Range Equals Size

First, note that the order of elements is unimportant. Hence, we can sort the input to make it easier to work with. With the input now sorted, let's consider a simpler version of the problem: one in which the values are pairwise *distinct*.

In the simpler version, the difference between maximum and minimum values (a.k.a., range) would be *strictly monotonic* in relation to subarray length: i.e., extending a subarray would increase the range, whereas shrinking it would do the opposite. Therefore, the problem would lend itself to a solution using two pointers.

However, in the original version, the strictness trait *does not hold*: extending a subarray does not necessarily increase the range. Therefore, even if the range is greater than the length, we could still match that range (or a larger one) by increasing the length. So it wouldn't always be clear which pointer we should move in each iteration of the naive solution.

Let's thus examine a different approach, keeping a few observations in mind:

1. We can count the occurrences of each number and work as if there were no duplicates.
2. We are interested in ranges that are no greater than their sequence length. This is because we can always form a sequence using a subset of the values (as long as we keep the maximum and minimum); the only exception being a unitary range, in which case it is not possible.

Our goal becomes: maximize the difference between any two numbers such that it is no greater than the length of the sequence containing all values within their range. To this effect, let's make some definitions:

- a_i — the i-th element in the sorted and deduplicated array
- b_i — the number of occurrences of a_i in the original array
- c_i — the total number of values up to, and including, a_i (i.e., $c_i = \sum_{j=1}^{i} b_j$)

In other words, we want to maximize the value of $a_j - a_i$ for any $i < j$ such that

$$a_j - a_i \leq c_j - c_i + b_i$$

or, equivalently,

$$a_j - c_j \leq a_i - c_i + b_i$$

Note that fixing a value of i also fixes the right side of this equation. Let's define $d_i = a_i - c_i$. Then,

$$d_j \leq d_i + b_i$$

Thus, it suffices to precompute all d_i and, for each i, find the maximum j among those that satisfy the equation above. This can be done efficiently using a Fenwick Tree.

Alternatively, one may compute c_i as the total number of values up to, but *excluding*, a_i. Then, by fixing a value of j, one can find the minimum i among those that satisfy $d_i \geq d_j - b_j$. This can be done incrementally while updating the Fenwick Tree.

Since we need to sort the input and use a Fenwick Tree, the overall time complexity is $O(n \log n)$.