1. Given an array of size 16, what is the maximum possible mixed-up-ness score? Explain why you think this (e.g., give a logical argument or provide an example)

Answer:
To answer this question, we first need to define what "mixed-up-ness score" means in the context of an array. Since this term is not standard in computer science or mathematics, I will assume it refers to some measure of how "mixed up" or "disordered" the elements of the array are.

One common measure of disorder in an array is the number of inversions, which is the number of pairs of elements that are out of order. For example, in the array [1, 3, 2, 4], there is one inversion (3, 2). Another measure could be the sum of the absolute differences between adjacent elements, or some other metric.

However, if we are looking for the "maximum possible mixed-up-ness score," we might interpret this as the maximum number of inversions possible in an array of size 16. In this case, the maximum mixed-up-ness score would occur when the array is sorted in reverse order, as this would create the maximum number of inversions.

For example, consider the array [16, 15, 14, ..., 1]. In this array, every pair of adjacent elements is an inversion, and there are a total of 15 + 14 + 13 + ... + 1 = 120 inversions. This is the maximum possible number of inversions for an array of size 16, and hence it represents the maximum possible mixed-up-ness score according to this metric.

If "mixed-up-ness" is measured differently, the answer would change accordingly. For example, if it were based on the sum of absolute differences between adjacent elements, the maximum score would occur when the array alternates between the maximum and minimum values, like [16, 1, 15, 2, 14, 3, ..., 8, 9]. In this case, the score would be the sum of the differences between adjacent elements, which would be much higher than in the previous example.

Without a specific definition of "mixed-up-ness," it's difficult to give a precise answer. However, if we interpret it as a measure of disorder or inversions, then the maximum possible score would occur when the array is sorted in reverse order or arranged in some other highly disordered way.

2. What is the worst-case runtime of the brute-force algorithm that you implemented? Give a proof (a convincing argument) of this.

Answer:
To provide a worst-case runtime analysis for a brute-force algorithm, we need to specify which algorithm we are referring to. Brute-force algorithms can vary depending on the problem they are trying to solve. However, I will provide a general analysis that can be applied to many brute-force algorithms.

Let's assume we have a brute-force algorithm that solves a problem by examining all possible solutions and checking if they satisfy the given conditions. The worst-case runtime of such an algorithm would occur when it has to examine all possible solutions before finding the correct one (or determining that no solution exists).

In the worst case, the algorithm would need to perform the following steps:

Generate all possible solutions.
Check each solution individually to determine if it satisfies the conditions.
Return the correct solution if found; otherwise, indicate that no solution exists.
The runtime of this algorithm would depend on the number of possible solutions and the time it takes to check each solution. Let's denote the number of possible solutions as (N) and the time it takes to check each solution as (T).

In the worst case, the algorithm would have to generate all (N) possible solutions and check each one individually. Therefore, the worst-case runtime would be (O(N \times T)), where (O) denotes the big-O notation used to describe the asymptotic behavior of algorithms.

To prove this, consider that in the worst case, the algorithm cannot terminate early because it has not found the correct solution (or determined that no solution exists) until it has checked all possible solutions. Therefore, it must perform (N) checks, each taking (T) time, resulting in a total runtime of (N \times T). Since we are interested in the asymptotic behavior, we use big-O notation to ignore constant factors and lower-order terms, resulting in a worst-case runtime of (O(N \times T)).

It's important to note that this analysis assumes that generating all possible solutions and checking each one takes constant time. In practice, the time it takes to generate or check solutions may vary depending on the problem and the specific implementation of the algorithm. Therefore, a more detailed analysis would be needed to account for these factors accurately. However, the general argument provided here gives a good intuition for understanding the worst-case runtime of brute-force algorithms.