import nltk

import numpy

import random

import timeit

def wrapper(func, \*args, \*\*kwargs):

def wrapped():

return func(\*args, \*\*kwargs)

return wrapped

def merge(l, r):

merged\_list = []

l\_i = 0

r\_i = 0

len\_list = len(l) + len(r)

len\_l = len(l)

len\_r = len(r)

while len(merged\_list) < len\_list:

if l\_i >= len\_l:

merged\_list += r[r\_i:]

elif r\_i >= len\_r:

merged\_list += l[l\_i:]

elif l[l\_i] < r[r\_i]:

merged\_list.append(l[l\_i])

l\_i += 1

else:

merged\_list.append(r[r\_i])

r\_i += 1

return merged\_list

def merge\_sort(u\_list):

len\_u\_list = len(u\_list)

if len\_u\_list == 1:

return u\_list

else:

l = merge\_sort(u\_list[:len\_u\_list/2])

r = merge\_sort(u\_list[len\_u\_list/2:])

return merge(l, r)

def partition(u\_list, lo, hi):

pivot = u\_list[lo]

i = lo-1

j = hi+1

while True:

i += 1

while u\_list[i] < pivot:

i += 1

j -= 1

while u\_list[j] > pivot:

j -= 1

if i >= j:

return j

temp = u\_list[i]

u\_list[i] = u\_list[j]

u\_list[j] = temp

def quick\_sort(u\_list, lo, hi):

if lo < hi:

p = partition(u\_list, lo, hi)

l\_list = quick\_sort(u\_list, lo, p)

r\_list = quick\_sort(u\_list, p+1, hi)

return l\_list

else:

return u\_list

def main():

unsorted = random.sample(xrange(1000000), 1000000)

#unsorted = list(set(nltk.corpus.gutenberg.words('austen-emma.txt')))

#unsorted = ["bobby", "crimi", "sam", "beckett", "backett"]

#print unsorted

wrapped = wrapper(merge\_sort, unsorted)

print timeit.timeit(wrapped, number=1)

wrapped = wrapper(quick\_sort, unsorted, 0, len(unsorted)-1)

print timeit.timeit(wrapped, number=1)

#sorted\_list = merge\_sort(unsorted)

if \_\_name\_\_ == "\_\_main\_\_":

main()

import numpy

class tree(object):

def \_\_init\_\_(self, root, max\_level):

self.root = root

self.max\_level = max\_level

def depth\_first(self, root):

if root.leaves == []:

print root.value

return

else:

print root.value

for leaf in root.leaves:

self.depth\_first(leaf)

def breadth\_first(self, root):

for level range(self.max\_level):

print

class node(object):

def \_\_init\_\_(self, value, level, leaves):

self.value = value

self.level = level

self.leaves = leaves

def main():

lf = node("F", 2, [])

le = node("E", 2, [])

ld = node("D", 2, [])

lc = node("C", 1, [le, lf])

lb = node("B", 1, [ld])

la = node("A", 0, [lb, lc])

t = tree(la, 2)

#t.depth\_first(la)

t.breadth\_first(la)

if \_\_name\_\_ == "\_\_main\_\_":

main()

import nltk

import math

import numpy as np

def hash\_function(inpt, N):

xlength = len(inpt)

s = 0

for i in range(xlength):

s += ord(inpt[i])

return s % N

def main():

hashTable = np.empty([10000, 1], dtype="S10")

words = list(set(nltk.corpus.gutenberg.words('austen-emma.txt')))

for word in words:

hashTable[hash\_function(word, 10000)] = word

testword = words[1500]

testindex = hash\_function(testword, 10000)

print hashTable[testindex], testword

if \_\_name\_\_ == "\_\_main\_\_":

main()