Solution

You cannot eat your cake and have it.
— JAMES JOYCE, »Ulysses« (1922)
All were wrong, so Shem himself, the doctator, took the cake, the correct solution being – all give it up?
— JAMES JOYCE, »Finnegans Wake« (1939)

Subtask 1

For every cake we store its current rank (ordered by deliciousness). The ranks can be updated trivially in $\mathcal{O}(N)$. Eating the cakes can be simulated in $\mathcal{O}(N)$, too. We get a time complexity of $\mathcal{O}(NQ)$.

Subtask 2

Updating the ranks can be implemented more efficiently, if we compute a new scale of deliciousness which can be used to compare the cakes at different times. After each enhancement we introduce a new number measuring the new deliciousness. This scale can be computed in $\mathcal{O}(e_{max}E + N \log N)$ where $e_{max} = 10$ and E is the number of enhancements. This rescaling will be done in all the following solutions. Afterwards we process the queries: Every enhancement can be performed $\mathcal{O}(1)$ by simply setting the deliciousness to the next value computed while constructing the scale. When still simulating the eating process, we get a complexity of $\mathcal{O}(FN + e_{max}E + N \log N)$ where F is the number of queries of type "F".

Subtask 3

To solve the third subtask we can store the order in which the cakes get eaten. Then using a binary search we can answer to queries of type "F" in $\mathcal{O}(\log N)$. However, in the worst case updating requires linear time. We get a complexity of $\mathcal{O}((N + F) \log N + (N + e_{max})E)$.

Full solutions

Segment tree

We create two segment trees with the same structure: one tree for the cakes on the left side of the initial cake and one tree for the cakes on the right side. The leafs represent the cakes ordered increasingly by the distance to the initial cake. We want to implement two types of queries:

- $maxTill(a)$: What is the largest deliciousness among the cakes up to cake number a?
- $initialLengthBelow(d)$: What is the length of the largest initial sequence of cakes which are less delicious than d?

This can be done with a complexity of $\mathcal{O}(\log n)$ when storing the greatest and the smallest deliciousness in every segment. Enhancements can also be performed in logarithmic time. Now a query "F a" can be answered by computing adding the the number of cakes from the initial cake to cake number a and

$\langle$tree not containing $a\rangle.initialLengthBelow(\langle$tree containing $a\rangle.maxTill(\langle$index of a in the tree$))$.

We get a complexity of $\mathcal{O}(Ee_{max} + (N + Q)\log N)$.

Stacks

Instead of segment trees we can also use stacks storing the longest increasing subsequences of deliciousnesses together with the corresponding positions.

Then *maxTill* and *initialLengthBelow* can be implemented using a binary search in the stacks with time complexity $\mathcal{O}(\log N)$. For making a cake the eth most delicious one, we pop all the cakes from the corresponding stack while remembering the last $e-1$ most delicious cakes on the stack until we reach a cake nearer to the initial cake than the cake we want to enhance. Now we decide whether we have to push this cake on the stack. Afterwards some of the $e-1$ most delicious cakes on the stack we have stored might have to be pushed onto the stack. While the number of popped cakes can be linear in N, the number of pushed cakes is bounded from above by e_{max}, and the initial size of the stack is bounded by N. Thus the amortised time complexity of an enhancement is $\mathcal{O}(e_{max})$. Overall we get the same complexity as in the previous solution.