# 495. Teemo Attacking

Easy Array Simulation

## Problem Description

In this problem, we are given a scenario where a character named Teemo is attacking another character named Ashe with poison attacks. When an attack occurs, it poisons Ashe for a fixed duration. Each attack time is recorded in an array called `timeSeries`, and this array is in non-decreasing order, which means the attacks are listed in the order they happened and no attack happens before the previous one. The `duration` is a fixed amount of time that the poison lasts after each attack. If Ashe gets attacked again before the previous poison has worn off, the poison duration is reset to last for `duration` more seconds from the time of the new attack.

Our goal is to calculate the total number of seconds Ashe is poisoned. Since an attack at second `t` means Ashe is poisoned up to `t + duration - 1`, we must be careful not to count any time twice if the poison effect from a previous attack is extended by a subsequent attack before it has ended.

## Intuition

To solve the problem, we need to calculate the periods Ashe is poisoned without counting any seconds multiple times. We initialize the total poisoned time with `duration` to account for the first attack, assuming there is at least one attack.

Then, we iterate through each pair of consecutive attacks (using a handy Python function called `pairwise` from the `itertools` module, that allows us to consider the attack at `timeSeries[i]` and `timeSeries[i+1]` at the same time). For each pair of attacks, we have two cases:

1. If the next attack happens after the current poison effect has ended, Ashe will be poisoned for a full `duration` again starting from the new attack.
2. If the next attack happens before the current poison has worn off, we only need to extend the poison effect to last `duration` seconds after this new attack, which may be less than a full `duration`.

The amount by which we extend the poison effect can be found by taking the minimum of `duration` and the difference between the current and next attack times, `b - a`. This ensures that if the new attack happens very shortly after the first one, we don't count more than the total `duration` for the poison effect that's been reset.

## Solution Approach

The solution uses a straightforward algorithm to implement the logic explained in the intuition:

1. Initialize a variable `ans` with the value of `duration`. This is to account for the initial state when Teemo attacks Ashe once. The postulated scenario ensures there is at least one attack. Thus, Ashe is at minimum poisoned for the length of `duration`.
2. Iterate through pairs of consecutive attacks. Python's `pairwise` function is used here from the `itertools` module, which takes an iterable and returns a new iterator over pairs of consecutive elements. For a list `timeSeries`, `pairwise(timeSeries)` yields `(timeSeries[0], timeSeries[1])`, `(timeSeries[1], timeSeries[2])`, and so on.
3. For each consecutive pair `(a, b)` of `timeSeries`, calculate the time the poison will last. If `b - a >= duration`, it means that the poison from the first attack would have worn off before the second attack occurs. If `b - a < duration`, the second attack effectively resets the timer, and the poison now extends `b - a` seconds past the time of the second attack.
4. For each pair `(a, b)`, add the minimum of `duration` and `b - a` to `ans`. This accounts for extending the poison duration correctly without double-counting any seconds.
5. After processing all pairs, `ans` holds the total time Ashe is poisoned, which is returned as the result.

The algorithm is efficient as it only goes through the list of attack times once, with a time complexity of O(n), where n is the number of attack times in `timeSeries`. No additional space apart from the input and a few variables is used, resulting in O(1) space complexity. The `pairwise` function is not included in the space complexity as it only returns an iterator and doesn't store the pairs in memory.

### Example Walkthrough

Let's illustrate the solution approach with a small example. Suppose we have `timeSeries = [1, 4, 5]` and `duration = 2`. This means Teemo attacks Ashe at times 1, 4, and 5, and each poison lasts for 2 seconds.

1. We start by initializing `ans` as `duration`. Therefore, `ans = 2`.
2. Iterate using `pairwise`, which will produce pairs `(1, 4)` and `(4, 5)` for our example.
3. For the pair `(1, 4)`, we see that `b - a = 4 - 1 = 3`, which is greater than `duration = 2`. This means the poison had worn off before the second attack. Thus, we add the full `duration` to `ans`. Now, `ans = ans + duration = 2 + 2 = 4`.
4. For the next pair `(4, 5)`, `b - a = 5 - 4 = 1`, which is less than `duration`. The second attack happens before the poison from the previous attack has worn off. We extend the poison duration by the amount `b - a`, which is 1 in this case. Hence, `ans = ans + (b - a) = 4 + 1 = 5`.
5. There are no more pairs. The total time Ashe is poisoned is stored in `ans`, which is 5 seconds.

To understand the final result, let's visualize the timeline:

- First attack at time `1`, poison lasts until the end of time `2`.
- Second attack at time `4`, poison lasts until the end of time `5`.
- Third attack at time `5`, extends the poison from the end of time `5` to the end of time `6`.

So visually:

```
Time:        1 2 3 4 5 6
Poison:      X X - X X X
Total time poisoned = 1-2 + 4-6 = 2 + 2 = 4 seconds.
```

Thus the total time poisoned is 5 seconds as calculated. This example confirms that the algorithm correctly calculates the total time Ashe is poisoned without double-counting any seconds.

## Solution Implementation

### Python

```python
from typing import List
from itertools import tee

class Solution:
    def findPoisonedDuration(self, time_series: List[int], duration: int) -> int:
        # Helper function to mimic the pairwise utility, yielding consecutive pairs from time_series
        def pairwise(iterable):
            a, b = tee(iterable)
            next(b, None)
            return zip(a, b)

        # If there are no time stamps, the poisoned duration is 0
        if not time_series:
            return 0

        # Initialize total poisoned duration with the duration of the last poisoning event
        total_duration = duration

        # Calculate the poisoned duration by comparing consecutive poison times
        for start, end in pairwise(time_series):
            time_diff = end - start
            total_duration += min(duration, time_diff)

        return total_duration
```

### Java

```java
class Solution {
    // Method to find total time for which a character remains poisoned
    public int findPoisonedDuration(int[] timeSeries, int duration) {
        // Length of the timeSeries array
        int n = timeSeries.length;

        // If there are no attacks, return 0 as the poisoned duration
        if (n == 0) {
            return 0;
        }

        // Initialize total poisoned duration with the duration of the first attack
        int totalPoisonedDuration = duration;

        // Iterate through the time series starting from the second attack
        for (int i = 1; i < n; ++i) {
            // Calculate the time difference between current and previous attack
            int timeDifference = timeSeries[i] - timeSeries[i - 1];

            // If the time difference is less than the duration of the poison,
            // it means the poison effect would have been refreshed and not lasted full duration,
            // so add the time difference, otherwise add the full duration
            totalPoisonedDuration += Math.min(duration, timeDifference);
        }

        // Return the total duration for which the character was poisoned
        return totalPoisonedDuration;
    }
}
```

### C++

```cpp
#include <vector>
#include <algorithm> // For std::min function

class Solution {
public:
    int findPoisonedDuration(std::vector<int>& timeSeries, int duration) {
        // The total poison duration starts with the duration of the first attack,
        // as there can be no overlap in this case
        int totalPoisonDuration = duration;

        // Number of time points in the series
        int numOfTimePoints = timeSeries.size();

        // Loop through all the time points starting from the second one
        for (int i = 1; i < numOfTimePoints; ++i) {
            // Calculate the poisoned time by the current attack.
            // If the time difference between the current attack and the previous one
            // is less than the duration, the poison duration is shortened to this time difference
            // to avoid counting the overlapping time more than once.
            totalPoisonDuration += std::min(duration, timeSeries[i] - timeSeries[i - 1]);
        }

        // Return the total calculated poison duration
        return totalPoisonDuration;
    }
};
```

### TypeScript

```typescript
function findPoisonedDuration(timeSeries: number[], duration: number): number {
    // 'n' represents the total number of time points in the timeSeries array.
    const n = timeSeries.length;

    // If there are no time points, the poisoned duration is 0.
    if (n === 0) return 0;

    // 'totalDuration' will accumulate the total duration for which the target is poisoned.
    let totalDuration = 0;

    // Iterate over the time points to calculate the poisoned duration.
    for (let i = 0; i < n - 1; ++i) {
        // Calculate the time difference between the current and the next time point.
        const timeDifference = timeSeries[i + 1] - timeSeries[i];

        // If the time difference is less than the 'duration',
        // add this time difference to 'totalDuration' as the poison
        // overlaps. Otherwise, add the full 'duration'.
        totalDuration += Math.min(duration, timeDifference);
    }

    // Add the 'duration' of the last poisoning effect,
    // since it will last the full 'duration' time.
    totalDuration += duration;

    // Return the sum of all the durations for which the target is poisoned.
    return totalDuration;
}


from typing import List
from itertools import tee

class Solution:
    def findPoisonedDuration(self, time_series: List[int], duration: int) -> int:
        # Helper function to mimic the pairwise utility, yielding consecutive pairs from time_series
        def pairwise(iterable):
            a, b = tee(iterable)
            next(b, None)
            return zip(a, b)

        # If there are no time stamps, the poisoned duration is 0
        if not time_series:
            return 0

        # Initialize total poisoned duration with the duration of the last poisoning event
        total_duration = duration

        # Calculate the poisoned duration by comparing consecutive poison times
        for start, end in pairwise(time_series):
            time_diff = end - start
            total_duration += min(duration, time_diff)

        return total_duration
```

## Time and Space Complexity

### Time Complexity

The time complexity of the provided code is `O(N)`, where `N` is the length of the `timeSeries` array. This is because the code iterates through the array once using a for loop with the `pairwise` function, which generates a new pair for consecutive elements on each iteration.

### Space Complexity

The space complexity of the code is `O(1)`. Regardless of the size of the input list, the only extra space required is for the variable `ans`. The `pairwise` function generates pairs on-the-fly on each iteration without storing them, thus not requiring any additional significant space.