Kick Start 2022 - Round B

Analysis: Palindromic Factors

A simple way to test if a number is palindromic is to convert it to a string first and then check if the string equals its reverse. Since $\mathbf{A}$ has no more than 11 digits, we will assume that it is a constant time operation.

C++:

```
bool isPalindrome(long long a) {
  string s = to_string(a);
  string rev(s.rbegin(), s.rend());
  return s == rev;
}
```

Python:

```
def isPalindrome(a):
  s = str(a)
  rev = s[::-1]
  return s == rev
```

Java:

```
public static boolean isPalindrome(long a) {
  String s = Long.toString(a);
  StringBuilder rev = new StringBuilder(s);
  rev.reverse();
  return s.equals(rev.toString());
}
```

Test Set 1

For the small test set, we can afford to find all factors of $\mathbf{A}$ by checking every integer $a \in \{1, 2, \ldots, \mathbf{A}\}$. For each factor of $\mathbf{A}$, we also check if it is a palindrome and increment the answer accordingly.

The time complexity of this brute-force solution is $O(\mathbf{A})$.

Test Set 2

Let a and b be two factors of $\mathbf{A}$ such that $\mathbf{A} = ab$ and $a \leq b$. Then $a \leq \sqrt{\mathbf{A}}$. It follows that we can find all factors of $\mathbf{A}$ by checking the first $\sqrt{\mathbf{A}}$ numbers only. For each factor $a \leq \sqrt{\mathbf{A}}$, the number $b = \frac{\mathbf{A}}{a} \geq \sqrt{\mathbf{A}}$ is also a factor of $\mathbf{A}$.

The time complexity of the optimized algorithm is $O(\sqrt{\mathbf{A}})$.