Written Exercise for HW5 Mixed-up-ness

Name: Rong Huang
Date:02/27/2023

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)

To determine the maximum possible mixed-up-ness score for an array of size 16, we need to consider the scenario where every possible pair of elements is in reverse order, maximizing the mixed-up-ness. The mixed-up-ness score is essentially the number of inversions in the array, where an inversion is defined as a pair of elements $a[i]$ and $a[j]$ such that $i < j$ but $a[i] > a[j]$.

For an array of size n, the maximum number of inversions occurs when the array is sorted in descending order. This is because every element will be larger than all elements that come after it, contributing to the inversion count. The total number of inversions, in this case, is given by the formula for the sum of the first $n-1$ positive integers, which is $\frac{n(n-1)}{2}$.

For an array of size 16, the maximum mixed-up-ness score can be calculated as:

$$\text{Maximum Score} = \frac{16 \times (16-1)}{2} = \frac{16 \times 15}{2} = 120.$$

This score represents the total number of pairs in reverse order when the array is in the worst possible order (e.g., fully reversed from the sorted order), making it the maximum possible mixed-up-ness score for an array of size 16.

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

The worst-case runtime of the brute-force algorithm for calculating the mixed-up-ness score of an array is $O(n^2)$, where n is the number of elements in the array.

This quadratic time complexity arises because the algorithm iteratively compares each element with every other element to count inversions. Specifically, for each element in the array, the algorithm performs a comparison with all subsequent elements to determine if an inversion exists.

Given an array of length n, the first element requires $n-1$ comparisons (with all subsequent elements), the second element requires $n-2$ comparisons (since it doesn't compare with the first element again), and so on, until the second-to-last element, which requires just 1 comparison. The last element does not require any comparison as all inversions involving it would have already been counted.

The total number of comparisons C required by this algorithm can be calculated as:

$$C = (n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2}$$

This formula represents the sum of the first $n-1$ positive integers, which simplifies to $\frac{n(n-1)}{2}$. This expression is in $O(n^2)$ because, in big O notation, we focus on the highest order term and ignore constants and lower order terms. The n^2 term dominates the growth of the runtime as n increases, leading to the conclusion that the algorithm has a quadratic runtime in the worst case.