## Problem 1

1. Initialize an empty hash table $H$.
2. For each element $x$ in the input array $A$:
   (a) Compute the hash value $h(x)$ using the independent uniform hash function $h$.
   (b) Check if $x$ is already in the hash table $H$ at index $h(x)$.
      - If $x$ is already in $H$, output `Not Unique` and return.
      - Otherwise, insert $x$ into $H$ at index $h(x)$.
3. If the loop terminates, output `Unique`.

Since we perform $n$ iterations, and each iteration involves a constant expected time for hashing, lookup, and insertion due to the uniform distribution of the hash function (ensuring that collisions are rare and evenly distributed), the total expected time of the algorithm is $n \times O(1) = O(n)$.

## Problem 2

1. $H_{p,\,m}$ is universal
   (a) Since $k_1 \neq k_2$, define $\delta = k_1 - k_2 \mod p$.
   (b) The condition $h(k_1) = h(k_2)$ implies

$$\begin{aligned}&((ak_1 + b) \mod p) \mod m = ((ak_2 + b) \mod p) \mod m\\ \Rightarrow\ &(a(k_1 - k_2) \mod p) \equiv 0 \mod m\\ \Rightarrow\ &(a\delta \mod p) \equiv 0 \mod m\end{aligned}$$

   (c) The number of values $t = a\delta \mod p$ such that $t \mod m \equiv 0$ is at most $\frac{p-1}{m}$.
   (d) The probability that $h_{a,b}(k_1) = h_{a,b}(k_2)$ is

$$\Pr\left[h_{a,b}(k_1) = h_{a,b}(k_2)\right] = \frac{\frac{p-1}{m}}{p-1} \leq \frac{1}{m}$$

   Therefore, $H_{p,\,m}$ is a universal hash family since the collision probability is at most $\frac{1}{m}$.

2. $H_{p,\ m}$ is 2-Independent
   (a) Define $s = ak_1 + b \mod p$ and $t = ak_2 + b \mod p$, then

   $$\begin{cases} ((ak_1 + b) \mod p) \mod m = s \mod m = \alpha \\ ((ak_2 + b) \mod p) \mod m = t \mod m = \beta \end{cases}$$

   (b) Define $\delta = k_1 - k_2$, we have

   $$(s - t) \mod m = (\alpha - \beta) \mod m = (a\delta) \mod p \mod m$$

   where there are $\frac{p-1}{m}$ possible values of $a$ satisfying the equation.
   (c) Once $a$ is fixed, the value of $b$ is uniquely determined by $b = s - ak_1 \mod p$. Therefore, for each valid $a$, there are exactly $\frac{p-1}{m}$ valid $b$.
   (d) The total number of valid pairs $(a, b)$ is $\frac{p-1}{m} \cdot \frac{p-1}{m} = \frac{(p-1)^2}{m^2}$.
   (e) Thus, the joint probability that both conditions hold is

   $$Pr\left[h_{a,b}(k_1) = \alpha \text{ and } h_{a,b}(k_2) = \beta\right] = \frac{\text{Number of valid } (a,b)}{\text{Total nimver of } (a,b)} = \frac{\frac{(p-1)^2}{m^2}}{(p-1)p} \leq \frac{1}{m^2}$$

   Therefore, $H_{p,m}$ is 2-independent because the joint probability of two keys mapping to specific values $\alpha$ and $\beta$ is at most $\frac{1}{m^2}$.
3. $H_{p,\ m}$ is not 3-Independent
   (a) Consider $h_{a,b}(0)$, $h_{a,b}(1)$, and $h_{a,b}(2)$

   $$\begin{cases} h_{a,b}(0) = (b \mod p) \mod m \\ h_{a,b}(1) = ((a + b) \mod p) \mod m \\ h_{a,b}(2) = ((2a + b) \mod p) \mod m \end{cases}$$

   (b) Observe that knowing $h_{a,b}(0)$ and $h_{a,b}(1)$ provides information about $h_{a,b}(2)$

   $$h_{a,b}(2) - 2h_{a,b}(1) + h_{a,b}(0) \equiv 0 \mod m$$

   The hash values of $x, y, z$ are not fully independent since they satisfy a linear relation modulo $m$. Therefore, $H_{p,m}$ is not 3-independent.

## Problem 3

Let $dp[i]$ be the minimum cost to reach square $i$ from square 1. The base case is $dp[1] = 0$ (the frog is already on square 1). For each square $i$ from 2 to $n$, we have two choices:

- if $i - 1 \geq 1$, the frog can move from square $i - 1$: $Cost_1 = dp[i-1] + |h[i] - h[i-1]|$.
- if $i - 2 \geq 1$, the frog can move from square $i - 2$: $Cost_2 = dp[i-2] + |h[i] - h[i-2]|$.
- The minimum cost to reach square $i$ is $dp[i] = \text{min}(Cost_1, Cost_2)$.

The answer is $dp[n]$, and the algorithm runs in $O(n)$ time.

## Problem 4

Let $dp[w][h]$ represent the minimum number of moves required to cut a $w \times h$ rectangle into squares. For all $w = h$, $dp[w][h] = 0$ (the rectangle is already a square). For all $w$ and $h$ where $w \neq h$, we have two choices:

- Vertical cuts: $VerticalCut[k] = \text{min}(dp[w][i] + dp[w][h-k] + 1)$ for $k = 1, 2, \ldots, w-1$.
- Horizontal cuts: $HorizontalCut[k] = \text{min}(dp[i][h] + dp[w-k][h] + 1)$ for $k = 1, 2, \ldots, h-1$.
- The minimum number of moves required to cut a $w \times h$ rectangle into squares is $dp[w][h] = \text{min}(\text{min}_{all\ k}(VerticalCut[k]), \text{min}_{all\ k}(HorizontalCut[k]))$.

The answer is $dp[a][b]$, and the algorithm runs in $O(a \times b \times (a + b))$ time.