SER 501 – ASSIGNMENT 2 **SNEHA VIDHYASHEKAR**
ASU ID : 1211274841

SCREEN SHOTS : **1. BRUTE FORCE**

assignment_2.py

```
# 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)

    """

    if len(A) == 0:
        return None

    start = 0
    end = len(A)
    maxsum = -999999  # Setting the value to infinity

    for i in range(start, end, 1):
        total = 0

        for j in range(i, end, 1):
            # add every element of the array to the variable sum
            total = total + A[j]

            if total > maxsum:        # check if sum is greater than max-sum
                maxsum = total
                start_index = i     # Assign the start index of the sub-array
                end_index = j       # Assign the last index of the sub-array

    return (start_index, end_index, maxsum)
```

TEST CASES:

1. Positive and Negative numbers in an array

```
STOCK_PRICE_CHANGES = [15, 20, 45, -80, 81]
```

```
assignment_2.py .
Brute Force Method
(0, 4)
```

2. Only Positive Numbers

```
STOCK_PRICE_CHANGES  = [100, 113, 110, 85, 105, 102, 86, 63, 81,
                       101, 94, 106, 101, 79, 94, 90, 97]
============================= test session starts ==============================
platform win32 -- Python 2.7.13, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: C:\Users\gu\Desktop, inifile:
collected 1 items
assignment_2.py .
Brute Force Method
(0, 16)
```

3. Only Negative numbers

```
STOCK_PRICE_CHANGES =[-13, -3, -25, -20, -3, -16, -23, -18,
  -20, -7, -12, -5, -22, -15, -4, -7]
```

```
============================= test session starts ==============================
platform win32 -- Python 2.7.13, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
rootdir: C:\Users\gu\Desktop, inifile:
collected 1 items
assignment_2.py .
Brute Force Method
(1, 1)
```

2. MAXIMUM SUB-ARRAY (RECURSIVE)

assignment_2.py

```
# 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].
    """
    # By following the algorithm in text_book
    # To set the values to infinity
    # can also use decimal(-Infinity) after importing decimal

    temp_sum1 = 0
    i = mid
    left_ptr = i

    left_sum = -999999

    # To find the maximum sum at the left sub-array
    # Left array is from low to mid

    while i >= low:
        temp_sum1 = temp_sum1 + A[i]
        if temp_sum1 > left_sum:
            left_sum = temp_sum1
            left_ptr = i
        i = i - 1

    # To find the maximum sum at the left sub-array
    # Right array is from mid+1 to high

    temp_sum2 = 0
    j = mid + 1

    right_sum = -999999

    right_ptr = j
    while j <= high:
        temp_sum2 = temp_sum2 + A[j]
        if temp_sum2 > right_sum:
            right_sum = temp_sum2
            right_ptr = j
        j = j + 1

    return left_sum + right_sum, left_ptr, right_ptr


# 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].


    """
    # Base case -> when there is only 1 element
    # Algorithm in textbook is implemented


    if low == high:
        return A[low], low, high
    else:
        mid = (low + high) // 2
        left_sum, left_low, left_high = \
            find_maximum_subarray_recursive_helper(A, low, mid)
        right_sum, right_low, right_high = \
            find_maximum_subarray_recursive_helper(A, mid + 1, high)
        cross_sum, cross_low, cross_high = \
            find_maximum_crossing_subarray(A, low, mid, high)
        if left_sum >= right_sum and left_sum >= cross_sum:
            return left_sum, left_low, left_high
        elif right_sum >= left_sum and right_sum >= cross_sum:
            return right_sum, right_low, right_high
        else:
            return cross_sum, cross_low, cross_high
    # The recursive method to solve max subarray problem

```

assignment_2.py

```
130
131
132     def find_maximum_subarray_recursive(A):
133         """
134         Return a tuple (i,j) where A[i:j] is the maximum subarray.
135         """
136         # To check if the array is empty
137
138         if len(A) == 0:
139             return None
140
141         return find_maximum_subarray_recursive_helper(A, 0, len(A) - 1)
142
143
144     # ============================================================
145
```

TEST CASES:

1. For mixture of positive and negative numbers

```
STOCK_PRICE_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18,
                       20, -7, 12, -5, -22, 15, -4, 7]
```

```
Recursive method
((7, 10), 43)
```

2. For only positive numbers

```
STOCK_PRICE_CHANGES = [100, 113, 110, 85, 105, 102, 86, 63, 81,
                       101, 94, 106, 101, 79, 94, 90, 97]
```

```
Recursive method
((0, 16), 1607)
```

3. For only negative numbers

```
STOCK_PRICE_CHANGES =[-13, -3, -25, -20, -3, -16, -23, -18,
  -20, -7, -12, -5, -22, -15, -4, -7]
```

```
Recursive method
((1, 1), -3)
```

3. ITERATIVE METHOD

assignment_2.py

```
# ============================================================

# The iterative method to solve max subarray problem
def find_maximum_subarray_iterative(A, low=0, high=-1):
    """

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

    """
    if len(A) == 0:
        return None

    low = 0
    high = len(A)
    totalSum = A[low]
    tempSum = 0
    tempLeftIndex = 0
    leftIndex = 0
    rightIndex = 0

    for i in range(low, high, 1):
        tempSum = max(A[i], (tempSum + A[i]))
        if tempSum == A[i]:
            tempLeftIndex = i
        if tempSum > totalSum:
            totalSum = tempSum
            rightIndex = i
            leftIndex = tempLeftIndex
    return (leftIndex, rightIndex, totalSum)

# ============================================================
```

TEST CASES:

1. For all negative numbers

```
STOCK_PRICE_CHANGES =[-13, -3, -25, -20, -3, -16, -23, -18,
  -20, -7, -12, -5, -22, -15, -4, -7]
```

```
Iterative method
(1, 1)
```

2. For all positive numbers

```
STOCK_PRICE_CHANGES = [100, 113, 110, 85, 105, 102, 86, 63, 81,
                       101, 94, 106, 101, 79, 94, 90, 97]
```

```
Iterative method
(0, 16)
```

3. For both positive and negative numbers

```
STOCK_PRICE_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18,
                       20, -7, 12, -5, -22, 15, -4, 7]
```

```
Iterative method
(7, 10)
```

4. Matrix multiplication

assignment_2.py

```
# ============================================================


def square_matrix_multiply(A, B):
    """

    Return the product AB of matrix multiplication.

    """

    A = asarray(A)
    B = asarray(B)

    assert A.shape == B.shape
    assert A.shape == A.T.shape

    # Since it is a sqaure matrix , row = column = n

    n = len(A)

    C = [[0 for i in range(n)] for j in range(n)]
    for i in range(0, n):
        for j in range(0, n):
            C[i][j] = 0
            for k in range(0, n):
                C[i][j] = C[i][j] + A[i][k] * B[k][j]

    return C

```

Output

<u>Calling function</u>

```
A = asarray([[1, 3], [7, 5]])
B = asarray([[6, 8], [4, 2]])
print
print("Normal square matrix multiplication")
print(square_matrix_multiply(A, B))
print
```

```
Normal square matrix multiplication
[[18, 14], [62, 66]]
```

5. Strassen's matrix multiplication

assignment_2.py

```
# ============================================================


def sum_up(A, B):
    n = len(A)
    result = [[0 for i in range(0, n)] for j in range(0, n)]
    for i in range(0, n):
        for j in range(0, n):
            result[i][j] = A[i][j] + B[i][j]
    return result

    # subtracts two matrices


def difference(A, B):
    n = len(A)
    result = [[0 for i in range(0, n)] for j in range(0, n)]
    for i in range(0, n):
        for j in range(0, n):
            result[i][j] = A[i][j] - B[i][j]
    return result


def square_matrix_multiply_strassens(A, B):
    A = asarray(A)
    B = asarray(B)

    assert A.shape == B.shape
    assert A.shape == A.T.shape

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

    n = len(A)

    if n == 1:
        C = [[0 for j in range(0, n)] for i in range(0, n)]
        for i in range(0, n):
            for j in range(0, n):
                C[i][j] = A[i][j] * B[i][j]
        return C
```

```
    else:  # dividing the input matrices A and B
        new_n = int(n / 2)

    a11 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    a12 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    a21 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    a22 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

    b11 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    b12 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    b21 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    b22 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

    aTemp = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
    bTemp = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

    for i in range(0, new_n):
        for j in range(0, new_n):
            a11[i][j] = A[i][j]
            a12[i][j] = A[i][j + new_n]
            a21[i][j] = A[i + new_n][j]
            a22[i][j] = A[i + new_n][j + new_n]

            b11[i][j] = B[i][j]
            b12[i][j] = B[i][j + new_n]
            b21[i][j] = B[i + new_n][j]
            b22[i][j] = B[i + new_n][j + new_n]

    aTemp = sum_up(a11, a22)
    bTemp = sum_up(b11, b22)
    p1 = square_matrix_multiply_strassens(aTemp, bTemp)

    aTemp = sum_up(a21, a22)
    p2 = square_matrix_multiply_strassens(aTemp, b11)

    bTemp = difference(b12, b22)
    p3 = square_matrix_multiply_strassens(a11, bTemp)
```

```

    bTemp = difference(b21, b11)
    p4 = square_matrix_multiply_strassens(a22, bTemp)

    aTemp = sum_up(a11, a12)
    p5 = square_matrix_multiply_strassens(aTemp, b22)

    aTemp = difference(a21, a11)
    bTemp = sum_up(b11, b12)
    p6 = square_matrix_multiply_strassens(aTemp, bTemp)

    aTemp = difference(a12, a22)
    bTemp = sum_up(b21, b22)
    p7 = square_matrix_multiply_strassens(aTemp, bTemp)

    aTemp = sum_up(p1, p4)
    bTemp = sum_up(aTemp, p7)
    c11 = difference(bTemp, p5)
    c12 = sum_up(p3, p5)
    c21 = sum_up(p2, p4)

    aTemp = sum_up(p1, p3)
    bTemp = sum_up(aTemp, p6)
    c22 = difference(bTemp, p2)

    C = [[0 for i in range(0, n)] for j in range(0, n)]
    for i in range(0, new_n):
        for j in range(0, new_n):
            C[i][j] = c11[i][j]
            C[i][j + new_n] = c12[i][j]
            C[i + new_n][j] = c21[i][j]
            C[i + new_n][j + new_n] = c22[i][j]
    return C

    pass
```

Output

```
    print("Strassen's matrix multiplication")
    print (square_matrix_multiply_strassens(A, B))

    pass
```

```
A = asarray([[1, 3], [7, 5]])
B = asarray([[6, 8], [4, 2]])
```

Result:

```
Strassen's matrix multiplication
[[18, 14], [62, 66]]
```

FLAKE8: No warnings and complexity<10

Command Prompt

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\gu>flake8 C:\Users\gu\Desktop\assignment_2.py

C:\Users\gu>
```

Command Prompt

```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\gu>flake8 C:\Users\gu\Desktop\assignment_2.py

C:\Users\gu>flake8 --max-complexity 10 C:\Users\gu\Desktop\assignment_2.py

C:\Users\gu>
```

PYTHON CODE

```
# -*- coding: utf-8 -*-

from numpy import asarray


# TODO: Replace all TODO comments (yes, this one too!)
"""

STOCK_PRICE_CHANGES  = [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]

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


# 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)

"""

```
if len(A) == 0:
    return None

start = 0
end = len(A)
maxsum = -999999  # Setting the value to infinity

for i in range(start, end, 1):
    total = 0

    for j in range(i, end, 1):
        # add every element of the array to the variable sum
        total = total + A[j]

        if total > maxsum:        # check if sum is greater than max-sum
            maxsum = total
            start_index = i     # Assign the start index of the sub-array
            end_index = j       # Assign the last index of the sub-array
```

```
    return (start_index, end_index, maxsum)


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


# 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].
    """
    # By following the algorithm in text_book
    # To set the values to infinity
    # can also use decimal(-Infinity) after importing decimal

    temp_sum1 = 0
    i = mid
    left_ptr = i

    left_sum = -999999

    # To find the maximum sum at the left sub-array
    # Left array is from low to mid
```

```
while i >= low:
    temp_sum1 = temp_sum1 + A[i]
    if temp_sum1 > left_sum:
        left_sum = temp_sum1
        left_ptr = i
    i = i - 1

# To find the maximum sum at the left sub-array
# Right array is from mid+1 to high

temp_sum2 = 0
j = mid + 1

right_sum = -999999

right_ptr = j
while j <= high:
    temp_sum2 = temp_sum2 + A[j]
    if temp_sum2 > right_sum:
        right_sum = temp_sum2
        right_ptr = j
    j = j + 1
```

```
    return left_sum + right_sum, left_ptr, right_ptr


# 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].


    """
    # Base case -> when there is only 1 element
    # Algorithm in textbook is implemented

    if low == high:
        return A[low], low, high
    else:
        mid = (low + high) // 2
        left_sum, left_low, left_high = \
            find_maximum_subarray_recursive_helper(A, low, mid)
        right_sum, right_low, right_high = \
            find_maximum_subarray_recursive_helper(A, mid + 1, high)
        cross_sum, cross_low, cross_high = \
```

```
        find_maximum_crossing_subarray(A, low, mid, high)
    if left_sum >= right_sum and left_sum >= cross_sum:
        return left_sum, left_low, left_high
    elif right_sum >= left_sum and right_sum >= cross_sum:
        return right_sum, right_low, right_high
    else:
        return cross_sum, cross_low, cross_high
# 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.
    """
    # To check if the array is empty

    if len(A) == 0:
        return None

    return find_maximum_subarray_recursive_helper(A, 0, len(A) - 1)


# ============================================================
```

```
# The iterative method to solve max subarray problem
def find_maximum_subarray_iterative(A, low=0, high=-1):
    """

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

    """
    if len(A) == 0:
        return None

    low = 0
    high = len(A)
    totalSum = A[low]
    tempSum = 0
    tempLeftIndex = 0
    leftIndex = 0
    rightIndex = 0

    for i in range(low, high, 1):
        tempSum = max(A[i], (tempSum + A[i]))
        if tempSum == A[i]:
            tempLeftIndex = i
        if tempSum > totalSum:
            totalSum = tempSum
```

```
        rightIndex = i
        leftIndex = tempLeftIndex
    return (leftIndex, rightIndex, totalSum)

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


def square_matrix_multiply(A, B):
    """
    Return the product AB of matrix multiplication.
    """
    A = asarray(A)
    B = asarray(B)

    assert A.shape == B.shape
    assert A.shape == A.T.shape

    # Since it is a sqaure matrix , row = column = n

    n = len(A)
```

```
    C = [[0 for i in range(n)] for j in range(n)]
    for i in range(0, n):
        for j in range(0, n):
            C[i][j] = 0
            for k in range(0, n):
                C[i][j] = C[i][j] + A[i][k] * B[k][j]

    return C

# ============================================================
# Addition

def sum_up(A, B):
    n = len(A)
    result = [[0 for i in range(0, n)] for j in range(0, n)]
    for i in range(0, n):
        for j in range(0, n):
            result[i][j] = A[i][j] + B[i][j]
    return result

    # subtracts two matrices


def difference(A, B):
```

```
    n = len(A)
    result = [[0 for i in range(0, n)] for j in range(0, n)]
    for i in range(0, n):
        for j in range(0, n):
            result[i][j] = A[i][j] - B[i][j]
    return result



def square_matrix_multiply_strassens(A, B):
    A = asarray(A)
    B = asarray(B)

    assert A.shape == B.shape
    assert A.shape == A.T.shape

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

    n = len(A)

    if n == 1:
        C = [[0 for j in range(0, n)] for i in range(0, n)]
        for i in range(0, n):
            for j in range(0, n):
                C[i][j] = A[i][j] * B[i][j]
```

```
    return C
else:  # dividing the input matrices A and B
    new_n = int(n / 2)

a11 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
a12 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
a21 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
a22 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

b11 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
b12 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
b21 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
b22 = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

aTemp = [[0 for i in range(0, new_n)] for j in range(0, new_n)]
bTemp = [[0 for i in range(0, new_n)] for j in range(0, new_n)]

for i in range(0, new_n):
    for j in range(0, new_n):
        a11[i][j] = A[i][j]
        a12[i][j] = A[i][j + new_n]
        a21[i][j] = A[i + new_n][j]
        a22[i][j] = A[i + new_n][j + new_n]
```

```
        b11[i][j] = B[i][j]
        b12[i][j] = B[i][j + new_n]
        b21[i][j] = B[i + new_n][j]
        b22[i][j] = B[i + new_n][j + new_n]

aTemp = sum_up(a11, a22)
bTemp = sum_up(b11, b22)
p1 = square_matrix_multiply_strassens(aTemp, bTemp)

aTemp = sum_up(a21, a22)
p2 = square_matrix_multiply_strassens(aTemp, b11)

bTemp = difference(b12, b22)
p3 = square_matrix_multiply_strassens(a11, bTemp)

bTemp = difference(b21, b11)
p4 = square_matrix_multiply_strassens(a22, bTemp)

aTemp = sum_up(a11, a12)
p5 = square_matrix_multiply_strassens(aTemp, b22)

aTemp = difference(a21, a11)
bTemp = sum_up(b11, b12)
p6 = square_matrix_multiply_strassens(aTemp, bTemp)
```

```
aTemp = difference(a12, a22)
bTemp = sum_up(b21, b22)
p7 = square_matrix_multiply_strassens(aTemp, bTemp)

aTemp = sum_up(p1, p4)
bTemp = sum_up(aTemp, p7)
c11 = difference(bTemp, p5)
c12 = sum_up(p3, p5)
c21 = sum_up(p2, p4)

aTemp = sum_up(p1, p3)
bTemp = sum_up(aTemp, p6)
c22 = difference(bTemp, p2)

C = [[0 for i in range(0, n)] for j in range(0, n)]
for i in range(0, new_n):
   for j in range(0, new_n):
      C[i][j] = c11[i][j]
      C[i][j + new_n] = c12[i][j]
      C[i + new_n][j] = c21[i][j]
      C[i + new_n][j + new_n] = c22[i][j]
return C
```

```
    pass

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


def test():
    # TODO: Test all of the methods and print results.

    # calling the brute force method of max_subarray problem

    index1, index2, maxSum =
(find_maximum_subarray_brute(STOCK_PRICE_CHANGES))
    print
    print("Brute Force Method")
    print (index1, index2)

    finalSum, l_index, r_index = \
        find_maximum_subarray_recursive(STOCK_PRICE_CHANGES)
    print
    print("Recursive method")
    print ((l_index, r_index), (finalSum))

    print
    l, r, sum = find_maximum_subarray_iterative(STOCK_PRICE_CHANGES)
    print("Iterative method")
```

```
    print(l, r)
    print


    # Taking Matrix A and B from textbook


    A = asarray([[1, 3], [7, 5]])
    B = asarray([[6, 8], [4, 2]])
    print
    print("Normal square matrix multiplication")
    print(square_matrix_multiply(A, B))
    print
    print("Strassen's matrix multiplication")
    print (square_matrix_multiply_strassens(A, B))


    pass



if __name__ == '__main__':


    test()


    # ================================================================
```