Band Protocol assignment (Software engineer)

Introduction:
Welcome to Band Protocol's assignment! This assignment is designed to assess your problem-solving skills, coding abilities, and software development knowledge. You will be given three questions to complete within a specified time frame. Please read the instructions carefully and complete the tasks to the best of your abilities.

Instructions:
1. Coding Algorithm Questions (2 questions):
   - Each question is accompanied by a description, input/output format, and example test cases.
   - You are required to write any programming language to solve each problem with the best solution in term of time and memory complexity.
   - Ensure that your code is well-structured, readable, and efficiently solves the given problem.
   - Ensure that you include explanation, comment, time and memory complexity of your solution.
   - Pay attention to edge cases and input constraints mentioned in the problem description.
   - Test your code with provided and additional test cases to verify correctness.
2. Software Development Question (1 question):
   - The question will present you with requirements for a software to do specific needs.
   - Consider scalability, reliability, and performance aspects in your code.
   - Explain your design decisions, trade-offs, and assumptions in detail.
   - You can use diagrams, text descriptions, or any other suitable format to present your software.

Guidelines:
1. Time Limit:
   - Manage your time wisely to ensure completion of all questions within the allocated time frame.
   - Keep track of time while working on each question and allocate sufficient time for testing and verification.
2. Submission:
   - Choose one of the following submission methods:
     i. Zip Package: Create a zip file containing your code for each task.
     ii. GitHub Repository: Establish a GitHub repository for your code for each task, including both the code and README.
3. Evaluation:
   - Your solutions will be evaluated based on correctness, efficiency, time and memory complexity, clarity, and adherence to instructions.
   - Provide concise and clear explanations for your solutions and design choices.
   - Demonstrate your problem-solving approach and coding style effectively.

Note:
- Plagiarism or dishonest behavior will result in disqualification from the interview process.

Good luck! Please feel free to reach out if you need any clarifications or have inquiries. We look forward to reviewing your submission!

Problem 1: Boss Baby's Revenge

Description:
Boss Baby, known for his cool and clever ways, deals with teasing from the neighborhood kids who shoot water guns at his house. In response, Boss Baby seeks revenge by shooting **at least one shot back**, but only if the kids have already shot at him first. As Boss Baby's assistant, your task is to check if Boss Baby has sought revenge for every shot aimed at him at least once and hasn't initiated any shooting himself.

Input:
A string (S, 1 <= len(S) <= 1,000,000) containing characters 'S' and 'R', where 'S' represents a shot and 'R' represents revenge. The string represents the sequence of events for one day.

Output:
Return "Good boy" if all shots have been revenged at least once and Boss Baby doesn't initiate any shots at the neighborhood kids first. Return "Bad boy" if these conditions are not satisfied.

Note:
- Boss Baby doesn't need to shoot back before the next shots of the kids. He can shoot back whenever he wants as long as he doesn't shoot first.

Example:

Input	**Output**
SRSSRRR	Good boy
RSSRR	Bad boy
SSSRRRRS	Bad boy
SRRSSR	Bad boy
SSRSRRR	Good boy

Explanation:
In the first example, the first shot “S” has been avenged by the second action. The next two shots “SS” have been avenged by the following retaliation shots “RRR”.

In the second example, the first action is revenge “R”, which makes Boss Baby a bad boy as he shouldn’t shoot first.

In the third example, the first three shots “SSS” are avenged by **at least 3** shots “RRRR”. However, the last shot has no revenge, hence making Boss Baby a bad boy.

Problem 2: Superman's Chicken Rescue

Description:
In a one-dimensional world, Superman needs to protect chickens from a heavy rainstorm using a roof of limited length. Given the positions of chickens and the length of the roof Superman can carry, determine the maximum number of chickens Superman can protect.

Input:
The input consists of two integers n and k (1 <= n,k <= 1,000,000), where n represents the number of chickens and k denotes the length of the roof Superman can carry. The next line contains n integers (1 <= x <= 1,000,000,000) representing the positions of the chickens on the one-dimensional axis.

Output:
Output a single integer, denoting the maximum number of chickens Superman can protect with the given roof length.

Note:
- Superman can position the roof starting at any point on the axis.
- The roof can cover chickens whose positions are within k units from its starting point. [p, p+k)
- It's not required to cover all chickens, but to maximize the number of chickens protected.
- It’s guaranteed that the given positions of the chickens will be sorted from lowest to highest.

Example:

Input	**Output**
5 5 2 5 10 12 15	2
6 10 1 11 30 34 35 37	4

Explanation:
In the first example, superman can position the roof starting at position 2 (roof at 2 - 6), covering chickens at positions 2 and 5. Thus, he can protect a maximum of 2 chickens.

In the second example, superman can position the roof starting at position 30 (roof at 30 - 39), covering chickens at positions 30, 34, 35, and 37. Thus, he can protect a maximum of 4 chickens.

Problem 3: Transaction Broadcasting and Monitoring Client

Description:
Your task is to develop a module/class in a programming language of your choice that interacts with an HTTP server. This client module will enable the broadcasting of a transaction and subsequently monitor its status until finalization.

Requirements:
Create a client module that is designed to be integrated into another application, with the following capabilities:

1. Broadcast Transaction: Construct a JSON payload and send a POST request to `https://mock-node-wgqbnxruha-as.a.run.app/broadcast`. The payload structure is as follows:

```
{
    "symbol": "string",         // Transaction symbol, e.g., BTC
    "price": uint64,            // Symbol price, e.g., 100000
    "timestamp": uint64       // Timestamp of price retrieval
}
```

```
Example payload and server response:
// Payload
{
    "symbol": "ETH",
    "price": 4500,
    "timestamp": 1678912345
}

// Server response
{
    "tx_hash": "e97ae379d666eed7576bf285c451baf9f0edba93ce718bf15b06c8a85d07b8d1"
}
```

2. Transaction Status Monitoring: Utilize the transaction hash obtained from the response to periodically issue GET requests to `https://mock-node-wgqbnxruha-as.a.run.app/check/<tx_hash>`. The response will be plaintext indicating the transaction status, which can be one of the following:
- `CONFIRMED`: Transaction has been processed and confirmed
- `FAILED`: Transaction failed to process
- `PENDING`: Transaction is awaiting processing
- `DNE`: Transaction does not exist

```
An example response is shown below:
{
    "tx_status": "CONFIRMED"
}
```

3. Documentation: Provide clear documentation and an example script that implements your client module. The documentation should explain how to use and integrate it, along with your strategies for handling each transaction status.