Solution 1

Let's fix v, the minimum value of $\lfloor \frac{a_i}{p_i} \rfloor$. Then, for all $1 \le i \le n$, we find the maximum value p_i such that $p_i \le k$ and $\lfloor \frac{a_i}{p_i} \rfloor \ge v$.

For some minimum value v, let's call the array described above $P(v)$, and let's define $M(v) = \max\limits_{1 \le i \le n} \lfloor \frac{a_i}{P(v)_i} \rfloor$. We can find the answer by taking the minimum of $M(v) - v$ across all $0 \le v \le a_1$, giving a $O(n \cdot a_i)$ solution.

To speed it up, let's consider how some element a_i will affect the values of $M(v)$.

First, notice that $\lfloor \frac{a_i}{q} \rfloor$ (where $1 \le q \le k$) can take on at most $O(\min(k, \sqrt{a_i}))$ distinct values. Let's denote these values (in increasing order) $s_1, s_2, s_3, \ldots, s_x$. Consider what happens when $v \le s_1$. Then, $M(v)$ must be at least s_1. What about when $s_1 < v \le s_2$? Then, $M(v)$ must be at least s_2. And so on, until $s_{x-1} < v \le s_x$, where $M(v)$ must be at least s_x.

This way, we can get lower bounds on value of $M(v)$. It is easy to see that the highest of these bounds is achievable.

Let's iterate over array a. Let $m[v]$ (here, $m = m[0], m[1], m[2], \ldots, m[a_1]$ is an array of length $a_1 + 1$) be the highest of lower bounds on $M(v)$ we already found. Initially, $m[v] = 0$ for all v. When we are dealing with a_i we want to do the following:

- For all $0 \le j \le x - 1$, we want to update $m[y] = \max(m[y], s_{j+1})$ for all $s_j + 1 \le y \le s_{j+1}$ (for convenience we define $s_0 = -1$).

Since $s_0 < s_1 < s_2 < \ldots < s_x$, this can be done without any fancy data structures — instead of updating all these ranges directly, we can set $m[s_j + 1] = \max(m[s_j + 1], s_{j+1}])$, so that $M(v)$ will be equal to $\max(m[0], m[1], \ldots, m[v])$.

Then, once m is computed, we can sweep through to find all values of $M(v)$ in with prefix maxes.

Once we have m computed, we can find $M(v) - v$ for all $0 \le v \le a_1$ in linear time. This gives a $\mathcal{O}(\sum\limits_{1 \le i \le n} \min(k, \sqrt{a_i}) + a_1)$ solution per test case, with total $\mathcal{O}(n + \max_a)$ memory across all tests.

Solution 2 (AlperenT)

Now, let's fix v as the maximum value of $\lfloor \frac{a_i}{p_i} \rfloor$. We now want to maximize the minimum value of $\lfloor \frac{a_i}{p_i} \rfloor$.

Let's now consider all elements a_i that satisfy $1 \le a_i \le v$. For these elements, it will be optimal to set $p_i = 1$, since we want to maximize them.

How about elements a_i satisfying $v + 1 \le a_i \le 2v$? We need to have $\lfloor \frac{a_i}{p_i} \rfloor \le v$, so for these elements, we must have $p_i \ge 2$. At the same time, we want to maximize them — so it will be optimal to set all these $p_i = 2$.

Continuing this logic, for all integers $u = 1, 2, \ldots, k$, we should check the elements a_i satisfying $(u-1) \cdot v + 1 \le a_i \le u \cdot v$, and set all these $p_i = u$.

How can we determine the minimum value of $\lfloor \frac{a_i}{p_i} \rfloor$ from this? For a fixed u, the minimum $\lfloor \frac{a_i}{u} \rfloor$ will come from the minimum a_i. So if we can determine the minimum a_i such that $(u-1) \cdot v + 1 \le a_i \le u \cdot v$, and calculate these values across all $u = 1, 2, \ldots, k$, then we will get the answer.

To help us, let's precompute an array $next_1, next_2, \ldots, next_{a_n}$. $next_j$ will store the minimum value of a_i such that $a_i \ge j$. Now, for a fixed u, we can check $next_{(u-1) \cdot v + 1}$. If this value is less than or equal to $u \cdot v$, it will be the minimum a_i that we divide by u.

Two important details:

1. If there exists some $a_i \ge (v+1) \cdot k$, then it is impossible to have the max element as v, and we should skip it.
2. For some value v, we only need to check u such that $(u-1) \cdot v + 1 \le a_n$.

Using this second detail, the solution runs in $\mathcal{O}\left(\sum_{i=1}^{a_n} \frac{a_n}{i}\right) = \mathcal{O}(a_n \cdot \log(a_n))$ time per test case. The memory usage is $\mathcal{O}(n + \max_a)$ across all tests.