Textbook anagram Time Complexity:

Anagram 1: doesn’t see if words are same size

Average Case:

O(n)

+O(1/2n^2)

=O(n^2)

def anagramSolution1(s1,s2):

alist = list(s2)

pos1 = 0

stillOK = True

while pos1 < len(s1) and stillOK:

pos2 = 0

found = False

while pos2 < len(alist) and not found:

if s1[pos1] == alist[pos2]:

found = True

else:

pos2 = pos2 + 1

if found:

alist[pos2] = None

else:

stillOK = False

pos1 = pos1 + 1

return stillOK

print(anagramSolution1('abcd','dcba'))

Anagram 2: doesn’t see if words are same size, can generate an error

O(2nlogn)

+O(n) -> at most

=O(nlogn)

def anagramSolution2(s1,s2):

alist1 = list(s1)

alist2 = list(s2)

#should have an if statement to compare lengths, return false if lengths are not equal

alist1.sort()

alist2.sort()

pos = 0

matches = True

while pos < len(s1) and matches:

if alist1[pos]==alist2[pos]:

pos = pos + 1

else:

matches = False

return matches

print(anagramSolution2('abcde','edcba'))

Anagram 4: ordinal

O(54)

+O(2n)

+O(26) -> worst case

=O(n)

def anagramSolution4(s1,s2):

c1 = [0]\*26

c2 = [0]\*26

#could have an if statement to compare lengths, return false if lengths are not equal

for i in range(len(s1)):

pos = ord(s1[i])-ord('a')

c1[pos] = c1[pos] + 1

for i in range(len(s2)):

pos = ord(s2[i])-ord('a')

c2[pos] = c2[pos] + 1

j = 0

stillOK = True

while j<26 and stillOK:

if c1[j]==c2[j]:

j = j + 1

else:

stillOK = False

return stillOK

print(anagramSolution4('apple','pleap'))