Problem 3376: Minimum Time to Break Locks I

Problem Information

Difficulty: Medium
Acceptance Rate: 31.43%
Paid Only: No
Tags: Array, Dynamic Programming, Backtracking, Bit Manipulation, Depth-First Search, Bitmask

Problem Description

Bob is stuck in a dungeon and must break `n` locks, each requiring some amount of **energy** to break. The required energy for each lock is stored in an array called `strength` where `strength[i]` indicates the energy needed to break the `ith` lock.

To break a lock, Bob uses a sword with the following characteristics:

* The initial energy of the sword is 0. * The initial factor `x` by which the energy of the sword increases is 1. * Every minute, the energy of the sword increases by the current factor `x`. * To break the `ith` lock, the energy of the sword must reach **at least** `strength[i]`. * After breaking a lock, the energy of the sword resets to 0, and the factor `x` increases by a given value `k`.

Your task is to determine the **minimum** time in minutes required for Bob to break all `n` locks and escape the dungeon.

Return the **minimum** time required for Bob to break all `n` locks.

Example 1:

Input: strength = [3,4,1], k = 1

Output: 4

Explanation:

Time | Energy | x | Action | Updated x ---|---|---|---|--- 0 | 0 | 1 | Nothing | 1 1 | 1 | 1 | Break 3rd Lock | 2 2 | 2 | 2 | Nothing | 2 3 | 4 | 2 | Break 2nd Lock | 3 4 | 3 | 3 | Break 1st Lock | 3 The locks cannot be broken in less than 4 minutes; thus, the answer is 4.

Example 2:

Input: strength = [2,5,4], k = 2

Output: 5

Explanation:

Time | Energy | x | Action | Updated x ---|---|---|---|--- 0 | 0 | 1 | Nothing | 1 1 | 1 | 1 | Nothing | 1 2 | 2 | 1 | Break 1st Lock | 3 3 | 3 | 3 | Nothing | 3 4 | 6 | 3 | Break 2nd Lock | 5 5 | 5 | 5 | Break 3rd Lock | 7 The locks cannot be broken in less than 5 minutes; thus, the answer is 5.

Constraints:

* `n == strength.length` * `1 <= n <= 8` * `1 <= K <= 10` * `1 <= strength[i] <= 106`

Code Snippets

C++:

```
class Solution {
public:
int findMinimumTime(vector<int>& strength, int k) {

}
};
```

Java:

```
class Solution {
public int findMinimumTime(List<Integer> strength, int k) {

}
}
```

Python3:

```
class Solution:
def findMinimumTime(self, strength: List[int], k: int) -> int:
```