```python
docdist1

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]
        sorted_word_list_1 = word_frequencies_for_file(filename_1)
        sorted_word_list_2 = word_frequencies_for_file(filename_2)
        distance = vector_angle(sorted_word_list_1, sorted_word_list_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(line)
        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
```

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 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)

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

docdist2

if __name__ == "__main__":
    import cProfile
    cProfile.run("main()")

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

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)
    insertion_sort(freq_mapping)
    return freq_mapping
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

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

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()
```python
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

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
```

The code above defines functions for processing text data. It includes a `get_words_from_string` function that takes a line of text, translates it to lowercase, and splits it into words. The `word_frequencies_for_file` function reads a file, processes each line, and returns a frequency mapping of the words. The `merge_sort` function implements a merge sort algorithm, and `merge` implements the merge operation used in merge sort.
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

def get_words_from_text(text):
    text = text.translate(translation_table)
    word_list = text.split()
    return word_list

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