Balatro

The first observation to make is that the product of all multiply cards is at most 10^9. So, we may assume that the number of multiply cards is at most 29, and that $k \le 29$ (note that k may be much larger than this in the input, but we can just replace it with $\min(k, 29)$).

We will construct an array s where $s[i][j]$ is the highest score achievable using any i mult and j add cards. Once we have this array, for each subsequence length $\ell = 1, \ldots, n$ we can just output

$$\max_{1 \le m \le \ell} \max_{\substack{i+j=m \\ 0 \le i \le k \\ 0 \le j \le n}} s[i][j].$$

This can be done in $O(nk)$ time assuming s has already been constructed.

So, how can we construct this array efficiently? We iterate through the cards in order, and each time we encounter a mult card, we process it along with all of the add cards that have not yet been processed (i.e., all add cards between this mult card and the previous mult card, or the beginning of the sequence if this is the first mult card). Note that we will do this at most 29 times. Say that the mult card we are processing has value m and the add cards have values $a_0, a_1, \ldots, a_r$.

We first update s to account for the new add cards $a_0, a_1, \ldots, a_r$. If we are choosing q of these cards, we will always want to choose the q highest value cards. So, sort the cards in non-increasing order of value and let $b_i = \sum_{0 \le j \le i} a_j$. Then, b_q is the best score we can get by picking q of these $r+1$ add cards. Now, to update s, we do the following for all j and for all i from 1 up to the number of mult cards we have already processed:

$$s[i][j] = \max_{\substack{x+y=j \\ 0 \le x \le j \\ 0 \le y \le r}} s[i][x] + b_y \tag{1}$$

Doing this naively requires $O(n^2)$ time for each i, but since the b array is concave, we can do better. For fixed i, Eq. (1) is an instance of a $(\max, +)$-convolution problem with one concave array, so we can solve it in $O(n \log n)$ using a Li Chao tree, or $O(n)$ using SMAWK (the time limit on the problem is set so that either of these approaches should get AC). There are good resources online describing how this can be done[1], so we omit the details here.

Now, we update s to account for the mult card m. Since we are only adding one additional card into consideration now, this is not hard. For all j and for all i from the number of mult cards processed so far down to 1, we do the following:

$$s[i][j] = \max(s[i][j], s[i-1][j] \cdot m)$$

[1] `https://codeforces.com/blog/entry/98663`

This takes $O(29 \cdot n)$ time. So, the overall running time is $O(29 \cdot n \log n)$ or $O(29 \cdot n)$ depending on which approach is used for the $(\max, +)$-convolution.