Code Jam 2021 - Round 1A

Analysis: Prime Time

Test Set 1

In Test Set 1, the total number of cards in the deck is at most 10. A small number like 10 suggests we can brute force and simulate all possible game scenarios. For each card we can decide whether it belongs to the first group or the second group. Next, we sum up the numbers on the first group cards and compare with the product of the numbers on the second group cards. If they are equal, current sum or product is one of the possible candidate scores. After checking all possible partitions, we output the maximum score or output the score 0 if there is no feasible way to partition the cards.

Two things to watch out for. First, each of the groups has to be non-empty. This is easy to handle. Second, the product of all the numbers may overflow the integer type of our choice (in the worst case there may be 10 cards, each with 499). There are many ways one can avoid this problem. One way can be to first find the sum of the first group numbers and then try to divide it by the second group numbers, one by one. If at some point we are unable to perform the division or if after all the divisions the result is not 1, we know current partition candidate can not be good. Another way would be to notice that the sum of the first group can not be very high. A naive upper bound is 4990 (all the 10 numbers are 499 and they are in the first group). Hence we can find the product of the second group numbers in floating point and compare with the sum of the first group numbers. Since the upper bound of the sum of the first group numbers is very low, precision error should not cause any trouble.

There are $2^{\mathbf{N}}$ ways to partition the cards where $\mathbf{N}$ is the number of cards. After each partition we can find the sum and the product of the groups with a linear loop. Hence the total run time is $O(2^{\mathbf{N}}\mathbf{N})$. We could also get rid of the linear loop but such optimization was not necessary.

Test Set 2

In Test Set 2, the total number of cards is at most 100. Performing all possible partitions would surely timeout. But notice that we cannot put many numbers into the second group because the product of positive numbers greater than 1 quickly increases. This time the upper limit for the sum of the first group numbers is 49900. That means, we can not have more than 15 numbers in the second group ($2^{16} > 49900$). We can use this observation to optimize our brute force solution.

Let S be the set of all possible partitions using the first i cards for some arbitrary ordering of the cards. Instead of keeping explicit partitions, we will represent the partition as (x, y) where x is the sum of the numbers in the first group and y is the product of the numbers in the second group. This helps us to reduce the state space. Once we partition the first i cards, it does not matter exactly which card went to which group. We just care about the sum and the product of the respective groups. Now let b be the value of the $i + 1$'th card. Hence for each partition (x, y) in S we will have two choices: $(x + b, y)$ and $(x, y \times b)$. If this makes the product of the second group numbers larger than 49900, we prune this state. After considering all $\mathbf{N}$ cards, we check if there is some partition where the sum of the first group numbers is equal to the product of the second group numbers (i.e., a tuple in S where both values are equal). We output the maximum of these sum or product, or 0 if there is no such partition.

We can have at most 100 numbers in the input. So it might be tempting to think that there may be $\binom{100}{15}$ possible valid partitions. But the product of the numbers in the second group can be at

most 49900 and each of these 49900 numbers can be uniquely represented as the product of primes. So the remaining numbers will go to the first group. Hence after considering each of the input numbers there can be at most 49900 possible partitions (49900 possible second groups and for each of these there is a unique first group). So this naive calculation gives us a limit of $49900 \times 100 \approx 5 \times 10^6$ operations.

So far we have used the fact that each score can be uniquely represented as a product to prove that our solution is fast enough, but it can also be used to obtain an alternative approach for Test Set 2: we have only 49900 candidates for the final score, so we can iterate over them one by one. For each candidate score, there is exactly one way to represent it as a product of primes. Assuming we know the prime factorization of the candidate how can we determine if the candidate is valid? Suppose the prime p appears in the prime factorization q times. Then, the prime p has to be in the input at least q times. This condition guarantees us that the second group is achievable. Next we need to come up with a condition to check the existence of the first group. If we know the sum of the input numbers and subtract the sum of the numbers in the second group we can get the sum of the first group which has to be equal to our candidate. This guarantees the first group existence. Both of these conditions can be checked very easily given the prime factorization of the candidate score.

It turns out that this approach generalizes very well to Test Set 3.

Test Set 3

In Test Set 3, $\mathbf{N}$ can be up to 10^{15}, which means the sum of the first group can be as high as 4.99×10^{17}. But the number of cards in the second group can not be very high. Since $2^{60} > 4.99 \times 10^{17}$, we can consider 60 as the upper bound of number of cards in the second group. Although the product of at most 60 cards in the second group can range from 2 to 2^{60}, the sum of the second group numbers can be only up to $60 \times 499 = 29940$ (the actual maximum possible sum of the second group numbers is 3025 under the problem's constraint but even a crude estimate of 29940 is enough to get a working solution). Let X be the sum of all the cards in the input. Then, the sum of the first group numbers must be between $X - 29940$ and X, inclusive. This means we have only 29941 candidates for the final score, so we could apply the second approach from Test Set 2 if only we could factorize those candidates.

Unfortunately factorizing a number as big as 10^{17} is not an easy task. A naive way may be to run a loop up to $\sqrt{10^{17}}$. However, in this problem we do not need to care about the primes higher than 499. To put it another way, if the candidate has a prime factor other than the input prime numbers, we cannot achieve this score in the second group. Hence, it is enough to try to factorize the 29940 candidates with only primes from 2 to 499. It takes about $29940 \times (95 + 60) \approx 4.6 \times 10^6$ operations to factorize the 29940 candidates (there are 95 primes between 2 and 499 and in total there can be at most 60 prime factors). We could also do sieve-like factorization reducing the number of operations to about $29940 \log \log 499 \approx 10^5$ but this is not necessary. After the factorization, we can run a loop to check if the number of each prime exceeds the input count and also to sum these primes. This takes $29940 \times 95 \approx 3 \times 10^6$ operations.