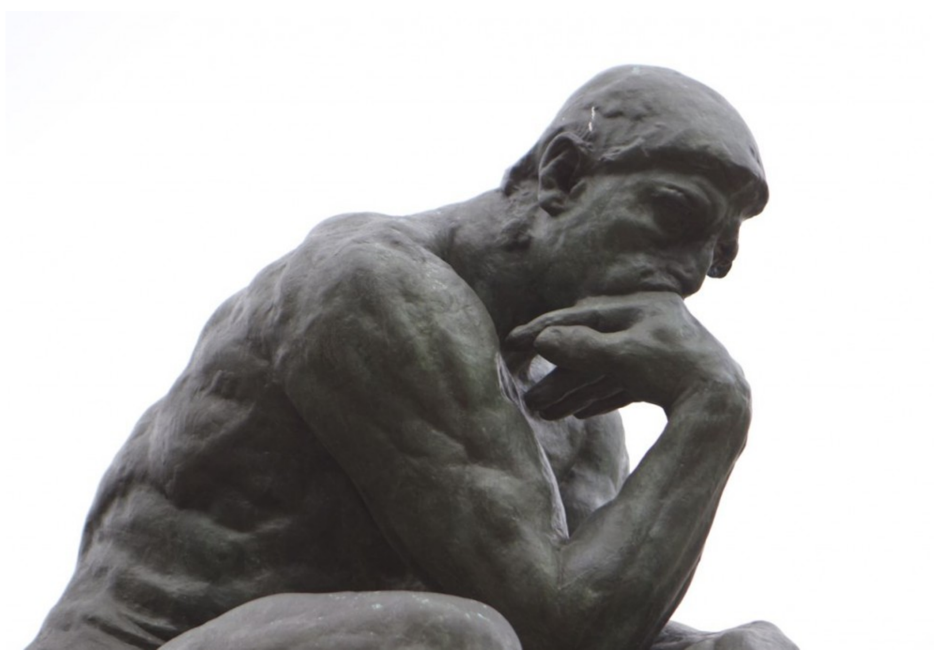
over-estimation tricks

→ **remove** a **negative** term
- $3n^2 - 2n \leq 3n^2$

→ **multiply** a **positive** term
- $5n^2 + n \leq 5n^2 + n \times n = 6n^2$

simplify the function without changing the highest degree

**now let's take a step back
and think about
what we have done**



all statements we have proven so far

$$3n^2 + 2n \in \mathcal{O}(n^2)$$

$$3n^2 + 2n + 5 \in \mathcal{O}(n^2)$$

$$7n^6 - 5n^4 + 2n^3 \in \mathcal{O}(6n^8 - 4n^5 + n^2)$$

$$n^3 \notin \mathcal{O}(3n^2)$$

$$2^n \notin \mathcal{O}(n^2)$$

$$n^2 + n \in \Omega(15n^2 + 3)$$

These are statements about specific functions.

It’s like …

Tim Horton’s is better than McDonalds.

Blue Jays is better than Yankees.

Ottawa is better Washington D.C.

Bieber is better than Lohan.

…

A general statement is more meaningful…

Canadian stuff is better than American stuff.

so, let's prove some general statements about big-Oh

a definition

$$\mathcal{F} : \{f : \mathbb{N} \to \mathbb{R}^{\geq 0}\}$$

*The **set** of all functions that take a natural number as input and return a non-negative real number.*

The set of all functions that we care about in CSC165.

now prove

$$\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$$

Intuition:

If ***f*** grows no faster than ***g***,
and ***g*** grows no faster than ***h***,
then ***f*** must grow no faster than ***h***.

thoughts

$$\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$$

want to find B”, c”, so that

f(n) ≤ c”h(n) beyond B”

Beyond B”:

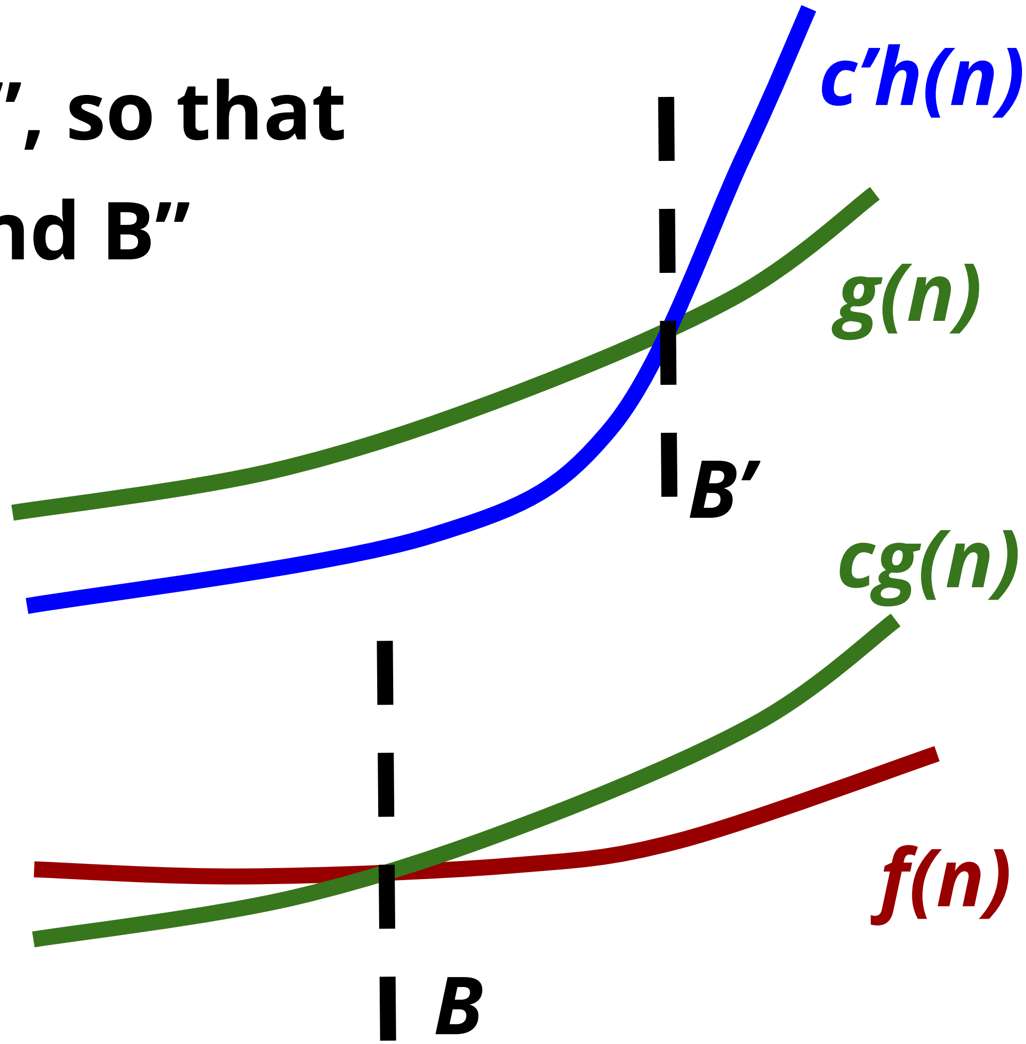
beyond both B & B’

B” = max(B, B’)

want f ≤ c” h

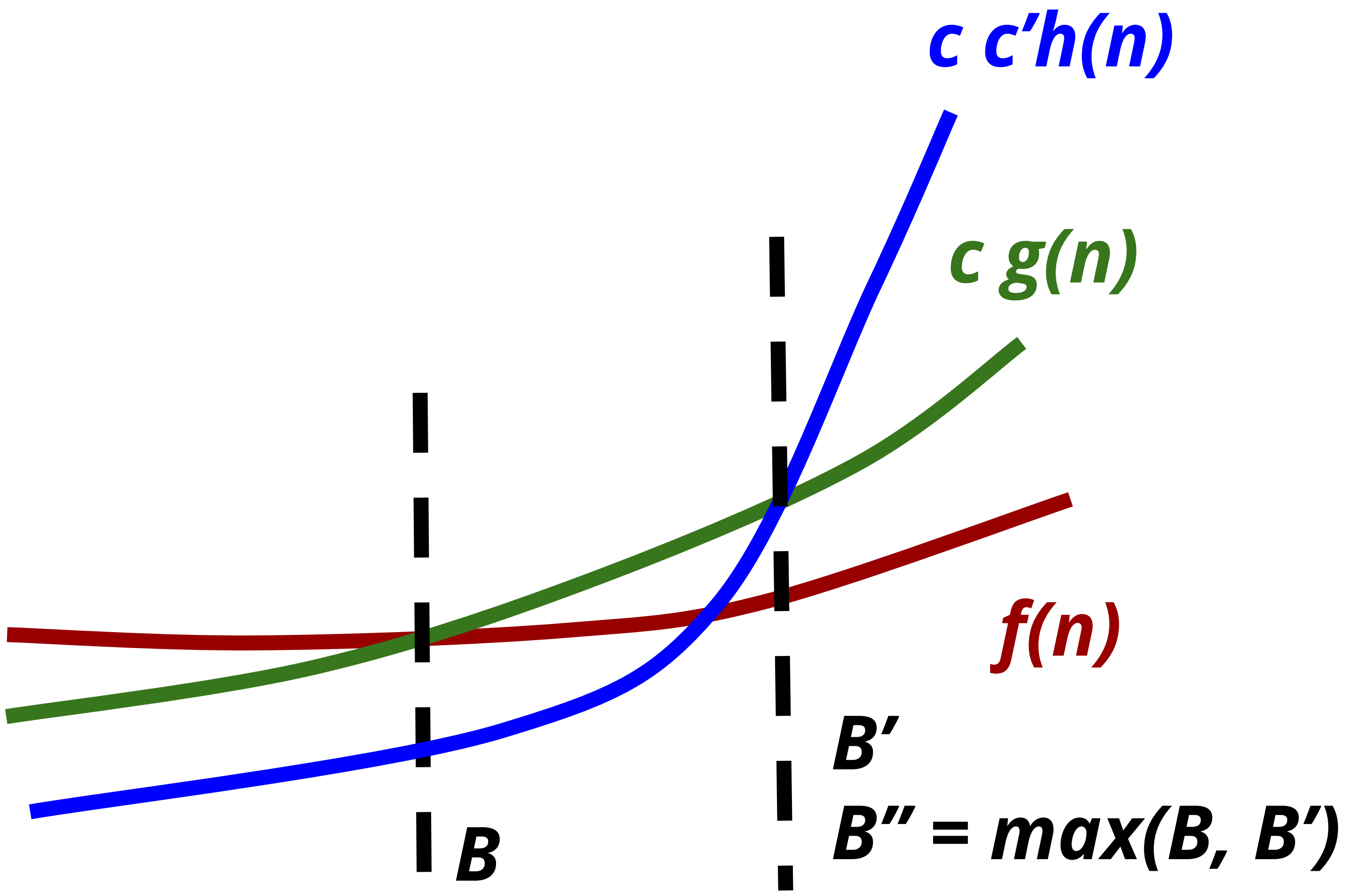
f ≤ cg ≤ c (c’h)

so **c” = cc’**



thoughts

$$\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$$



$$f \in \mathcal{O}(h) : \exists c'' \in \mathbb{R}^+, \exists B'' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B'' \Rightarrow f(n) \leq c''h(n)$$

Proof: $\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$

assume $f, g, h \in \mathcal{F}, f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)$

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \leq cg(n)$

then $\exists c' \in \mathbb{R}^+, \exists B' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B' \Rightarrow g(n) \leq c'h(n)$

pick $c'' = c \cdot c'$, then $c'' \in \mathbb{R}^+$

pick $B'' = \max(B, B')$, then $B'' \in \mathbb{N}$

assume $n \in \mathbb{N}, n \geq B''$

then $f(n) \leq cg(n)$ # f ∈ O(g) and n ≥ B

also $g(n) \leq c'h(n)$ # g ∈ O(h) and n ≥ B'

then $f(n) \leq cg(n) \leq cc'h(n) = c''h(n)$

then $\forall n \in \mathbb{N}, n \geq B'' \Rightarrow f(n) \leq c''h(n)$

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B'' \Rightarrow f(n) \leq c''h(n)$

then $\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h)$

another general statement

Prove $\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f)$

Intuition:

if ***f*** grows no faster than ***g***,

then ***g*** grows no slower than ***f***.

Prove $\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f)$

thoughts:

Assume $\quad f \in \mathcal{O}(g): \quad \boxed{n \geq B} \Rightarrow \boxed{f \leq cg} \quad g \geq \frac{1}{c} f$

Want to pick B', c' $\quad g \in \Omega(f): \quad \boxed{n \geq B'} \Rightarrow \boxed{g \geq c' f} \quad \text{pick } c' = \frac{1}{c}$

$$\text{pick } B' = B$$

$g \in \Omega(f) : \exists c' \in \mathbb{R}^+, \exists B' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B' \Rightarrow g \geq c'f$

Proof $\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f)$

assume $f, g \in \mathcal{F}, f \in \mathcal{O}(g)$ **# generic functions, *and antecedent***

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow f \leq cg$ **# def of O**

pick $c' = 1/c$, then $c' \in \mathbb{R}^+$ **# c > 0 so 1/c > 0**

pick $B' = B$, then $B' \in \mathbb{N}$ **# B is natural number**

assume $n \in \mathbb{N}, n \geq B'$ **# generic natural num, *and antecedent***

then $n \geq B$ **# since B' = B**

then $f \leq cg$ **# n ≥ B => f ≤ cg**

then $(1/c)f \leq g$ **# divide both sides by c > 0**

then $g \geq (1/c)f = c'f$ **# reverse inequality and c' = 1/c**

then $\forall n \in \mathbb{N}, n \geq B' \Rightarrow g \geq c'f$ **# intro ∀ and =>**

then $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B' \Rightarrow g \geq c'f$ **# intro ∃**

then $g \in \Omega(f)$ **# def of Ω**

then $\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow g \in \Omega(f)$ **# intro ∀ and =>**

yet another general statement

Proof writing left as exercise

Prove: $\forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(h) \wedge g \in \mathcal{O}(h)) \Rightarrow (f+g) \in \mathcal{O}(h)$

thoughts:

Assume $f \in \mathcal{O}(h) : \boxed{n \geq B} \Rightarrow \boxed{f \leq ch}$

and $g \in \mathcal{O}(h) : \boxed{n \geq B'} \Rightarrow \boxed{g \leq c'h}$

$$(f+g) \leq (c+c')h$$

Pick B″ = max(B, B′)

(make sure to be beyond both B and B′)

Pick c″ = c + c′

Want to pick B″, c″ $(f+g) \in \mathcal{O}(h) : n \geq B'' \Rightarrow (f+g) \leq c''h$

yet another one, trickier

$$\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow f \in \mathcal{O}(g \cdot g)$$

$$f \in \mathcal{O}(n) \Rightarrow f \in \mathcal{O}(n^2)$$

$$f \in \mathcal{O}(n^3) \Rightarrow f \in \mathcal{O}(n^6)$$

$$f \in \mathcal{O}\left(\frac{1}{n}\right) \Rightarrow f \in \mathcal{O}\left(\frac{1}{n^2}\right)$$

so $g = \dfrac{1}{n}$ gives the counterexample $\quad \mathcal{F} : \{f : \mathbb{N} \to \mathbb{R}^{\geq 0}\}$

more precisely $g = \dfrac{1}{n+1}$ **# so that n can be 0**

now we want to show

$$\frac{1}{n+1} \notin \mathcal{O}\left(\frac{1}{(n+1)^2}\right)$$ which by definition is

$$\forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, \boxed{n \geq B} \wedge \underline{\frac{1}{n+1} > \frac{c}{(n+1)^2}}$$

pick ***n*** wisely

$$n + 1 > c$$

$$\boxed{n > c - 1}$$

n = max(⌈c⌉, B) $\in \mathbb{N}$

$$f \notin \mathcal{O}(g \cdot g) : \forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N}, n \geq B \wedge f(n) > c \cdot g(n) \cdot g(n)$$

Disproof: $\forall f, g \in \mathcal{F}, f \in \mathcal{O}(g) \Rightarrow f \in \mathcal{O}(g \cdot g)$

Proof: $\exists f, g \in \mathcal{F}, f \in \mathcal{O}(g) \wedge f \notin \mathcal{O}(g \cdot g)$

Pick $f = g = \dfrac{1}{n+1}$, then $f, g \in \mathcal{F}, f \in \mathcal{O}(g)$ **# f is no faster than itself**

then $g \cdot g = \dfrac{1}{(n+1)^2}$ **# algebra**

assume $c \in \mathbb{R}^+, B \in \mathbb{N}$

pick $n = \max(\lceil c \rceil, B)$, then $n \in \mathbb{N}$ **# ceiling, B are both in N**

then $n + 1 > c$ **# n ≥ c, by def of ceiling and max**

then $\dfrac{1}{n+1} > \dfrac{c}{(n+1)^2}$ **# divide both sides by (n+1)²**

then $f > c \cdot g \cdot g$ **# because the choice of f, g**

also $n \geq B$ **# def of max**

then $n \geq B \wedge f > c \cdot g \cdot g$ **# conjunction introduction**

...introduce quantifiers and finish the proof (omitted)...

Summary of Chapter 4

- **definition** of big-Oh, big-Omega
- big-Oh proofs for **polynomials** (standard procedure with over/underestimates)
- big-Oh proofs for **non-polynomials** (need to use limits and L'Hopital's rule)
- proofs for **general** big-Oh statements (pick **B** and **c** based on known **B's** and **c's**)

all the proofs we have done establish your confidence in talking like this in the future

"The worst-case runtime of bubble-sort is in $\boxed{O(n^2)}$."

"I can sort it in $\boxed{n \log n \text{ time}}$."

"That's too slow, make it $\boxed{\text{linear-time}}$."

"That problem cannot be solved in $\boxed{\text{polynomial time}}$."

Chapter 5

Introduction to computability

why computers suck

… at certain things

- Computers solve problems using algorithms, in a systematic, repeatable way
- However, there are some problems that are not easy to solve using an algorithm
- What questions do you think an algorithm cannot answer?
- Some questions look like easy for computers to answer, but not really.

a python function f(n) that may or may not halt

```
def f(n):

    if n % 2 == 0:
        while True:
            pass
    else:
        return
```

```
def halt(f, n):
    '''return True iff
    f(n) halts'''

    return n % 2 != 0
```

only works for this particular f

Now devise an algorithm **halt(f, n)** that predicts whether this function **f** with input **n** eventually halts, i.e., will it ever stop?

another function f(n) that may or may not halt

```
def f(n):

   while n > 1:
      if n % 2 == 0:
         n = n / 2
      else:
         n = 3n + 1
   return “i is 1”
```

AFAIK, nobody knows how to write **halt(f, n)** for this function.

People know that **f(n)** halts for every single n up to more than 2^{58}. But we don't know whether it halts for ***all*** **n**.

Is it possible at all to write a **halt(f, n)** for this f ?

Answer: **not sure**.

what we are sure about

It is **impossible** to write **one halt(f, n)** that works for **all functions** f.

```
def halt(f, n):
    '''return True if f(n) halts, return false otherwise'''
    ...
```

It's not like "we don't know how to implement **halt(f, n)**".

It's like "nobody can possibly implement it, not in Python, or in any other programming language".

Why are we so sure about this?
Because we can **prove** it.

a naive thought of writing halt(f, n)

Why don’t we just implement **halt(f, n)** by calling **f(n)** and see if it halts?

If **f(n)** doesn’t halt, **halt(f, n)** never returns.
(it is supposed to return **False** in this case)

Prove: nobody can write halt(f, n)

thoughts:

suppose we could write it, construct a clever function that leads to **contradiction**

Now suppose we **can write** a **halt(f, n)** that works for all functions.

Prove: nobody can write halt(f, n)

```
def clever(f):

    while halt(f, f):
        pass
    return 42
```

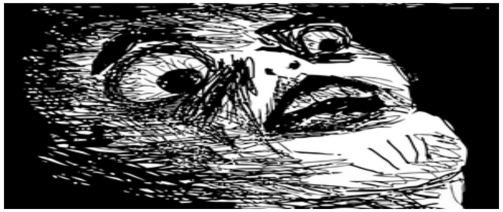
Now consider:

clever(clever)

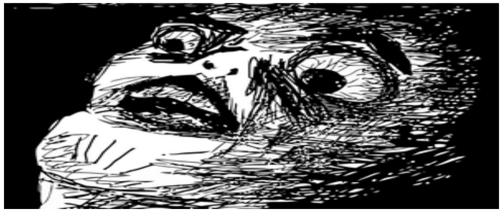
Does it halt?

Case 1:
assume **clever(clever)** **halts**
then **halt(clever, clever)** is **true**
then entering an infinite loop,
then **clever(clever)** does **not halt**

Case 2:
assume **clever(clever)** **doesn't halt**
then **halt(clever, clever)** is **false**
then just return 42
then **clever(clever)** **halts**



Contradiction in both cases, so we cannot write halt(f, n)

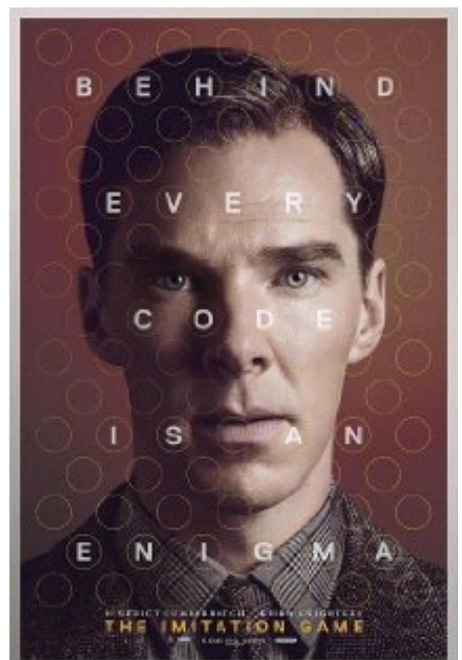


computers cannot solve the halting problem

The proof was first done Alonzo Church and Alan Turing, independently, in 1936.

(Yes, that’s before computers even existed!)

Alonzo Church
- ➔ Lambda calculus (CSC324)



Alan Turing
- ➔ Turing machine
- ➔ Turing test
- ➔ Turing Award

terminology

A function ***f*** is **well-defined** iff we can tell **what** ***f(x)*** is for every ***x*** in some domain

A function ***f*** is **computable** iff it is well-defined and we can tell **how** to compute ***f(x)*** for every ***x*** in the domain.

Otherwise, ***f(x)*** is **non-computable**.

halt(f, n) is well-defined and non-computable.

what we learn to do in CSC165

Given any function, decide whether it is computable not not.

how we do it

use **reductions**

Reductions

If function ***f(n)*** can be implemented by extending another function ***g(n)***, then we say ***f*** **reduces** to ***g***

```
def f(n):

    return g(2n)
```

g computable => f computable

f non-computable => g non-computable

f reduces to g

g computable => f computable

f non-computable => g non-computable

To prove a function **computable**

➔ show that this function reduces to a function **g** that is computable

To prove a function **non-computable**

➔ show that a non-computable function **f** reduces to this function

```python
def initialized(f, v):
    ‘’’return whether variable v is
       guaranteed to be initialized
       before its first use in f’’’
    ...
    return True/False
```

```python
def f1(x):
    return x + 1
    print y
```

```python
def f2(x):
    return x + y + 1
```

`initialized(f1, y)` **TRUE, because we never use y in f1**

`initialized(f2, y)` **FALSE, because we could use y before it is initialized**

```
def initialized(f, v):
    '''return whether variable v is
       guaranteed to be initialized
       before its first use in f'''
    ...
    return True/False
```

now prove: initialized(f, v) is non-computable

f reduces to g

f non-computable => g non-computable

halt(f, n)

Find a **non-computable function** that reduces to **initialized(f, v)**.

*A computable **initialized** would allow a computable **halt**. No way that's possible!*

all we need to show:
halt(f, n) reduces to initialized(f, v)

in other words, implement **halt(f, n)** using **initialized(f, v)**

```
def halt(f, n):

    def initialized(g, v):
        ...implementation of initialized...

    # code that scan code of f, and figure out
    # a variable name "v" that does not
    # appear anywhere in the code of f

    def f_prime(x):
       # ignore arg x, call f with fixed arg n
       # (the one passed in to halt)
       f(n)
       print(v)    # or exec("print(%s)" % v)

    return not initialized(f_prime, v)
```

If **f(n)** halts,
then, in f_prime, we get to **print(v)**, where **v** is not initialized, thus
not initialized(f_prime, v)
returns **true**.

If **f(n)** does not halt,
then, in **f_prime**, we never get to **print(v)**, thus
not initialized(f_prime, v)
returns **false**.

correct implementation!

summary

- ➔ Fact: **halt(f, n)** is non-computable
- ➔ use reductions to prove other functions being non-computable

next next week (last lecture)

- ➔ more on computability
- ➔ review for final exam