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

from numpy import asarray

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

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

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


# The brute force method to solve first problem
def find_significant_energy_increase_brute(A):

    """
    Return a tuple (i,j) where A[i:j] is the most significant
    energy increase period.
    time complexity = O(n^2)
    """
    # TODO
    print("Question 1 A")
    maximum = -99999
    start = 0
    last = 0
    for x in range(len(A)):
        for y in range(x+1, len(A)):
            if maximum < A[y]-A[x]:

                maximum = A[y]-A[x]
                start = x
                last = y
    print((start, last))
    return(start, last)

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


# The recursive method to solve first problem
def find_significant_energy_increase_recursive(A):

    """
    Return a tuple (i,j) where A[i:j]
    is the most significant energy increase period.
    time complexity = O (n logn)
    """
    # TODO
    print("Question 1 B")
    x = max_diff(A, 0, len(A)-1)
    print((x[1], x[2]))
    return (x[1], x[2])


def max_diff(A, low, high):
    if((low == high) | (low == high-1)):
        return [A[high] - A[low], low, high]
    else:
        mid = (low+high)//2
        left = max_diff(A, low, mid)
```

```
        right = max_diff(A, mid+1, high)
        cross = max_diff_cross(A, low, mid, high)
        mx = max(left, right, cross)
        if mx == left:
            return left
        elif mx == right:
            return right
        else:
            return cross


def max_diff_cross(A, low, mid, high):
    d = 0
    lm = -99999
    rm = -99999
    lp = low
    rp = high

    for i in range(low, mid):
        d = A[mid]-A[i]
        if d > lm:
            lm = d
            lp = i

    for i in range(mid+1, high):
        d = A[i] - A[mid+1]
        if d > rm:
            rm = d
            rp = i
    return [lm+rm, lp, rp]


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

# The iterative method to solve first problem
def find_significant_energy_increase_iterative(A):

    """
    Return a tuple (i,j) where A[i:j]
    is the most significant energy increase period.
    time complexity = O(n)
    """
    # TODO
    print("Question 1 C")
    maximum = A[1] - A[0] - 1
    la = A[0]
    s = 0
    e = 0

    for x in range(len(A)):
        if(maximum < A[x] - la):
            maximum = A[x] - la
            s = A.index(la)
            e = x
        if A[x] < la:
            la = A[x]
    print((s, e))
    return(s, e)
```

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


# The Strassen Algorithm to do the matrix multiplication
def square_matrix_multiply_strassens(A, B):

    """
    Return the product AB of matrix multiplication.
    Assume len(A) is a power of 2
    """
    print("Question 2 A")
    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"

    # TODO
    m1 = (A[0][0] + A[1][1]) * (B[0][0] + B[1][1])
    m2 = (A[1][0] + A[1][1]) * B[0][0]
    m3 = A[0][0] * (B[0][1] - B[1][1])
    m4 = A[1][1] * (B[1][0] - B[0][0])
    m5 = (A[0][0] + A[0][1]) * B[1][1]
    m6 = (A[1][0] - A[0][0]) * (B[0][0] + B[0][1])
    m7 = (A[0][1] - A[1][1]) * (B[1][0] + B[1][1])
    c = [[m1 + m4 - m5 + m7, m3 + m5], [m2 + m4, m1 - m2 + m3 + m6]]
    print(c)
    return c


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

# Calculate the power of a matrix in O(k)
def power_of_matrix_navie(A, k):
    """
    Return A^k.
    time complexity = O(k)
    """
    # TODO
    print("Question 2 B")
    s = []
    mat = A
    for i in range(0, k - 1):
        if s == []:
            s = square_matrix_multiply_strassens(mat, mat)
        else:
            s = square_matrix_multiply_strassens(s, mat)
    print(s)
    return s
# ============================================================


# Calculate the power of a matrix in O(log k)
def power_of_matrix_divide_and_conquer(A, k):
    """
```

```
    Return A^k.
    time complexity = O(log k)
    """
    # TODO
    print("Question 2 C")
    if k == 2:
        return square_matrix_multiply_strassens(A, A)
    elif k == 1:
        return A
    else:
        k1 = k // 2
        x = power_of_matrix_divide_and_conquer(A, k1)
        y = power_of_matrix_divide_and_conquer(A, k1+1)
        if k % 2 == 0:
            return square_matrix_multiply_strassens(x, x)
        else:
            return square_matrix_multiply_strassens(x, y)

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


def test():

    assert(find_significant_energy_increase_brute(ENERGY_LEVEL) == (7, 11))
    assert(find_significant_energy_increase_recursive(ENERGY_LEVEL) == (7, 11))
    assert(find_significant_energy_increase_iterative(ENERGY_LEVEL) == (7, 11))
    assert((square_matrix_multiply_strassens([[0, 1], [1, 1]],
                                             [[0, 1], [1, 1]]) ==
           asarray([[1, 1], [1, 2]])).all())
    assert((power_of_matrix_navie([[0, 1], [1, 1]], 3) ==
           asarray([[1, 2], [2, 3]])).all())
    assert((power_of_matrix_divide_and_conquer([[0, 1], [1, 1]], 3) ==
           asarray([[1, 2], [2, 3]])).all())
    # TODO: Test all of the methods and print results.


if __name__ == '__main__':

    test()

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