Homework 5:

Q1:

Maximum mixed-up-ness score occurs when the array is sorted in reverse order. Hence, we can calculate mixed-up-ness score by using the linear series formula: **(n*(n-1)) / 2**

If n = 16

(16*(15))/2 = 120

Hence, maximum mixed-up-ness score of an array of size 16 will be 120.

Q2:

The worst-case runtime of brute-force algorithm will be **O(n^2).** This is because 2 loops are used. We use these 2 loops to iterate every element in the array and check if they can be swapped. Here is the convincing argument for worst case scenario.

1. In the brute force algorithm, we use 2 loops to iterate through the array. The outer loop iterates through each element of array. And the inner loop compares the rest of elements with the current item of arrays.
2. The worst-case scenario is that the array is in reverse order. In this case, we will have a maximum number of inversions. Due to which, maximum number of comparisons occur, and they have been calculated before.
3. The nested loop results in (n*(n-1))/2 comparisons, in worst case. It grows quadratically with the size of array.
4. Asymptotically, the quadratic term dominates the runtime, resulting in a worst-case time complexity of O(n^2).

Q3:

The recurrence for the merge-sort in part 3 will be:

T(n) = 2 * T(n/2) + O(n)

This is because this algorithm is a recursive algorithm, and at each call the array is divided into 2 parts. Whereas the O(n) indicates that it takes O(n) complexity to combine the array or merge it.

Q4:

We need to establish the base case for n = 1. For n = 1;

T(1) = 2 * T(1/2) + O(1).

Here, T(1/2) = T(1) = 1.

So: T(1) = 2 * 1 + O(1) = 2 + O(1) = O(1)

Now Proving:

T(n) = 2 * [c * (n/2) * log(n/2)] + O(n) T(n)

= c * n * log(n/2) + O(n)

T(n) = c * n * (log(n) - log(2)) + O(n) T(n)

= c * n * (log(n) - 1) + O(n) T(n)

= c * n * log(n) - c * n + O(n)

= c * n * log(n) + O(n) - c * n T(n)

= c * n * log(n) + O(n * (1 - c))

T(n) = c * n * log(n) + O(n * (1 - c))

In practice, we typically ignore constant factors and lower-order terms when analyzing algorithmic time complexity. Therefore, we can approximate the recurrence as:

T(n) = O(n * log(n))

Q5:
As **T(n) = 2 * T(n/2) + O(n)**

a = 2 , b = 2 , k = 1 , p = 0

Here: $a = b^k$ (case 2)

ii) p > -1

$T(n) = O(n^{logba} \log^{p+1} n)$

$T(n) = O(n^{\log_2 2} \log^{0+1} n)$ … $\log_2 2 = 1$

T(n) = O(n log n)

Hence, it's proved that recurrence in part 4 is same with master theorem.