**Question 1 AIM : To develop program for maximum sub array**

a)**BRUTEFORCE**

from numpy import asarray

import numpy as np

import sys

#STOCK\_PRICES = [100, 113, 110, 85, 105, 102, 86, 63, 81, 101, 94, 106, 101, 79, 94, 90, 97]

STOCK\_PRICE\_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]

B = [[1, 2, 3, 4], [5, 6, 7, 8], [3, 4, 5, 6], [7, 8, 9, 5]]

D = [[2, 9, 8, 3], [5, 6, 0, 1], [1, 3, 2, 3], [1, 1, 2, 0]]

E = [[1, 2, 3, 4], [5, 6, 7, 8], [3, 4, 5, 6], [7, 8, 9, 5]]

F = [[2, 9, 8, 3], [5, 6, 0, 1], [1, 3, 2, 3], [1, 1, 2, 0]]

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

# The brute force method to solve max subarray problem

def find\_maximum\_subarray\_brute(A):

"""

Return a tuple (i,j) where A[i:j] is the maximum subarray.

time complexity = O(n^2)

"""

A = asarray(A)

if len(A) == 0:

return None

if len(A) == 1:

return (0, 0)

max = -sys.maxint-1

sum = 0

start = 0

end = 0

for i in range(0, A.size):

sum = 0

for j in range(i, A.size):

sum = sum + A[j]

if sum > max:

max = sum

start = i

end = j

return (start, end)

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

b)**RECURSIVE**

# The maximum crossing subarray method for solving the max subarray problem

def find\_maximum\_crossing\_subarray(A, low, mid, high):

"""

Find the maximum subarray that crosses mid

Return a tuple ((i, j), sum) where sum is the maximum subarray of A[i:j].

"""

left\_max = -sys.maxint-1

right\_max = -sys.maxint-1

sum = 0

i = mid

j = mid+1

while i >= low:

sum = sum + A[i]

if sum > left\_max:

left\_max = sum

cross\_low = i

i = i-1

sum = 0

while j <= high:

sum = sum+A[j]

if sum > right\_max:

right\_max = sum

cross\_high = j

j = j+1

return ((cross\_low, cross\_high), left\_max+right\_max)

# The recursive method to solve max subarray problem

def find\_maximum\_subarray\_recursive\_helper(A, low=0, high=-1):

"""

Return a tuple ((i, j), sum) where sum is the maximum subarray of A[i:j].

"""

if low == high:

return ((low, high), A[low])

else:

mid = int((low + high) / 2)

l\_index, l\_max = find\_maximum\_subarray\_recursive\_helper(A, low, mid)

r\_index, r\_max = find\_maximum\_subarray\_recursive\_helper(A, mid+1, high)

c\_index, c\_max = find\_maximum\_crossing\_subarray(A, low, mid, high)

if l\_max >= r\_max and l\_max >= c\_max:

return (l\_index, l\_max)

elif r\_max >= l\_max and r\_max >= c\_max:

return (r\_index, r\_max)

else:

return (c\_index, c\_max)

# The recursive method to solve max subarray problem

def find\_maximum\_subarray\_recursive(A):

"""

Return a tuple (i,j) where A[i:j] is the maximum subarray.

"""

A = asarray(A)

if len(A) == 0:

return None

return find\_maximum\_subarray\_recursive\_helper(A, 0, len(A) - 1)[0]

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

c) ITERATIVE

# The iterative method to solve max subarray problem

def find\_maximum\_subarray\_iterative(A):

"""

Return a tuple (i,j) where A[i:j] is the maximum subarray.

"""

A = asarray(A)

if len(A) == 0:

return None

start\_of\_max = 0

start = 0

end = 0

max\_computing = A[0]

max\_till\_now = A[0]

for i in range(1, A.size):

if A[i] > (max\_computing + A[i]):

max\_computing = A[i]

start = i

else:

max\_computing = A[i] + max\_computing

if max\_computing > max\_till\_now:

max\_till\_now = max\_computing

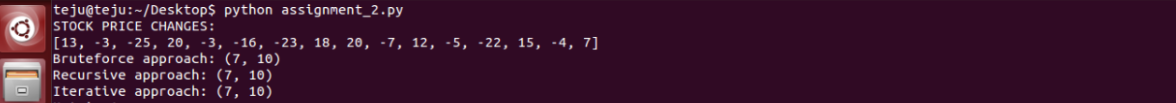
start\_of\_max = start

end = i

return (start\_of\_max, end)

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

OUTPUT:



**QUESTION 2 AIM : To implement Matrix Multiplication**

1. Square matrix multiplication

def square\_matrix\_multiply(A, B):

"""

Return the product AB of matrix multiplication.

"""

A = asarray(A)

B = asarray(B)

assert A.shape == B.shape

if A.size == 0:

return 0

assert A.shape == A.T.shape

A = asarray(A)

B = asarray(B)

assert A.shape == B.shape

assert A.shape == A.T.shape

num\_rows, num\_cols = A.shape

C = np.zeros(shape=(num\_rows, num\_cols))

for i in range(0, num\_rows):

for j in range(0, num\_rows):

for k in range(0, num\_rows):

C[i][j] = C[i][j] + A[i][k] \* B[k][j]

return C.astype(int)

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

1. Strassens Algorithm

def square\_matrix\_multiply\_strassens(A, B):

"""

Return the product AB of matrix multiplication.

Assume len(A) is a power of 2

"""

A = asarray(A)

B = asarray(B)

assert A.shape == B.shape

if A.size == 0:

return 0

assert A.shape == A.T.shape

assert (len(A) & (len(A) - 1)) == 0, "A is not a power of 2"

n = A.shape[0]

if n == 1:

return A \* B

else:

A11 = A[:int(n/2), :int(n/2)]

A12 = A[:int(n/2), int(n/2):]

A21 = A[int(n/2):, :int(n/2)]

A22 = A[int(n/2):, int(n/2):]

B11 = B[:int(n/2), :int(n/2)]

B12 = B[:int(n/2), int(n/2):]

B21 = B[int(n/2):, :int(n/2)]

B22 = B[int(n/2):, int(n/2):]

S1 = B12 - B22

S2 = A11 + A12

S3 = A21 + A22

S4 = B21 - B11

S5 = A11 + A22

S6 = B11 + B22

S7 = A12 - A22

S8 = B21 + B22

S9 = A11 - A21

S10 = B11 + B12

P1 = square\_matrix\_multiply\_strassens(A11, S1)

P2 = square\_matrix\_multiply\_strassens(S2, B22)

P3 = square\_matrix\_multiply\_strassens(S3, B11)

P4 = square\_matrix\_multiply\_strassens(A22, S4)

P5 = square\_matrix\_multiply\_strassens(S5, S6)

P6 = square\_matrix\_multiply\_strassens(S7, S8)

P7 = square\_matrix\_multiply\_strassens(S9, S10)

C11 = P5 + P4 - P2 + P6

C12 = P1 + P2

C21 = P3 + P4

C22 = P5 + P1 - P3 - P7

C = np.zeros(shape=(n, n))

C[:int(n/2), :int(n/2)] = C11

C[:int(n/2), int(n/2):] = C12

C[int(n/2):, :int(n/2)] = C21

C[int(n/2):, int(n/2):] = C22

return C.astype(int)

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

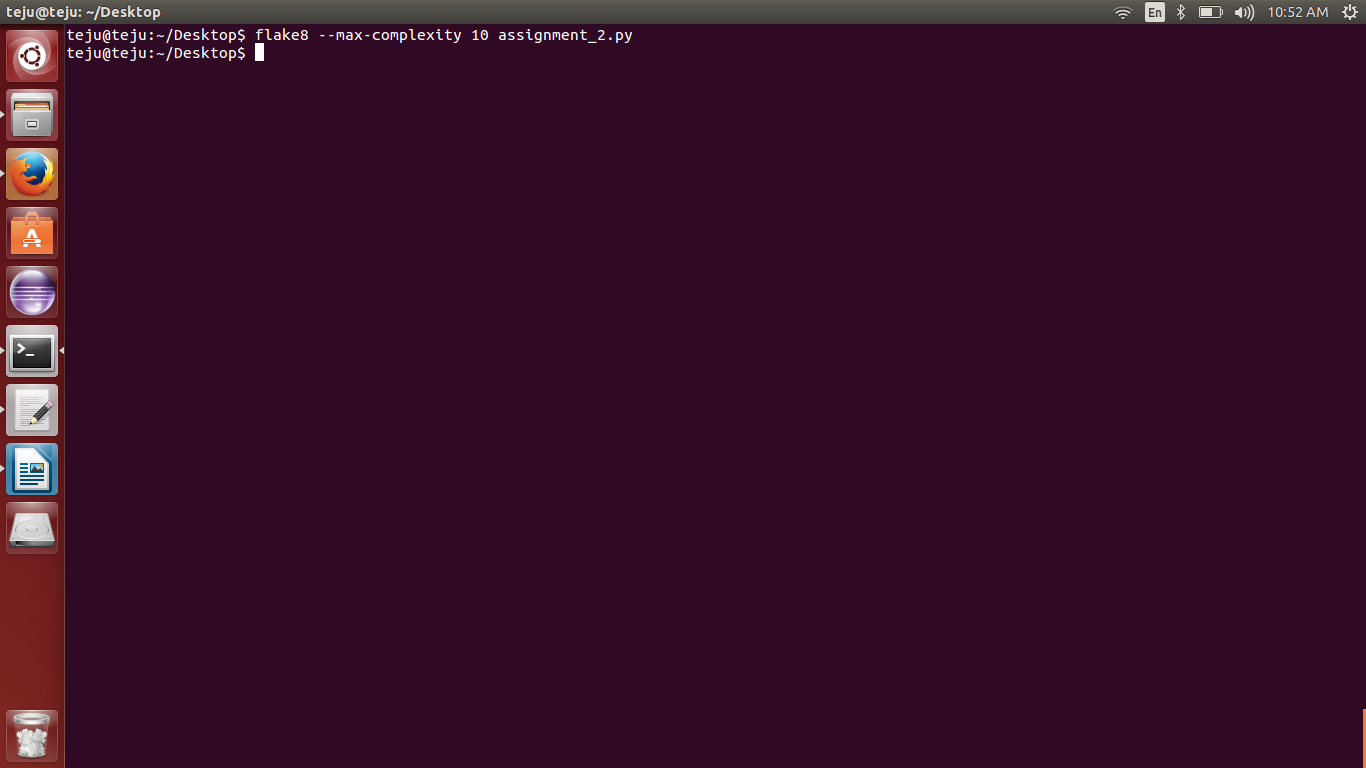
OUTPUT: 

FLAKE OUTPUT:

1. To Check warnings using flake:



1. To check McCabe complexity (<10):



Test Method:

def test():

print "STOCK PRICE CHANGES:"

print(STOCK\_PRICE\_CHANGES)

res1 = find\_maximum\_subarray\_brute(STOCK\_PRICE\_CHANGES)

res2 = find\_maximum\_subarray\_recursive(STOCK\_PRICE\_CHANGES)

res3 = find\_maximum\_subarray\_iterative(STOCK\_PRICE\_CHANGES)

print "Bruteforce approach: %s" % (res1, )

print "Recursive approach: %s" % (res2, )

print "Iterative approach: %s" % (res3, )

print "Matrix 1:"

print(asarray(B))

print "Matrix 2:"

print(asarray(D))

print "Matrix 3:"

print(asarray(E))

print "Matrix 4:"

print(asarray(F))

res4 = square\_matrix\_multiply(E, F)

res5 = square\_matrix\_multiply\_strassens(B, D)

print("Square Matrix Multiplication(Mat1\*Mat2):")

print(res4)

print("Square Matrix Multiplication using Strassens(Mat3\*Mat4):")

print(res5)

OVERALL OUTPUT:

