Kick Start 2020 - Round A

Analysis: Workout

Test set 1

Since **K**=1, all that we need to do is to find the maximum difference and split it into 2 halves. For example, given a sequence [2, 12, 18] and **K** = 1, the *difficulty* is 10, since the maximum difference is in [2, 12]. The best way to minimize this is to take the maximum difference and split it in half giving us the final sequence of [2, 7, 12, 18]. The *difficulty* for this final sequence now is 6. The time complexity is O(**N**).

Test set 2

For this test case, we cannot perform such direct splits because repeatedly splitting the maximum difference into halves is not optimal. For example, given a sequence [2, 12] and **K** = 2, splitting into halves will result in [2, 12] → [2, 7, 12] → [2, 7, 9, 12]. This way, the *difficulty* would be 5. However, if we perform [2, 12] → [2, 5, 12] → [2, 5, 8, 12], the *difficulty* would be 4. This clearly demonstrates that continuous halving of the maximum difference is sub-optimal. Okay, so how do we do this?

Consider the i-th adjacent pair of training sessions with an initial difference d_i. If we want to insert some number of training sessions in between this pair such that the maximum difference among those is at most a certain value, let's say $d_{optimal}$, then *how many training sessions can be inserted in between?* The answer to this is *ceiling(d_i / $d_{optimal}$)-1*. Let's call that k'_i. Doing this for all **N**-1 adjacent pairs in the given array would give us k'[1, ..., **N**-1]. Let's denote k'_{sum} = k'_1+k'_2+ ....+$k'_{\mathbf{N}-1}$. From the constraints, we can insert at most **K** training sessions. Therefore, we need to make sure $k'_{sum} \le$ **K** while minimizing $d_{optimal}$ as much as possible.

If you observe, $d_{optimal}$ can lie anywhere between [1, max(d_i)] (1 ≤ i ≤ **N**-1). Linear search would be to check every value here starting from 1 and output the first value that satisfies the above condition. A quicker way to do this is using binary search. On closer observation, you can see that increasing the value of $d_{optimal}$ decreases the value of *ceiling(d_i / $d_{optimal}$)-1* and hence smaller is the value of k'_{sum}. Therefore, we can perform a binary search in the range [1, max(d_i)] to find the least value of $d_{optimal}$ that makes $k'_{sum} \le$ **K**. That is our answer.

Since the max(d_i) could be as much as 10^9, we might have to search [1, 10^9] making time complexity of the solution is O(log(10^9)*$\mathbf{N}$).