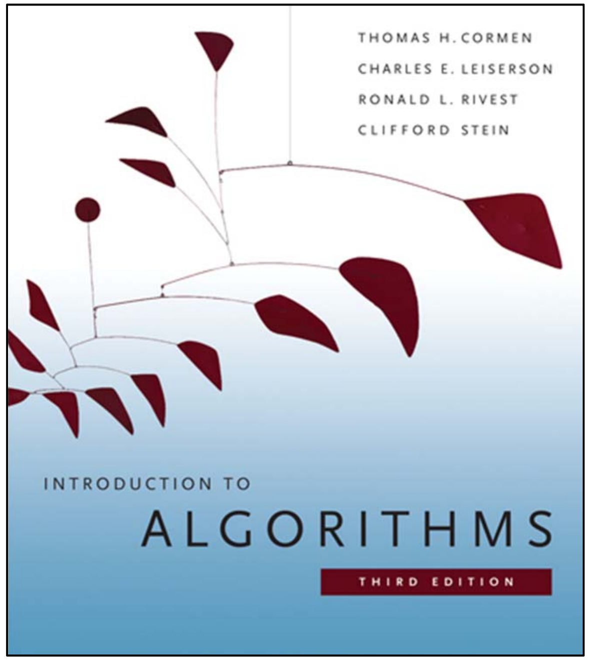
6.006

Introduction to Algorithms



Lecture 1: Document Distance

Prof. Erik Demaine

Your Professors

Prof. Erik Demaine

Prof. Piotr Indyk

Prof. Manolis Kellis

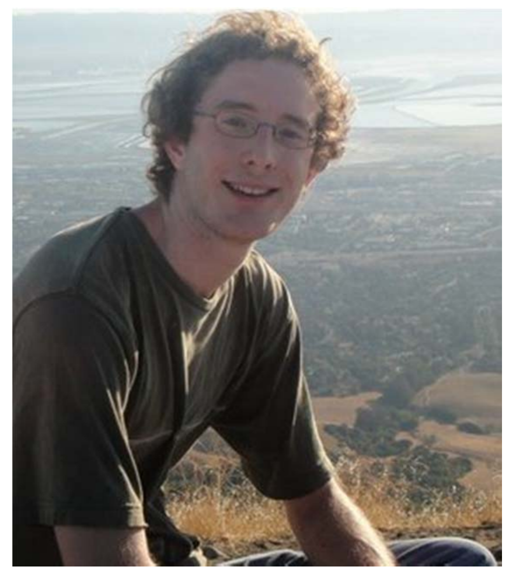
Your TAs

Kevin Kelley

Joseph Laurendi

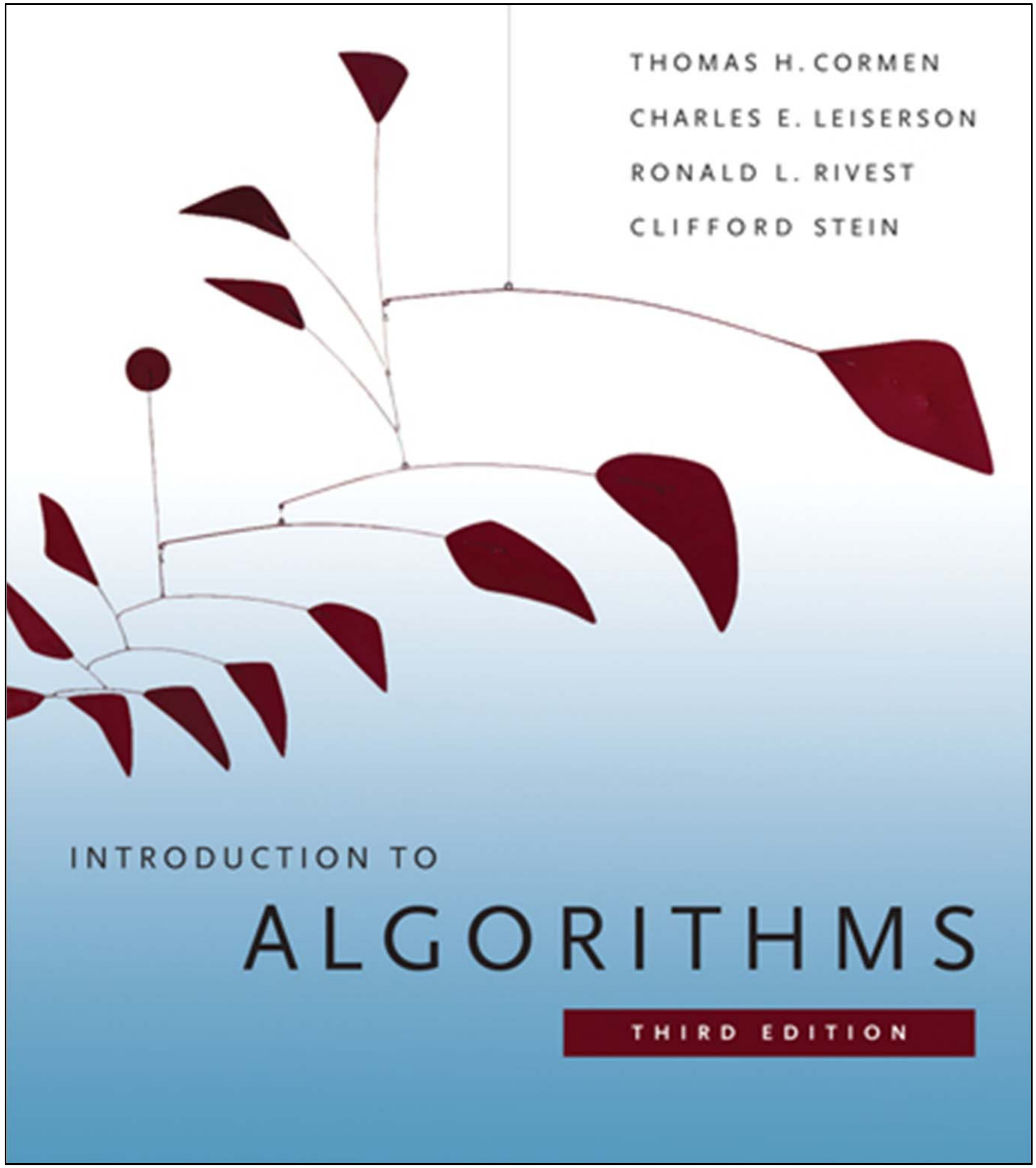
Tianren Qi

David Wen



Nicholas Zehender

Your Textbook



Administrivia

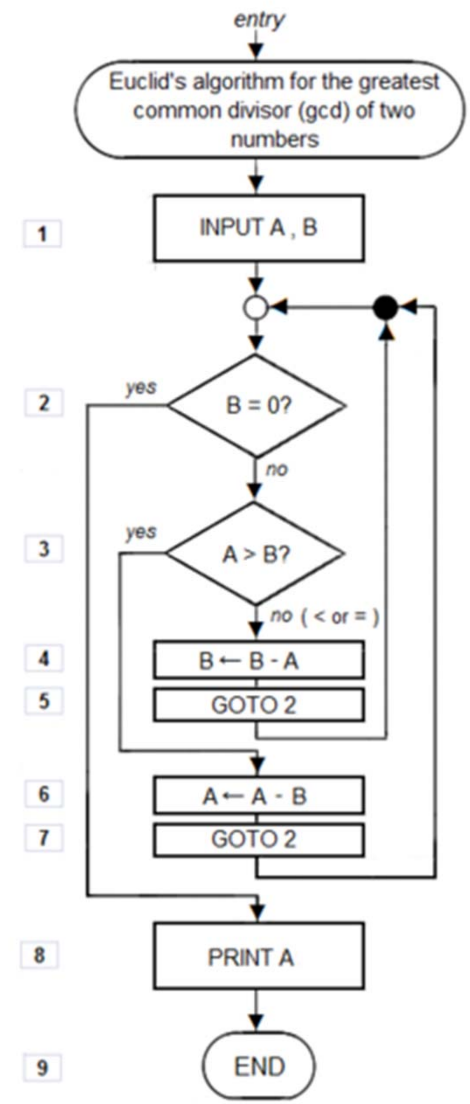
- **Handout:** Course information
- **Webpage:** http://courses.csail.mit.edu/6.006/spring11/
- **Sign up for recitation** if you didn't fill out form already
- **Sign up for problem set server:** https://alg.csail.mit.edu/
- **Sign up for Piazzza** account to ask/answer questions: http://piazzza.com/
- Prereqs: 6.01 (Python), 6.042 (discrete math)
- Grades: Problem sets (30%)
  Quiz 1 (20%; Mar. 8 @ 7.30–9.30pm)
  Quiz 2 (20%; Apr. 13 @ 7.30–9.30pm)
  Final (30%)
- Lectures & Recitations; Homework labs; Quiz reviews
- Read collaboration policy!

Today

- Class overview
  - What's a (good) algorithm?
  - Topics
- Document Distance
  - Vector space model
  - Algorithms
  - Python profiling & gotchas

What's an Algorithm?

- Mathematical abstraction of computer program
- Well-specified method for solving a computational problem
  - Typically, a finite sequence of operations
- Description might be structured English, pseudocode, or real code
- <u>Key:</u> no ambiguity



http://en.wikipedia.org/wiki/File:Euclid_flowchart_1.png

al-Khwārizmī

(c. 780–850)

- "al-kha-raz-mi"

http://en.wikipedia.org/wiki/Al-Khwarizmi

http://en.wikipedia.org/wiki/File:Abu_Abdullah_Muhammad_bin_Musa_al-Khwarizmi_edit.png

al-Khwārizmī

(c. 780–850)

- “al-kha-raz-mi”
- Father of algebra
  - *The Compendious Book on Calculation by Completion and Balancing* (c. 830)
  - Linear & quadratic equations: some of the first algorithms

http://en.wikipedia.org/wiki/Al-Khwarizmi

http://en.wikipedia.org/wiki/File:Image-Al-Kit%C4%81b_al-mu%E1%B8%ABta%E1%B9%A3ar_f%C4%AB_%E1%B8%A5is%C4%81b_al-%C4%9Fabr_wa-l-muq%C4%81bala.jpg

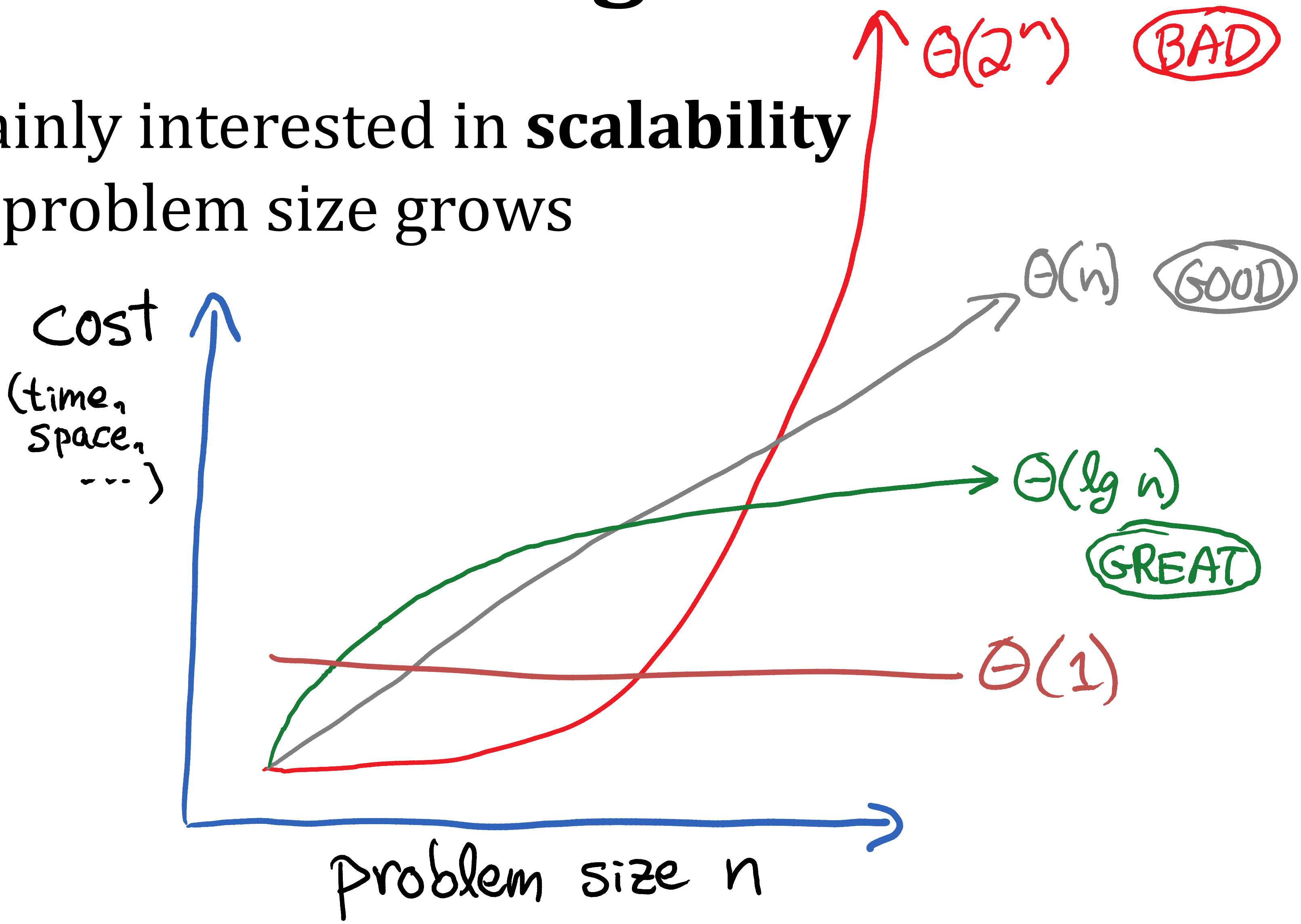
Efficient Algorithms

- Want an algorithm that's
  - **Correct**
  - **Fast**
  - Small space
  - General
  - Simple
  - Clever

"cool"

Efficient Algorithms

- Mainly interested in **scalability** as problem size grows



Why Efficient Algorithms?

- Save wait time, storage needs, energy consumption/cost, …
- Scalability = win
  - Solve bigger problems given fixed resources (CPU, memory, disk, etc.)
- Optimize travel time, schedule conflicts, …

How to Design an Efficient Algorithm?

7 EASY STEPS!

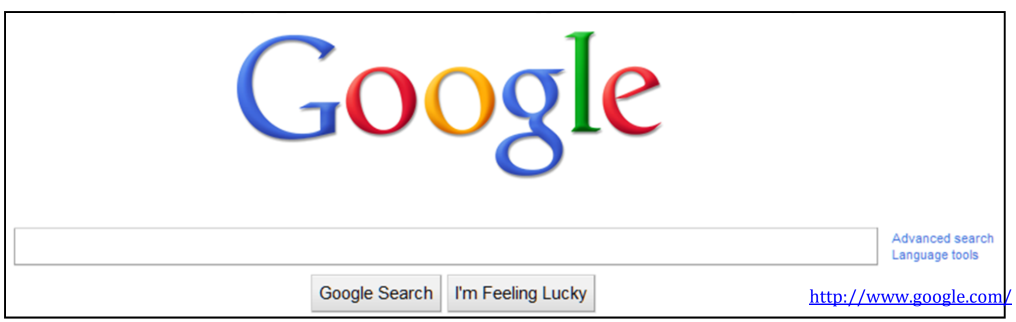
1. Define computational problem
2. Abstract irrelevant detail
3. Reduce to a problem you learn here
   (or 6.046 or algorithmic literature)
4. Else design using “algorithmic toolbox”
5. Analyze algorithm’s scalability
6. Implement & evaluate performance
7. Repeat (optimize, generalize)

Modules & Applications

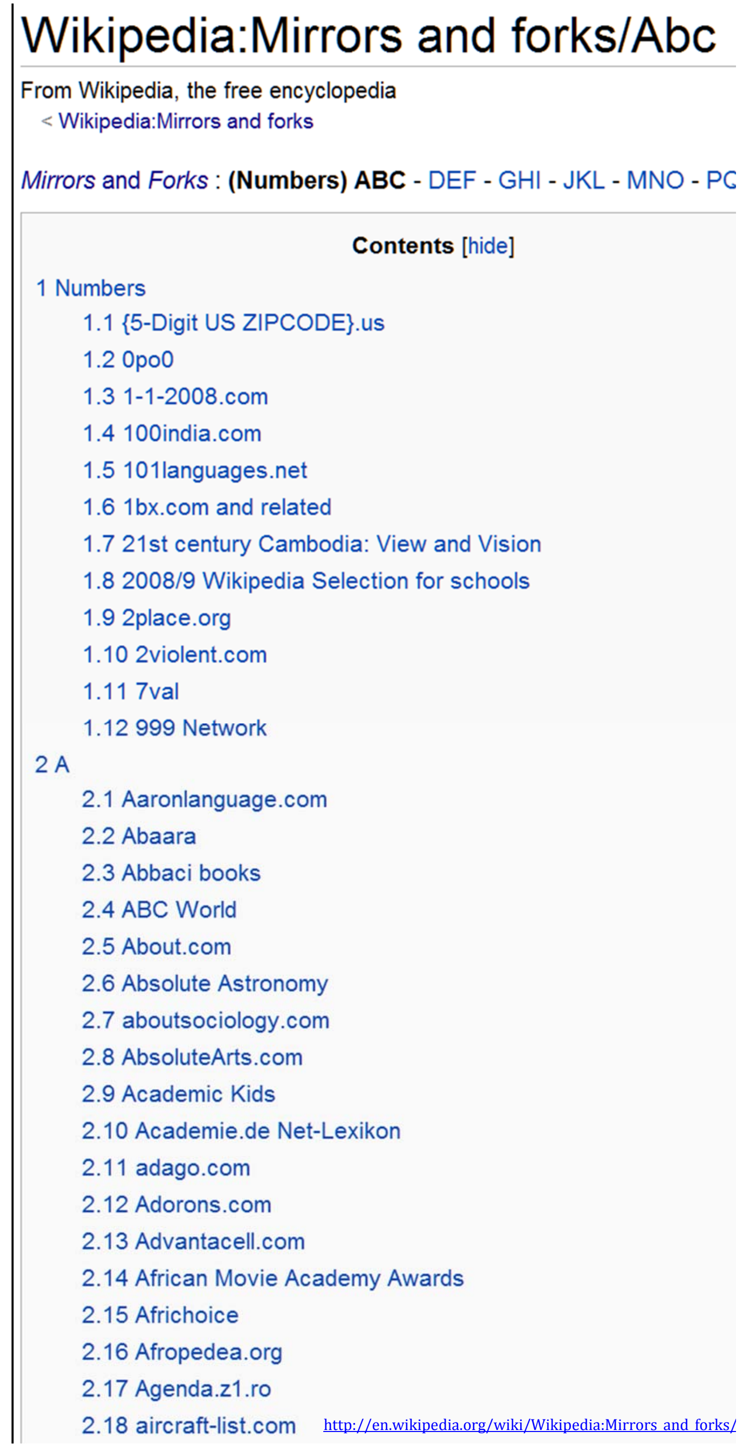
1. Introduction	Document similarity
2. Binary Search Trees	Scheduling
3. Hashing	File synchronization
4. Sorting	Spreadsheets
5. Graph Search	Rubik's Cube
6. Shortest Paths	Google Maps
7. Dynamic Programming	Justifying text, packing, ..
8. Numbers Pictures (NP)	Computing π, collision detection, hard problem
9. Beyond	Folding, streaming, bio

Document Distance

- Given two documents, how similar are they?
- Applications:
  - Find similar documents
  - Detect plagiarism / duplicates
  - Web search



http://www.google.com/



Wikipedia:Mirrors and forks/Abc

From Wikipedia, the free encyclopedia
< Wikipedia:Mirrors and forks

Mirrors and *Forks* : **(Numbers) ABC** - DEF - GHI - JKL - MNO - PC

Contents [hide]

http://en.wikipedia.org/wiki/Wikipedia:Mirrors_and_forks/

Document Distance

- How to define "document"?
- **Word** = sequence of alphanumeric characters
- **Document** = sequence of words
  - Ignore punctuation & formatting

Collaboration policy

The goal of homework is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate on problem sets. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study group meeting. Specifically, you should spend at least 30-45 minutes trying to solve each problem beforehand. If your group is unable to solve a problem, try asking questions via Piazzza so that other groups and the course staff can be helpful.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You are asked on problem sets to identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that you cannot orally explain to a member of the course staff.**

Code you submit must also be written by yourself. You may receive help from your classmates during deugging. Don't spend hours trying to debug a problem in your code before asking for help. However, regardless of who is helping you, only you are allowed to make changes to your code. **Both manual and automatic mechanisms will be employed to detect plagiarism in code.**

No other 6.006 student may use your solutions; this includes your writing, code, tests, docu- mentation, etc. **It is a violation of the 6.006 collaboration policy to permit anyone other than 6.006 staff and yourself to see your solutions to either theory or code questions.**

Plagiarism and other anti-intellectual behavior cannot be tolerated in any academic environment that prides itself on individual accomplishment. If you have any questions about the collaboration policy, or if you feel that you may have violated the policy, please talk to one of the course staff. Although the course staff is obligated to deal with cheating appropriately, we often have the ability to be more understanding and lenient if we find out from the transgressor himself or herself rather than from a third party.

Document Distance

- How to define “distance”?
- <u>Idea:</u> focus on shared words
- **Word frequencies:**
  - $D(w)$ = # occurrences of word w in document D

Collaboration policy

The goal of homework is to give you practice in mastering the course material. Consequently, you are encouraged to collaborate on problem sets. In fact, students who form study groups generally do better on exams than do students who work alone. If you do work in a study group, however, you owe it to yourself and your group to be prepared for your study group meeting. Specifically, you should spend at least 30-45 minutes trying to solve each problem beforehand. If your group is unable to solve a problem, try asking questions via Piazzza so that other groups and the course staff can be helpful.

You must write up each problem solution by yourself without assistance, even if you collaborate with others to solve the problem. You are asked on problem sets to identify your collaborators. If you did not work with anyone, you should write "Collaborators: none." If you obtain a solution through research (e.g., on the web), acknowledge your source, but write up the solution in your own words. **It is a violation of this policy to submit a problem solution that you cannot orally explain to a member of the course staff.**

Code you submit must also be written by yourself. You may receive help from your classmates during deugging. Don't spend hours trying to debug a problem in your code before asking for help. However, regardless of who is helping you, only you are allowed to make changes to your code. **Both manual and automatic mechanisms will be employed to detect plagiarism in code.**

No other 6.006 student may use your solutions; this includes your writing, code, tests, docu- mentation, etc. **It is a violation of the 6.006 collaboration policy to permit anyone other than 6.006 staff and yourself to see your solutions to either theory or code questions.**

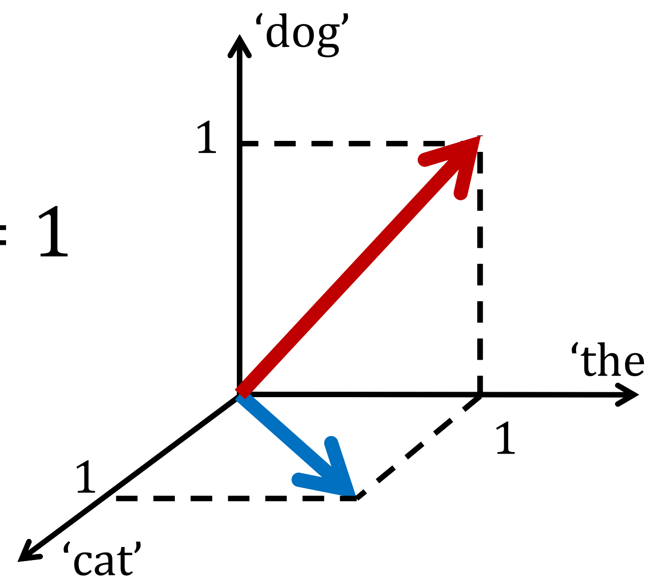
Plagiarism and other anti-intellectual behavior cannot be tolerated in any academic environment that prides itself on individual accomplishment. If you have any questions about the collaboration policy, or if you feel that you may have violated the policy, please talk to one of the course staff. Although the course staff is obligated to deal with cheating appropriately, we often have the ability to be more understanding and lenient if we find out from the transgressor himself or herself rather than from a third party.

Vector Space Model

[Salton, Wong, Yang 1975]

- Treat each document D as a vector of its words
  - One coordinate $D(w)$ for every possible word w
- Example:
  - D_1 = "the cat"
  - D_2 = "the dog"
- Similarity between vectors?
  - **Dot product:**

$$D_1 \cdot D_2 = 1$$



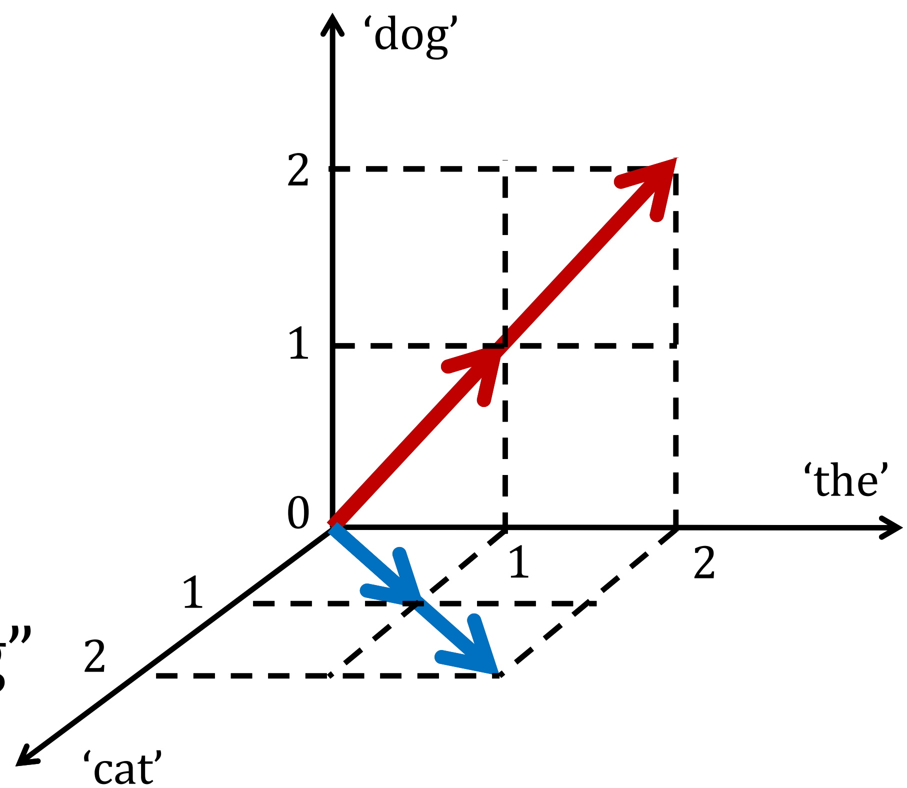
$$D_1 \cdot D_2 = \sum_w D_1(w) \cdot D_2(w)$$

http://portal.acm.org/citation.cfm?id=361220

Vector Space Model

[Salton, Wong, Yang 1975]

- <u>Problem:</u> Dot product not scale invariant
- Example 1:
  - D_1 = "the cat"
  - D_2 = "the dog"
  - $D_1 \cdot D_2 = 1$
- Example 2:
  - D_1 = "the cat the cat"
  - D_2 = "the dog the dog"
  - $D_1 \cdot D_2 = 2$



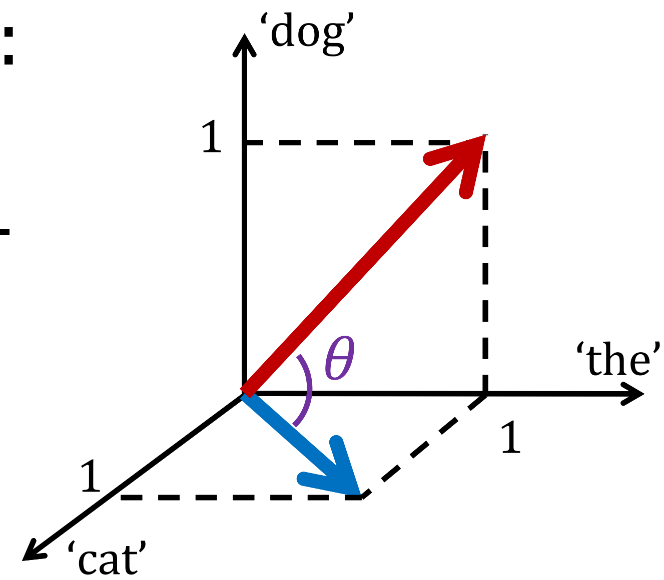
http://portal.acm.org/citation.cfm?id=361220

Vector Space Model

[Salton, Wong, Yang 1975]

- Idea: Normalize by # words:

$$\frac{D_1 \cdot D_2}{||D_1|| \cdot ||D_2||}$$



- Geometric solution: angle between vectors

$$\theta(D_1, D_2) = \arccos \frac{D_1 \cdot D_2}{||D_1|| \cdot ||D_2||}$$

  – 0 = "identical", 90° = orthogonal (no shared words)

http://portal.acm.org/citation.cfm?id=361220

Algorithm

1. Read documents
2. Split each document into words
3. Count word frequencies (document vectors)
4. Compute dot product

Algorithm

1. Read documents
2. **Split each document into words**
   - `re.findall('\w+', doc)`
   - But how does this actually work?
3. Count word frequencies (document vectors)
4. Compute dot product

Algorithm

1. Read documents
2. **Split each document into words**

   $O\left(\sum_{line} |line|\right) = O(|document|)$

   – For each line in document:

     For each character in line:

       If not alphanumeric:

         Add previous word

           (if any) to list

         Start new word

   $O(1)$ $O(|line|)$

3. Count word frequencies (document vectors)
4. Compute dot product

Algorithm

1. Read documents
2. Split each document into words
3. **Count word frequencies (document vectors)**
   a. Sort the word list $\leftarrow O(w \lg w)$, $w = \#$ words

      MODULE 4
   b. For each word in word list:
      – If same as last word: $\leftarrow O(|word|)$
        Increment counter
      – Else:
        Add last word and its counter to list
        Reset counter to 0

      $O(1)$

      $O\left(\sum_{word} |word|\right)$
      $= O(|document|)$
4. Compute dot product

Algorithm

1. Read documents
2. Split each document into words
3. Count word frequencies (document vectors)
4. **Compute dot product:**

   For every possible word: $\leftarrow \infty$ iterations...

      Look up frequency in each document

      Multiply

      Add to total

Algorithm

1. Read documents
2. Split each document into words
3. Count word frequencies (document vectors)
4. **Compute dot product:**

   For every word in first document: ← w_1 iterations

       If it appears in second document: ← $O(w_2)$

           Multiply word frequencies

           Add to total

   $O(1)$

   $O(w_1 w_2)$

Algorithm

1. Read documents
2. Split each document into words
3. Count word frequencies (document vectors)
4. **Compute dot product:**
    a. Start at first word of each document (in sorted order)
    b. If words are equal:
        Multiply word frequencies
        Add to total
    c. In whichever document has lexically lesser word, advance to next word
    d. Repeat until either document out of words

O(|word|)

O(|doc|)

Algorithm

1. Read documents
2. Split each document into words
3. **Count word frequencies** (document vectors)
   a. Initialize a dictionary mapping words to counts
   b. For each word in word list:
      – If in dictionary:
        Increment counter
      – Else:
        Put 0 in dictionary

$O(|word|) + O(1)$ "with high probability"

$O(|doc|)$ with high probability

4. Compute dot product

$\boxed{\text{MODULE } 3}$

Algorithm

1. Read documents
2. Split each document into words
3. Count word frequencies (document vectors)
4. **Compute dot product:**

   For every word in first document: ← w_1 iterations

       If it appears in second document: ← $O(|\text{word}|)$ with high probability

           $\left.\begin{array}{l}\text{Multiply word frequencies}\\ \text{Add to total}\end{array}\right\} O(1)$

$O(|doc_1|)$ with high probability

Python Implementations

```
##################################
# Operation 1: read a text file ##
##################################
def read_file(filename):

#################################################
# Operation 2: split the text lines into words ##
#################################################
def get_words_from_line_list(L):

##############################################
# Operation 3: count frequency of each word ##
##############################################
def count_frequency(word_list):

###############################################################
# Operation 4: sort words into alphabetic order             ###
###############################################################
def insertion_sort(A):
```

Python Profiling

```
import profile
profile.run("main()")
```

```
File t2.bobsey.txt : 6667 lines, 49785 words, 3354 distinct words
File t3.lewis.txt : 15996 lines, 182355 words, 8530 distinct words
The distance between the documents is: 0.574160 (radians)
         3397380 function calls in 192.500 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 :0(acos)
  1241849   10.325    0.000   10.325    0.000 :0(append)
  1300248   10.501    0.000   10.501    0.000 :0(isalnum)
   232140    2.076    0.000    2.076    0.000 :0(join)
   368314    2.960    0.000    2.960    0.000 :0(len)
   232140    1.856    0.000    1.856    0.000 :0(lower)
        2    0.000    0.000    0.000    0.000 :0(open)
        2    0.000    0.000    0.000    0.000 :0(range)
        2    0.028    0.014    0.028    0.014 :0(readlines)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
        1    0.000    0.000    0.000    0.000 :0(sqrt)
        1    0.008    0.008  192.500  192.500 <string>:1(<module>)
        2   16.121    8.061   16.121    8.061 docdist2.py:112(insertion_sort)
        2    0.000    0.000  191.748   95.874 docdist2.py:134(word_frequencies_f
or_file)
        3    0.384    0.128    0.732    0.244 docdist2.py:152(inner_product)
        1    0.000    0.000    0.732    0.732 docdist2.py:178(vector_angle)
        1    0.012    0.012  192.492  192.492 docdist2.py:188(main)
        2    0.000    0.000    0.028    0.014 docdist2.py:40(read_file)
        2   63.724   31.862  120.416   60.208 docdist2.py:55(get_words_from_line
_list)
    22663   29.414    0.001   56.691    0.003 docdist2.py:67(get_words_from_stri
ng)
        2   55.091   27.546   55.183   27.592 docdist2.py:95(count_frequency)
        1    0.000    0.000  192.500  192.500 profile:0(main())
        0    0.000             0.000          profile:0(profiler)
```

Culprit

```
##################################################
# Operation 2: split the text lines into words ##
##################################################
def get_words_from_line_list(L):
    """
    Parse the given list L of text lines into words.
    Return list of all words found.
    """

    word_list = []
    for line in L:
        words_in_line = get_words_from_string(line)
        word_list = word_list + words_in_line
    return word_list
```

$A+B$ costs $\Theta(|A|+|B|)$

total: $\sum_{j} \sum_{i \le j} (\#\text{ words in line } i) = \Theta(w^2)$ in worst case

Fix

```
##################################################
# Operation 2: split the text lines into words ##
##################################################
def get_words_from_line_list(L):
    """
    Parse the given list L of text lines into words.
    Return list of all words found.
    """

    word_list = []
    for line in L:
        words_in_line = get_words_from_string(line)
        # Using "extend" is much more efficient
        word_list.extend(words_in_line)
    return word_list
```

A += B has same benefit

A.extend(B) costs $\Theta(|B|)$

<u>total</u>: $O(\sum_i (\#\text{words in line } i)) = \Theta(w)$

Python Implementations

docdist1	initial version	
docdist2	add profiling	192.5 sec
docdist3	replace + with extend	126.5 sec
docdist4	count frequencies using dictionary	73.4 sec
docdist5	split words with string.translate	18.1 sec
docdist6	change insertion sort to merge sort	11.5 sec
docdist7	no sorting, dot product with dictionary	1.8 sec
docdist8	split words on whole document, not line by line	0.2 sec

Experiments on Intel Pentium 4, 2.8GHz, Python 2.6.2, Linux 2.6.18.
Document 1 (t2.bobsey.txt) has 268,778 lines, 49,785 words, 3,354 distincts.
Document 2 (t3.lewis.txt) has 1,031,470 lines, 182,355 words, 8,530 distincts.

Don’t Forget!

- **Webpage:** http://courses.csail.mit.edu/6.006/spring11/
- **Sign up for recitation** if you didn’t already receive a recitation assignment from us
- **Sign up for problem set server:** https://alg.csail.mit.edu/
- **Sign up for Piazzza** account to ask/answer questions: http://piazzza.com/