Max Minus Min Equals Size (Hard)

Problem category: Greedy
Expected difficulty: 1500

Solution

Unlike the easy version of this problem, the monotonicity property does *not* apply here: when we extend a subarray within the sorted array, we might not increase the difference between its minimum and maximum elements (even though we are increasing its length).

This means that a two-pointer approach would not work: even if the difference between the elements at the right and left pointers is greater than the length of the range, we could still form a group with that difference (or a larger difference) by extending to the right. So it is not always clear which pointer we should move in each iteration.

Therefore, we are going to use a dynamic programming approach. First, let's make some observations:

1. We can count the occurrences of each number and work as if there were no duplicates.
2. We are only interested in differences that are no greater than their group's size.

To rephrase: we want to maximize the difference between any two numbers such that it is no greater than the size of the group containing all elements between them (inclusive). 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 elements 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 call d_j the left side. Then,

$$d_j \leq d_i + b_i$$

Thus, it suffices to precompute all d_i and, for each i, find the 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 elements up to, but *excluding*, a_i. Then, by fixing a value of j, one can find the minimum i among those that satisfy $d_i \geq d_j - b_j$.

Complexity

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

Code

```
void solve() {
  Int n;
  vector<Int> a(n);
  map<int, int> cnt;
  for (auto &&ai : a) { // O(n*log n)
    cnt[ai]++;
```

```
  }
  map<int, int> diff;
  int total = 0;
  for (auto &&[x, c] : cnt) { // O(n*log n)
    diff.emplace(x - total, 0);
    total += c;
  }
  int j = diff.size();
  for (auto &&[d, i] : diff) { // O(n)
    i = --j;
  }
  FenTree<int> fen(diff.size(), Min<int>{}, INT_MAX);
  total = 0;
  int ans = 0;
  for (auto &&[x, c] : cnt) { // O(n*log n)
    auto it = diff.lower_bound(x - total - c);
    assert(it != diff.end());
    auto y = fen.query(it->second);
    if (x - y > 1) {
      ans = max(ans, x - y);
    }
    fen.update(diff[x - total], x);
    total += c;
  }
  println(ans);
}
```