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(nlogn)

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

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

The cost model of code

Python cost model

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

Commands	complexity (time)	
L[i]=x	Θ(1)	
L.append(x)	Θ(1)	Add x to L
L_1.extend(L_2)	Θ(|L_2|)	Add L_2 to L_1
A = L_1+L_2	Θ(|L_1|+|L_2|)	Add L_2 and L_1 to A
x in L	Θ(|L|)	Search for x

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

Commands	complexity (time)	
D[x_i]=y_i	Θ(1)	X is index, y is value
x in D	Θ(1)	Search for x

Document distance

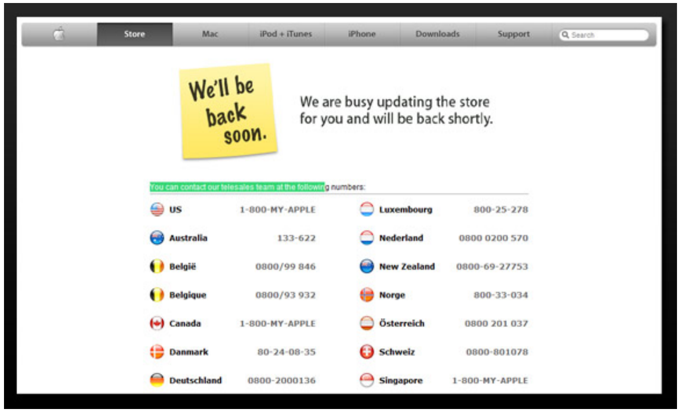
- *Document* means sequence of words
- The Problem: Given 2 documents, find out how similar are they
- The Algorithm: we will talk about it

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



We'll be back soon.

We are busy updating the store for you and will be back shortly.

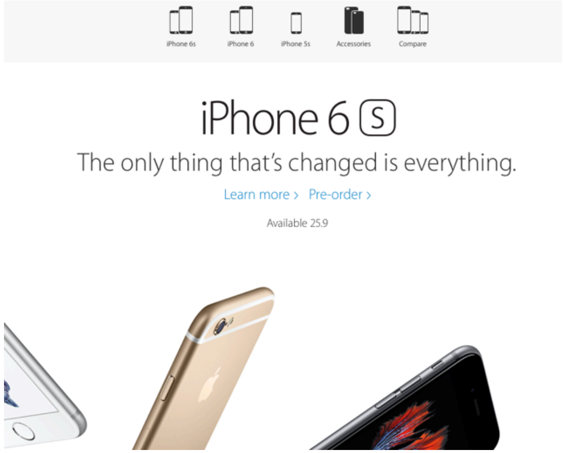
You can contact our telesales team at the following numbers:

US	1-800-MY-APPLE	Luxembourg	800-25-278
Australia	133-622	Nederland	0800 0200 570
België	0800/99 846	New Zealand	0800-69-27753
Belgique	0800/93 932	Norge	800-33-034
Canada	1-800-MY-APPLE	Österreich	0800 201 037
Danmark	80-24-08-35	Schweiz	0800-801078
Deutschland	0800-2000136	Singapore	1-800-MY-APPLE

```
<!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=MnxzZ3xzZ3x8ZW5fU0d8Y29uc3VtZXJ8aW50ZXJ
uZXR8MHwwfDE=; 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



iPhone 6 Ⓢ

The only thing that's changed is everything.

Learn more › Pre-order ›

Available 25.9

```
<!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=MnxzZ3xzZ3x8ZW5fU0d8Y29uc3VtZXJ8aW50ZXJ
uZXR8MHwwfDE=; 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(“good”) = 3

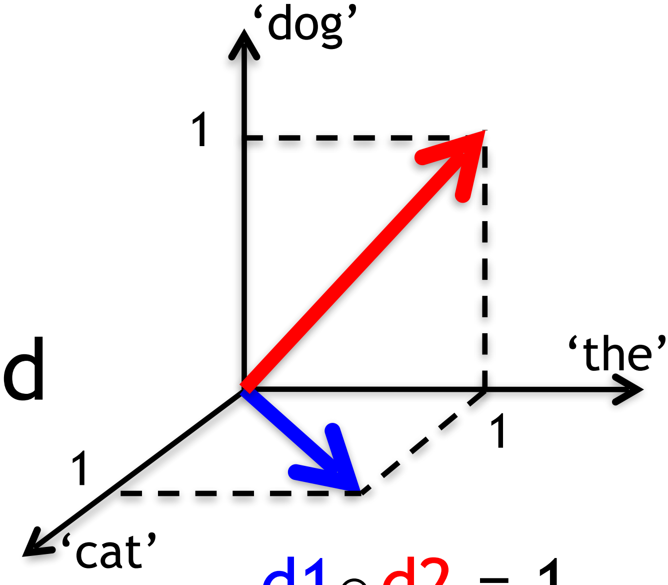
What we will do today

- 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”



d1∘d2 = 1

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 x 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 x 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

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

$$D_1 = aD_2$$

  - measure distance by angle between vectors:

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

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*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*eof*

- Read file

Algorithm

aa*b*cv*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*eof*

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

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

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

Algorithm

aa*b*cv*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*eof*

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

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

↓

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

Algorithm

aa*b*cv*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*eof*

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

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

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

Algorithm

aa*b*cv*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*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) x #words(D_2)

Algorithm

aa*b*cv*bnl*bg*b*aa*nl*aau*b*aa*nl*cv*b*xf*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

      $\#\text{words}(D_1) \times \#\text{words}(D_2)$
  - micro-optimization:
    - sort documents into word order (alphabetically)
    - compute inner product in time

      $\#\text{words}(D_1) + \#\text{words}(D_2)$

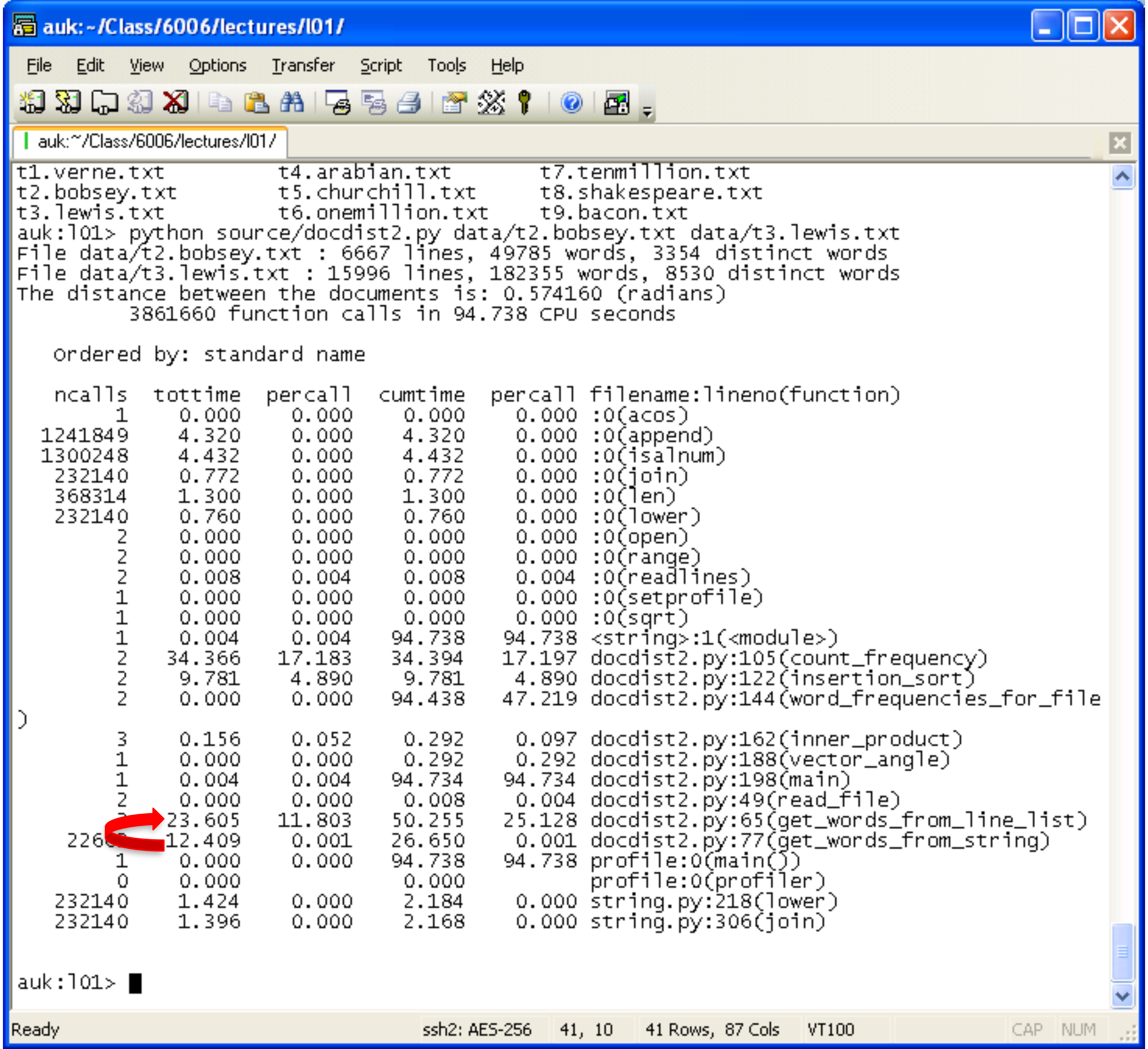
Python Implementation

- 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()”)

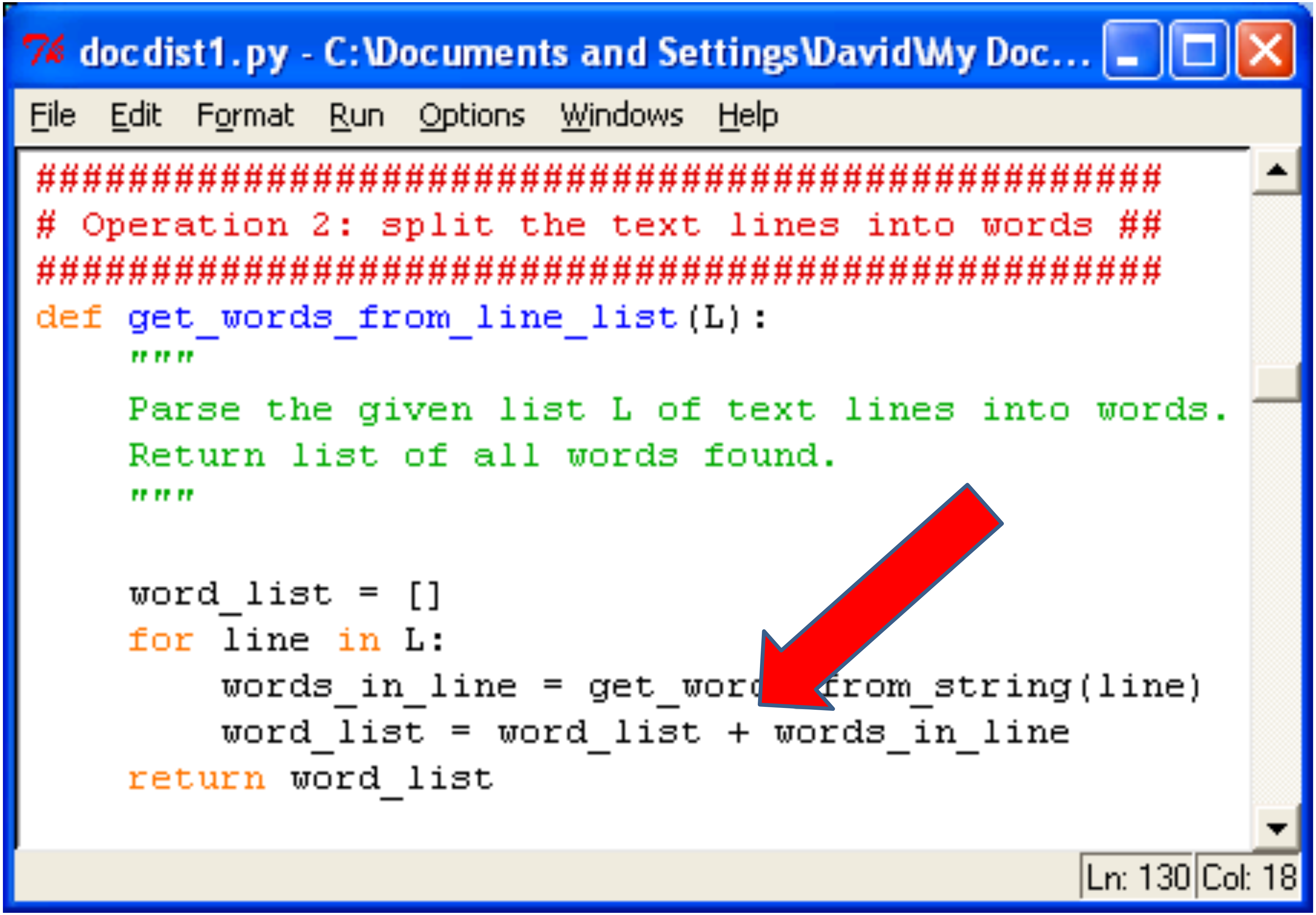


```
t1.verne.txt          t4.arabian.txt        t7.tenmillion.txt
t2.bobsey.txt         t5.churchill.txt      t8.shakespeare.txt
t3.lewis.txt          t6.onemillion.txt     t9.bacon.txt
auk:l01> python source/docdist2.py data/t2.bobsey.txt data/t3.lewis.txt
File data/t2.bobsey.txt : 6667 lines, 49785 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[illegible]   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          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:l01> █
```

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



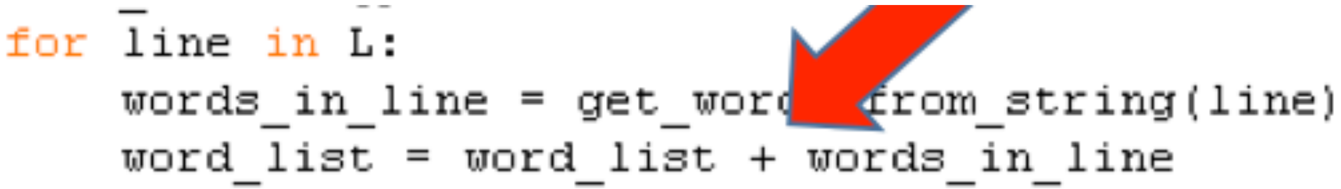
```
##################################################
# Operation 2: split the text lines into words ##
##################################################
def get_words_from_line_list(L):
    """
    Parse the given list L of text lines into words.
    Return list of all words found.
    """

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

+ is slow, need to improve it

- Look at word_list = word_list + words_in_line
- + : concatenation of arrays

- running time = $T(k)$
  - k = length (word_list) + length(words_in_line)

- Total $T(n)$ is $1+2+\ldots+n = n(n+1)/2 = \Theta(n^2)$



```
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

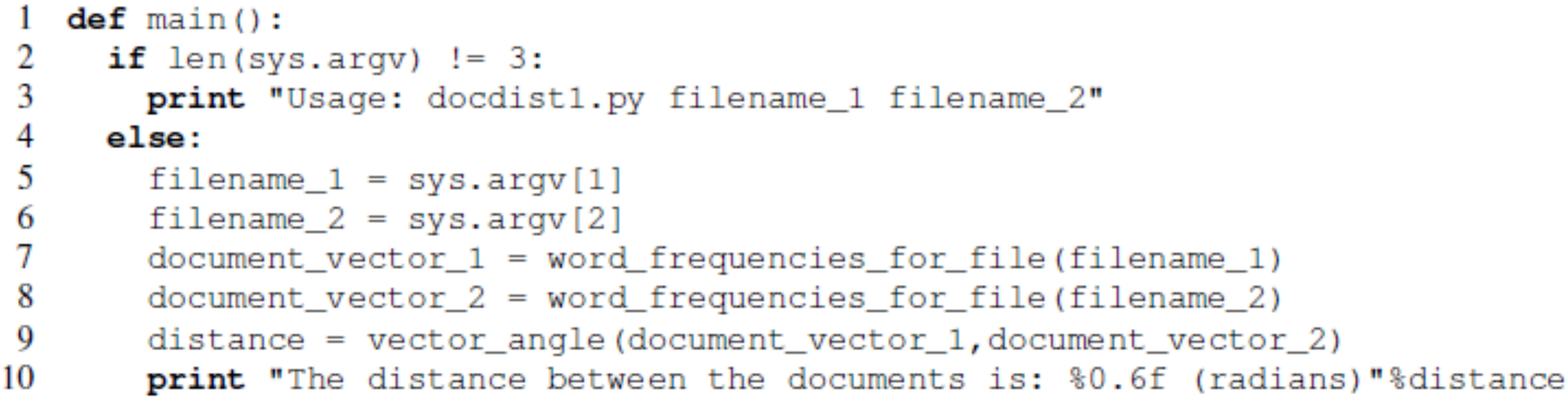
- Docdist3.py: sort the data first (2.657s)
- Docdist4.py: Instead of inserting words in list, insert in dictionary: (0.735s)
- Docdist5.py: Process words instead of chars (0.138s)
- Docdist6.py: better sorting algorithm (0.069s)
  - merge sort instead of insertion sort
- Docdist7.py: dictionary instead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

Appendix –
how to optimize the code

docdist<1-8>.py – Iterative improvements

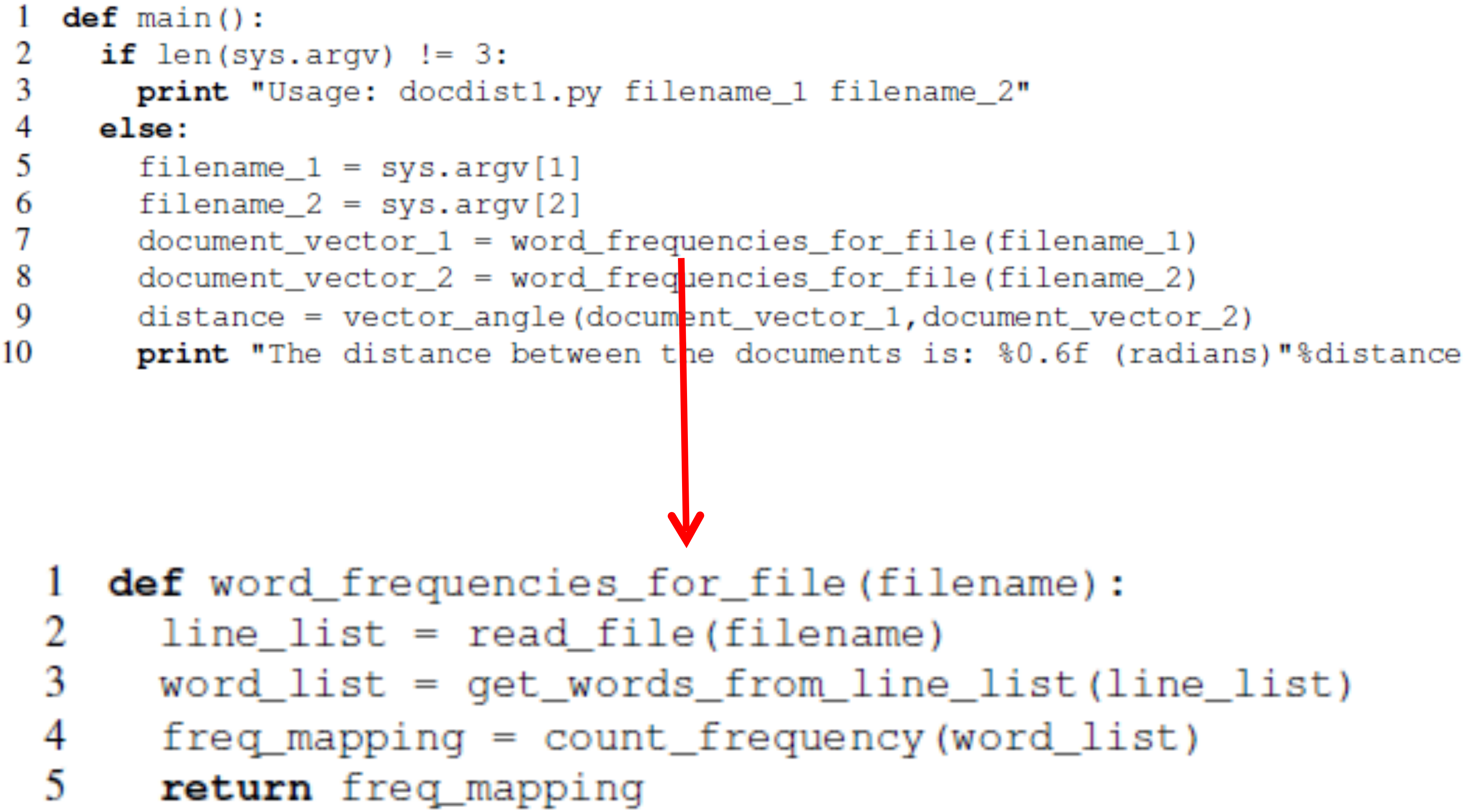
- Objective
  - To arrive at the most efficient implementation in terms of running time
- Approach
  - At each iteration, analyze the running times of various parts of the code
  - Based on the biggest consumer of running time, refactor the code by
    - Introducing faster algorithms
    - Using features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

docdist1 – Code structure



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

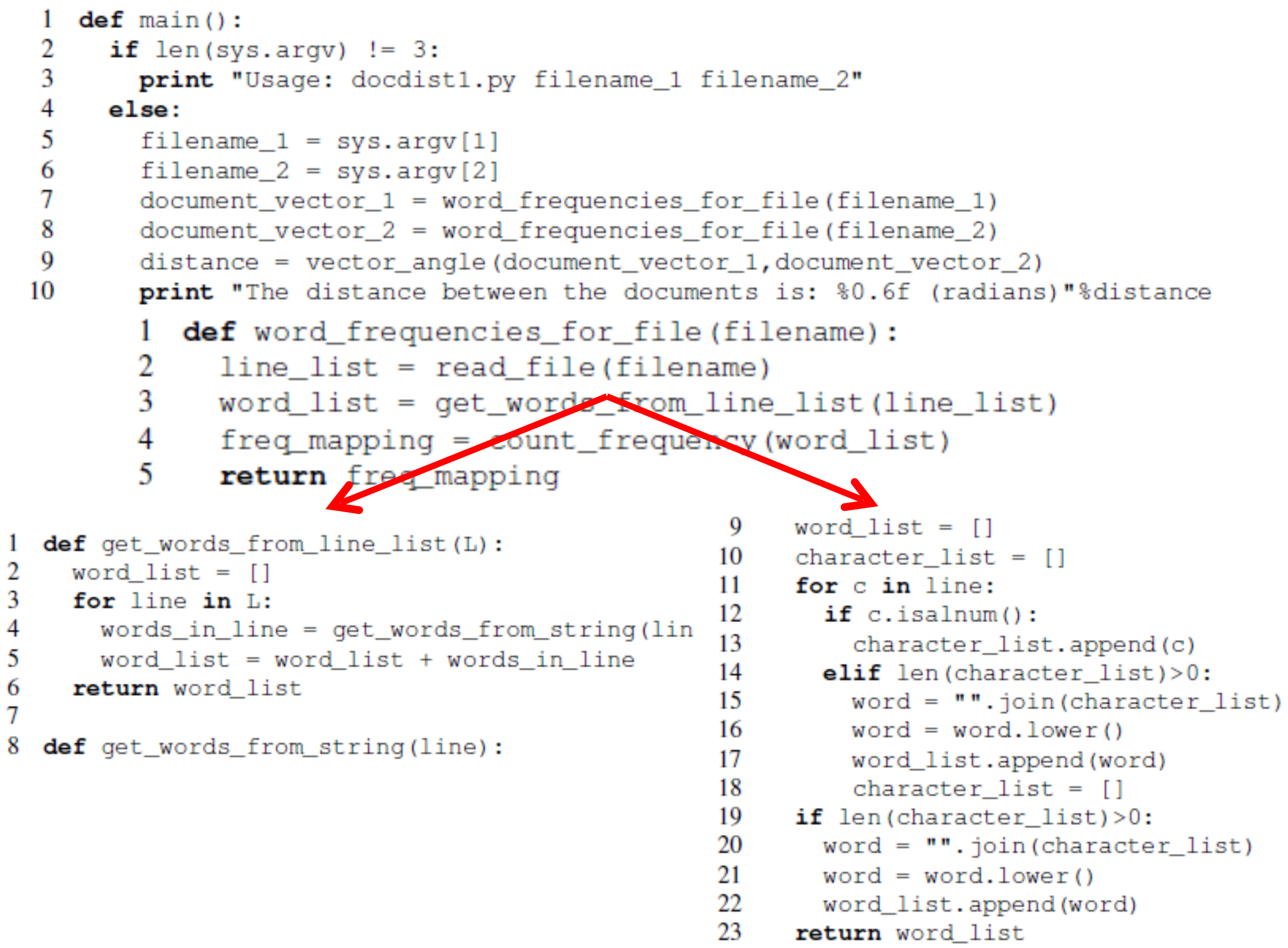
docdist1 – Code structure



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

```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```

docdist1 – Code structure

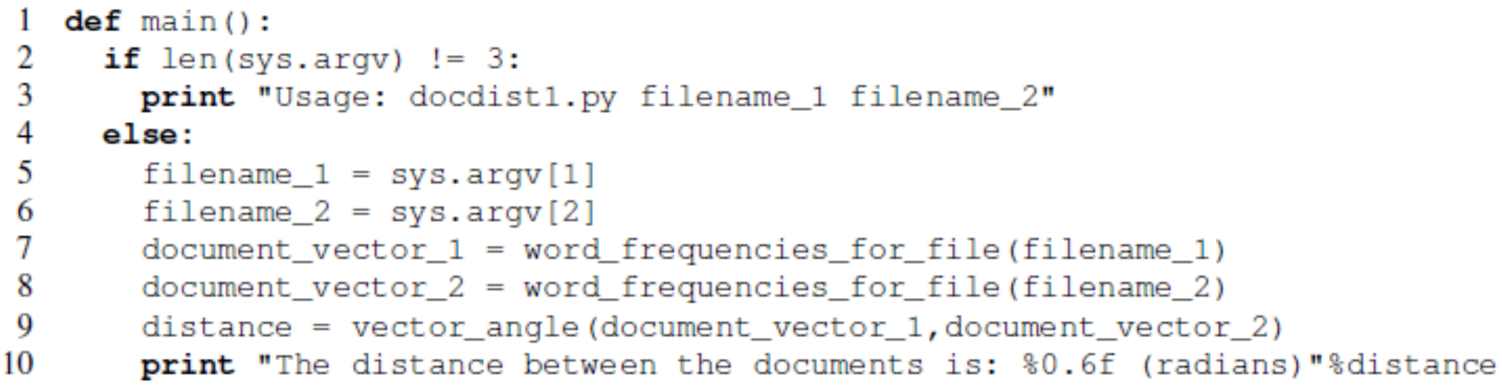


```
def main():
  if len(sys.argv) != 3:
    print "Usage: docdist1.py filename_1 filename_2"
  else:
    filename_1 = sys.argv[1]
    filename_2 = sys.argv[2]
    document_vector_1 = word_frequencies_for_file(filename_1)
    document_vector_2 = word_frequencies_for_file(filename_2)
    distance = vector_angle(document_vector_1,document_vector_2)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

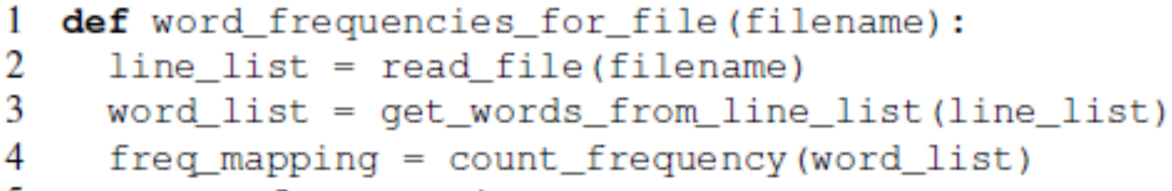
```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```

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

def get_words_from_string(line):
  word_list = []
  character_list = []
  for c in line:
    if c.isalnum():
      character_list.append(c)
    elif len(character_list)>0:
      word = "".join(character_list)
      word = word.lower()
      word_list.append(word)
      character_list = []
  if len(character_list)>0:
    word = "".join(character_list)
    word = word.lower()
    word_list.append(word)
  return word_list
```

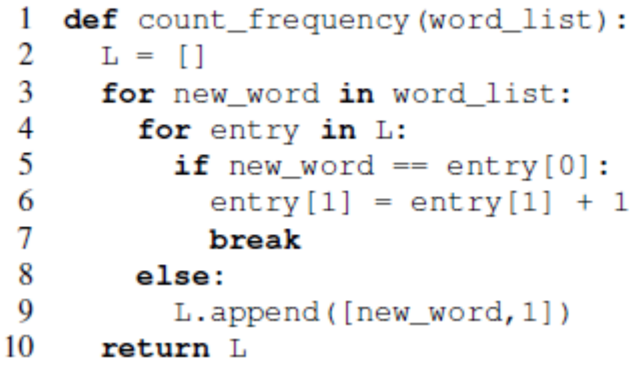
docdist1 – Code structure



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

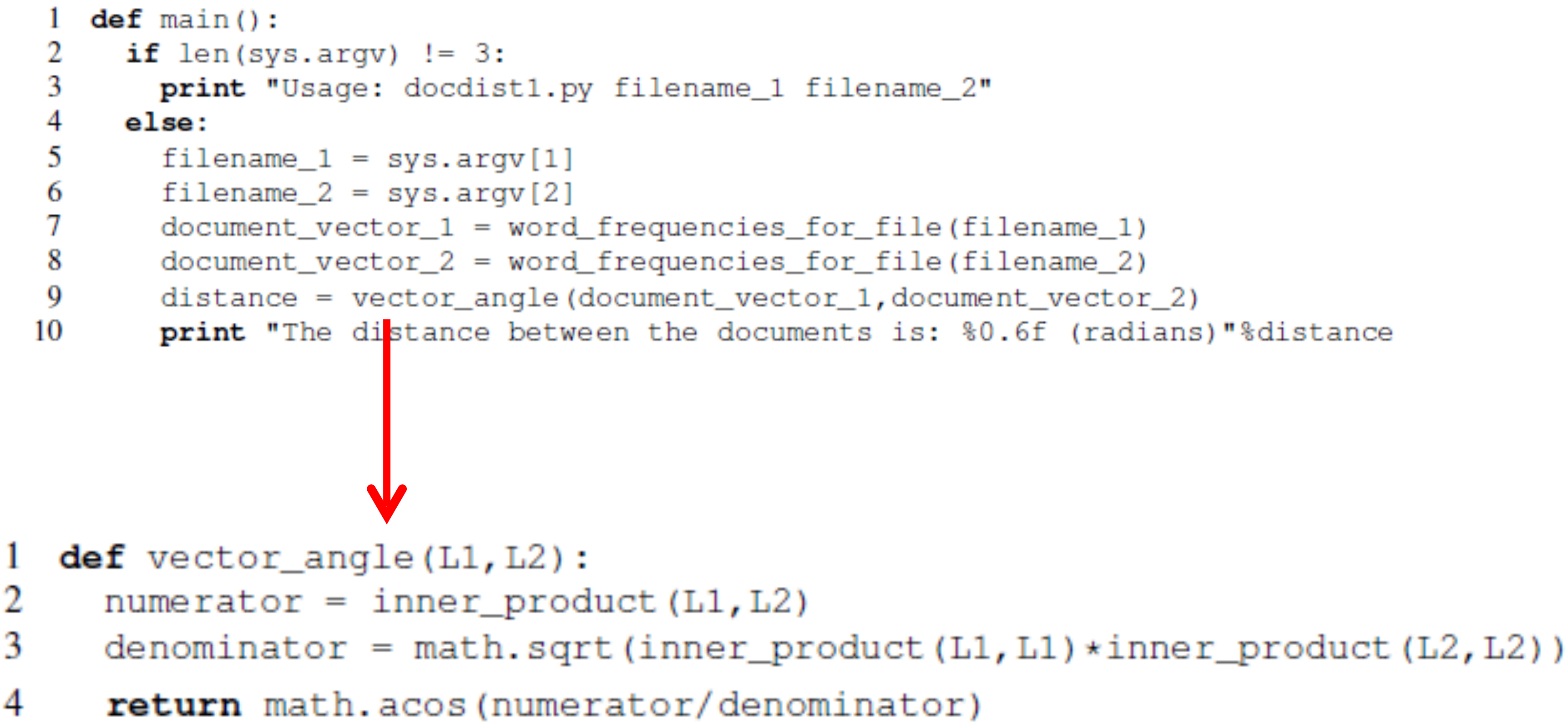


```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```



```
def count_frequency(word_list):
  L = []
  for new_word in word_list:
    for entry in L:
      if new_word == entry[0]:
        entry[1] = entry[1] + 1
        break
    else:
      L.append([new_word,1])
  return L
```

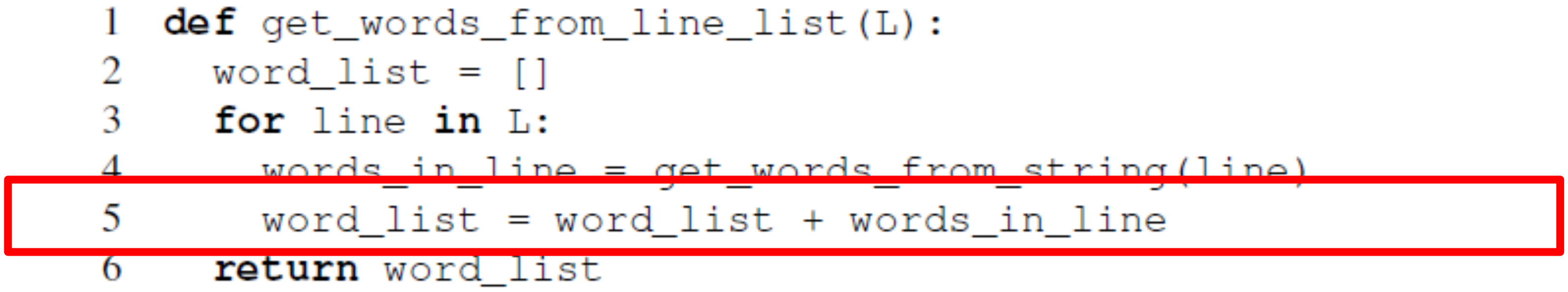
docdist1 – Code structure



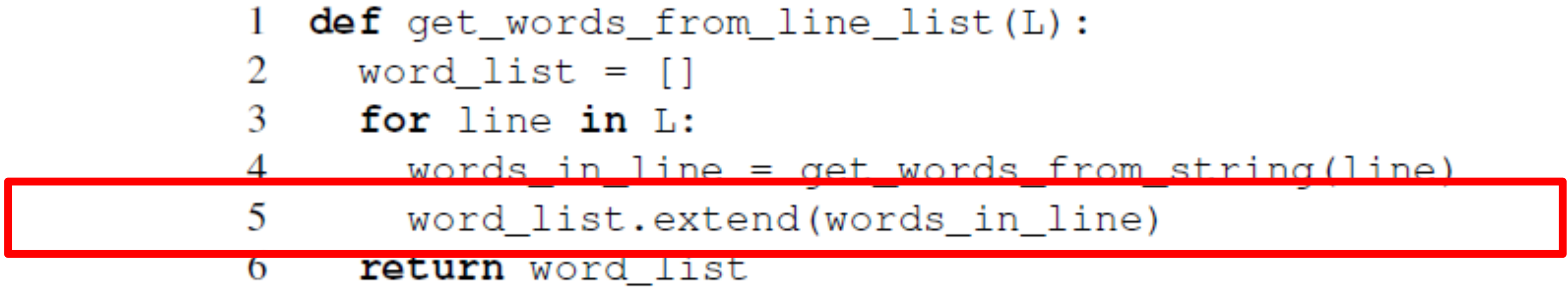
```
def main():
  if len(sys.argv) != 3:
    print "Usage: docdist1.py filename_1 filename_2"
  else:
    filename_1 = sys.argv[1]
    filename_2 = sys.argv[2]
    document_vector_1 = word_frequencies_for_file(filename_1)
    document_vector_2 = word_frequencies_for_file(filename_2)
    distance = vector_angle(document_vector_1,document_vector_2)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

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

docdist1 vs. docdist2

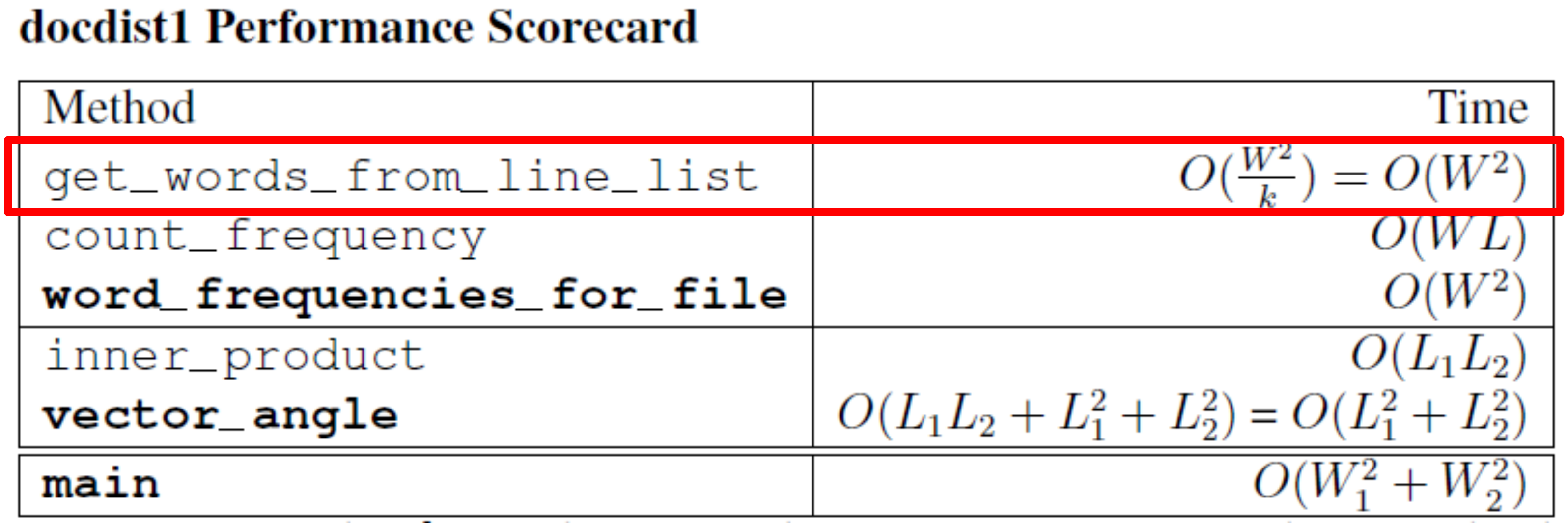


```
def get_words_from_line_list(L):
  word_list = []
  for line in L:
    words_in_line = get_words_from_string(line)
    word_list = word_list + words_in_line
  return word_list
```



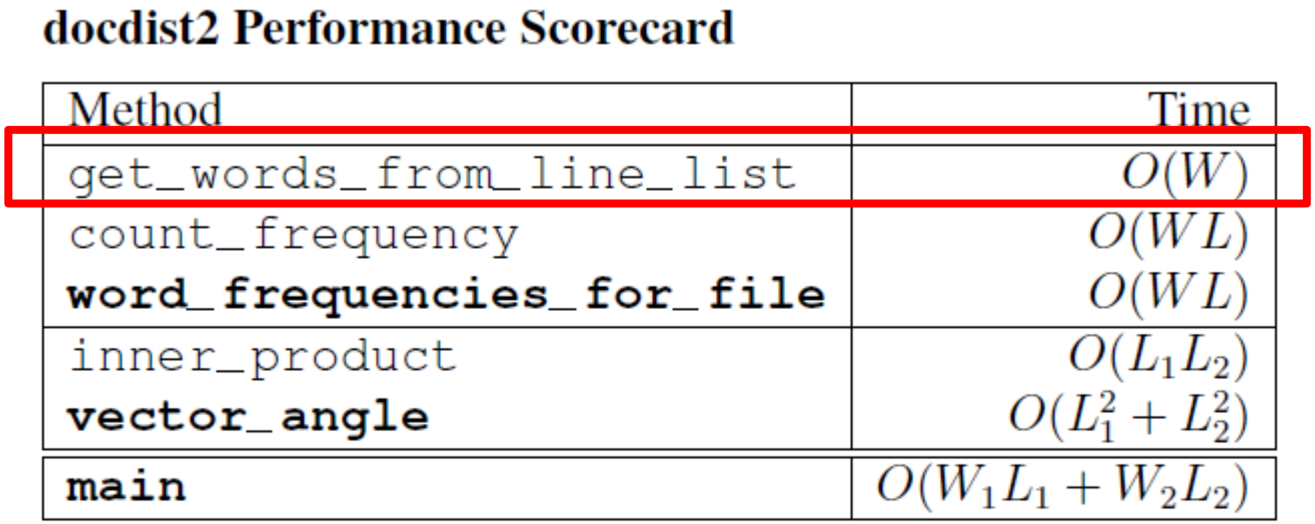
```
def get_words_from_line_list(L):
  word_list = []
  for line in L:
    words_in_line = get_words_from_string(line)
    word_list.extend(words_in_line)
  return word_list
```

docdist1 vs. docdist2



docdist1 Performance Scorecard

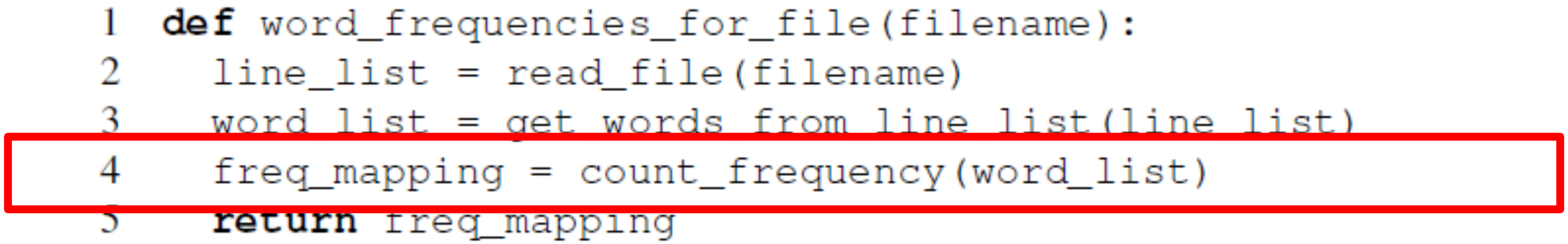
Method	Time
get_words_from_line_list	$O(\frac{W^2}{k}) = O(W^2)$
count_frequency	$O(WL)$
word_frequencies_for_file	$O(W^2)$
inner_product	$O(L_1 L_2)$
vector_angle	$O(L_1 L_2 + L_1^2 + L_2^2) = O(L_1^2 + L_2^2)$
main	$O(W_1^2 + W_2^2)$



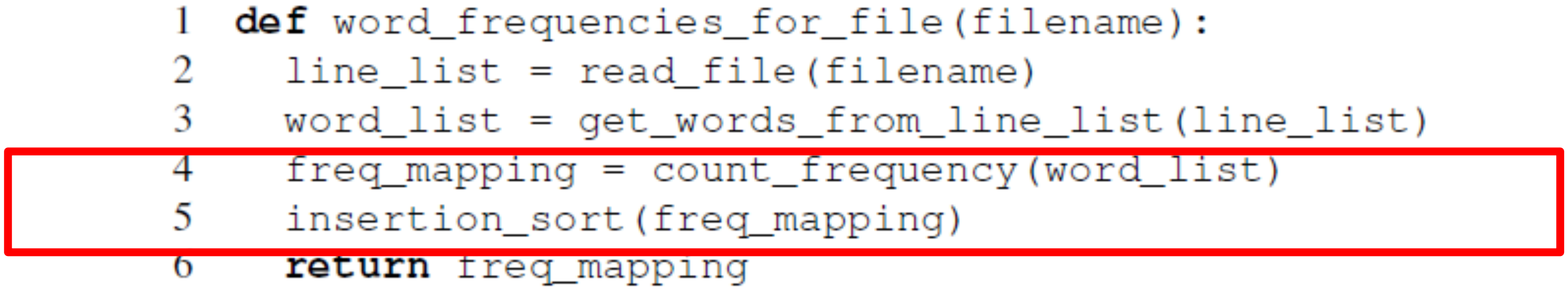
docdist2 Performance Scorecard

Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
word_frequencies_for_file	$O(WL)$
inner_product	$O(L_1 L_2)$
vector_angle	$O(L_1^2 + L_2^2)$
main	$O(W_1 L_1 + W_2 L_2)$

docdist2 vs. docdist3

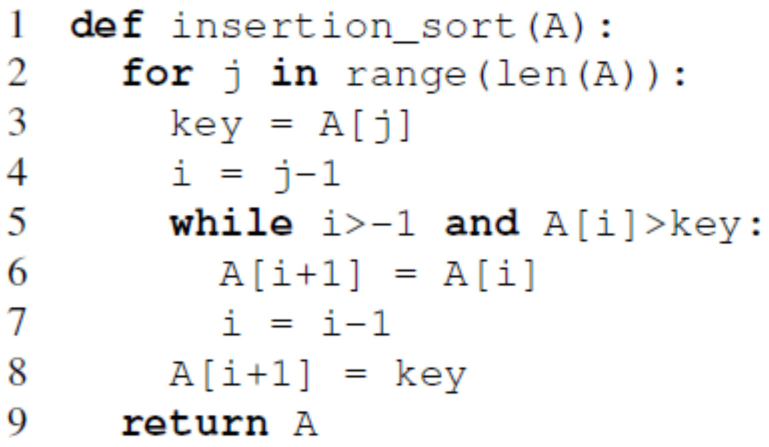


```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```



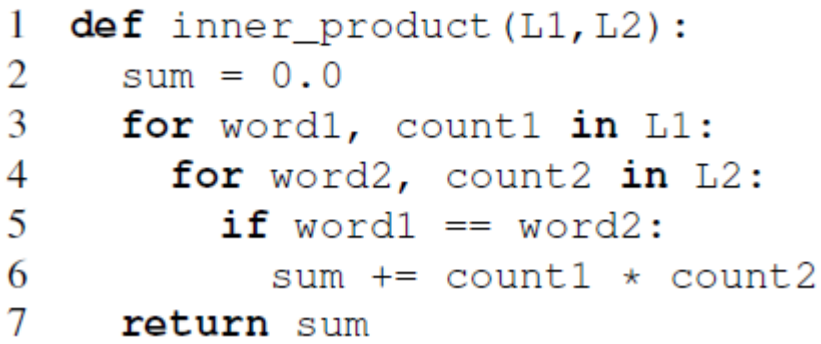
```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  insertion_sort(freq_mapping)
  return freq_mapping
```

docdist3 – Implementing insertion_sort()

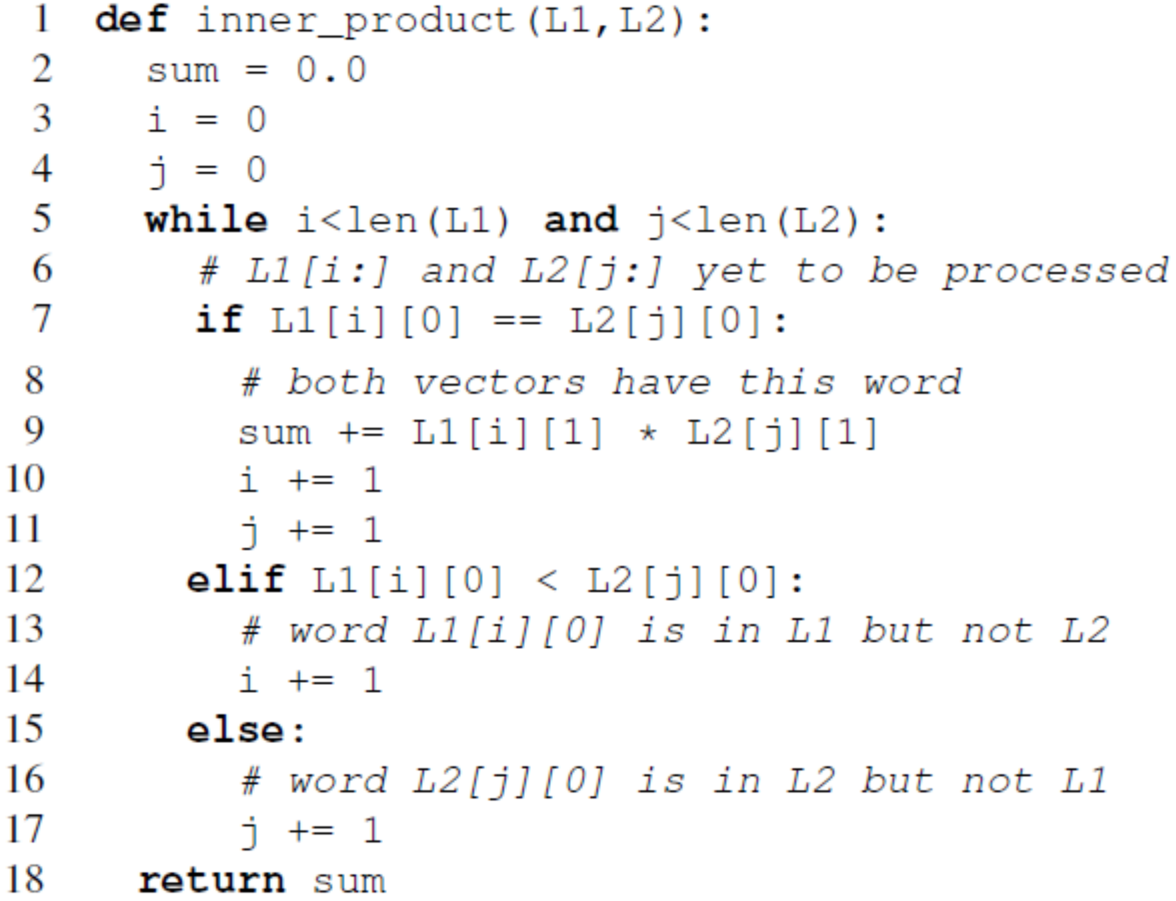


```
def insertion_sort(A):
  for j in range(len(A)):
    key = A[j]
    i = j-1
    while i>-1 and A[i]>key:
      A[i+1] = A[i]
      i = i-1
    A[i+1] = key
  return A
```

docdist3 - Reimplementing inner_product()

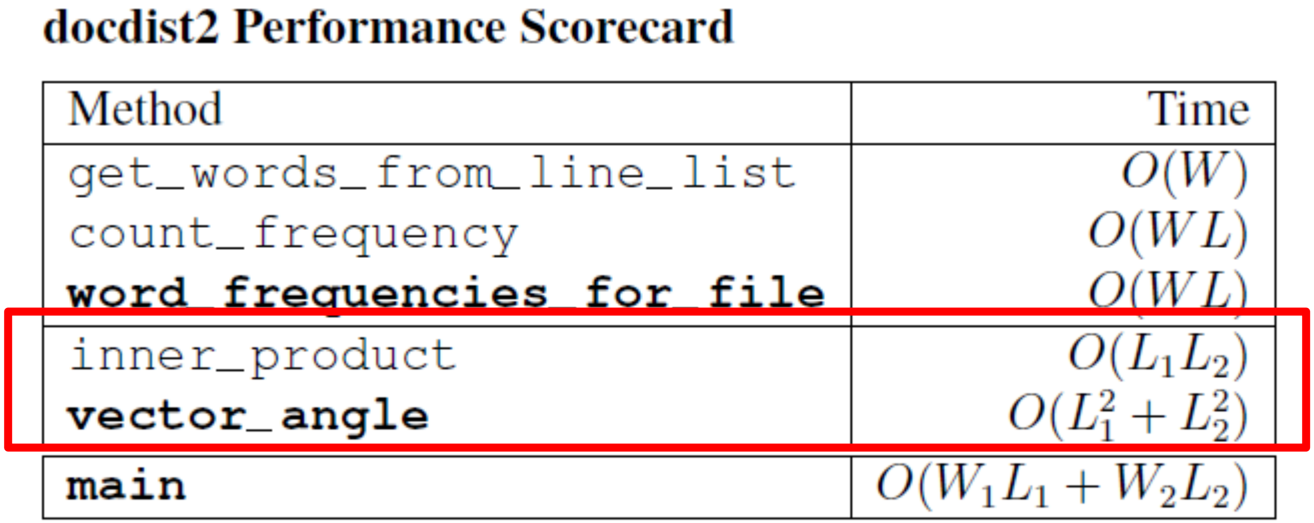


```
def inner_product(L1,L2):
  sum = 0.0
  for word1, count1 in L1:
    for word2, count2 in L2:
      if word1 == word2:
        sum += count1 * count2
  return sum
```



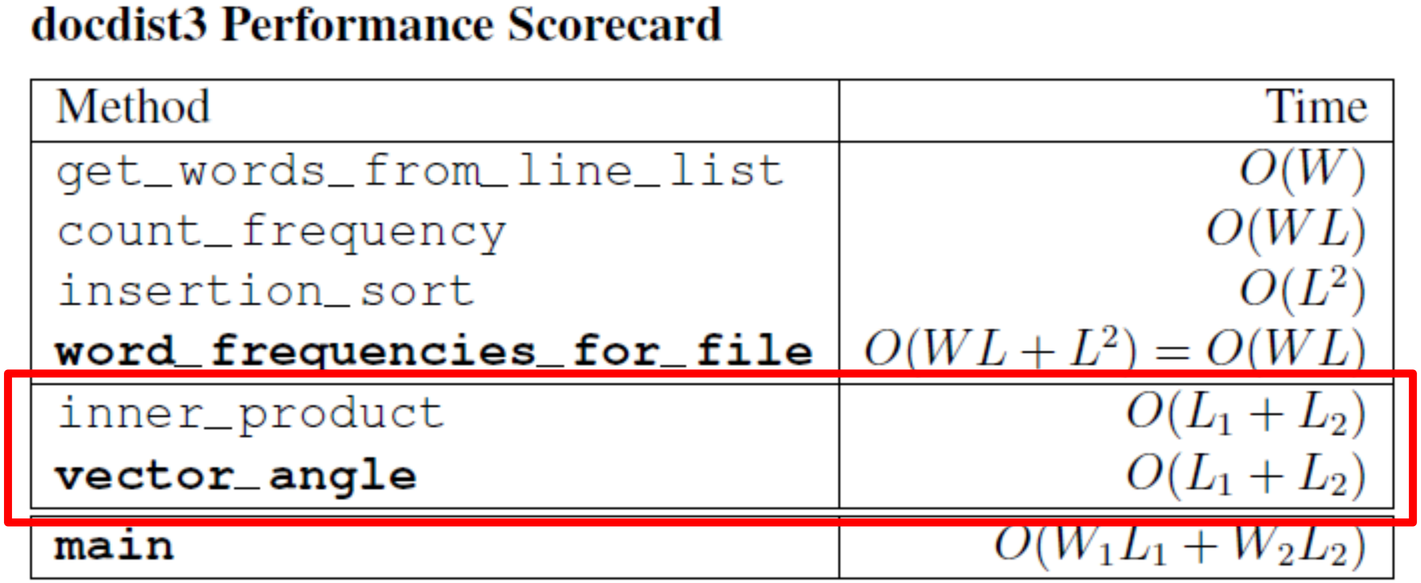
```
def inner_product(L1,L2):
  sum = 0.0
  i = 0
  j = 0
  while i<len(L1) and j<len(L2):
    # L1[i:] and L2[j:] yet to be processed
    if L1[i][0] == L2[j][0]:
      # both vectors have this word
      sum += L1[i][1] * L2[j][1]
      i += 1
      j += 1
    elif L1[i][0] < L2[j][0]:
      # word L1[i][0] is in L1 but not L2
      i += 1
    else:
      # word L2[j][0] is in L2 but not L1
      j += 1
  return sum
```

docdist2 vs. docdist3



docdist2 Performance Scorecard

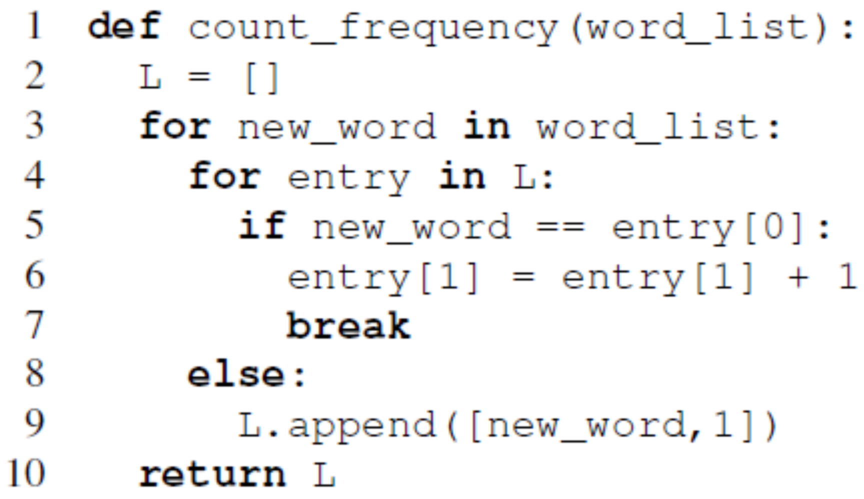
Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
word_frequencies_for_file	$O(WL)$
inner_product	$O(L_1 L_2)$
vector_angle	$O(L_1^2 + L_2^2)$
main	$O(W_1 L_1 + W_2 L_2)$



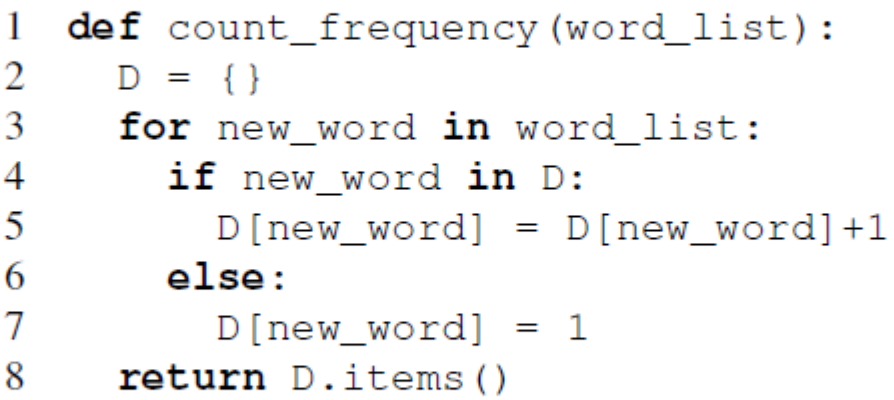
docdist3 Performance Scorecard

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

docdist4 – Refining count_frequency()

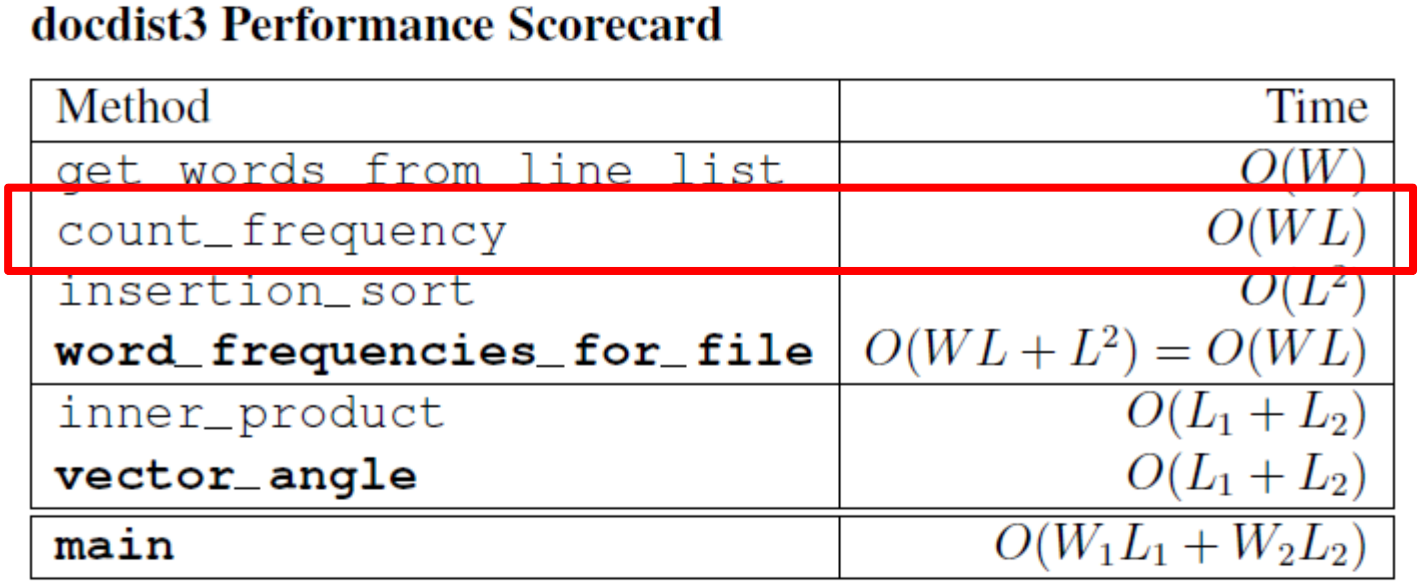


```
def count_frequency(word_list):
  L = []
  for new_word in word_list:
    for entry in L:
      if new_word == entry[0]:
        entry[1] = entry[1] + 1
        break
    else:
      L.append([new_word,1])
  return L
```



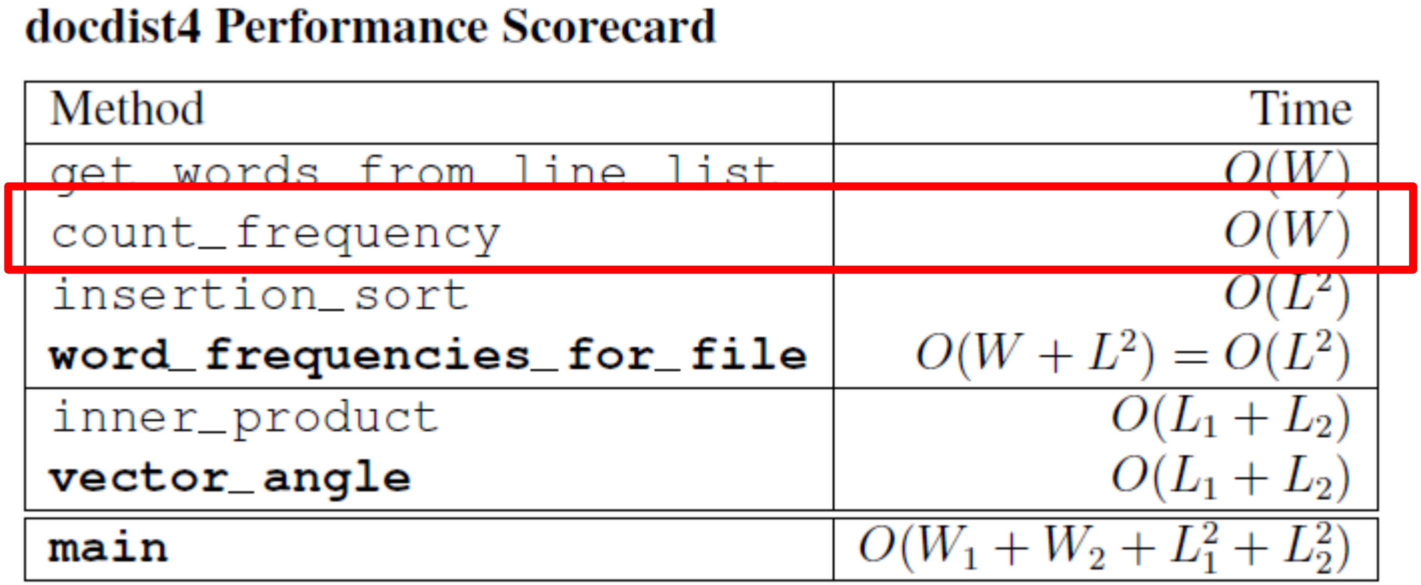
```
def count_frequency(word_list):
  D = {}
  for new_word in word_list:
    if new_word in D:
      D[new_word] = D[new_word]+1
    else:
      D[new_word] = 1
  return D.items()
```

docdist3 vs. docdist4



docdist3 Performance Scorecard

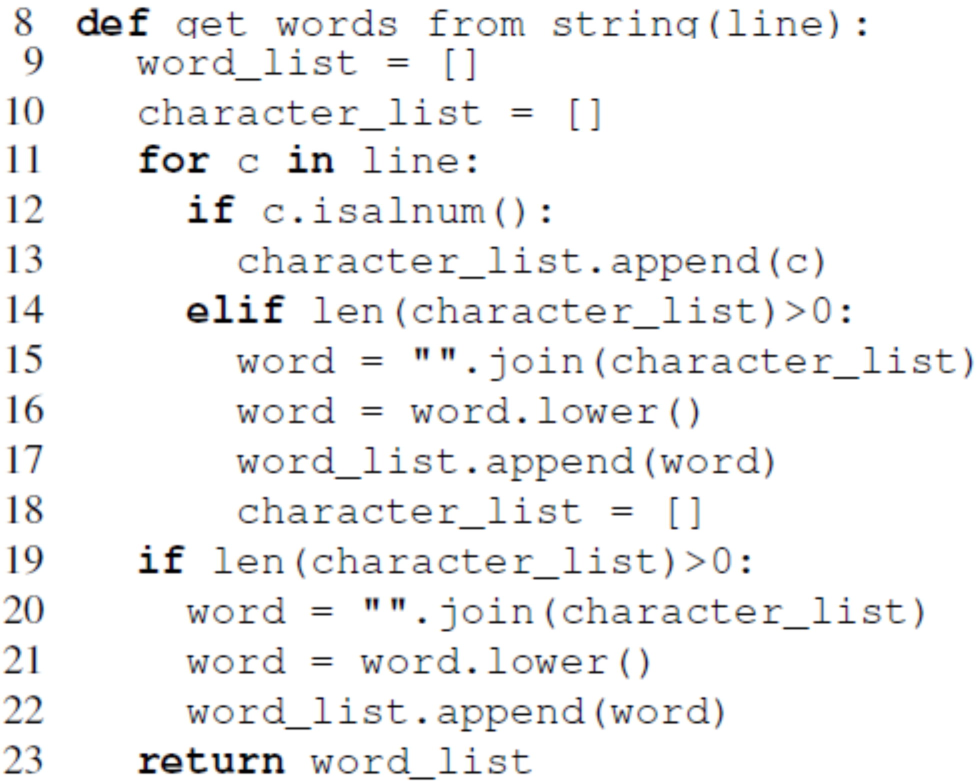
Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(WL)$
insertion_sort	$O(L^2)$
word_frequencies_for_file	$O(WL+L^2)=O(WL)$
inner_product	$O(L_1+L_2)$
vector_angle	$O(L_1+L_2)$
main	$O(W_1L_1+W_2L_2)$



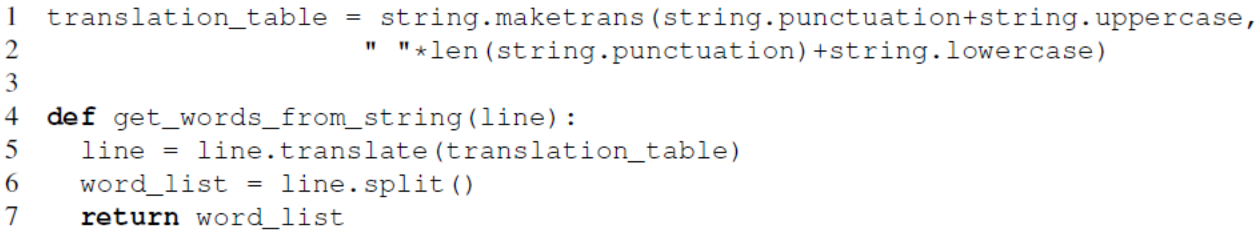
docdist4 Performance Scorecard

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

docdist5 – Simplifying get_words_from_string()

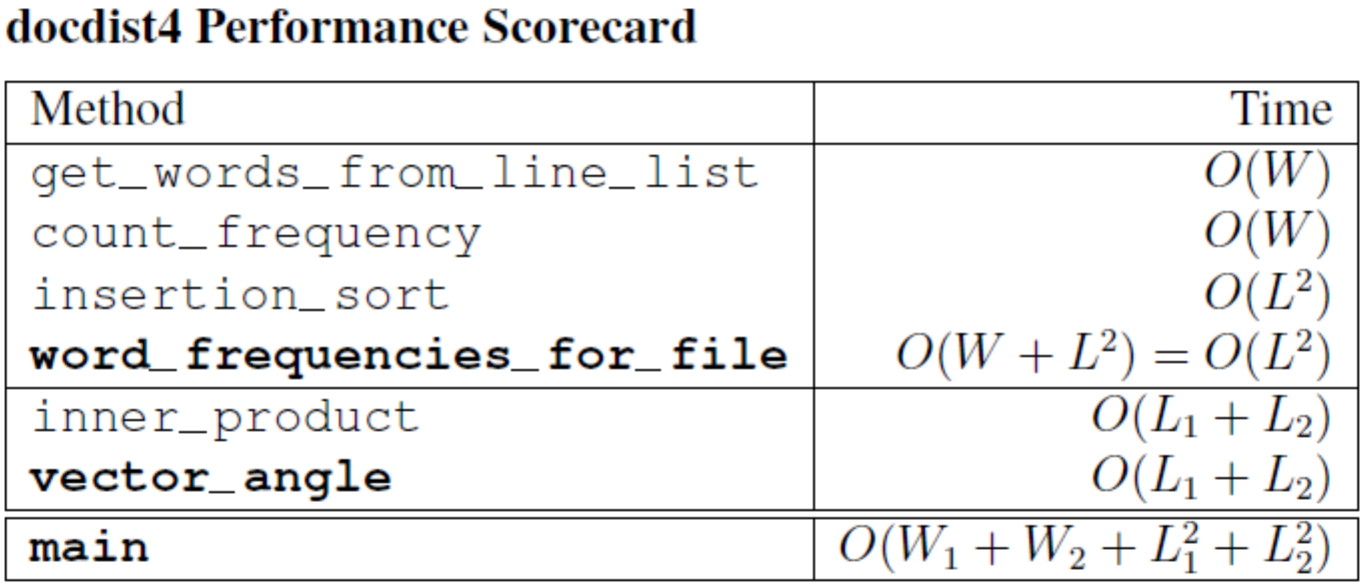


```
def get words from string(line):
  word_list = []
  character_list = []
  for c in line:
    if c.isalnum():
      character_list.append(c)
    elif len(character_list)>0:
      word = "".join(character_list)
      word = word.lower()
      word_list.append(word)
      character_list = []
  if len(character_list)>0:
    word = "".join(character_list)
    word = word.lower()
    word_list.append(word)
  return word_list
```



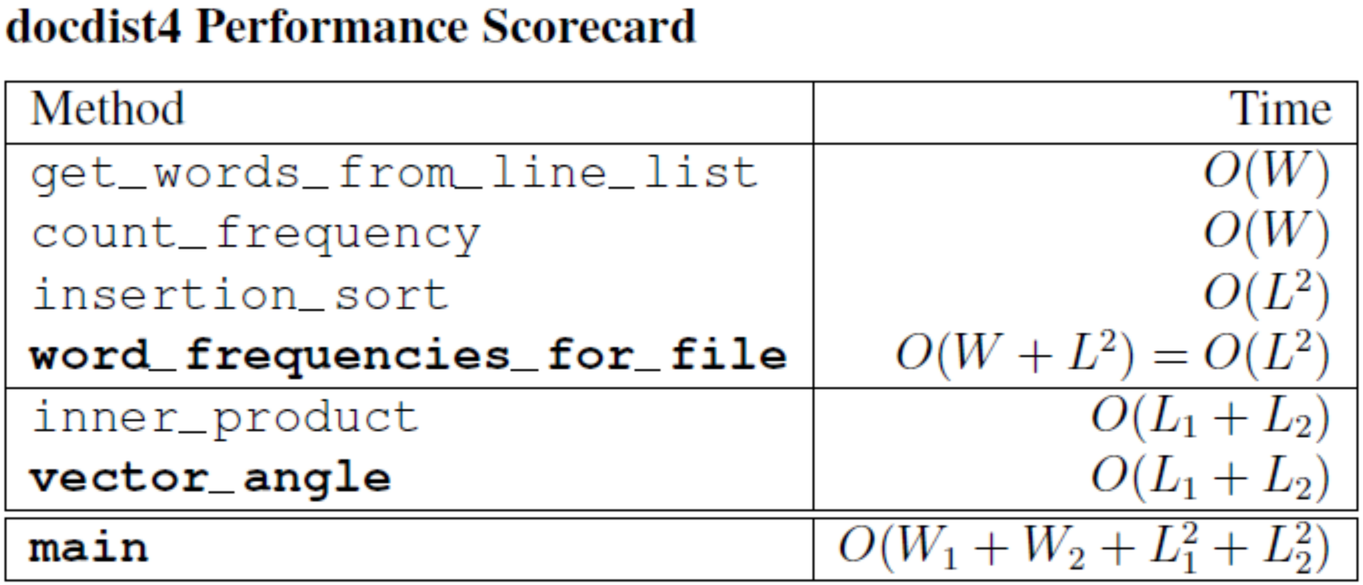
```
translation_table = string.maketrans(string.punctuation+string.uppercase,
                   " "*len(string.punctuation)+string.lowercase)

def get_words_from_string(line):
  line = line.translate(translation_table)
  word_list = line.split()
  return word_list
```

docdist4 vs. docdist5 (same as docdist4)



docdist4 Performance Scorecard

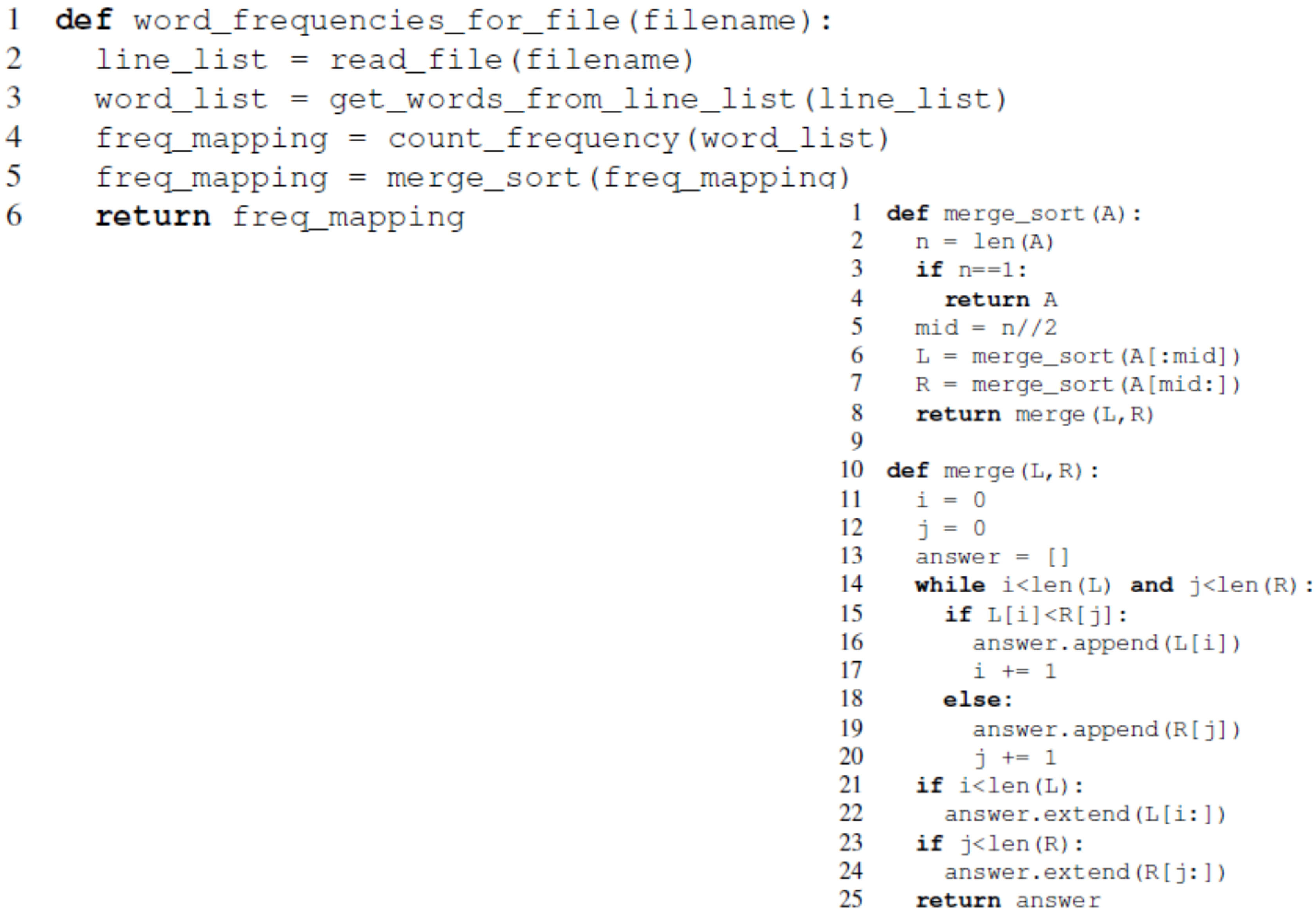
Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
insertion_sort	$O(L^2)$
word_frequencies_for_file	$O(W + L^2) = O(L^2)$
inner_product	$O(L_1 + L_2)$
vector_angle	$O(L_1 + L_2)$
main	$O(W_1 + W_2 + L_1^2 + L_2^2)$



docdist4 Performance Scorecard

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

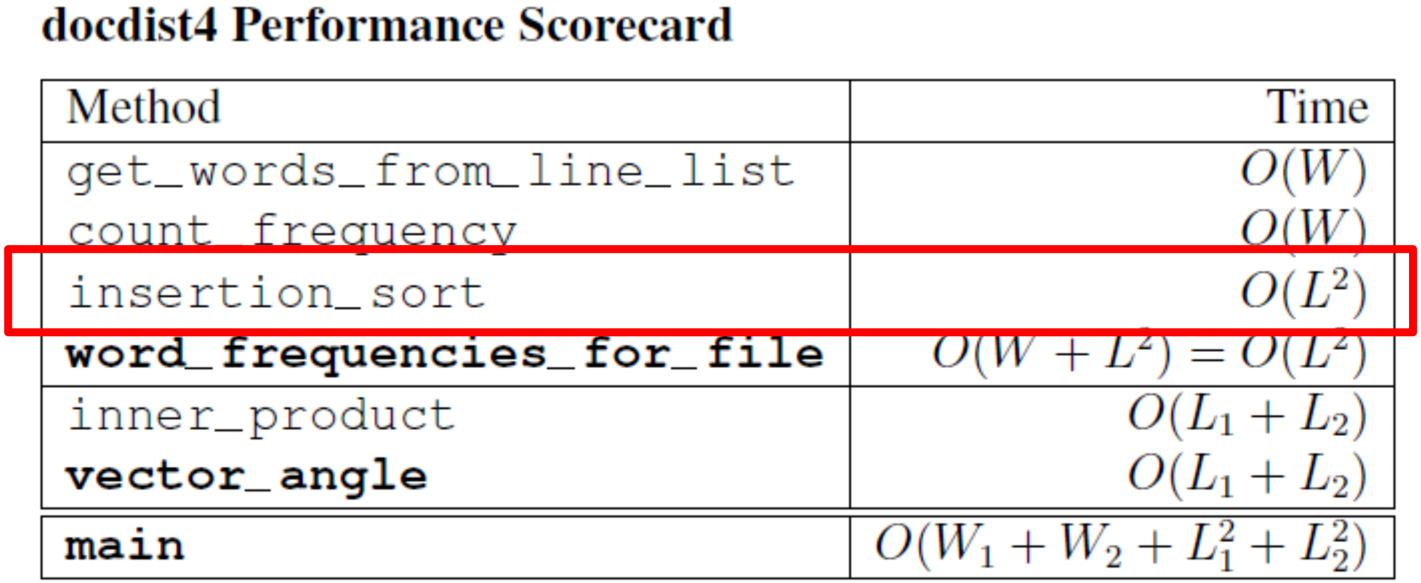
docdist6 – Replacing insertion sort by merge sort



```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  freq_mapping = merge_sort(freq_mapping)
  return freq_mapping
```

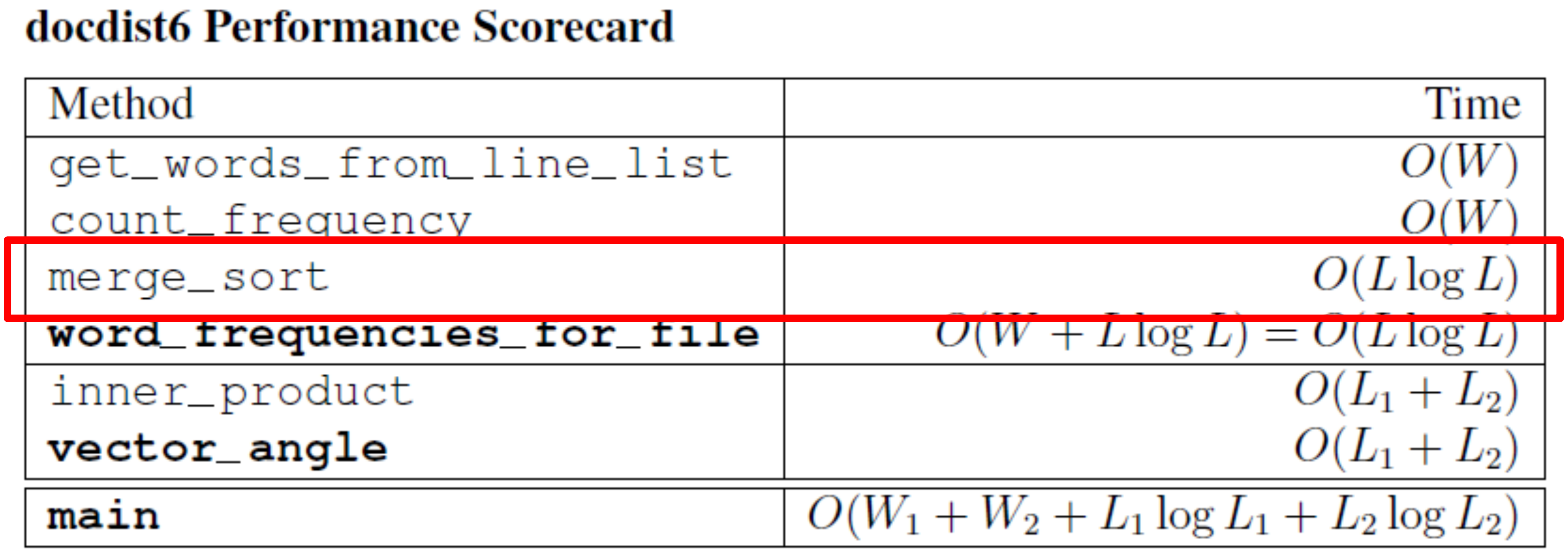
```
def merge_sort(A):
  n = len(A)
  if n==1:
    return A
  mid = n//2
  L = merge_sort(A[:mid])
  R = merge_sort(A[mid:])
  return merge(L,R)

def merge(L,R):
  i = 0
  j = 0
  answer = []
  while i<len(L) and j<len(R):
    if L[i]<R[j]:
      answer.append(L[i])
      i += 1
    else:
      answer.append(R[j])
      j += 1
  if i<len(L):
    answer.extend(L[i:])
  if j<len(R):
    answer.extend(R[j:])
  return answer
```

docdist5 vs. docdist6



docdist4 Performance Scorecard

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



docdist6 Performance Scorecard

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

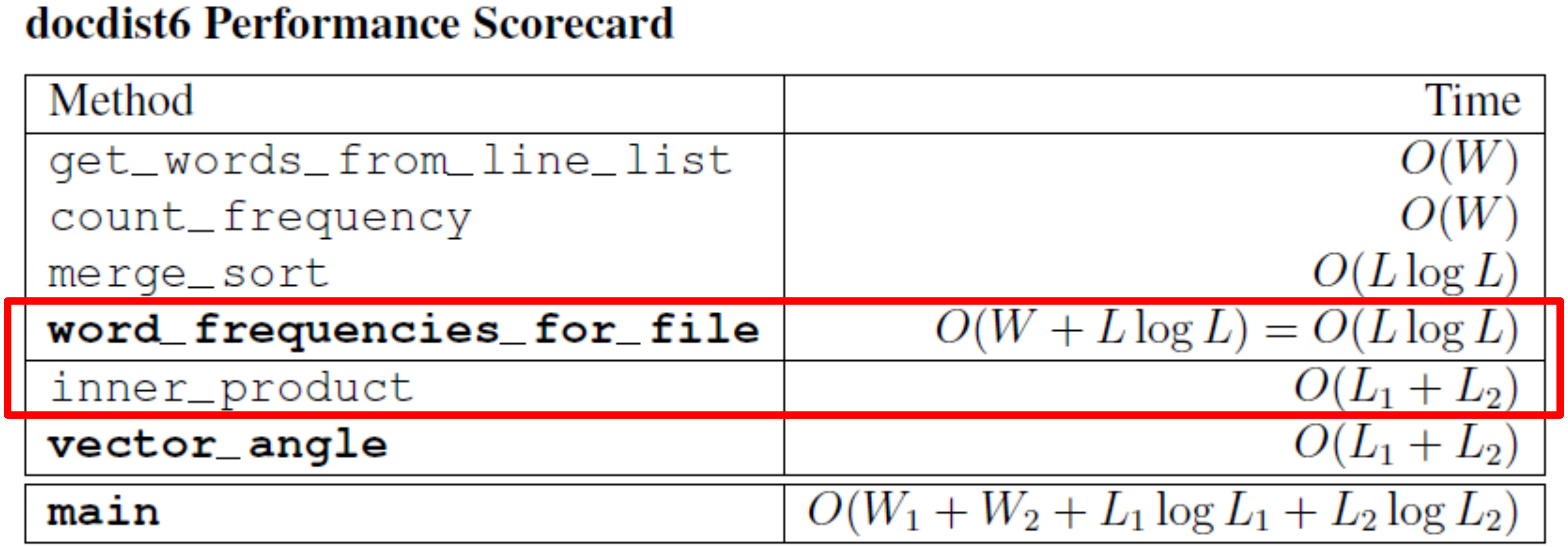
docdist7 – From sorted list to document vectors

```
def count_frequency(word_list):
  D = {}
  for new_word in word_list:
    if new_word in D:
      D[new_word] = D[new_word]+1
    else:
      D[new_word] = 1
  return D
```

```
def word_frequencies_for_file(filename):
  line_list = read_file(filename)
  word_list = get_words_from_line_list(line_list)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```

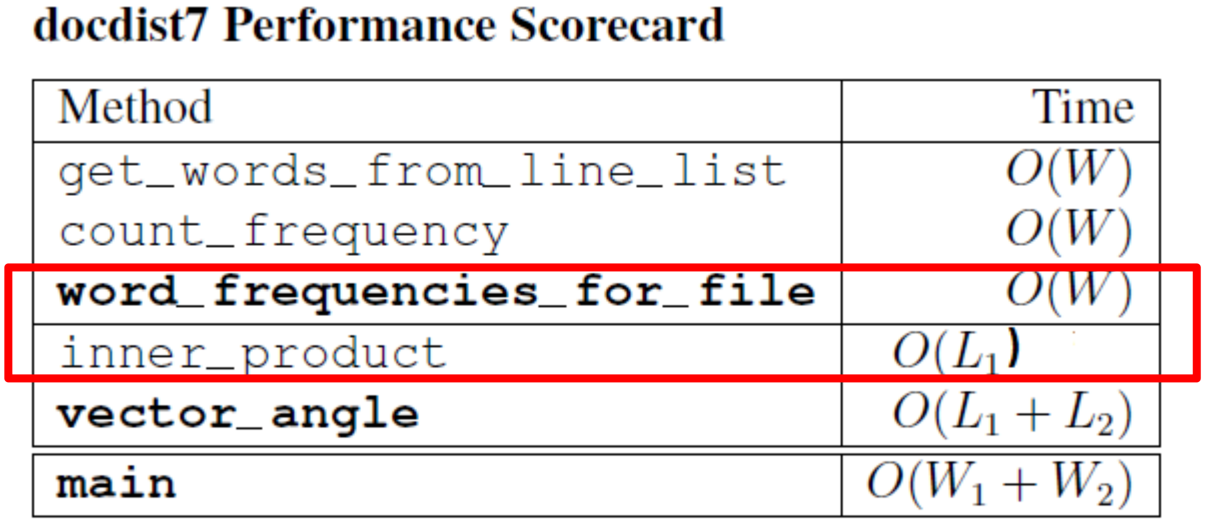
```
def inner_product(D1,D2):
  sum = 0.0
  for key in D1:
    if key in D2:
      sum += D1[key] * D2[key]
  return sum
```

docdist6 vs. docdist7



docdist6 Performance Scorecard

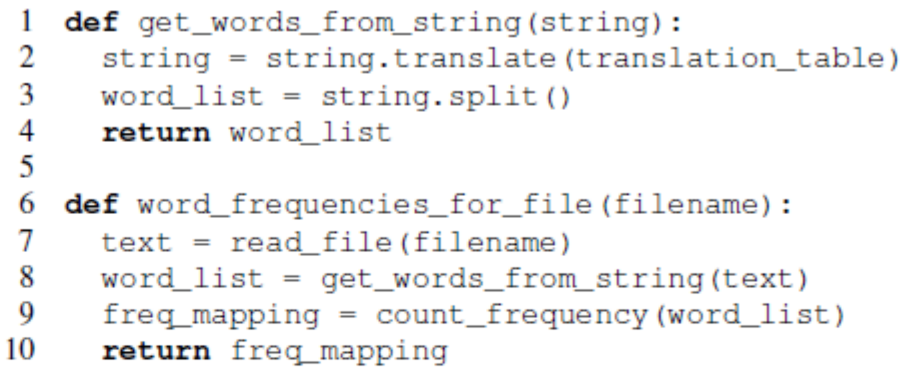
Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
merge_sort	$O(L \log L)$
word_frequencies_for_file	$O(W + L \log L) = O(L \log L)$
inner_product	$O(L_1 + L_2)$
vector_angle	$O(L_1 + L_2)$
main	$O(W_1 + W_2 + L_1 \log L_1 + L_2 \log L_2)$



docdist7 Performance Scorecard

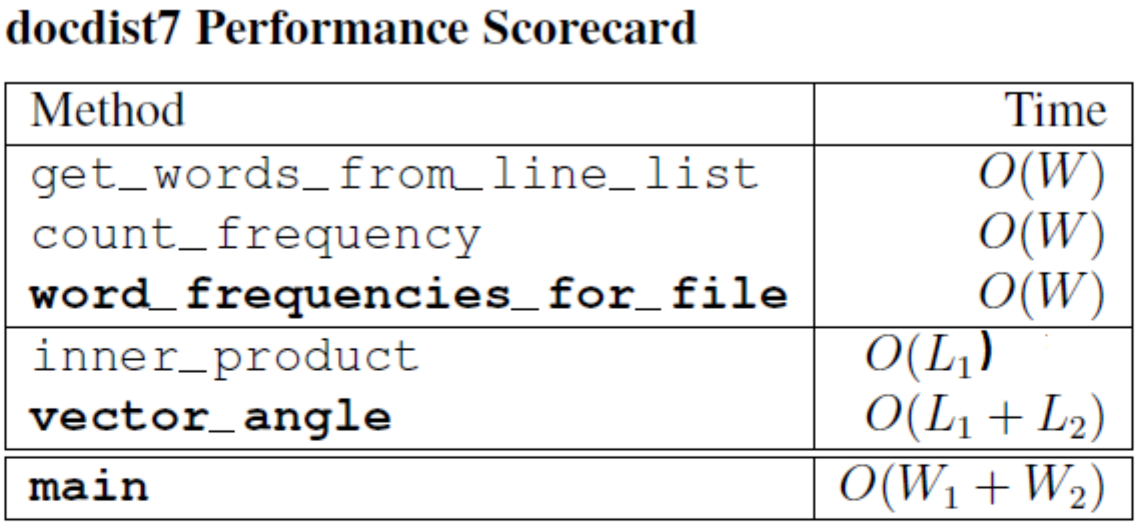
Method	Time
get_words_from_line_list	$O(W)$
count_frequency	$O(W)$
word_frequencies_for_file	$O(W)$
inner_product	$O(L_1)$
vector_angle	$O(L_1 + L_2)$
main	$O(W_1 + W_2)$

docdist8 – Simplifying the code



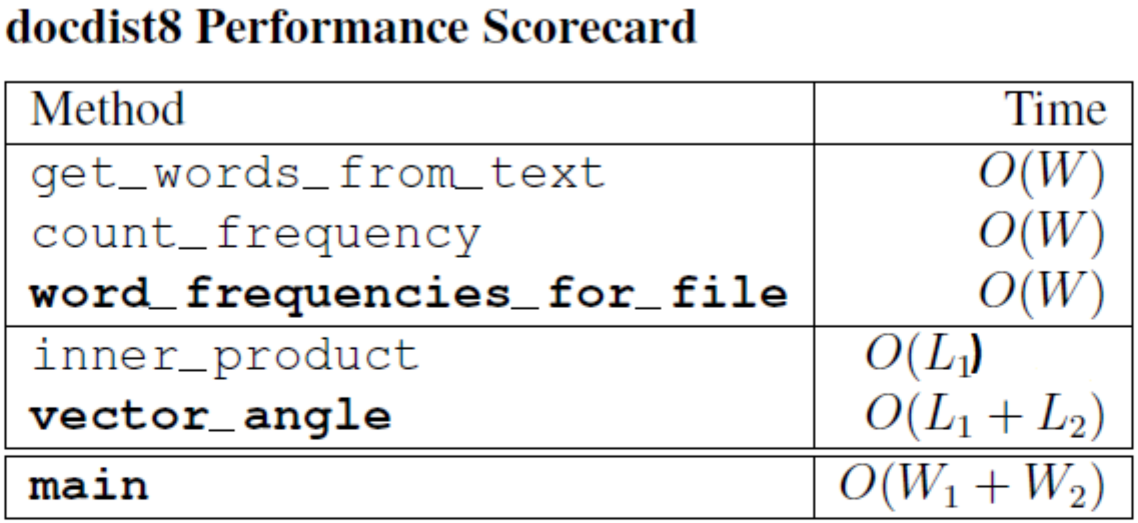
```
def get_words_from_string(string):
  string = string.translate(translation_table)
  word_list = string.split()
  return word_list

def word_frequencies_for_file(filename):
  text = read_file(filename)
  word_list = get_words_from_string(text)
  freq_mapping = count_frequency(word_list)
  return freq_mapping
```

docdist7 vs. docdist8



docdist7 Performance Scorecard

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



docdist8 Performance Scorecard

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