**Problem 2**

```python
# Time Complexity O(n)
# Space Complexity O(n)
def get_maximum_number_of_chicken_protected_by_superman(
    n: int, k: int, pos_chicken_list: list[int]
) -> int:
    max_help = 1
    start_help_index = 0

    for i in range(1, n):
        while pos_chicken_list[i] > pos_chicken_list[start_help_index] + k - 1:
            start_help_index += 1

        max_help = max(max_help, i - start_help_index + 1)

    return max_help
```

To maximize the numbers of chickens protected by superman, the starting point of the roof should align with a chicken's position. If it does not, I will waste a portion of the roof before the starting point.

For example:

Roof length = 5

Chicken positions = [6,7,8,9,10]

Starting the roof at 5 will only cover 4 chickens (5 to 9). However, for our algorithm, the roof starting point is 6, allowing it to cover all the chickens.

**Script Explanation**

```
max_help = 1
start_help_index = 0
```

Initially, set the maximum number of chickens to 1 and start_help_index to position 0.

Since the problem guarantees that the given positions of the chickens will be sorted from lowest to highest, I will structure the script accordingly:

```
for i in range(1, n):
    while pos_chicken_list[i] > pos_chicken_list[start_help_index] + k - 1:
        start_help_index += 1

    max_help = max(max_help, i - start_help_index + 1)

return max_help
```

Now, I have two pointers:

1. i => the last index of the chicken position list that the roof must cover.
2. start_help_index=> the starting point of roof

In the while loop, I iterate through the position list starting from start_help_index up to 'i' to find the optimal starting point for the roof that will cover the chicken(i) and still yield the maximum number of the protected chickens.

I can use this approach because previous starting points will not be used as the starting point for next positions. If a starting point cannot cover the current position, it also cannot cover the next positions, as the position list is arranged in ascending order.

Then, if the current starting point of the roof can protect more chickens, set the max_help to the new value. Otherwise, keep it the same value.

The response of this function is the max_help, which represents the maximum number of chickens protected by superman.

Complexity Analysis

```
# Time Complexity O(n)
# Space Complexity O(n)
def get_maximum_number_of_chicken_protected_by_superman(
```

Time Complexity : O(n)=> loop through the list of length n

Space Complexity : O(n) => the length of input list is n