Solution for Fish

Observe that the order in which the fish eat each other is irrelevant. All that matters is whether the jewels of a given set of fish can be "united" into one fish (through the eating process described in the problem statement). We can arrive at the following lemma:

Lemma 1
The jewels that you get might be those of a given set of fish X if and only if the longest fish in X is at least twice as long as the second longest fish in X.
Proof
If the longest fish in X cannot eat the second longest, their jewels can never be united unless a fish not in X is involved. If the longest fish is at least twice as long as the second longest, then the longest one can eat everyone else in X successively.

The tricky part of this problem is to avoid counting some possible combination of jewels more than once. One way to avoid this is to map every possible combination of jewels to the longest fish that can have that combination in its stomach. We then attempt to count, for each fish, the number of combinations mapped to it.

Note: For simplicity we will say that a fish, which was originally given a jewel of kind X, is itself "a fish of kind X".

Lemma 2
Unless a fish is the longest one of its kind, it will have no combinations mapped to it.
Proof
If we have a fish A of kind J and a longer fish B also of kind J, then whatever can be eaten by A can also be eaten by B. So any jewel set that's mapped to A can also involve the longer fish B instead of A, which is a contradiction with the mapping method defined above.

Lemma 3
If the longest fish A of kind J_1 can eat N fish of kind J_1 and another fish B of kind J_2 can eat more than N fish of kind J_1, then no combinations that contain any jewel J_2 would be mapped to the longest fish of kind J_1. (Observe that in such a case the fish B must necessarily be longer than the fish A.)
Proof
This is a similar case to the one above. Any such set that's mapped to the longest fish of kind J_1 can also be found in the stomach of the longer fish of kind J_2, leading to another contradiction.

Lemma 4
If the longest fish of kind J_1 can eat N fish of kind J_1 and another longer fish of kind J_2 can also eat N, but not N+1 fish of kind J_1, then the only combinations that can potentially be mapped to the longest fish of kind J_1 would be ones which either have N+1 jewels J_1 or no jewels J_2.
Proof
Again, using the superset principle we find out that if a combination has at most N jewels of kind J_1 and at least one jewel of kind J_2, then it can be found in the stomach of the longer fish of kind J_2 and thus cannot be mapped to a fish of kind J1.

Knowing this, we can build an algorithm to tell us which combinations are mapped to a given fish.

Note: For simplicity we will denote by $E(J_1, J_2)$ the number of fish of kind J_2 that can be eaten by the longest fish of kind J_1.

Following lemma 2, we're only interested in the longest fish of their respective kind. For each such fish F_1 of kind J_1, we count two types of combinations that are mapped to it. First, combinations that have the maximum number of jewels of kind J_1 (which is $E(J_1, J_1)$ plus one). These are called the "full" combinations. Second, all other combinations mapped to F_1. These are called the "partial" combinations.

Now, for every other kind J_2, with longest fish F_2, we count how many jewels of kind J_2 can be part of a "full" or a "partial" combination mapped to F_1. The above lemmas give rise to three scenarios:

* If $E(J_2, J_1)$ is more than $E(J_1, J_1) + 1$, meaning that F_2 can eat more fish of kind J_1 than F_1, then no jewels of kind J_2 can be part of any combination mapped to F_1.

* If F_2 is longer than F_1, but doesn't fall into the above category, there can be no jewels of kind J_2 in the "partial" category of

F_1, but there can be anywhere between 0 and $E(J_1, J_2)$ jewels in the "full" category.

* If F_2 is shorter than F_1, then any number between 0 and $E(J_1, J_2)$ of fish of kind J_2 can participate in either "full" or "partial" combinations of F_1.

Except for J_1, the number of jewels of any two colors are independent of each other (i.e., the first count doesn't influence the feasibility of the second count in any way and vice versa).

A naïve implementation of this algorithm gives O(K*F) running time since for each longest fish of its kind, we look through all other fish and determine the E values. This should be sufficient for 56 points.

We can improve the naïve implementation if we realize that we need only one loop over all the individual fish, as long as the fish are sorted by length. Then we can go from smallest to longest and count how many fish of each kind we have seen so far. We will use $H(J_X)$ to denote how many fish of kind J_X we have seen. Then when we reach the largest fish that can be eaten by a fish of kind J, we have the values for $E(J, J_X)$ in the current values of $H(J_X)$. Implementing this properly yields a $O(F*\log F + K^2)$ algorithm with the O(F*log F) bottleneck being the sorting of the F fish and the $O(K^2)$ bottleneck being the evaluation of the products of the E numbers for each jewel kind. This solution is sufficient for 70~75 points.

We now work towards an O(F*log F) solution, which gets 100 points for this problem. The key observation here is that if we have the kinds sorted by the length of their respective longest fish, when we compute the number of combinations mapped to a given kind J_1, all we do is adding together a few numbers, each of which is a product of a some consecutive $E(J_1,J_X)$ numbers (where "consecutive" refers to J_X).

This means that we can achieve an O(F*log F) solution by keeping the array $H(J_X)$ (which at given times becomes $E(J_1, J_X)$ for each kind J_1) in a data structure that allows us to modify the array in O(log F) time and to extract the product of a continuous section of the array in O(log F) time as well. This can be done using a binary tree data structure with the leaves of the tree storing the $H(J_X)$ numbers and each node in the tree storing the product of the numbers in its sub-tree. A primitive illustration of this (with the leaves on top) would be as follows:

Note: [a, b] indicates that the node is keeping the product of H(a), H(a+1), H(a+2), …, H(b).

[1,1], [2,2], [3,3], …
[1,2], [3,4], [5,6]...
[1,4], [5,8], [9,12]...
[1,8], [9,16], [17,24]...
...
$[1,2^k]$, $[2^k+1,2*2^k]$, $[2*2^k+1,3*2^k]$...

Clearly each change in the array affects the value stored in O(log F) of these nodes, hence any updates to the data structure can be achieved in O(log F) time by combining the values of the node's children in every affected node, going from the affected leaf to the root. It can also be shown that any interval of the array can be decomposed into at most 2*log F of these intervals, so the product of values in any interval can be calculated in O(log F) time as well.

Final Note: The author also developed an extended version of this problem in which you catch 2 fish, rather than just 1, and the jewels from the two fish are counted together. It is also doable in polynomial time, although it is much more complicated. If you are enthusiastic about solving this version, you should feel free to send your solutions to Velin.Tzanov@deshaw.com.