Senate Building Decoration

Mikhail Pyaderkin

Setup

Two-player game. Given an array, the players repeatedly remove **everything but a subarray at least half the length**, until only one element is left.

The first player wants to **maximize** the final number, while the second player wants to **minimize** it.

Subtask: A is nondecreasing

Clearly, out of all subarrays of a given length L, the first player will want to choose the rightmost one and the second player the leftmost one.

Also, it is never useful to give out a subarray that is longer than needed: the opponent is always able to give back a new array that contains smaller numbers than if we gave them the shortest array.

Thus, we can repeatedly cut off the left or right halves to find the last remaining position. This can be easily implemented in $n \log n$ time.

Subtask: $n \leq 250$ (n^4)

We can do **dynamic programming** with $O(n^2)$ states: for each subarray, remember what's the highest value you would get if player 1 started from that subarray, and the lowest value if player 2 started from that subarray.

In $O(n^2)$, for every state we can check the corresponding value for the other player for all its subarrays, for a total cost of $O(n^4)$

Subtask: $n \leq 800$ (n^3)

We can still do the same dynamic programming solution, but we can compute the minimum/ maximum value for each length of subarray in $O(n)$ with a Range Minimum Query structure.

This gives a total cost of $O(n^3)$.

Full solution ($n^2 \log n$)

Let's **binary search** on the final value in the array. We now need to figure out whether the final value is $\geq$ or $<$ of the target.

This is equivalent to solving the problem on a A array that is either 0 or 1.

To solve this special case, we just need to keep track of the **number** of 0s (or 1s) answers you can get to from a given configuration.

If we process sub-arrays in order of length, we can update these numbers when going from length l to length $l + 1$ for **all** sub-arrays of that length in $O(n)$.

This makes the total cost $O(n^2 \log n)$