Third laboratory.

Iván Alejandro Cruz Tole.

08.10.2018

Let A be a permutation of n different numbers without any inversion (i.e. Sorted) and B be a permutation such as A but with as many inversions as possible (i.e. Reversely sorted, thus having $\frac{n(n-1)}{2}$ inversions), and let us measure complexity by counting the worst-case number of comparisons made between numbers of the permutation.

1 Complexity of the brute force counting on A and B.

The brute force approach does as follows for any number in the ith index of the permutation: Traverse the permutation, one by one, and from the $(i+1)$th index to the last one, adding up the number of inversions found with respect to the number in the ith index. This, in turn, means that we'll execute n^2 comparisons each time we run the brute force approach - Thus, no matter how many inversions the permutation has, it will always take the same number of comparisons for an n sized permutation.

2 Complexity of the divide and conquer counting on A and B.

The divide and conquer approach is built upon the merge sort algorithm in such way that the former takes the exact same number of comparisons as the latter, thus having a total of $n \log_2 n$ comparisons. Once again, this is invariable: It doesn't matter how the permutation's elements are arranged, but how many they are.

3 Time counts (in seconds) for the brute force and divide and conquer approaches in Python.

Brute force approach:

- HackerEarth case: 703.353.

- Increasingly sorted case: 602.012.
- Decreasingly sorted case: 771.211.

Merge sort approach:

- HackerEarth case: 1.311.
- Increasingly sorted case: 1.089.
- Decreasingly sorted case: 1.281.

4 Time counts (in seconds) for the brute force and divide and conquer approaches in C.

Brute force approach:

- HackerEarth case: 26.2.
- Increasingly sorted case: 12.1.
- Decreasingly sorted case: 16.0.

Merge sort approach:

- HackerEarth case: 0.3.
- Increasingly sorted case: 0.1.
- Decreasingly sorted case: 0.2.