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

\[ 7n^6 - 5n^4 + 2n^3 \in O(6n^8 - 4n^5 + n^2) \]

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

**assume** \( n \geq 1 \)

**under-estimate**

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

\[ 6n^8 - 4n^5 + n^2 \]
\[ 6n^8 - 4n^5 \]
\[ 6n^8 - 4n^8 = 2n^8 \]
\[ 9n^6 \leq \frac{9}{2} \cdot 2n^8 \]

**over-estimate**

\[ 7n^6 + 2n^6 = 9n^6 \]
\[ 7n^6 + 2n^3 \]
\[ 7n^6 - 5n^4 + 2n^3 \]
\[ \exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \text{such that } \forall n \in \mathbb{N}, n \geq B \Rightarrow f(n) \geq cn^2 \]

**now a new proof**

Prove \[ n^2 + n \in \Omega(15n^2 + 3) \]

**pick** \( B = 1 \) *(magic brk-pt)*  
**assume** \( n \geq 1 \)

\[ n^2 + n \]

\[ \frac{1}{18} \cdot (18n^2) \]

\[ c \cdot (18n^2) \]

\[ c \cdot (15n^2 + 3n^2) \]

\[ c \cdot (15n^2 + 3) \]

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

**over-estimate**

**under-estimate**

**large**

**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 \( n \geq 1 \)  
   \# \( n \geq B \), the antecedent

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

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

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

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 \ \forall

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

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

\[ \# \text{ intro } \exists \]
choose Magic Breakpoint $B = 1$, then we can assume $n \geq 1$

**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

\[ 3n^2 + 2n \in O(n^2) \]
\[ 3n^2 + 2n + 5 \in O(n^2) \]
\[ 7n^6 - 5n^4 + 2n^3 \in O(6n^8 - 4n^5 + n^2) \]
\[ n^3 \notin O(3n^2) \]
\[ 2^n \notin O(n^2) \]
\[ n^2 + n \in \Omega(15n^2 + 3) \]

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

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

Intuition:

If \( f \) grows no faster than \( g \), and \( g \) grows no faster than \( h \), then \( f \) must grow no faster than \( h \).
thoughts

\[ \forall f, g, h \in \mathcal{F}, (f \in \mathcal{O}(g) \land g \in \mathcal{O}(h)) \Rightarrow f \in \mathcal{O}(h) \]

want to find \( B'' \), \( c'' \), so that \( f(n) \leq c''h(n) \) beyond \( B'' \)

Beyond \( B'' \):

beyond both \( B \) & \( B' \)

\( B'' = \max(B, B') \)

want \( f \leq c''h \)

\( f \leq cg \leq c \ (c'h) \)

so \( c'' = cc' \)
thoughts

∀f, g, h ∈ 𝐹, (f ∈ 𝑂(𝑔) ∧ g ∈ 𝑂(ℎ)) ⇒ f ∈ 𝑂(ℎ)

\[ B' \]

\[ B'' = \max(B, B') \]
Proof: \( \forall f, g, h \in \mathcal{F}, (f \in \Theta(g) \land g \in \Theta(h)) \Rightarrow f \in \Theta(h) \)

assume \( f, g, h \in \mathcal{F}, f \in \Theta(g) \land g \in \Theta(h) \)

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 \( f(n) \leq cg(n) \) \# \( f \in \Theta(g) \) and \( n \geq B \)

also \( g(n) \leq c'h(n) \) \# \( g \in \Theta(h) \) and \( n \geq B' \)

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 \( \forall f, g, h \in \mathcal{F}, (f \in \Theta(g) \land g \in \Theta(h)) \Rightarrow f \in \Theta(h) \)
another general statement
Prove $\forall f, g \in F, f \in O(g) \Rightarrow g \in \Omega(f)$

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 \( f \in \mathcal{O}(g) : n \geq B \Rightarrow f \leq cg \)

Want to pick \( B', c' \)

\( g \in \Omega(f) : n \geq B' \Rightarrow g \geq c'f \)

pick \( B' = B \)

\[ g \geq \frac{1}{c} f \]

pick \( c' = \frac{1}{c} \)
Proof: \( \forall f, g \in \mathcal{F}, f \in O(g) \Rightarrow g \in \Omega(f) \)

assume \( f, g \in \mathcal{F}, f \in O(g) \)  \# 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 \( f \leq cg \) \# \( n \geq B \Rightarrow f \leq cg \)

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 \( \forall \) 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 \( \exists \)

then \( g \in \Omega(f) \) \# def of \( \Omega \)

then \( \forall f, g \in \mathcal{F}, f \in O(g) \Rightarrow g \in \Omega(f) \) \# intro \( \forall \) and \( => \)
yet another general statement
Proof: \( \forall f, g, h \in \mathcal{F}, (f \in O(h) \land g \in O(h)) \Rightarrow (f + g) \in O(h) \)

thoughts:

Assume \( f \in O(h) : n \geq B \Rightarrow f \leq c h \)

and \( g \in O(h) : n \geq B' \Rightarrow g \leq c' h \)

Pick \( B'' = \max(B, B') \)

(make sure to be beyond both \( B \) and \( B' \))

Want to pick \( B'', c'' \)

\( (f + g) \in O(h) : n \geq B'' \Rightarrow (f + g) \leq c'' h \)

Pick \( c'' = c + c' \)
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 \( g = \frac{1}{n} \) gives the counterexample \( \mathcal{F} : \{ f : \mathbb{N} \to \mathbb{R}_{\geq 0} \} \)

more precisely \( g = \frac{1}{n + 1} \)  

# so that n can be 0
now we want to show

\[ \frac{1}{n+1} \notin O \left( \frac{1}{(n+1)^2} \right) \]

which by definition is

\[ \forall c \in \mathbb{R}^+, \forall B \in \mathbb{N}, \exists n \in \mathbb{N} \quad n \geq B \wedge \frac{1}{n+1} > \frac{c}{(n+1)^2} \]

pick \( n \) wisely

\[ n = \max(\lceil c \rceil, B) \]

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

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

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

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

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

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

pick \( n = \max([c], B) \), then \( n \in \mathbb{N} \) \# ceiling, \( B \) are both in \( \mathbb{N} \)

then \( n + 1 > c \) \# \( n \geq c \), by def of ceiling and max

then \( \frac{1}{n + 1} > \frac{c}{(n + 1)^2} \) \# 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 \( n \geq B \land f > c \cdot g \cdot g \) \# 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 $O(n^2)$.”

“I can sort it in $n \log n$ time.”

“That’s too slow, make it linear-time.”

“That problem cannot be solved in 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:

```python
def f(n):
    if n % 2 == 0:
        while True:
            pass
    else:
        return
```

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

```python
def halt(f, n):
    '''return True iff f(n) halts'''
    return n % 2 != 0
```

Only works for this particular `f`
another function \( f(n) \) that may or may not halt

```python
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 \texttt{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 \textit{all} \( n \).

Is it possible at all to write a \texttt{halt(f, n)} for this \( f \)?

Answer: \texttt{not sure}. 
what we are sure about

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

```python
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 $\text{halt}(f, n)$

Why don’t we just implement $\text{halt}(f, n)$ by calling $f(n)$ and see if it halts?

If $f(n)$ doesn’t halt, $\text{halt}(f, n)$ never returns. (it is supposed to return $\text{False}$ in this case)
Prove: nobody can write $\text{halt}(f, n)$

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

Now suppose we can write a $\text{halt}(f, n)$ that works for all functions.
Prove: nobody can write \( \text{halt}(f, n) \)

```
def clever(f):
    while \text{halt}(f, f):
        pass
    return 42
```

Now consider:

\text{clever(clever)}

Does it halt?

**Case 1:**

assume \text{clever(clever)} \text{ halts}

then \text{halt(clever, clever)} is \text{ true}

then entering an infinite loop,

then \text{clever(clever)} does \text{ not halt}

**Case 2:**

assume \text{clever(clever)} \text{ doesn’t halt}

then \text{halt(clever, clever)} is \text{ false}

then just return 42

then \text{clever(clever)} \text{ halts}

Contradiction in both cases, so we cannot write \( \text{halt}(f, n) \)
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**.

$\text{halt}(f, n)$ is well-defined and non-computable.
what we learn to do in CSC165

Given any function, decide whether it is computable or 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 $g$.

```python
def f(n):
    return g(2n)
```

g computable $\Rightarrow$ f computable

f non-computable $\Rightarrow$ g non-computable
f reduces to g

\[ g \text{ computable} \implies f \text{ computable} \]
\[ f \text{ non-computable} \implies g \text{ non-computable} \]

To prove a function \textbf{computable}

\rightarrow show that this function reduces to a function \textbf{g} that is computable

To prove a function \textbf{non-computable}

\rightarrow show that a non-computable function \textbf{f} reduces to this function
def initialized(f, v):
    '''return whether variable v is guaranteed to be initialized before its first use in f'''
    ...
    return True/False

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

initialized(f1, y)

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

initialized(f2, y)

TRUE, because we never use y in f1

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

Find a non-computable function that reduces to
initialized(f, v).
all we need to show: \( \text{halt}(f, n) \) reduces to \text{initialized}(f, v) \\

in other words, implement \text{halt}(f, n) using \text{initialized}(f, v) \\

```python
def halt(f, n):
    def initialized(g, v):
        ...implementation of \text{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 \text{halt})
        f(n)
        print(v)    # or exec("print(%s)") % v

    return not initialized(f_prime, v)
```

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

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

\[ \text{correct implementation!} \]
summary

➔ Fact: \texttt{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