

L01.02

# Coding Cost model Document Distance

50.004 Introduction to Algorithm

Gemma Roig

(slides adapted from Dr. Simon LUI)

ISTD, SUTD

# About the coding requirement of the course

# coding requirement of this course

- we will be **reading and understanding** code / pseudo code
- You need to do some coding in exercise / homework / problem set
- You need to read/write code in exam
  - E.g. read a code, tell me if it is  $O(n)$  or  $O(n \log n)$

# coding requirement of this course

**What is pseudo code**

```
if a is divisible by 3  
    print “OH YEAH”
```

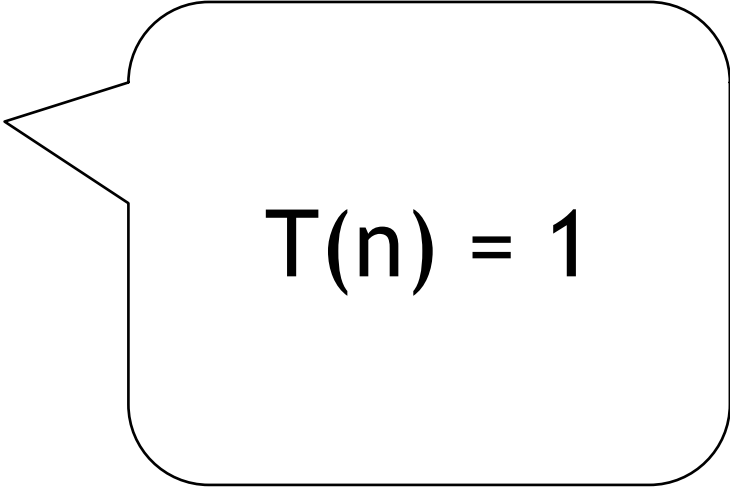
**What is code**

```
if (a%3 == 0)  
    print “OH YEAH”;
```

How to read code and find its  
 $T(n)$

# How to read code

`Print("GREAT");`

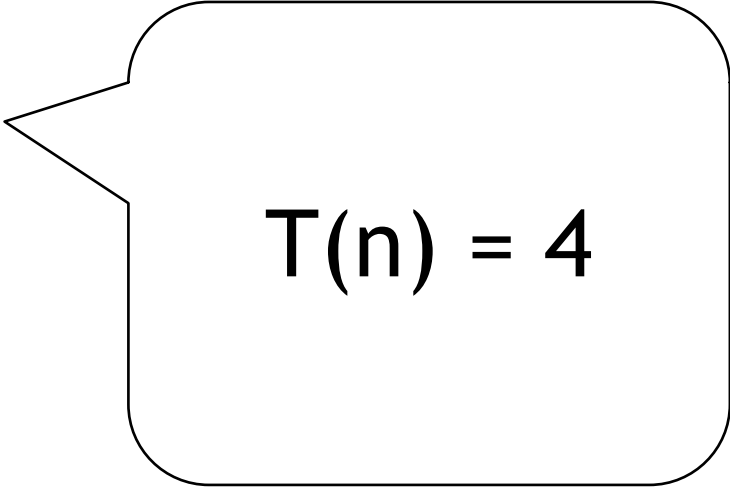

$$T(n) = 1$$

i.e. the number of steps  
required to complete this  
program is 1

**This code is  $\Theta(1)$**

# How to read code

```
Print("GREAT");  
Print("GREAT");  
Print("GREAT");  
Print("GREAT");
```

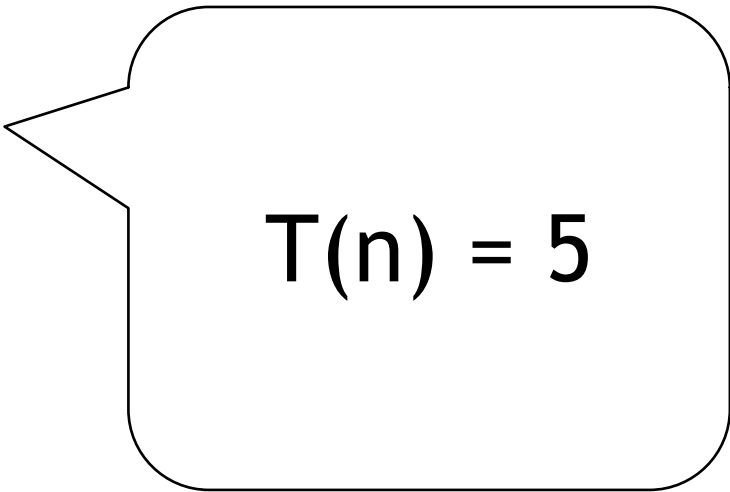

$$T(n) = 4$$

i.e. the number of steps  
required to complete this  
program is 4

**This code is  $\Theta(1)$**

# How to read code

```
For (i=0;i<5;i++)  
    Print("GREAT");
```


$$T(n) = 5$$

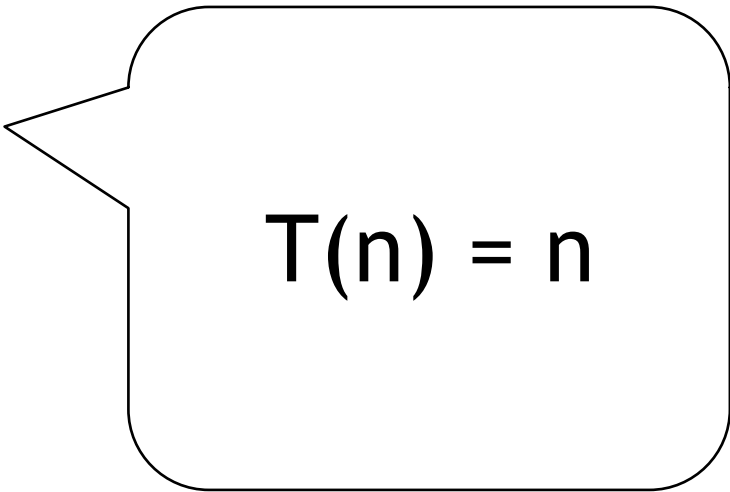
i.e. the number of steps  
required to complete this  
program is 5

**This code is  $\Theta(1)$**



# How to read code

```
For (i=0;i<n;i++)  
    Print("GREAT");
```

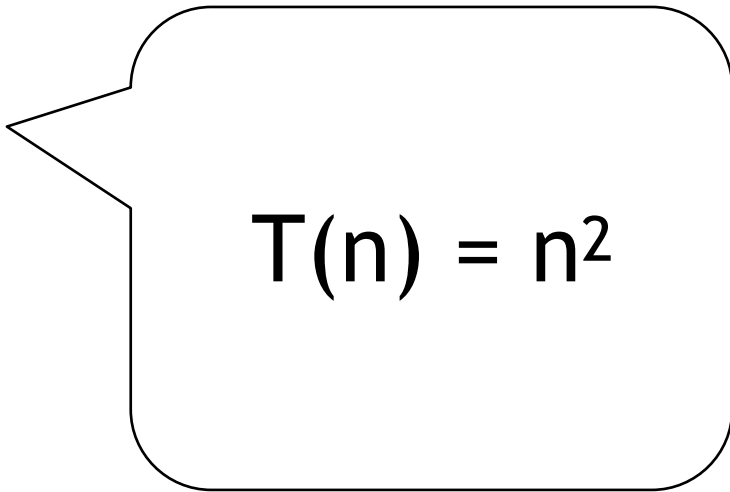

$$T(n) = n$$

i.e. the number of steps  
required to complete this  
program is  $n$

**This code is  $\Theta(n)$**

# How to read code

```
For (int i=0;i<n;i++)  
  For (int j=0;j<n;j++)  
    Print("GREAT");
```


$$T(n) = n^2$$

i.e. the number of steps  
required to complete this  
program is  $n^2$

This code is  
 $\Theta(n^2)$

# How to read code

```
For (i=0;i<n;i++)  
    For (j=0;j<n;j++)  
        Print("GREAT");  
For (k=0;k<n;k++)  
    print("GREAT");
```

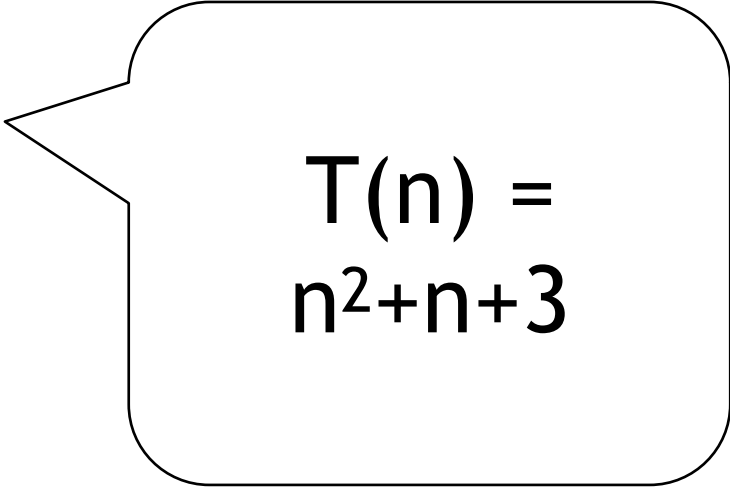

$$T(n) = n^2 + n$$

i.e. the number of steps  
required to complete this  
program is  $n^2 + n$

**This code is**  
 **$\Theta(n^2)$**

# How to read code

```
For (i=0;i<n;i++)  
    For (j=0;j<n;j++)  
        Print("GREAT");  
For (k=0;k<n;k++)  
    print("GREAT");  
print("GREAT");  
print("GREAT");  
print("GREAT");
```


$$T(n) = n^2 + n + 3$$

i.e. the number of steps required to complete this program is  $n^2 + n + 3$

This code is  
 $\Theta(n^2)$

# The cost model of code

# Python cost model

$L = [a_1, a_2, \dots, a_n]$   $L$  is a list

Commands	complexity (time)	
$L[i]=x$	$\Theta(1)$	
$L.append(x)$	$\Theta(1)$	Add $x$ to $L$
$L_1.extend(L_2)$	$\Theta( L_2 )$	Add $L_2$ to $L_1$
$A = L_1+L_2$	$\Theta( L_1 + L_2 )$	Add $L_2$ and $L_1$ to $A$
$x \text{ in } L$	$\Theta( L )$	Search for $x$

$D = \{x_1 : y_1, x_2 : y_2, \dots, x_n : y_n\}$   $D$  is a dictionary

Commands	complexity (time)	
$D[x_i]=y_i$	$\Theta(1)$	$X$ is index, $y$ is value
$x \text{ in } D$	$\Theta(1)$	Search for $x$

# Document distance

- *Document* means sequence of words
- The **Problem**: Given 2 documents, find out how similar are they
- The **Algorithm**: we will talk about it

# Document distance - usage

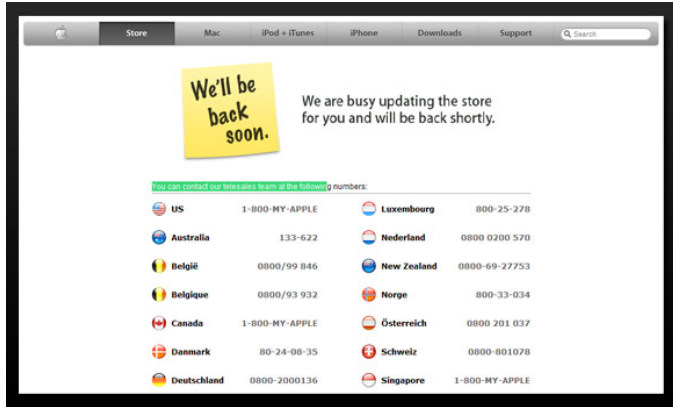
*1. detect plagiarism*

*2. Web update check*

- Simon loves iPhone. He wants to be the world first person to buy it*
- So he write a **DOCUMENT DISTANCE code** to check if the apple webpage has updated*
  - If updated, probably there are new product.*
  - So, send a email to notify Simon*



# Document distance - usage

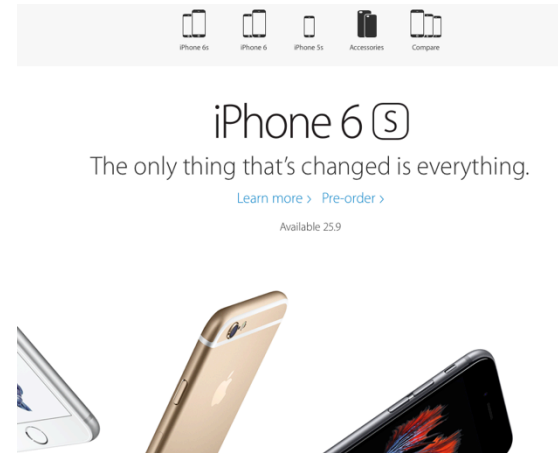


```
<!DOCTYPE html>
<html class="singapore sg nojs en en-sg apac" lang="en-SG">
  <head>
    <meta name="viewport" content="width=1024" />
    <title>Pre-order iPhone 6s and iPhone 6s Plus - Apple (SG)</title>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible"
content="IE=edge,chrome=1" />
    <meta name="format-detection" content="telephone=no" />
    <meta http-equiv="Set-Cookie"
content="as_sfa=MnxzZ3xzZ3x8ZW5fU0d8Y29uc3VtZXJ8aW50ZXJuZXR8MHwwfDE=; path=/; domain=.apple.com; expires=Mon,
15-Sep-2025 04:18:11 GMT;" />

    <meta property="og:type"
content="product" />
```

Apple.com/iphone.html  
...at 11:58am

←→  
**COMPARE**



```
<!DOCTYPE html>
<-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
<meta name="format-detection" content="telephone=no" />
<meta http-equiv="Set-Cookie"
content="as_sfa=MnxzZ3xzZ3x8ZW5fU0d8Y29uc3VtZXJ8aW50ZXJuZXR8MHwwfDE=; path=/; domain=.apple.com; expires=Mon,
15-Sep-2025 04:18:11 GMT;" />

    <meta property="og:type"
content="product" />
    <meta property="og:title" content="iPhone
6s" />

    <meta property="og:url"
content="http://www.apple.com/sg/shop/buy-iphone/
iphone6s" />

    <meta prope
```

Apple.com/iphone.html  
...at 11:59am

# Problem definition

- Document  $D$ : sequence of words
- Word  $w$ : sequence of characters
- **Word frequency**  $D(w)$ : number of occurrences of  $w$  in  $D$
- For example
  - $D$ : “good morning good good”
  - $w$ : {“good”, “morning”, “good”, “good”}
    - $w[0]$  = “good”
    - $w[1]$  = “morning”
    - $w[2]$  = “good”
    - $w[3]$  = “good”
  - $D(\text{“good”}) = 3$

# What we will do today

- I will show you some VERY COMPLICATED CODE
- No need to understand them (out of syllabus)
- I just want see
  - the running time of  $O(1)$ ,  $O(n)$  and  $O(n^2)$  function can be very different
  - How to improve the speed of a code

## Algorithm for document distance - Vector space model

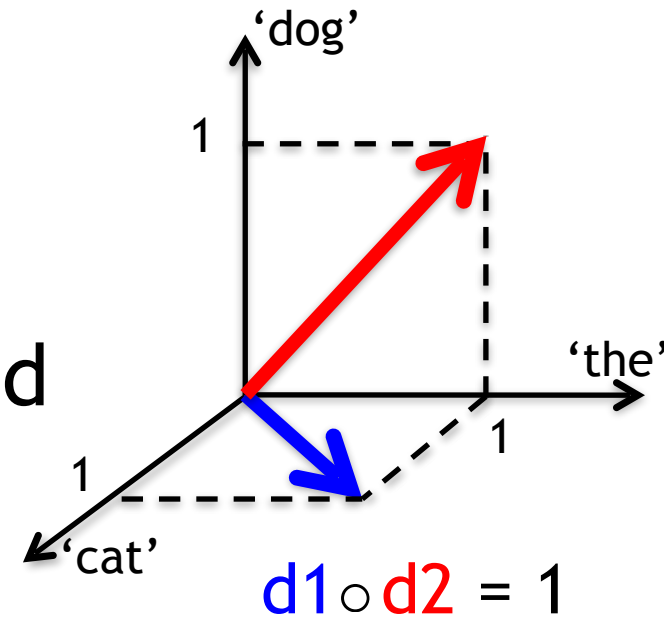
- Treat each document as a vector of its words
  - one coordinate per word of the English dictionary

e.g. **doc1** = “the cat”  
**doc2** = “the dog”

The dot product between D1 and D2

represent their similarity

$$D_1 \circ D_2 \equiv \sum_w D_1(w) \cdot D_2(w)$$



# Algorithm for document distance - Vector space model

e.g.

“the cat”。 ”the dog”

= [the x the + cat x dog]

= 1 + 0

= 1

# Algorithm for document distance - Vector space model

e.g.

“the the cat cat”。 ”the the dog dog”

= the x the + the x the + cat x dog + cat x dog

= 1 + 1 + 0 + 0

= 2

# Algorithm for document distance - Vector space model

... wait,

A: “the cat” vs “the dog”

B: “the the cat cat” vs “the the dog dog”

Both problem A and B are 50% similar.

Why B score more (2) than A (1)?

So, we need to **normalize** the result

# Algorithm for document distance - Vector space model

e.g.

["the cat", "the dog"] /  $\sqrt{2 \times 2}$

= [the x the + cat x dog] / 2

= [1 + 0] / 2

= 50%



# Algorithm for document distance - Vector space model

e.g.

["the the cat cat"。 "the the dog dog"] /  $\sqrt{4 \times 4}$

= [the x the + the x the + cat x dog + cat x dog]  
/ 4

= [1 + 1 + 0 + 0] / 4

= 50%

# Vector space model

- Normalization
  - divide by the length of the vectors

$$\frac{D_1 \circ D_2}{||D_1|| \cdot ||D_2||} \quad D_1 = aD_2$$

- measure distance by angle between vectors:

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

e.g.  $\theta=0$  documents “identical”  
(if of the same size, permutations of each other)  
 $\theta=\pi/2$  not even share a word

# Algorithm

aa**b**cv**bn**lbg**b**aa**n**laaub**a**aa**n**lcv**b**xf**eof**

- Read file

# Algorithm

aa**b**cv**bn**!bg**b**aa**n**!aaub**a**aa**n**!cv**b**xf**eof**

- Read file
- Make word list (divide file into words)

[aa,cv] + [bg,aa] + [aa,aa] + [cv,xf]

→ [aa, cv, bg, aa, aa, aa, cv,xf]

# Algorithm

aa**bc**cv**bn**lbg**ba**an**la**au**ba**an**lc**vb**xf****eof**

- Read file
- Make word list (divide file into words)
- Count frequencies of words

[aa, cv, bg, aa, aa, aa, cv,xf]



[(aa,3), (cv,2), (bg,1), (aa,1),(xf,1)]

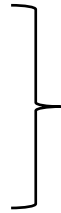
# Algorithm

aa**b**cv**bn**l**b**g**b**aa**n**laau**b**aa**n**lcv**b**xf**eof**

- Read file
- Make word list (divide file into words)
- Count frequencies of words
- Compute dot product

$D_1: [(aa, 3), (aa, 1), (bg, 1), (cv, 2)]$

$D_2: [(aa, 2), (ba, 2), (bg, 2), (ca, 1), (fv, 2)]$



# Algorithm

aa**b**cv**bn**lbg**b**aa**n**laa**u**baa**n**lc**v**bx**f**eof

- Read file
- Make word list (divide file into words)
- Count frequencies of words
- Compute dot product
  - for every word in the first document, check if it appears in the other document; if yes, multiply their frequencies and add to the dot product
    - worst case time: order of
$$\#words(D_1) \times \#words(D_2)$$

# Algorithm `aabcvnlbgbaanlaaubbaanlcvbxfeof`

- Read file
- Make word list (divide file into words)
- Count frequencies of words
- Compute dot product
  - for every word in the first document, check if it appears in the other document; if yes, multiply their frequencies and add to the dot product
    - worst case time: order of
$$\#words(D_1) \times \#words(D_2)$$
  - micro-optimization:
    - sort documents into word order (alphabetically)
    - compute inner product in time
$$\#words(D_1) + \#words(D_2)$$



# Python Implementation

- Docdist1.py (see handout)
- Read file: `read_file(filename)`
  - Output: list of lines (strings)
- Make word list: `get_words_from_line_list(L)`
  - Output: list of words (array)
- Count frequencies: `count_frequency(word list)`
  - Output: list of word-frequency pairs
- Dot product: `inner_product(D1, D2)`
  - Output: number

# cProfiling.run()

- return  $T(n)$ 
  - import profile
  - profile.run("main()")



auk:~/Class/6006/lectures/I01/

```
t1.verne.txt      t4.arabian.txt      t7.tenmillion.txt
t2.bobsey.txt     t5.churchill.txt    t8.shakespeare.txt
t3.lewis.txt      t6.onemillion.txt   t9.bacon.txt
auk:I01> python source/docdist2.py data/t2.bobsey.txt data/t3.lewis.txt
File data/t2.bobsey.txt : 6667 lines, 49785 words, 3354 distinct words
File data/t3.lewis.txt : 15996 lines, 182355 words, 8530 distinct words
The distance between the documents is: 0.574160 (radians)
3861660 function calls in 94.738 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	4.320	0.000	4.320	0.000	:0(append)
1300248	4.432	0.000	4.432	0.000	:0(isalnum)
232140	0.772	0.000	0.772	0.000	:0(join)
368314	1.300	0.000	1.300	0.000	:0(len)
232140	0.760	0.000	0.760	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	0.008	0.004	0.008	0.004	: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	0.004	0.004	94.738	94.738	<string>:1(<module>)
2	34.366	17.183	34.394	17.197	docdist2.py:105(count_frequency)
2	9.781	4.890	9.781	4.890	docdist2.py:122(insertion_sort)
2	0.000	0.000	94.438	47.219	docdist2.py:144(word_frequencies_for_file)
3	0.156	0.052	0.292	0.097	docdist2.py:162(inner_product)
1	0.000	0.000	0.292	0.292	docdist2.py:188(vector_angle)
1	0.004	0.004	94.734	94.734	docdist2.py:198(main)
2	0.000	0.000	0.008	0.004	docdist2.py:49(read_file)
2	23.605	11.803	50.255	25.128	docdist2.py:65(get_words_from_line_list)
226	12.409	0.001	26.650	0.001	docdist2.py:77(get_words_from_string)
1	0.000	0.000	94.738	94.738	profile:0(main())
0	0.000	0.000	0.000	0.000	profile:0(profiler)
232140	1.424	0.000	2.184	0.000	string.py:218(lower)
232140	1.396	0.000	2.168	0.000	string.py:306(join)

auk:I01> █

# docdist1

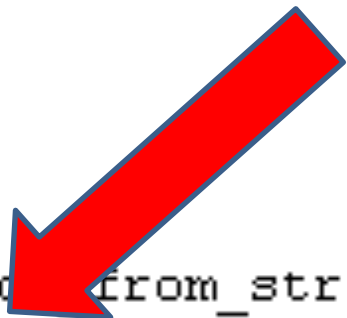
- Docdist1.py is slow
  - Docdist2.py is faster
  - Docdist3.py is even faster
  - ....
  - Docdist8.py is the best
- 
- Now let's look at docdist1.py

76 docdist1.py - C:\Documents and Settings\David\My Doc...



File Edit Format Run Options Windows Help

```
#####  
# 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
```




Ln: 130 Col: 18

# + is slow, need to improve it

- Look at word\_list = word\_list + words\_in\_line
- + : concatenation of arrays
- running time =  $T(k)$ 
  - $k = \text{length}(\text{word\_list}) + \text{length}(\text{words\_in\_line})$
- Total  $T(n)$  is  $1+2+\dots+n = n(n+1)/2 = \Theta(n^2)$

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



# Solution

- word\_list = word\_list + words\_in\_line:  $\Theta(n^2)$ 
  - docdist1.py (3.7s)
- word\_list.extend(words\_in\_line) :  $\Theta(n)$ 
  - docdist2.py (2.722s)
- How to run:
  - Python docdist1.py file5.txt file4.txt

# Further Improvements

- Docdist3.py: sort the data first (2.657s)
- Docdist4.py: Instead of inserting words in list, insert in dictionary: (0.735s)
- Docdist5.py: Process words instead of chars (0.138s)
- Docdist6.py: better sorting algorithm (0.069s)
  - merge sort instead of insertion sort
- Docdist7.py: dictionary instead of sort. (0.052s)



# Appendix - how to optimize the code

# docdist<1-8>.py - Iterative improvements

- Objective
  - To arrive at the most efficient implementation in terms of running time
- Approach
  - At each iteration, analyze the running times of various parts of the code
  - Based on the biggest consumer of running time, refactor the code by
    - Introducing faster algorithms
    - Using features of the Python language
    - Simplifying and optimizing code

# docdist1 - Code structure

```
1 def main():
2     if len(sys.argv) != 3:
3         print "Usage: docdist1.py filename_1 filename_2"
4     else:
5         filename_1 = sys.argv[1]
6         filename_2 = sys.argv[2]
7         document_vector_1 = word_frequencies_for_file(filename_1)
8         document_vector_2 = word_frequencies_for_file(filename_2)
9         distance = vector_angle(document_vector_1, document_vector_2)
10        print "The distance between the documents is: %0.6f (radians)"%distance
```

# docdist1 - Code structure

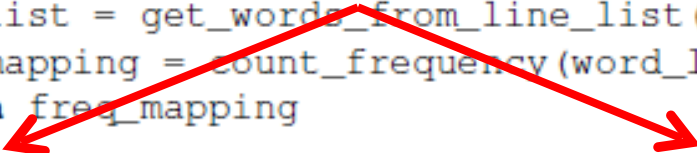
```
1 def main():
2     if len(sys.argv) != 3:
3         print "Usage: docdist1.py filename_1 filename_2"
4     else:
5         filename_1 = sys.argv[1]
6         filename_2 = sys.argv[2]
7         document_vector_1 = word_frequencies_for_file(filename_1)
8         document_vector_2 = word_frequencies_for_file(filename_2)
9         distance = vector_angle(document_vector_1, document_vector_2)
10        print "The distance between the documents is: %.6f (radians)"%distance
```



```
1 def word_frequencies_for_file(filename):
2     line_list = read_file(filename)
3     word_list = get_words_from_line_list(line_list)
4     freq_mapping = count_frequency(word_list)
5     return freq_mapping
```

# docdist1 - Code structure

```
1 def main():
2     if len(sys.argv) != 3:
3         print "Usage: docdist1.py filename_1 filename_2"
4     else:
5         filename_1 = sys.argv[1]
6         filename_2 = sys.argv[2]
7         document_vector_1 = word_frequencies_for_file(filename_1)
8         document_vector_2 = word_frequencies_for_file(filename_2)
9         distance = vector_angle(document_vector_1, document_vector_2)
10        print "The distance between the documents is: %0.6f (radians)"%distance
11
12    def word_frequencies_for_file(filename):
13        line_list = read_file(filename)
14        word_list = get_words_from_line_list(line_list)
15        freq_mapping = count_frequency(word_list)
16        return freq_mapping
17
18    def get_words_from_line_list(L):
19        word_list = []
20        for line in L:
21            words_in_line = get_words_from_string(line)
22            word_list = word_list + words_in_line
23        return word_list
24
25    def get_words_from_string(line):
26        word_list = []
27        character_list = []
28        for c in line:
29            if c.isalnum():
30                character_list.append(c)
31            elif len(character_list)>0:
32                word = "".join(character_list)
33                word = word.lower()
34                word_list.append(word)
35                character_list = []
36        if len(character_list)>0:
37            word = "".join(character_list)
38            word = word.lower()
39            word_list.append(word)
40        return word_list
```



# docdist1 - Code structure

```
1 def main():
2     if len(sys.argv) != 3:
3         print "Usage: docdist1.py filename_1 filename_2"
4     else:
5         filename_1 = sys.argv[1]
6         filename_2 = sys.argv[2]
7         document_vector_1 = word_frequencies_for_file(filename_1)
8         document_vector_2 = word_frequencies_for_file(filename_2)
9         distance = vector_angle(document_vector_1, document_vector_2)
10        print "The distance between the documents is: %0.6f (radians)"%distance
11
12    def word_frequencies_for_file(filename):
13        line_list = read_file(filename)
14        word_list = get_words_from_line_list(line_list)
15        freq_mapping = count_frequency(word_list)
16        return freq_mapping
```



```
1 def count_frequency(word_list):
2     L = []
3     for new_word in word_list:
4         for entry in L:
5             if new_word == entry[0]:
6                 entry[1] = entry[1] + 1
7             break
8         else:
9             L.append([new_word, 1])
10    return L
```

# docdist1 - Code structure

```
1 def main():
2     if len(sys.argv) != 3:
3         print "Usage: docdist1.py filename_1 filename_2"
4     else:
5         filename_1 = sys.argv[1]
6         filename_2 = sys.argv[2]
7         document_vector_1 = word_frequencies_for_file(filename_1)
8         document_vector_2 = word_frequencies_for_file(filename_2)
9         distance = vector_angle(document_vector_1, document_vector_2)
10        print "The distance between the documents is: %0.6f (radians)"%distance
```



```
1 def vector_angle(L1, L2):
2     numerator = inner_product(L1, L2)
3     denominator = math.sqrt(inner_product(L1, L1) * inner_product(L2, L2))
4     return math.acos(numerator/denominator)
```

# docdist1 vs. docdist2

```
1 def get_words_from_line_list(L):  
2     word_list = []  
3     for line in L:  
4         words_in_line = get_words_from_string(line)  
5         word_list = word_list + words_in_line  
6     return word_list
```

```
1 def get_words_from_line_list(L):  
2     word_list = []  
3     for line in L:  
4         words_in_line = get_words_from_string(line)  
5         word_list.extend(words_in_line)  
6     return word_list
```



# docdist1 vs. docdist2

## docdist1 Performance Scorecard

Method	Time
get_words_from_line_list	$O(\frac{W^2}{k}) = O(W^2)$
count_frequency	$O(WL)$
<b>word_frequencies_for_file</b>	$O(W^2)$
inner_product	$O(L_1L_2)$
<b>vector_angle</b>	$O(L_1L_2 + L_1^2 + L_2^2) = O(L_1^2 + L_2^2)$
<b>main</b>	$O(W_1^2 + W_2^2)$

## docdist2 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
<b>word_frequencies_for_file</b>	$O(WL)$
inner_product	$O(L_1L_2)$
<b>vector_angle</b>	$O(L_1^2 + L_2^2)$
<b>main</b>	$O(W_1L_1 + W_2L_2)$

# docdist2 vs. docdist3

```
1 def word_frequencies_for_file(filename):  
2     line_list = read_file(filename)  
3     word_list = get_words_from_line_list(line_list)  
4     freq_mapping = count_frequency(word_list)  
5     return freq_mapping
```

```
1 def word_frequencies_for_file(filename):  
2     line_list = read_file(filename)  
3     word_list = get_words_from_line_list(line_list)  
4     freq_mapping = count_frequency(word_list)  
5     insertion_sort(freq_mapping)  
6     return freq_mapping
```

# docdist3 - Implementing insertion\_sort()

```
1 def insertion_sort(A):
2     for j in range(len(A)):
3         key = A[j]
4         i = j-1
5         while i>-1 and A[i]>key:
6             A[i+1] = A[i]
7             i = i-1
8         A[i+1] = key
9     return A
```

# docdist3 - Reimplementing inner\_product()

```
1 def inner_product(L1,L2):
2     sum = 0.0
3     for word1, count1 in L1:
4         for word2, count2 in L2:
5             if word1 == word2:
6                 sum += count1 * count2
7     return sum
```

```
1 def inner_product(L1,L2):
2     sum = 0.0
3     i = 0
4     j = 0
5     while i<len(L1) and j<len(L2):
6         # L1[i:] and L2[j:] yet to be processed
7         if L1[i][0] == L2[j][0]:
8             # both vectors have this word
9             sum += L1[i][1] * L2[j][1]
10            i += 1
11            j += 1
12        elif L1[i][0] < L2[j][0]:
13            # word L1[i][0] is in L1 but not L2
14            i += 1
15        else:
16            # word L2[j][0] is in L2 but not L1
17            j += 1
18    return sum
```

SLIDE 10.12 Select filters for algorithms

# docdist2 vs. docdist3

## docdist2 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
<b>word_frequencies_for_file</b>	$O(WL)$
inner_product	$O(L_1L_2)$
<b>vector_angle</b>	$O(L_1^2 + L_2^2)$
main	$O(W_1L_1 + W_2L_2)$

## docdist3 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(WL + L^2) = O(WL)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
main	$O(W_1L_1 + W_2L_2)$

# docdist4 - Refining count\_frequency()

```
1 def count_frequency(word_list):
2     L = []
3     for new_word in word_list:
4         for entry in L:
5             if new_word == entry[0]:
6                 entry[1] = entry[1] + 1
7                 break
8         else:
9             L.append([new_word, 1])
10    return L
```

```
1 def count_frequency(word_list):
2     D = {}
3     for new_word in word_list:
4         if new_word in D:
5             D[new_word] = D[new_word]+1
6         else:
7             D[new_word] = 1
8     return D.items()
```

# docdist3 vs. docdist4

## docdist3 Performance Scorecard

Method	Time
get words from line list	$O(W)$
count_frequency	$O(WL)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(WL + L^2) = O(WL)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1L_1 + W_2L_2)$

## docdist4 Performance Scorecard

Method	Time
get words from line list	$O(W)$
count_frequency	$O(W)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(W + L^2) = O(L^2)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1^2 + L_2^2)$

# docdist5 - Simplifying get\_words\_from\_string()

```
8 def get_words_from_string(line):
9     word_list = []
10    character_list = []
11    for c in line:
12        if c.isalnum():
13            character_list.append(c)
14        elif len(character_list)>0:
15            word = "".join(character_list)
16            word = word.lower()
17            word_list.append(word)
18            character_list = []
19    if len(character_list)>0:
20        word = "".join(character_list)
21        word = word.lower()
22        word_list.append(word)
23    return word_list
```

```
1 translation_table = string.maketrans(string.punctuation+string.uppercase,
2                                     " "*len(string.punctuation)+string.lowercase)
3
4 def get_words_from_string(line):
5     line = line.translate(translation_table)
6     word_list = line.split()
7     return word_list
```



# docdist4 vs. docdist5 (same as docdist4)

## docdist4 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(W + L^2) = O(L^2)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1^2 + L_2^2)$

## docdist4 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(W + L^2) = O(L^2)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1^2 + L_2^2)$

# docdist6 - Replacing insertion sort by merge sort

```
1 def word_frequencies_for_file(filename):
2     line_list = read_file(filename)
3     word_list = get_words_from_line_list(line_list)
4     freq_mapping = count_frequency(word_list)
5     freq_mapping = merge_sort(freq_mapping)
6     return freq_mapping

1 def merge_sort(A):
2     n = len(A)
3     if n==1:
4         return A
5     mid = n//2
6     L = merge_sort(A[:mid])
7     R = merge_sort(A[mid:])
8     return merge(L,R)
9
10 def merge(L,R):
11     i = 0
12     j = 0
13     answer = []
14     while i<len(L) and j<len(R):
15         if L[i]<R[j]:
16             answer.append(L[i])
17             i += 1
18         else:
19             answer.append(R[j])
20             j += 1
21     if i<len(L):
22         answer.extend(L[i:])
23     if j<len(R):
24         answer.extend(R[j:])
25     return answer
```

# docdist5 vs. docdist6

## docdist4 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
insertion_sort	$O(L^2)$
<b>word_frequencies_for_file</b>	$O(W + L^2) = O(L^2)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1^2 + L_2^2)$

## docdist6 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
merge_sort	$O(L \log L)$
<b>word_frequencies_for_file</b>	$O(W + L \log L) = O(L \log L)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1 \log L_1 + L_2 \log L_2)$

# docdist7 - From sorted list to document vectors

```
1 def count_frequency(word_list):
2     D = {}
3     for new_word in word_list:
4         if new_word in D:
5             D[new_word] = D[new_word]+1
6         else:
7             D[new_word] = 1
8     return D
```

```
1 def word_frequencies_for_file(filename):
2     line_list = read_file(filename)
3     word_list = get_words_from_line_list(line_list)
4     freq_mapping = count_frequency(word_list)
5     return freq_mapping
```

```
1 def inner_product(D1,D2):
2     sum = 0.0
3     for key in D1:
4         if key in D2:
5             sum += D1[key] * D2[key]
6     return sum
```

# docdist6 vs. docdist7

## docdist6 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
merge_sort	$O(L \log L)$
<b>word_frequencies_for_file</b>	$O(W + L \log L) = O(L \log L)$
inner_product	$O(L_1 + L_2)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2 + L_1 \log L_1 + L_2 \log L_2)$

## docdist7 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
<b>word_frequencies_for_file</b>	$O(W)$
inner_product	$O(L_1)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2)$

# docdist8 - Simplifying the code

```
1 def get_words_from_string(string):  
2     string = string.translate(translation_table)  
3     word_list = string.split()  
4     return word_list  
5  
6 def word_frequencies_for_file(filename):  
7     text = read_file(filename)  
8     word_list = get_words_from_string(text)  
9     freq_mapping = count_frequency(word_list)  
10    return freq_mapping
```

# docdist7 vs. docdist8

## docdist7 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
<b>word_frequencies_for_file</b>	$O(W)$
inner_product	$O(L_1)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2)$

## docdist8 Performance Scorecard

Method	Time
get_words_from_text	$O(W)$
count_frequency	$O(W)$
<b>word_frequencies_for_file</b>	$O(W)$
inner_product	$O(L_1)$
<b>vector_angle</b>	$O(L_1 + L_2)$
<b>main</b>	$O(W_1 + W_2)$