PP01: Divide and Conquer

Name: Manohara Rao Penumala
ID: 1209455582

Python Code:

```
1.  # -*- coding: utf-8 -*-
2.  import numpy
3.  # STOCK_PRICES  = [100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97]
4.  STOCK_PRICE_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
5.
6.
7.  # Implement pseudo code from the book
8.  def find_maximum_sub_array_brute(A, low=0, high=-1):
9.      """
10.    Return a tuple (i,j) where A[i:j] is the maximum subarray.
11.    Implement the brute force method from chapter 4
12.    time complexity = O(n^2)
13.
14.    >>> STOCK_PRICE_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
15.    >>> find_maximum_sub_array_brute(STOCK_PRICE_CHANGES, 0, 15)
16.    (7, 10)
17.    """
18.     left_position = low
19.     right_position = low
20.     maximum = 0
21.     for i in range(0, high):
22.         current_max = 0
23.         for j in range(i, high):
24.             current_max = current_max + A[j]
25.             if maximum < current_max:
26.                 maximum = current_max
27.                 left_position = i
28.                 right_position = j
29.     return (left_position, right_position)
30.
31.
32. # Implement pseudocode from the book
33. def find_maximum_crossing_sub_array(A, low, mid, high):
34.     """
```

```
    Find the maximum subarray that crosses mid
    Return a tuple (i,j) where A[i:j] is the maximum subarray.

    >>> STOCK_PRICE_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
    >>> find_maximum_crossing_sub_array(STOCK_PRICE_CHANGES, 0, 7, 15)
    (7, 10, 43)
    """
     left_max = 0
     maximum = 0
     left_position = mid
     for i in range(mid-1, low-1, -1):
         maximum = maximum + A[i]
         if left_max < maximum:
             left_max = maximum
             left_position = i
     right_max = 0
     maximum = 0
     right_position = mid
     for j in range(mid, high):
         maximum = maximum + A[j]
         if right_max < maximum:
             right_max = maximum
             right_position = j
     return (left_position, right_position, left_max+right_max)


def find_maximum_sub_array_recursive(A, low=0, high=-1):
     """
    Return a tuple (i,j) where A[i:j] is the maximum subarray.
    Recursive method from chapter 4

    >>> STOCK_PRICE_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
    >>> find_maximum_sub_array_recursive(STOCK_PRICE_CHANGES, 0, 15)
    (7, 10, 43)
    """
     if high == low:
         return (low, high, A[0])
     else:
         mid = ((low + high)/2)
         left_tuple = find_maximum_sub_array_recursive(A, low, mid)
         right_tuple = find_maximum_sub_array_recursive(A, mid+1, high)
```

```
        cross_tuple = find_maximum_crossing_sub_array(A, low, mid, high)
        if left_tuple[2] >= right_tuple[2] and left_tuple[2] >= cross_tuple[2]:
            return (left_tuple[0], left_tuple[1], left_tuple[2])
        elif right_tuple[2] >= left_tuple[2] and right_tuple[2] >= cross_tuple[2]:
            return (right_tuple[0], right_tuple[1], right_tuple[2])
        else:
            return (cross_tuple[0], cross_tuple[1], cross_tuple[2])


def find_maximum_sub_array_iterative(A, low=0, high=-1):
    """
   Return a tuple (i,j) where A[i:j] is the maximum subarray.
   Do problem 4.1-5 from the book.
   Assuming that at least one of the input values will be positive.

   >>> STOCK_PRICE_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
   >>> find_maximum_sub_array_iterative(STOCK_PRICE_CHANGES, 0, 15)
   (7, 10)
   """
    left_position = 0
    right_position = 0
    current_position = 0
    positive_element_exists = 0
    maximum = A[low]
            S = [0]*len(A)
            if A[low] > 0:
                S[0] = A[low]
                positive_element_exists = 1
            for i in range(low+1, high):
                if A[i] > 0:
                    positive_element_exists = 1
                S[i] = S[i-1] + A[i]
                if S[i] > S[i-1] and S[i-1] <= 0:
                        current_position = i
                if S[i] < 0:
                    S[i] = 0
                if S[i] > maximum:
                    maximum = S[i]
                    left_position = current_position
                    right_position = i
            if positive_element_exists == 1:
```

```
            return (left_position, right_position)
        else:
            return (0, 0)


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

       >>> A = [[36, 54, 24, 38], [54, 50, 19, 68], [26, 79, 57, 49], [94, 59, 20, 97]]
       >>> B = [[46, 68, 27, 38], [57, 94, 74, 20], [46, 0, 52, 69], [20, 65, 37, 26]]
       >>> square_matrix_multiply(A, B)
       array([[  6598.,   9994.,   7622.,   5092.],
              [  7568.,  12792.,   8662.,   6131.],
              [  9301.,  12379.,  11325.,   7775.],
              [ 10547.,  18243.,  11533.,   8654.]])
       """
        A = numpy.asarray(A)
        B = numpy.asarray(B)
        assert A.shape == B.shape
        assert A.shape == A.T.shape
        dimensions = A.shape
        C = numpy.zeros(dimensions)
        for i in range(0, dimensions[0]):
            for j in range(0, dimensions[0]):
                for k in range(0, dimensions[0]):
                    C[i][j] = C[i][j] + (A[i][k]*B[k][j])
        return C


    def square_matrix_multiply_strassens(A, B):
        """
       Return the product AB of matrix multiplication.
       Assume len(A) is a power of 2

       >>> A = [[36, 54, 24, 38], [54, 50, 19, 68], [26, 79, 57, 49], [94, 59, 20, 97]]
       >>> B = [[46, 68, 27, 38], [57, 94, 74, 20], [46, 0, 52, 69], [20, 65, 37, 26]]
       >>> square_matrix_multiply(A, B)
       array([[  6598.,   9994.,   7622.,   5092.],
              [  7568.,  12792.,   8662.,   6131.],
              [  9301.,  12379.,  11325.,   7775.],
```

```
                [ 10547.,  18243.,  11533.,   8654.]])
        """
        A = numpy.asarray(A)
        B = numpy.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"
        dimensions = A.shape
        C = numpy.zeros(shape=(dimensions[0],dimensions[0]))
        if dimensions[0] == 1:
            C[0][0] = A[0][0] * B[0][0]
        else:
            # Partition the given 2 matrices
            partition = dimensions[0]/2
            A11 = A[:partition, :partition]
            A12 = A[:partition, partition:]
            A21 = A[partition:, :partition]
            A22 = A[partition:, partition:]
            B11 = B[:partition, :partition]
            B12 = B[:partition, partition:]
            B21 = B[partition:, :partition]
            B22 = B[partition:, partition:]

            # Evaluate P
            P1 = square_matrix_multiply_strassens(A11, B12 - B22)
            P2 = square_matrix_multiply_strassens(A11 + A12, B22)
            P3 = square_matrix_multiply_strassens(A21 + A22, B11)
            P4 = square_matrix_multiply_strassens(A22, B21 - B11)
            P5 = square_matrix_multiply_strassens(A11 + A22, B11 + B22)
            P6 = square_matrix_multiply_strassens(A12 - A22, B21 + B22)
            P7 = square_matrix_multiply_strassens(A11 - A21, B11 + B12)

            # Evaluate the product matrix
            C[:partition, :partition] = P5 + P4 - P2 + P6
            C[:partition, partition:] = P1 + P2
            C[partition:, :partition] = P3 + P4
            C[partition:, partition:] = P1 + P5 - P3 - P7

        return C
        pass

```

```
199.
200.        def test():
201.            C = [numpy.random.randint(-99, 99)]*1
202.            array_length = numpy.random.randint(1, 20)
203.            for x in range(1, array_length):
204.                C.append(numpy.random.randint(-99, 99))
205.
206.            brute_force_sub_array = find_maximum_sub_array_brute(C, 0, len(C)-1)
207.            crossing_sub_array = find_maximum_crossing_sub_array(C, 0, (len(C)-1)/2, len(C)-1)
208.            recursive_sub_array = find_maximum_sub_array_recursive(C, 0, len(C)-1)
209.            iterative_sub_array = find_maximum_sub_array_iterative(C, 0, len(C)-1)
210.            print(C)
211.            print(brute_force_sub_array)
212.            print(crossing_sub_array)
213.            print(recursive_sub_array)
214.            print(iterative_sub_array)
215.
216.            matrix_size = 2**numpy.random.randint(1, 3)
217.            A = numpy.random.randint(99, size=(matrix_size, matrix_size))
218.            B = numpy.random.randint(99, size=(matrix_size, matrix_size))
219.            square_matrix = square_matrix_multiply(A, B)
220.            strassens_matrix = square_matrix_multiply_strassens(A, B)
221.            print(A)
222.            print(B)
223.            print(A.dot(B))
224.            print(square_matrix)
225.            print(strassens_matrix)
226.
227.            pass
228.
229.
230.        if __name__ == '__main__':
231.            test()
```

Output for one of the random inputs:

```
C:\Python27\python.exe "C:/Users/MadhulikaBushi/Desktop/501/Coding Assignments/Coding Assignment 1/dc_algorithms.py"
[-72, 27, -25, 8, -78, 77, -99, -96, 37, -62]
(5, 5)
(1, 4, 10)
(5, 5, 77)
(5, 5)
[[96 90 50  6]
 [74 14 58 37]
 [25 61 60 47]
 [82 46 81 52]]
[[20 85 94 51]
 [15 73 78 18]
 [ 3 97 45 38]
 [44 71 35 66]]
[[ 3684 20006 18504  8812]
 [ 3492 15565 11953  8672]
 [ 3663 15735 11453  7755]
 [ 4861 21877 16761 11520]]
[[  3684.  20006.  18504.   8812.]
 [  3492.  15565.  11953.   8672.]
 [  3663.  15735.  11453.   7755.]
 [  4861.  21877.  16761.  11520.]]
[[  3684.  20006.  18504.   8812.]
 [  3492.  15565.  11953.   8672.]
 [  3663.  15735.  11453.   7755.]
 [  4861.  21877.  16761.  11520.]]

Process finished with exit code 0
```

Doctest and flake8 outputs:

Input used for doctests:

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

Expected Output:

```
(7, 10)
```

Doctest passed for all the functions as seen in below screenshot.

```
2 items had no tests:
    dc_algorithms
    dc_algorithms.test
6 items passed all tests:
   2 tests in dc_algorithms.find_maximum_crossing_sub_array
   2 tests in dc_algorithms.find_maximum_sub_array_brute
   2 tests in dc_algorithms.find_maximum_sub_array_iterative
   2 tests in dc_algorithms.find_maximum_sub_array_recursive
   3 tests in dc_algorithms.square_matrix_multiply
   3 tests in dc_algorithms.square_matrix_multiply_strassens
14 tests in 8 items.
14 passed and 0 failed.
Test passed.
```

Output of flake8 (complexity 9):

```
C:\Users\MadhulikaBushi\Desktop\501\Coding Assignments\Coding Assignment 1>flake8 --max-complexity 9 dc_algorithms.py

C:\Users\MadhulikaBushi\Desktop\501\Coding Assignments\Coding Assignment 1>
```