1.

**import** sys  
**import** time  
**'''  
Project created on Oct 21, 2017  
 created by Yifu Liu  
'''  
'''The program should perform better than nklogn + nklogk'''  
  
  
class** Main():  
  
 **def** \_\_init\_\_(self, file):  
 self.file = file  
  
 *#tuples of a list, for every word in original document make a word into a tuple,  
 #which includes a sorted\_word and a original word. e.g.word ---> (dorw, word)* **def** make\_tuple(self, file, list):  
 num\_of\_words = 0 *#How many words in the original file* sort\_time = time.time()  
 **for** word **in** open(sys.argv[int(file\_index)], **'r'**): *#Go through the list, O(n)* word = word.strip(**'\n'**) *#get rid of \n at the end of each word, O(1)* sorted\_word = SortWord(word).quick\_sort(word) *#sort each word using quick\_sort()  
 #sorted\_word = Mergesort(word).merge\_sort(word) ###Mergesort() is slower than Quick\_sort() in this case###  
 ###Mergesort() uses 3.6s, but Quick\_sort() uses 1.5s###  
 #So, the Quick\_sort() should be O(klog(k)) in this case* list.append((sorted\_word, word)) *#append tuples, each tuple contains (index of the original table, sorted\_word, word), O(1)* num\_of\_words += 1 *#O(1)* print(**'The number of words:'**, num\_of\_words)  
 **return** self.sort(list)  
  
 *#When test\_word has the same word with sorted word in next tuple, then append original word to list  
 #if not, replace the next\_word with sorted word in next\_tuple, then move the original word into the list* **def** sort(self, list):  
 sort = Mergesort(list)  
 sorted\_word = sort.merge\_sort(list) *#Using Merge\_sort to sort the whole list, O(nklog(n))  
 #sorted\_word = sorted(list, key = itemgetter(1)) #built-in sort method* test\_word = sorted\_word[0][0] *#Take the sorted\_word from first tuple* final\_list = [[]] *#the list\_of\_list will be returned* index = 0  
 **for** elem **in** sorted\_word: *#O(n)* **if** test\_word != elem[0]: *#different word with test\_word* test\_word = elem[0] *#replace the test\_word* final\_list.append([elem[1]])  
 index += 1  
 **else**:  
 final\_list[index].append(elem[1]) *#same sorted\_word but different original word* **return** final\_list *#The final output is list\_of\_list. e.g.[['abc', 'cab'], ['igkl'], ['defgh', 'hgfed']]***class** SortWord():  
  
 **def** \_\_init\_\_(self, word):  
 self.word = word  
  
 *#the worst case of quick\_sort is O(n^2), however, it does not really affect the run time of this program,  
 #because the len(words) are not large. Sorting the same amount of word use Mergesort() will spend more time at this point.* **def** quick\_sort(self, word):  
 **if** word == **''**:  
 **return ''** pivot = word[0]  
 l, r = self.quick\_sort\_helper(word, pivot)  
 left = self.quick\_sort(**''**.join(l)) *#starts with None, add all of chars that smaller than the pivot.* right = self.quick\_sort(**''**.join(r)) *#starts with None, add all of chars that larger than the pivot.* **return** left + pivot + right  
  
 **def** quick\_sort\_helper(self, word, pivot):  
 partition\_left = [elem **for** elem **in** word[1::] **if** elem < pivot] *#move all of chars that smaller than pivot in the word to the left.* partition\_right = [elem **for** elem **in** word[1::] **if** elem >= pivot] *#move all of chars in word to the right.* **return** partition\_left, partition\_right  
  
  
**class** Mergesort():  
  
 **def** \_\_init\_\_(self, list):  
 self.list = list  
  
 *#It sorts the whole list of words based on sorted words  
 #e.g. [('dgo', 'dog'), ('abc', 'cba)] ---> [('abc', 'cba), ('dgo', 'dog')]* **def** merge\_sort(self, list): *#which accepts a tuple of list [(sorted\_word, word)]* **if** len(list) < 2: *#when the len(list) < 2, return the list to last recurision* **return** list  
 middle = len(list) // 2 *#get the middle index of list. get lower index when the len(list) is even* left\_list = self.merge\_sort(list[:middle]) *#merge\_sort() the left\_list of a list* right\_list = self.merge\_sort(list[middle:]) *#merge\_sort() the right\_list of a list* **return** self.merge\_sort\_helper(left\_list, right\_list)  
  
 *#Combine two lists into a single list,  
 #First it compares the length of list. If the length is same, then compares the words based on the words themselves  
 #e.g. 'abc' < 'abcd'; 'abc' < 'abd'  
 #append the smaller one into the result[]* **def** merge\_sort\_helper(self, left, right):  
 i, j = 0, 0  
 result = []  
 **while**(len(result) < len(left) + len(right)):  
 **if** i != len(left) **and** j != len(right) **and** len(left[i][0]) < len(right[j][0]):  
 result.append(left[i])  
 i += 1  
 **elif** i != len(left) **and** j != len(right) **and** len(left[i][0]) > len(right[j][0]):  
 result.append(right[j])  
 j += 1  
 **elif** i != len(left) **and** j != len(right) **and** left[i][0] == right[j][0]:  
 result.append(left[i])  
 result.append(right[j])  
 i += 1  
 j += 1  
 **elif** i != len(left) **and** j != len(right) **and** left[i][0] < right[j][0]:  
 result.append(left[i])  
 i += 1  
 **elif** i != len(left) **and** j != len(right) **and** left[i][0] > right[j][0]:  
 result.append(right[j])  
 j += 1  
 **elif** i == len(left) **or** j == len(right):  
 result.extend(left[i:] **or** right[j:])  
 **return** result  
  
  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 sys.argv.append(**"dict1.txt"**)  
 sys.argv.append(**"dict2.txt"**)  
 sys.argv.append(**"dict3.txt"**)  
 list = []  
 file\_index = input(**'Please enter the file name (1/2) that you want to use :'**)  
 main = Main(sys.argv[int(file\_index)])  
 **while True**:  
 number = 0  
 **if** 0 < int(file\_index) < 4:  
 print(**'correct number'**)  
 start = time.time()  
 list\_of\_list = main.make\_tuple(sys.argv[int(file\_index)], list)  
 print(**"time"**, time.time() - start)  
 f = open(**"anagram"** + file\_index+ **".txt"**, **"w+"**)  
 **for** elem **in** list\_of\_list:  
 f.write(**"%s\n"** % elem)  
 **if** len(elem) >= 5:  
 number += 1  
 print(number, **'.'**, elem)  
 f.close()  
 print(len(list\_of\_list))  
  
 print(**"To grader: there is a blank row at the end of each file, I deleted that, so the number should be less 1 than original file"**)  
 **break  
 else**:  
 print(**'incorrect number, please run again'**)  
 sys.exit()

2.

Example: {abc, defgh, igkl, m, n, op, q, rstuvwxyz, cab, hgfed}

Step1: Turn the input file in to list, O(n)

[abc, defgh, igkl, m, n, op, q, rstuvwxyz, cab, hgfed]

Step2: Using quick sort to sort each word and bind them into a tuple of list.O(nklogk)

[('abc', 'abc'), ('defgh', 'defgh'), ('gikl', 'igkl'), ('m', 'm'), ('n', 'n'), ('op', 'op'), ('q', 'q'), ('rstuvwxyz', 'rstuvwxyz'), ('abc', 'cab'), ('defgh', 'hgfed')]

Step3: Based on the length and order of words, sort them as below.O(nlogn)

[('m', 'm'), ('n', 'n'), ('q', 'q'), ('op', 'op'), ('abc', 'abc'), ('abc', 'cab'), ('gikl', 'igkl'), ('defgh', 'defgh'), ('defgh', 'hgfed'), ('rstuvwxyz', 'rstuvwxyz')]

Step4: Using a test\_word to check if next word has the same key, if the key is same, then append the original word to the end of the last element of final\_list. If not, append a new list to final\_list and update the test\_word.O(n)

Finally, the program will give an output as below.

[['m'], ['n'], ['q'], ['op'], ['abc', 'cab'], ['igkl'], ['defgh', 'hgfed'], ['rstuvwxyz']]

Total: nklogk + nlogn+n

Observation of dict1:

Average time: 2.464

Time spent on each word: 0.00003428

Time spent on sorting words: 1.0392

Time spent on sorting single word: 0.00001446

Time spent on sorting whole list: 1.3474

Time spent on sorting whole list for each word: 0.00001874

Observation of dict2:

Average time: 220.344

Time spent on each word: 0.0006869

Time spent on sorting words: 66.590

Time spent on sorting single word: 0.000207

Time spent on sorting whole list: 143.842

Time spent on sorting whole list for each word: 0.000448

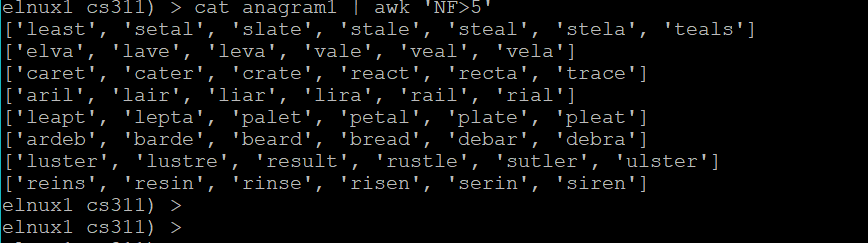
3.When I type time a.out <dict1 >anagram1 in the console, it says a.out command not found. Please do it yourself.

Windows

4.

dict1: 67605

dict2: 320750



PS: I know the run time is not fast enough and the program is not performing good as well. However, I think that’s enough fast enough for future if some people want to use this algorithm for their own purpose.