Abdul Samad Khan

akhan51@asu.edu

SER-501-A2

Q1) a) max subarray by Brute force

```
def find_maximum_subarray_brute(A):
    """
    Return a tuple (i,j) where A[i:j] is the maximum subarray.
    time complexity = O(n^2)
    """
    max = 0
    start = 0
    end = 0
    if (len(A) == 0):
        return None
    # loop running till the length of the array
    for i in range(0, len(A)):
        sum = 0
        j = i
        sum = sum + A[i]
        # loop for taking in account the size of array we are considering
        for j in range(j, len(A)):
            sum = sum + A[j]
            if sum > max:
                max = sum
                start = i
                end = j
    tup = (start, end)
    return tup
#=============================
```

Q1) b) maximum crossing subarray

```
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].
    """
    max_left = 0
    max_right = 0
    left_sum = 0
    sum = 0
    i = mid
```

```
    # taking sum of left side
    while i >= low:
        sum = sum + A[i]
        if (sum > left_sum):
            left_sum = sum
            max_left = i
        i = i - 1
    j = mid + 1
    right_sum = 0
    sum = 0
    # taking sum of right side
    while (j <= high):
        sum = sum + A[j]
        if (sum > right_sum):
            right_sum = sum
            max_right = j
        j = j + 1
    # returning sum of both sides
    tup = (max_left, max_right)
    tup1 = (tup, left_sum + right_sum)
    return tup1
# 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
    if (low == high):
        tup = (low, high)
        return (tup, A[low])
    # finding midpoint of the array
    mid = ((low + high) // 2)
    # calling function for left array
    ((left_low, left_high),
     left_sum) = find_maximum_subarray_recursive_helper(A, low, mid)
    # calling function for right array
    ((right_low, right_high),
     right_sum) = find_maximum_subarray_recursive_helper(A, mid + 1, high)
    # calling function for crossing array
    ((cross_low, cross_high),
     cross_sum) = find_maximum_crossing_subarray(A, low, mid, high)
    print(cross_sum)
    # if the sum of left array is greater
    if (left_sum >= right_sum) and (left_sum >= cross_sum):
        tup_left = (left_low, left_high)
```

```
        tup_left1 = (tup_left, left_sum)
        return (tup_left1)
    # if the sum of right array is greater
    elif (right_sum >= left_sum) and (right_sum >= cross_sum):
        tup_right = (right_low, right_high)
        tup_right1 = (tup_right, right_sum)
        return (tup_right1)
    # if the sum of crossing array is greater
    else:
        tup_cross = (cross_low, cross_high)
        tup_cross1 = (tup_cross, cross_sum)
        return (tup_cross1)
# 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.
    """
    if (len(A) == 0):
        return None
    # if the array is not of length of power2 we apply padding to left
    A = PadLeft(A)
    return find_maximum_subarray_recursive_helper(A, 0, len(A) - 1)[0]


def PadLeft(A):
    # checking the closest power of 2 to the len(A)
    nextPower = NextPowerOfTwo(len(A))
    deficit = int(math.pow(2, nextPower) - len(A))
    # concatenate 0 with the original array
    for x in range(0, deficit):
        A = np.concatenate((A, [0]))
    return A


def NextPowerOfTwo(number):
    # Returns next power of two following 'number'
    return math.ceil(math.log(number, 2))
#===========================
```

Q1) c) max subarray by iterative method

```
def find_maximum_subarray_iterative(A):

    """
```

```
    Return a tuple (i,j) where A[i:j] is the maximum subarray.
    """
    arr_sum = 0
    arr_val = 0
    left = -1
    right = -1
    left1 = 0
    if (len(A) == 0):
        return None
    # loop runing till the length of A
    for i in range(0, (len(A))):
        # keeping sum after every index
        arr_sum = arr_sum + A[i]
        # comparing two sums
        if (arr_sum > arr_val):
            arr_val = arr_sum
            left = left1
            right = i
        # if arr_sum get less then 0 we reset arr_sum to zero
        if arr_sum < 0:
            arr_sum = 0
            left1 = i + 1
    tup = (left, right)
    return tup
#=============================
```

Q2) a) 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
    assert A.shape == A.T.shape
    n = len(A)
    # intializing an n*n matrix c
    C = [[0 for i in range(n)] for j in range(n)]
    # assigning values to c
    for i in range(n):
        for j in range(n):
            for k in range(n):
                C[i][j] = C[i][j] + (A[i][k] * B[k][j])
    C = np.array(C)
    return C

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

Q2) b) matrix multiplication by Strassen method

```
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
    assert A.shape == A.T.shape
    assert (len(A) & (len(A) - 1)) == 0, " A is not a power of 2 "
    # base case
    if len(A) == 1:
        base = [[0]]
        base[0][0] = A[0][0] * B[0][0]
        return base
    else:
        # spliting matrices into 4
        A11, A12, A21, A22 = split(A)
        B11, B12, B21, B22 = split(B)
        # Finding the values of M to be applied
        # M1=(A11+A22)*(B11+B22)
        M1 = square_matrix_multiply_strassens(np.add(A11, A22),
                                              np.add(B11, B22))
        # M2=(A21+A22)*(B11)
        M2 = square_matrix_multiply_strassens(np.add(A21,
                                                  A22), B11)
        # M3=(A11)*(B12-B22)
        M3 = square_matrix_multiply_strassens(A11,
                                              np.subtract(B12, B22))
        # M4=(A22)*(B21-B11)
        M4 = square_matrix_multiply_strassens(A22, np.subtract(B21, B11))
        # M5=(A11+A12)*(B22)
        M5 = square_matrix_multiply_strassens(np.add(A11, A12), B22)
        # M6=(A21-A11)*(B11+B12)
        M6 = square_matrix_multiply_strassens(np.subtract
                                              (A21, A11), np.add(B11, B12))
        # M7=(A12-A22)*(B21+B22)
        M7 = square_matrix_multiply_strassens(np.subtract
                                              (A12, A22), np.add(B21, B22))
        # assingning values of the above to the final matrix
        X11 = np.add(np.subtract(np.add(M1, M4), M5), M7)
        X12 = np.add(M3, M5)
        X21 = np.add(M2, M4)
        X22 = np.add(np.subtract(np.add(M1, M3), M2), M6)
        c = [[0 for row in range(len(X11) * 2)] for col in range(len(X11) * 2)]
```

```
        for i in range(len(X11)):
            for j in range(len(X11)):
                c[i][j] = X11[i][j]
                c[i][j + len(X11)] = X12[i][j]
                c[i + len(X11)][j] = X21[i][j]
                c[i + len(X11)][j + len(X11)] = X22[i][j]
        c = np.array(c)
        return c
    pass

def split(matrix):
    # split matrix into quarters
    m = len(matrix) // 2
    a = [[0 for i in range(m)] for j in range(m)]
    b = [[0 for i in range(m)] for j in range(m)]
    c = [[0 for i in range(m)] for j in range(m)]
    d = [[0 for i in range(m)] for j in range(m)]
    for i in range(0, m):
        for j in range(0, m):
            a[i][j] = matrix[i][j]
            b[i][j] = matrix[i][m + j]
            c[i][j] = matrix[i + m][j]
            d[i][j] = matrix[i + m][m + j]
            j = j + 1
        j = j + 1
    return a, b, c, d



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

Output

```
In [42]: runfile('I:/MS/SER501/Assignment2/assignment_2.py', wdir='I:/MS/SER501/Assignment2')
Q1a) maximum subarray by brute force
(7, 10)
Q1b) maximum subarray by recursive method
(7, 10)
Q1c) maximum subarray by iterative method
(7, 10)
Q2a) matrix multiplication
[[ 96  68  69  69]
 [ 24  56  18  52]
 [ 58  95  71  92]
 [ 90 107  81 142]]
Q2b) matrix multiplication by strassen method
[[ 96  68  69  69]
 [ 24  56  18  52]
 [ 58  95  71  92]
 [ 90 107  81 142]]


In [43]:
```

Results after running flake8 and McCabe complexity command