It suffices to count the number of pairs $(i, j)$ that satisfies $a[i]$ xor $a[j] \leq x$.
1. Use trie. $O(nW)$.
2. Similar to 421. Maximum XOR of Two Numbers in an Array. First sort the integers, then build the compressed trie in $O(n)$ time, which has $O(n)$ nodes and edges. Perform a dfs on the trie to enumerate all possible $a[i]$'s, and the value $a[i]$ xor $x$ will also traverse the trie once (we can simulate the traversal using bit operations), in the meantime we maintain the number of possible $j$'s, so the running time is $O(n)$. $O(\mathrm{sort}(n))$.
3. Suppose we want to count the number of pairs $a[i]$ xor $a[j] < x$. Let $k$ denote their leftmost differing position, we must have $(a[i]$ xor $a[j])[k] = 0$ and $x[k] = 1$. In other words, $a[i][1..k-1]$ xor $a[j][1..k-1] = x[1..k-1]$, $a[i][k]$ xor $a[j][k] = 0$, and $x[k] = 1$. So we only need to count the frequency of the first $i$ bits of the input numbers (in $[2^i]$), for $i = 1, \ldots, \log U$. $O(\min\{U, n \log \frac{U}{n}\})$.
4. Fast Walsh-Hadamard transform. $O(U \log U)$.

# References