Coding Competitions Farewell Rounds - Round B

Analysis: Spacious Sets

For simplicty let us define a function $\mathrm{minDiff(X)}$, which takes a list of numbers X as input and returns the minimum absolute difference between any two elements in the list.

Test Set 1

Since $\mathbf{N}$ is very small, for each i we can find all subsets of $\mathbf{A}$ containing $\mathbf{A_i}$ in $O(2^{\mathbf{N}})$ and then we can filter the subsets that have $\mathrm{minDiff(subset)} \geq \mathbf{K}$ in $O(\mathbf{N}^2)$. We can find the size of such subsets in $O(\mathbf{N})$ and then we can find the maximum size of all such subsets while iterating through the subsets. So the total time complexity will be $O(\mathbf{N} \cdot 2^{\mathbf{N}} \cdot \mathbf{N}^2) = O(2^{\mathbf{N}} \cdot \mathbf{N}^3)$ which should work for Test Set 1.

Test Set 2

The previous algorithm would be too slow for Test Set 2, so we have to find another efficient solution.

Since the order of elements in the subset does not matter, let us sort the list $\mathbf{A}$ in increasing order and call it B. The output list L for B can be rearranged to find the output list of $\mathbf{A}$.

For simplicty, let us use the below definition on some key lists:
E_i is the maximum length of some subsequence s of B such that s ends with B_i and $\mathrm{minDiff(s)} \geq \mathbf{K}$.
S_i is the maximum length of some subsequence s of B such that s starts with B_i and $\mathrm{minDiff(s)} \geq \mathbf{K}$.

Now, L_i can be written as $L_i = E_i + S_i - 1$ (since B_i would be repeated). So we can derive L by computing E and S.

We can use dynamic programming to compute E and S:
$E_i = E_{left_i} + 1$, where left_i is the largest index less than i such that $B_i - B_{left_i} \geq \mathbf{K}$.
$S_i = S_{right_i} + 1$, where right_i is the smallest index greater than i such that $B_{right_i} - B_i \geq \mathbf{K}$.

Binary Search

Now the problem reduces to compute the left and right lists efficiently. Since B is sorted, by using binary search we can find left_i and right_i for each i in $O(\log \mathbf{N})$. Once we know left and right lists then we can easily compute the E and S lists in $O(\mathbf{N})$, and then we can compute the L list in $O(\mathbf{N})$. The total complexity will be $O(\mathbf{N} \log \mathbf{N} + \mathbf{N} \log \mathbf{N} + \mathbf{N} + \mathbf{N}) = O(\mathbf{N} \log \mathbf{N})$.

Sweeping Technique

Instead of binary search, there is a more efficient algorithm if we observe the pattern of left_i and right_i values. The observation is: as i increases, left_i increases, and as i decreases, right_i decreases.

Now initialize a variable $x = 0$ and try to find left_i while iterating through i. At the i-th iteration, keep incrementing x by 1 till you find $B_i - B_{x+1} < \mathbf{K}$ or $x = i - 1$. Now $\mathrm{left}_i = x$. At the end of the i-th step $x = \mathrm{left}_i$.

As x always increases, total increment step of x can be at max length of list which is $\mathbf{N}$. So the amortized time complexity of finding the left list is $O(\mathbf{N})$.
Similarly, right list can found by iterating from right to left (from $\mathbf{N}$ to 1), and initializing the x with $\mathbf{N}$ and keep decrementing in $O(\mathbf{N})$.
Once we compute the left and right lists, we can compute L list in $O(\mathbf{N})$. Total time complexity is $O(\mathbf{N} \log \mathbf{N} + \mathbf{N}) = O(\mathbf{N} \log \mathbf{N})$.