# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# INTERVIEW PREP - Programming Interview Questions from

# Elements of Programming Interviews

# Cracking the Coding Interview

# Hackerrank and Leetcode

# Author: Alex H Chao

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import numpy as np

from random import randint

print(randint(0,9))

# Elements of Programming Interviews

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# To do Still:

# - Implement 3 sorting algos (mergesort, quicksort, etc)

# - Implement linked list and BST

# - Implement stack / queue

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Bitwise Ops

#USING XOR to single out non-pairs

arr = [1,1,2,2,1]

result = arr[0]

for i in range(1,len(arr)):

result = result ^ arr[i]

print result

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - ch 6 - Arrays

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Example: given an array of integers, reorder so that its even entries appear first

#swap, pass by reference

def swap(a,i,j):

"""

input: array a, swap indicies i,j

"""

temp = a[i]

a[i] = a[j]

a[j] = temp

return a

a = [x for x in range(1,10)]

swap(a,0,1)

a # a is changed!

#generate a random vector

a = [randint(1,9) for x in range(10)]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# ch 6 - Arrays - From EPI ch6 Arrays, p 54,

# Given array, reorder so even entries appear first

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#TRICK - DIVIDE INTO 3 SUBARRAYS - EVENS, UNCLASSIFIED, ODD

# ITERATE THROUGH ALL UNCLASSIFIED

#even = 0

def even\_odds\_2(a):

ptr = 0

odd = len(a)-1

while ptr < odd:

if a[ptr] % 2 != 0: # if its odd, swap

swap(a,odd,ptr) # decrement odd

odd -= 1

else: # if even. move to next element

ptr += 1

#even += 1

return a

even\_odds\_2([randint(1,9) for x in range(10)])

def evens\_odds(a):

"""

From EPI ch6 Arrays, p 54, Given array, reorder so even entries appear first

"""

# initialize 2 points, one at each end

start\_evens = 0

end\_odds = len(a)-1

while start\_evens < end\_odds: # these 2 pointers will move inwards until they meet in middle

#if its even, move on to the next one

if a[start\_evens] % 2 == 0:

start\_evens += 1

else: # otherwise, swap it out

swap(a,start\_evens,end\_odds)

#move odds pointer inwards

end\_odds -= 1

return a

evens\_odds([randint(1,9) for x in range(10)])

from collections import deque

#rotate an array (Hacker rank)

#trick is to find a formula for the new position (k mod n)

s = "saveChangesToSpace"

if s is None:

print 0

count = 1

for letter in s:

if letter.lower() != letter:

count += 1

print count

#matching socks

# Check if n is prime

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def is\_prime(n):

for i in range(2,int(np.math.sqrt(n)+1)):

if n % i == 0:

return 0

return 1

print [x for x in range(20)]

print [is\_prime(x) for x in range(20)]

# 8.6 - Write a program tha takes an integer n and returns all primes between 1 and n

# EPI p65

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#tip: for each i, only go up to sqrt(i)

#tip: seive or remove all multiples of any primes from future calculations

# also known as Sieve of eratosthenes

N = 20

def generate\_all\_primes(N):

#keep track of all primes

is\_prime\_arr = [1 for i in range(N)]

is\_prime\_arr[0] = is\_prime\_arr[1] = 0

primes = []

for n in range(2,N): # start at 2

#divide each i by 2 to srt(i)

if is\_prime\_arr[n]:

primes.append(n)

# seive out multiples

for j in range(n,N,n):

is\_prime\_arr[j] = 0

return primes

generate\_all\_primes(50)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Write program that takes an array A and index i , and rearranges elements so that all

# elements < A[i] (the pivot) appear 1st, then elements == p, then elements > p

# INCOMPLETE, NEED TO REVISIT

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def dutch\_flag(p, s):

"""

keep invariants:

bottom group: [0, smaller-1]

middle group: [smaller, equal-1]

unclassified: [equal, larger-1]

largergroup: [larger, N]

"""

smaller = 0

larger = len(s)-1 #initialize at end

equal = 0 # this will be our "main" pointer

pivot = s[p] # initialize pivot

while equal < larger:

if s[equal] < pivot:

#put in smaller group

swap(s, equal,smaller)

#increment both

equal += 1

smaller += 1

elif s[equal] > pivot:

#put in larger group

swap(s,equal,larger)

larger -= 1

#equal += 1 # dont increment

else:

equal += 1

print s

return s

#generate a random vector

a = [randint(1,3) for x in range(10)]

a

p=0

dutch\_flag(p,a)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI 6.2, p 59

# WRite a program that increments a decimal number from D to D+1, e.g. [1,2,9] -> [1,3,0]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

a = [1,8,8]

for i in reversed(range(len(a))):

if a[i] == 9:

a[i] = 0

else:

a[i] += 1

break

print a

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI 6.6, p 62

# Write program that takes an array of daily stock prices and returns max profit of selling

# 1 share

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def max\_profit(price):

max\_so\_far = 0

min\_price\_thus\_far = 9999

for i in range(len(price)):

min\_price\_thus\_far = min(min\_price\_thus\_far, price[i])

profit\_if\_sold\_today = price[i] - min\_price\_thus\_far

max\_so\_far = max(max\_so\_far, profit\_if\_sold\_today)

return max\_so\_far

price = [randint(80,100) for x in range(10)]

price

max\_profit(price)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Cracking the Coding Interview (CTCI) - Ch 17 - Moderate

# 17.8 - Given array of integers (both positive and negative) find contiguous max subarray with largest sum

# WARNING: DOESNT SUPPORT ARRAYS WITH ALL NEGATIVE NUMBERS OR EMPTY ARRAYS (RETURNS 0)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

arr = [2,3,-8,-1,2,4,-2,3]

arr = [-1,-2,-3,-4,-5]

def max\_contiguous\_subarray(arr,verbose=False):

max\_so\_far = 0

max\_ending\_here = 0

for i in range(len(arr)):

max\_ending\_here += arr[i]

if max\_ending\_here > max\_so\_far:

max\_so\_far = max\_ending\_here

# IMPORTANT: CHECK IF max\_ending\_here < 0 (WHY? anytime our running max is negative, restart the sum

# as we would never include a subset that sums to be negative overall

if max\_ending\_here < 0:

max\_ending\_here = 0

if verbose:

print "arr[i] = ", arr[i]

print "max ending here = ", max\_ending\_here

print "max\_so\_far = ", max\_so\_far

return max\_so\_far

max\_contiguous\_subarray(arr)

# KADANES ALGORITHM (UTILIZING A RNNING MIN) - GREEN BOOK p181

# ========================================================================

def max\_cont\_subarray\_2(arr): # includes trough to peak indices

T = arr[0]

running\_max = T

running\_min = min(0,T)

a=0; b=0

for j in range(1,len(arr)):

T = T + arr[j]

if T - running\_min > running\_max:

running\_max = T - running\_min # we found a new max

b=j # save the index (trough)

if T < running\_min:

running\_min = T # we found a new min

a=j # save the index (peak)

return running\_max,a,b

arr = [1,2,-5,-8,-2,2,6,8,-1,-2,5]

max\_contiguous\_subarray(arr)

max\_cont\_subarray\_2(arr)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# TWO SUM (2sum) PROBLEM (LEET CODE)

# given an array of sorted numbers, find 2 numbers which sum up to a target number,

# return the indices

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

numbers = [1,3,4,5,5,7,9]

target = 9

twoSum(numbers, target)

def twoSum(numbers, target):

# 2 pointers

start = 0

end = len(numbers)-1

while start < end:

sum = numbers[start] + numbers[end]

if sum > target: # sum too big, decrement end

end -= 1

elif sum < target: # sum too small, increment start

start += 1

elif sum == target:

#return\_string = "index1=",start,"index2=",end

return start+1, end+1

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Longest substring without repeating chars - PROBLEM (LEET CODE)

# Given a string, find the length of the longest substring without repeating characters.

# e.g. Given "abcabcbb", the answer is "abc", which the length is 3.

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

s = 'dvdf'

lengthOfLongestSubstring(s)

s[0:1]

def lengthOfLongestSubstring(s):

max\_ending\_here = 0

max\_so\_far = 0

max\_index\_ending\_here = 0

letters\_seen = {}

for i in range(0,len(s)):

#if we've seen this letter before, reset the counter

if letters\_seen.has\_key(s[i]) and s[i-1] != s[i]:

max\_ending\_here = 2

elif letters\_seen.has\_key(s[i]) and s[i-1] == s[i]:

max\_ending\_here = 1

else: # new character

max\_ending\_here += 1

if max\_ending\_here > max\_so\_far:

max\_so\_far = max\_ending\_here

max\_index\_ending\_here = i

#add to hashmap to keep track of letters weve seen before

letters\_seen[s[i]] = 1

#return max\_so\_far

return max\_so\_far, s[(max\_index\_ending\_here-max\_so\_far+1):max\_index\_ending\_here+1]

"""

:type s: str

:rtype: int

"""

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - Ch 17 Dynamic Programming - p273 - Example

# Find max sum over all subarrays

# SUPPORTS ALL CASES INCL ALL NEGATIVE NUMBERS

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def max\_sub\_array\_2(arr,verbose=False):

min\_sum = 0

sum\_here = 0

max\_sum = 0

for i in range(len(arr)):

sum\_here += arr[i]

if sum\_here < min\_sum:

min\_sum = sum\_here # new trough

if sum\_here - min\_sum > max\_sum: # new peak

max\_sum = sum\_here - min\_sum

if verbose:

print "arr[i] = ", arr[i]

print "sum ending here = ", sum\_here

print "max\_sum\_so\_far = ", max\_sum

print "min\_sum\_so\_far = ", min\_sum

return max\_sum

max\_sub\_array\_2(arr,verbose=True)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - Ch 25 Honors Class - 25.5 Compute longest increasing contiguous subarray

# longest contiguous increasing subarray, monotonic

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Longest subarray ending at j+1

# 1. A[j+1], if A[j+1] <= A[j]

# 2. longest subarray ending at j + A[j+1], if A[j+1] > A[j]

# 3. Keep additional variables to store max\_so\_far, index, length, etc

# Matrix Multiplication

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

A = [[1,2],[3,4],[5,6]]

A[1][1]

len(A) # 3

len(A[0]) # 2

B = [[1,2,3],[3,2,1]]

A = [1,2,3,4,5]

B = [[cols for cols in range(1,2)] for rows in range(0,5)]

#B = [[y for y in range(1,3)] for x in range(3,5)]

A.shape

def matrix\_multiply(A,B):

#compute dimensions

# A = [n,m], B= [p,q]

n = len(A) # num rows

m = len(A[0]) # num cols

p = len(B)

q = len(B[0])

if m != p:

raise NameError('Matrix Dimensions are not compatible for multiplication') # ERROR, DIMENSIONS DONT MATCH

C = [[0 for cols in range(q)] for rows in range(n)] # new matrix will be n \* q

for row in range(n): # row by row

for col in range(q): # for each col

sum = 0

for i in range(p): # each element

sum += A[row][i] \* B[i][col]

C[row][col] = sum

return C

matrix\_multiply(A,B)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 7 - EPI - Strings p7.1 p 86

# Write program that takes a string representing an integer and return the interger,

# and vice versa

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def int\_to\_string(num):

#hint: x mod 10, x/10

while x:

last\_digit = x % 10

remaining\_digits = x/10

# STILL IN PROGRESS

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Implement mergesort in python

# WORKS but needs review

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Merge 2 sorted arrays

m = mid - low

n = hi - (mid+1)

def merge(left, right): # merge 2 arrays

c = []

a,b = 0,0

#left = [left]

#right = [right]

while a < len(left) and b < len(right): # 2 pointers - a,b

if left[a] < right[b]: # add min(a,b) to our new array, increment pointer

c.append(left[a])

a += 1

elif left[a] > right[b]:

c.append(right[b])

b += 1

else: # left[a] == right[b]

c.append(left[a]) # if equal, add both, increment both pointers

c.append(right[b])

a += 1

b += 1

#need to add leftovers

while a < len(left):

c.append(left[a])

a += 1

while b < len(right):

c.append(right[b])

b += 1

return c

# ======== TEST ==========

left = [1,3,5]

right = [2,6,7]

merge(left,right)

# Merge Sort - working

def mergeSort(s):

if len(s) <2:

return s

else:

mid = int(len(s)/2)

left = mergeSort(s[:mid])

right = mergeSort(s[mid:])

print "calling merge(left,right) on", left, right

return merge(left,right)

s = [randint(1,9) for x in range(20)]

mergeSort(s)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Implement quicksort in python

# STILL IN PROGRESS

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def quicksort(s, low, high):

if ()

def partition(s,low,high):

p = high # set last item as partition pivot (this can be chosen to be anywhere)

high = high-1

while (low < high):

#increment low, decrement high, until both numbers are "out of place"

#out of place - meaning, number in low > pivot and number in high < pivot

#swap these two

print s, low,high

if s[low] >= s[p] and s[high] <= s[p]: #NEED TO SWAP

print "SWAPPING"

swap(s,low,high)

high -= 1

low += 1

elif s[high] >= s[p]:

high -= 1

elif s[low] <= s[p]:

low += 1

else:

high -= 1

low += 1

#swap low with pivot

swap(s,low,p)

return s

a = [randint(1,9) for x in range(10)]

#swap(a,1,2)

s = partition(a,0,9)

def quicksort(s,high,low):

if low < high:

p = partition()

# PYTHONIC WAY - found online

# ==========================

def quick\_sort(l):

if len(l) == 0:

return l

pivot = l[0]

pivots = [x for x in l if x == pivot]

smaller = quick\_sort([x for x in l if x < pivot])

larger = quick\_sort([x for x in l if x > pivot])

return smaller + pivots + larger

a = [randint(1,100) for x in range(100)]

a

quick\_sort(a)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Implement linked list

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class node:

#constructor

def \_\_init\_\_(self, data = None, next\_node = None):

self.data = data

self.next = next\_node

def \_\_str\_\_(self):

return str(self.data)

class linked\_list:

def \_\_init\_\_(self, head = None):

self.head = head

def append(self, data):

# if its empty

current = self.head

if current is None:

self.head = node(data)

else:

#traverse the list

while current.next is not None:

current = current.next

current.next = node(data)

def remove(self, data):

current = self.head

#if we are removing the head

if current.data == data:

self.head = current.next

else:

while current.next.data != data:

current = current.next

#stop if next is the one to delete

current.next = current.next.next

def \_\_str\_\_(self):

#traverse the list

if self.head == None:

return "Empty List"

else:

total\_string = ''

current = self.head

while current is not None:

value\_string = str(current.data)

current = current.next

total\_string += ' -> ' + value\_string

return total\_string

# ===========

l = linked\_list()

l.append(1)

l.append(2)

l.append(3)

print l

l.remove(2)

print l

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 8 - EPI - Linked Lists q8.1 on p 102

# Problem 8.1

# We have 2 singly linked lists, each node holds a number. Each list is sorted, Merge the

# lists so that they are still in order

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

l1 = linked\_list()

l2 = linked\_list()

l1.append(1)

l1.append(3)

l1.append(5)

l2.append(2)

l2.append(6)

l2.append(7)

l2.append(8)

print l1, l2

#initialize new linked list

l\_new = linked\_list()

#compare heads, take min

cur1 = l1.head

cur2 = l2.head

#set temp vars

temp1 = cur1.next

temp2 = cur2.next

if cur1.data < cur2.data:

min\_node = cur1

other\_node = cur2

else:

min\_node = cur2

other\_node = cur1

# compare other node vs next node

if other\_node < min\_node.next:

#order should be min\_node -> other\_node -> min\_node.next

other\_node.next = min\_node.next

min\_node.next = other\_node

l\_new = linked\_list(min\_node)

print l\_new

print min\_node

print other\_node.next

# use recursion

# NOT WORKING...

def merge\_lists(node\_a, node\_b):

#base case, node is null, weve reached the end

if node\_a == None:

return node\_b

if node\_b == None:

return node\_a

if node\_a.data < node\_b.data:

smaller\_node = node\_a

smaller\_node.next = merge\_lists(node\_a.next, node\_b)

else:

smaller\_node = node\_b

smaller\_node.next = merge\_lists(node\_a, node\_b.next)

return smaller\_node

new\_l = merge\_lists(l1.head,l1.head)

print new\_l

def merge\_lists(left\_head, right\_head):

c = []

left\_ptr = left\_head

right\_ptr = right\_head

while left\_ptr and right\_ptr:

if left\_ptr.data < right\_ptr.data:

c.append(left\_ptr.data)

left\_ptr = left\_ptr.next

elif right\_ptr.data < left\_ptr.data:

c.append(right\_ptr.data)

right\_ptr = right\_ptr.next

else: # left[a] == right[b]

c.append(left\_ptr)

c.append(right\_ptr)

left\_ptr = left\_ptr.next

right\_ptr = right\_ptr.next

#leftovers

while left\_ptr:

c.append(left\_ptr.data)

left\_ptr = left\_ptr.next

while right\_ptr:

c.append(right\_ptr.data)

right\_ptr = right\_ptr.next

return c

merge\_lists(l1.head,l2.head)

#found online - recursive merge lists

def MergeLists(headA, headB):

#base cases

if headA is None and headB is None:

return None

if headA is None:

return headB

if headB is None:

return headA

#recursion

if headA.data < headB.data:

new\_node = headA

new\_node.next = MergeLists(headA.next,headB)

else:

new\_node = headB

new\_node.next = MergeLists(headA,headB.next)

return new\_node

print l1,l2

new\_head = MergeLists(l1.head,l2.head)

l3 = linked\_list(new\_head)

print l3

current = new\_head

while current is not None:

print current.data

current = current.next

# REVERSE A LINKED LIST

# ==============================

print l1

print reverse\_list\_iter(l1)

def reverse\_list\_iter(l):

if l.head is None:

return None

elif l.head.next is None:

return l

else:

prev = l.head

current = l.head.next

while current is not None:

print current.data

#first save the next node

nextNode = current.next

current.next = prev

prev = current

current = nextNode

l.head = prev

return l

def reverse\_list(l):

#helper function for recursion

def reverse\_list\_helper(current):

#base case

if current.next is None:

return current

else:

last = reverse\_list\_helper(current.next)

last.next = current

#Example on p117

# print linked list in reverse using recursion

def print\_list\_reverse(l):

traverse\_list(l.head)

def traverse\_list(current):

if current.next is None:

print current

else:

traverse\_list(current.next)

print current

print l1

print\_list\_reverse(l1)

# print linked list in reverse using a stack

def print\_list\_reverse\_stack(l):

stck = []

current = l.head

while current != None:

stck.append(current.data)

current = current.next

while stck:

print stck.pop()

print\_list\_reverse\_stack(l1)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 8.7 - EPI p 109 - Remove the k-th last element from a list

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Here, the trick is to iterate 2 ptrs, one at head, the other at head + k, when

# faster pointer hits end, we know the slower one is where we want

def remove\_kth\_last\_element(l,k):

slow\_ptr = l.head

fast\_ptr = l.head

for i in range(k): # advance fast ptr k nodes

fast\_ptr = fast\_ptr.next

if fast\_ptr is None: # k is > size of list

return 'Nil'

#advance both ptrs together

while fast\_ptr.next is not None:

fast\_ptr = fast\_ptr.next

slow\_ptr = slow\_ptr.next

#now that fast ptr is at the end, slow\_ptr must point to the k+1 last node

slow\_ptr.next = slow\_ptr.next.next

#TEST

l = linked\_list()

for i in range(1,11):

l.append(i)

print l

remove\_kth\_last\_element(l,20)

print l

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Recursion

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# GOOD RECURSION EXAMPLE from Brett Bernstein

n=3

def p(n):

if n == 0:

return

print "before: ",n

p(n-1)

print "after: ",n

p(3)

# Implement binary search

# make sure to include "return" on "return binary search(...)

def binary\_search(arr,left,right,target):

#base case

if left == right:

return -1 # not found

mid = (left + right) / 2

if arr[mid] == target: # found it

return mid

elif target < arr[mid]: # go lower

return binary\_search(arr,left, mid,target)

else:

return binary\_search(arr,mid+1,right,target)

arr = [randint(1,9) for x in range(10)]

arr

print binary\_search(arr,0,len(arr)-1,4)

found

(len(arr)-1)/2

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# ch 9 - Stacks and Queues - EPI - p117

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Implement a stack, stack = LIFO, similar to linked list

class node:

#constructor

def \_\_init\_\_(self, data = None, next\_node = None):

self.data = data

self.next = next\_node

def \_\_str\_\_(self):

return str(self.data)

class stack:

def \_\_init\_\_(self, head = None):

self.head = head

def push(self, data):

# if its empty

current = self.head

if current is None:

self.head = node(data)

else:

#add to begining

new\_head = node(data)

new\_head.next = self.head

self.head = new\_head

def pop(self):

if self.head is None:

return None

to\_remove = self.head

self.head = self.head.next

return to\_remove.data

def peek(self):

if self.head is None:

return None

return self.head.data

def \_\_str\_\_(self):

#traverse the list

if self.head == None:

return "Empty List"

else:

total\_string = ''

current = self.head

while current is not None:

value\_string = str(current.data)

current = current.next

total\_string += ' -> ' + value\_string

return total\_string

# ===================

s = stack()

s.push(1)

s.push(2)

s.push(3)

s.peek()

s.pop()

print s

s.pop()

s.pop()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 9.1 Implement a stack with Max (and Min) API

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# NEED TO FIX: NEED TO ADD running min, not just min

class stack\_max\_min:

def \_\_init\_\_(self, head = None):

self.head = head

self.max = stack()

self.min = stack()

def push(self, data):

# if its empty

current = self.head

if current is None:

self.head = node(data)

#update max and min

self.max.push(data)

self.min.push(data)

else:

#add to begining

new\_head = node(data)

new\_head.next = self.head

self.head = new\_head

#update max or min

if data >= self.max.peek(): # we have a new max

self.max.push(data)

if data <= self.min.peek(): # we have a new min

self.min.push(data)

def pop(self):

if self.head is None:

return None

to\_remove = self.head

self.head = self.head.next

#update max or min

if to\_remove.data == self.min.peek(): # we are removing current min

self.min.pop()

if to\_remove.data == self.max.peek(): # we are removing current max

self.max.pop()

return to\_remove.data

def peek(self):

return self.head.data

def get\_max(self):

return self.max.peek()

def get\_min(self):

return self.min.peek()

def \_\_str\_\_(self):

#traverse the list

if self.head == None:

return "Empty List"

else:

total\_string = ''

current = self.head

while current is not None:

value\_string = str(current.data)

current = current.next

total\_string += ' -> ' + value\_string

return total\_string

# ===========

s = stack\_max\_min()

for i in [2,3,2,5,1,6,9,1,2]:

s.push(i)

print s, " : ", s.get\_max(), s.get\_min()

for i in range(9):

s.pop()

print s, " : ", s.get\_max(), s.get\_min()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 9.9 Implement a Stack using 2 Queues

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class MyQueue(object):

def \_\_init\_\_(self):

self.first = []

self.second = []

#look at top element

def peek(self):

if len(self.second)==0: # Only do this if 2nd stack is empty

while len(self.first) > 0:

self.second.append(self.first.pop())

if len(self.second)==0: # if its still empy

return 'empty'

else:

return self.second[len(self.second)-1]

def pop(self):

if len(self.second)==0: # Only do this if 2nd stack is empty

while len(self.first) > 0:

self.second.append(self.first.pop())

if len(self.second)==0: # if its still empy

return 'empty'

else:

return self.second.pop()

def put(self, value):

#put in stack 1

self.first.append(value)

q = MyQueue()

q.put(1)

q.put(2)

q.put(3)

q.pop()

q.peek()

q.pop()

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 11 - Heaps EPI p158

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import heapq as hq

# 11.5 Compute running median

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# TRICK: maintain 2 heaps, a min and max heap

min\_heap = []

max\_heap = []

hq.

for i in [randint(0,99) for x in range(20)]:

heapq.heappush(min\_heap,i)

print min\_heap

?heapq

len(min\_heap)

#min and max heaps [max heap] [median] [min heap]

def running\_median(n, min\_heap, max\_heap):

if len(min\_heap) == 0 and len(max\_heap) == 0:

hq.heappush(min\_heap,n)

return min\_heap[0]

else:

if n >= min\_heap[0]:

#put in the min\_heap

hq.heappush(min\_heap, n)

else: # put in max heap

hq.heappush(max\_heap,-n)

if len(min\_heap) > len(max\_heap) +1:

#uneven, pop from min\_heap to max heap

hq.heappush(max\_heap,-hq.heappop(min\_heap))

elif len(max\_heap) > len(min\_heap):

hq.heappush(min\_heap,-hq.heappop(max\_heap))

if len(max\_heap) == len(min\_heap): #if equal lengths, return average

return 0.5 \* (min\_heap[0] - max\_heap[0])

else:

return min\_heap[0]

for i in [10,1,2,3,4,5]:

print running\_median(i,min\_heap,max\_heap)

#HACKER RANK (DOESNT PASS ALL TESTS)

def calculate\_median(n, min\_heap,max\_heap):

# if heaps are empty

if len(min\_heap) == 0 and len(max\_heap) == 0:

hq.heappush(min\_heap,n)

return float(min\_heap[0])

else:

# if a < min\_heap[0] => put in max\_heap, else put in min\_heap

if n >= min\_heap[0]:

hq.heappush(min\_heap,n)

else:

hq.heappush(max\_heap,-n) #negate, as all heaps are min\_heaps

#rebalance heaps if they get too out of balance

if len(min\_heap) > len(max\_heap)+1:

hq.heappush(max\_heap,-hq.heappop(min\_heap)) #negate before pushing into max heap

#if odd numbers, just return min\_heap[0], else average the two

if (len(min\_heap) + len(max\_heap)) % 2 == 0:

return 0.5 \* (min\_heap[0] - max\_heap[0]) #dont forget to negate the max\_heap

else:

return float(min\_heap[0])

# TICKET SALES

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

"""

1. Cre­ate a max-heap of size of num­ber of win­dows. (Click here read about max-heap and pri­or­ity queue.)

2. Insert the num­ber of tick­ets at each win­dow in the heap.

3. Extract the ele­ment from the heap k times (num­ber of tick­ets to be sold).

4. Add these extracted ele­ments to the rev­enue. It will gen­er­ate the max rev­enue since extract­ing for heap will give you the max ele­ment which

is the max­i­mum num­ber of tick­ets at a win­dow among all other win­dows, and price of a ticket will be num­ber of tick­ets remain­ing at each

window.

5. Each time we extract an ele­ment from heap and add it to the rev­enue, reduce the ele­ment by 1 and insert it again to the heap since after

num­ber of tick­ets will be one less after selling.

"""

def maximumAmount(a, k):

pq = queue.PriorityQueue() # well use a min-heap and just negate all the values

for i in a:

pq.put(-i) # negate values

total = 0

for i in range(k):

sales = pq.get() #sell the maximum which is at root of heap

total += sales

pq.put(sales+1) # put back maximum -1 into the heap

return -total

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 12 - Searching 0 EPI p172

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Search a sorted array for 1st occurance of K

def search\_array\_first(arr,k):

left =0

right = len(arr)-1

while left < right:

mid = (left + right)/2

if k < arr[mid]:

right = mid-1

elif k == arr[mid]: # found one occurence

right = mid

else: # k > arr[mid]

left = mid+1

return right

arr = [1,3,4,4,5,5,5,5,5,5,7,8,9,9]

k = 5

search\_array\_first(arr,k)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - 12.8 0 Find the k-th largest element (similar to find the median of array), p 180

# kth largest element

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#BRUTE FORCE: sort -> O(nlogn)

# BETTER:

# USE MIN HEAP, store the top k largest elements, if new num < k, discard. O(nlogk)

# BEST:

#TRICK: pivot, similar to quicksort -> O(n) on average

# pick random "pivot", bucket all < pivot to the left, all > pivot to the right

# each time, discarding (half) the array

# IN PROGRESS

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - Ch 25 Honors Class - 25.15 - Search array of unknown length

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# USe binary search

# increment by 2^i

# as soon as we find invalid value, say 2^i - 1, use binary search on interval [2^(i-1), 2^i-2]

# O(logn)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 13 - Hash Tables EPI p190

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 13.1

# Write a function to test whether the letters forming a string can be permuted to form a palindrome

# (spells the same backwards as forwards)

# form palindrome if: num letters is even, each has 2 occurences

# num letters is odd, each has 2 occurences, one has one occurence

def can\_form\_palindrome(s):

my\_dict = {}

for i,letter in enumerate(s):

if my\_dict.has\_key(letter):

my\_dict[letter]+= 1

else:

my\_dict[letter] = 1

counts = []

#once dict is populated, count all occurences

for i,j in my\_dict.iteritems():

counts.append(j)

mods = sum(map(lambda x: int(x) % 2, counts)) # sum up each reminder

if len(s) % 2 == 0: # IF EVEN

return mods == 0

else:

return mods == 1

#TESTING

s = 'levels'

print can\_form\_palindrome(s)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 14 - Sorting EPI p213

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 14.1

# Intersection of 2 sorted arrays, intersecting sorted arrays

# Write a function that takes as inputs, 2 sorted arrays, returns new array containing

# elements present in BOTH input arrays (no dupes)

#ALGO

# - 2 pointers 1 to a, 1 to b

# - increment each and compare

# - if both equal, add to new array, increment both

# - Complexity = O(n + m)

def intersect\_arrays(arr\_a,arr\_b):

ptr\_a = 0

ptr\_b = 0

out = []

while ptr\_b <= len(arr\_b)-1 and ptr\_a <= len(arr\_a)-1:

if arr\_a[ptr\_a] == arr\_b[ptr\_b]: # we have a match

if len(out) == 0:

last\_value = None

else:

last\_value = out[len(out)-1] # save last value

if last\_value != arr\_a[ptr\_a]: # make sure u dont add dupes

out.append(arr\_a[ptr\_a])

ptr\_b += 1

ptr\_a += 1

elif arr\_a[ptr\_a] > arr\_b[ptr\_b]:

ptr\_b += 1 #increment ptr\_B

elif arr\_b[ptr\_b] > arr\_a[ptr\_a]:

ptr\_a += 1 # increment ptr a

return out

arr\_a = [2,3,3,4,4,5,6,6,8,10,12]

arr\_b = [5,5,6,8,8,9,10,10]

print intersect\_arrays(arr\_a,arr\_b)

out = []

len(out)

# 14.2

# ================================

# Merge 2 sorted arrays

# Write a function that takes as inputs, 2 sorted arrays, merged arrays in place of

# 1st array

# O(m+n)

# TRICK -> fill in starting from the end (to avoid shifting)

#m = length of arr\_a, n = length of arr\_b

def merge\_2\_sorted\_arrays(arr\_a, arr\_b, m, n):

a = m-1

b = n-1

c = m+n-1 # start at end

while a >= 0 and b >= 0:

if arr\_a[a] > arr\_b[b]:

arr\_a[c] = arr\_a[a]

a -= 1

elif arr\_b[b] >= arr\_a[a]:

arr\_a[c] = arr\_b[b]

b -= 1

c -= 1

#print arr\_a, a, b,c

# we still may have some left over in B

while b >=0:

arr\_a[c] = arr\_b[b]

b -= 1

c -= 1

return arr\_a

arr\_a = [2,3,3,4,'','','','','','']

arr\_b = [1,1,1,5,6]

m=4

n=5

print merge\_2\_sorted\_arrays(arr\_a, arr\_b, m, n)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# EPI - problem 14.5 - Merging Intervals, p218

# Take as input an array of disjoint closed intervals and an interval to be added and returns union

# of all intervals

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Algo:

# Iterate through all intervals that appear completely before the interval to be added (call it A)

# once we encounter an interval which intersects A, compute its union and add it

# compare newly formed interval to the next interval, union if necessary

# iterate through remaining intervals

# Complexity -> O(n)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 15 - Binary Search Trees - EPI - p 253

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

class bst\_node:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

def insert(self, data):

if data < self.data:

if self.left is None:

self.left = bst\_node(data)

else:

self.left.insert(data)

else: # data > self.data

if self.right is None:

self.right = bst\_node(data)

else:

self.right.insert(data)

def print\_in\_order(self):

#print in order traversal

if self.left is not None:

self.left.print\_in\_order()

print self.data," -> ",

if self.right is not None:

self.right.print\_in\_order()

root = bst\_node(50)

for i in [randint(1,99) for x in range(5)]:

root.insert(i)

root.print\_in\_order()

# 15.1 Test if BST satisfies the BST condition

# ==================================================

# TRICK: when traversing the tree, check to make sure each nodes value falls within an interval [-infinity, root] (for left subtree)

# or [root, parent root] and keep updating the min,max intervals

int\_min = -999999

int\_max = 999999

def BST\_check(root,my\_min,my\_max):

if root is None:

return

elif root.data > my\_max or root.data < my\_min: # if this node is outside our range

return "False"

else:

my\_data = root.data

print "checking: ", my\_data, " in (", my\_min, " : ", my\_max, ")"

BST\_check(root.left,my\_min,my\_data)

BST\_check(root.right,my\_data,my\_max)

return "True"

BST\_check(root,int\_min,int\_max)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 16 - Recursion - EPI - p 253

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Example - calculate GCD of 2 numbers

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#hint -> gcd(x,y) = gcd(x, y-x) if y > x

# this implied gcd(x,y) = gcd(x, y mod x)

def gcd(x,y):

if y==0:

return x

elif y > x:

return gcd(x, y%x)

elif y < x:

return gcd(y, x%y)

else: # x == y

return x

gcd(144,8)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 16.1 - Towers of Hanoi

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#represent 3 towers as 3 individual stacks

s1 = [3,2,1]

s2 = []

s3 = []

def towers\_hanoi(n # how many remaining

,source #source pile (a stack)

,helper # helper pile ( a stack)

,target): #target pile, a stack):

if n > 0:

print s1,s2,s3

#move n-1 tower over to helper

towers\_hanoi(n-1,source,target,helper)

if len(source) > 0: #if source not empty

#move from source to target

target.append(source.pop())

print source, target

#move from helper to target

towers\_hanoi(n-1,helper,source,target)

towers\_hanoi(3,s1,s2,s3)

print s1,s2,s3

# DONT USE RETURN STATEMENT HERE!!

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 17 - Dynamic Programming - EPI - p 272

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Example - fibonacci

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def fib\_slow(n):

if n < 2:

return n

else:

return fib\_slow(n-1) + fib\_slow(n-2)

fib\_slow(50)

fib(50)

#Enhanced version (cache values in a dict?)

cache = {}

def fib(n):

if n < 2:

return n

else:

# check if f(n-1) and f(n-2) exist in our dict

# if not, calculate it (use recursion)

if cache.has\_key(n-1):

fib\_n\_1 = cache[n-1]

else:

fib\_n\_1 = fib(n-1)

cache[n-1] = fib\_n\_1

if cache.has\_key(n-2):

fib\_n\_2 = cache[n-2]

else:

fib\_n\_2 = fib(n-2)

cache[n-2] = fib\_n\_2

return fib\_n\_1 + fib\_n\_2

#BETTER, FOUND ONLINE

# https://jeremykun.com/2012/01/12/a-spoonful-of-python/

def fib(n):

saved\_fib = [0,1]

for i in range(2,n+1):

saved\_fib.append(saved\_fib[i-1]+saved\_fib[i-2])

return saved\_fib[n]

for i in range(10):

print fib(i)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# HACKER RANK - compute factorial

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#save calculated factorials

saved\_facts = {}

def factorial(n):

if n <= 2:

return n

else:

if n not in saved\_facts:

saved\_facts[n] = n\*factorial(n-1)

return saved\_facts[n]

factorial(15)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 17.1 - Football Scores - write program that takes final score and outputs number of combinations

# of plays that result in the final score

# NOTE: same as the coin change problem

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# NOT WORKING

def football(n):

#define base cases

if n <= 0:

return 0

elif n==3 or n ==2:

return 1

else:

return 1+ football(n-2) + football(n-3) + football(n-7)

football(12)

n=12

scores = [2,3,7]

#wrap into a function

def football(n,scores): # Similar to knapsack problem

# Need to populate the table

# rows = number of point combos (2,3,7)

# cols = total score, up to 12

A = np.zeros([len(scores),n+1])

#1st row

for col in range(0,n+1,2):

A[0,col]=1

for row in range(1,len(scores)):

for col in range(0,n+1): #e.g. start at 2

if col < scores[row]:

A[row,col] = A[row-1,col] #less than the current number, copy over from above row

else:

A[row,col] = A[row-1,col] + A[row,col-scores[row]]

return A[len(scores)-1,n]

football(12,[1,2,3])

#WHAT IF NUMPY IS NOT ALLOWED?

# THIS WORKS

# SAME AS COIN CHANGE PROBLEM

def football(n,scores):

# Need to populate the table

# rows = number of point combos (2,3,7)

# cols = total score, up to 12

#A = np.zeros([len(scores),n+1])

#just in case u cant use numpy

A =[[0 for col in range(n+1)] for row in range(len(scores))]

#1st row

for col in range(0,n+1,scores[0]):

A[0][col]=1

#for the rest of the rows

for row in range(1,len(scores)):

for col in range(0,n+1): #e.g. start at 2

if col < scores[row]:

A[row][col] = A[row-1][col] #less than the current number, copy over from above row

else:

A[row][col] = A[row-1][col] + A[row][col-scores[row]]

return A[len(scores)-1][n], A

answer, A = football(4,[1,2,3])

A

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 17.3 - Count number of ways to traverse a 2D array

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# works! but needs review

#Recursion plus cache -> keep track of visited nodes with num\_ways matrix

def count\_ways\_to\_xy(x,y, num\_ways):

#base case

if x==0 and y==0: # we are at the origin

return 1

if num\_ways[x,y]==0: #not set yet

if x == 0:

ways\_x = 0

else:

ways\_x = count\_ways\_to\_xy(x-1,y,num\_ways)

if y == 0:

ways\_y = 0

else:

ways\_y = count\_ways\_to\_xy(x, y-1,num\_ways)

num\_ways[x,y] = ways\_x + ways\_y

print x,y,num\_ways[x,y]

return num\_ways[x,y]

n=5

num\_ways = np.zeros([n,n])

count\_ways\_to\_xy(n-1,n-1,num\_ways)

num\_ways

# =========== final submission for Hackerrank (Blue Mountain Capital Coding Test) ======================

# WORKING!!!

# Complete the function below.

def canReach(a, b, c, d):

num\_ways = [[0 for y in range(d+1)] for x in range(c+1)]

if a > c or b > d: #if starting position is greater than ending position, this is impossible

return 'No'

if count\_ways\_to\_xy(c,d,a,b,num\_ways) > 0:

return 'Yes'

else:

return 'No'

def count\_ways\_to\_xy(x,y,a,b,num\_ways):

#if we found a way, break

if num\_ways[a][b] > 0:

return 1

#base case

if x==a and y==b: # we are at the origin

return 1

if x < a or y < b or x == 0 or y ==0:

return 0

if num\_ways[x][y]==0: #not set yet

if x < a or x-y < 0: #passed origin, no paths found

ways\_x = 0

else:

ways\_x = count\_ways\_to\_xy(x-y,y,a,b,num\_ways)

if y < b or y-x < 0: #passed origin, no paths found

ways\_y = 0

else:

ways\_y = count\_ways\_to\_xy(x, y-x,a,b,num\_ways)

num\_ways[x][y] = ways\_x + ways\_y

return num\_ways[x][y]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 17.6 - The Knapsack Problem - works but needs review

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#our saved max profit matrix

values = [60,50,70,30]

weights = [5,3,4,2]

capacity = 5

V = np.zeros([len(values),capacity+1])

def knapsack(values, weights, capacity):

#if our total capacity is 0, then our optimal value has to be 0

V[:,0] = 0

for i in range(len(values)):

for w in range(capacity+1): # need to start at 0

#if current object weights too much

if weights[i] > w:

V[i,w] = V[i-1,w] # cant add the new item

else: #take max of 1) take this item, 2) dont take this item

V[i,w] = max(V[i-1,w-weights[i]]+values[i], V[i-1,w] )

return V[i,w]

knapsack(values, weights, capacity)

print V

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 17.9 - Pick up coins for max gain - p291

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

coins = [25,5,10,5,10,5,10,25,1,25,1,25,1,25,5,10]

coins = [10,15,30,20]

#save all profits in a table of (a,b)

#profit = np.zeros([len(coins), len(coins))

#keep pointers a,b

def coins\_max\_gain(a,b):

#base case

if a > len(coins)-1 or b < 0:

return 0

if a >= b:

return 0

else:

#pick a (plus min of next choice, as player B will minimize our choice)

pick\_a = coins[a] + min(coins\_max\_gain(a+1,b-1), coins\_max\_gain(a+2,b) )

#pick b (plus min of next choice, as player B will minimize our choice)

pick\_b = coins[b] + min(coins\_max\_gain(a+1,b-1), coins\_max\_gain(a,b-2))

#profit[a,b] = max(pick\_a, pick\_b)

return max(pick\_a, pick\_b)

#return profit[a,b]

coins\_max\_gain(0,len(coins)-1)

# Recursive Staircase problem (Davis Staircase)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

saved\_staircase = {1:1, 2:2, 3:4}

def staircase(n):

#base case

if n <= 3:

return saved\_staircase[n]

elif n not in saved\_staircase:

saved\_staircase[n] = staircase(n-1) + staircase(n-2) + staircase(n-3)

return saved\_staircase[n]

n=7

staircase(7)

#save into dict

# Hackerrank - Cracking the coding interview

# DP - Coin Change Problem

# Given a number of dollars, , and a list of dollar values for distinct coins,

# find and print the number of different ways you can make change for dollars if each coin

# is available in an infinite quantity.

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# similar to generalized staircase problem

coins = [1,2,3]

saved\_staircase = {}

def staircase(n):

#base case

if n <= 1:

return 1

elif n not in saved\_staircase:

for i in range(1,n)

saved\_staircase[n] = staircase(n-1) + staircase(n-2) + staircase(n-3)

return saved\_staircase[n]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Ch 18 - Greedy Algorithms and Invariants - EPI - p 272

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Example - coin change - make change for a given number of cents

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

coins = [1,5,10,25,100]

def make\_change(n,coins):

change = {} # a dictionary to keep track of how many quarters, dimes, etc

for i in reversed(coins): # iterate in descending order

if i <= n:

num\_this\_coin = int(n/i)

n = n % i

change[i]= num\_this\_coin

return change

make\_change(194,coins)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# GS Interview Q - find the median of two sorted arrays

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Cracking Coding Interview - Sorting and Search - Ch 11

# 11.6 - 2-D Array Search, or sorted matrix search (M by N)

# OR EPI problem 12.6 p 177

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Algo:

# Start from the left-most column

# if num > target: move left

# if num < target: move down

# Complexity -> worst case, insect m+n-1 elements, so O(m+n)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Leet Code - Is integer a perfect square? (hint, use binary search)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print isPerfectSquare(82)

#inefficient o(n) soln

def isPerfectSquare(num):

for i in range(1,num/2+1):

sq = i\*i

if num == sq:

return True

elif num < sq:

return False

return False

def isPerfectSquare(num):

n = num/2+1

while n <= num/2+1:

sq = i\*i

if num == sq:

return True

elif num < sq:

return False

return False