Problem 362: Design Hit Counter

Problem Information

Difficulty: Medium
Acceptance Rate: 69.36%
Paid Only: Yes
Tags: Array, Binary Search, Design, Queue, Data Stream

Problem Description

Design a hit counter which counts the number of hits received in the past `5` minutes (i.e., the past `300` seconds).

Your system should accept a `timestamp` parameter (**in seconds** granularity), and you may assume that calls are being made to the system in chronological order (i.e., `timestamp` is monotonically increasing). Several hits may arrive roughly at the same time.

Implement the `HitCounter` class:

* `HitCounter()` Initializes the object of the hit counter system. * `void hit(int timestamp)` Records a hit that happened at `timestamp` (**in seconds**). Several hits may happen at the same `timestamp`. * `int getHits(int timestamp)` Returns the number of hits in the past 5 minutes from `timestamp` (i.e., the past `300` seconds).

Example 1:

Input ["HitCounter", "hit", "hit", "hit", "getHits", "hit", "getHits", "getHits"] [[], [1], [2], [3], [4], [300], [300], [301]] **Output** [null, null, null, null, 3, null, 4, 3] **Explanation** HitCounter hitCounter = new HitCounter(); hitCounter.hit(1); // hit at timestamp 1. hitCounter.hit(2); // hit at timestamp 2. hitCounter.hit(3); // hit at timestamp 3. hitCounter.getHits(4); // get hits at timestamp 4, return 3. hitCounter.hit(300); // hit at timestamp 300. hitCounter.getHits(300); // get hits at timestamp 300, return 4. hitCounter.getHits(301); // get hits at timestamp 301, return 3.

Constraints:

* `1 <= timestamp <= 2 * 109` * All the calls are being made to the system in chronological order (i.e., `timestamp` is monotonically increasing). * At most `300` calls will be made to `hit` and `getHits`.

Follow up: What if the number of hits per second could be huge? Does your design scale?

Code Snippets

C++:

```
class HitCounter {
public:
HitCounter() {

}

void hit(int timestamp) {

}

int getHits(int timestamp) {

}
};

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter* obj = new HitCounter();
* obj->hit(timestamp);
* int param_2 = obj->getHits(timestamp);
*/
```

Java:

```
class HitCounter {

public HitCounter() {

}

public void hit(int timestamp) {
}

public int getHits(int timestamp) {

}
}

/**
* Your HitCounter object will be instantiated and called as such:
* HitCounter obj = new HitCounter();
* obj.hit(timestamp);
* int param_2 = obj.getHits(timestamp);
*/
```

Python3:

```
class HitCounter:

def __init__(self):



def hit(self, timestamp: int) -> None:



def getHits(self, timestamp: int) -> int:




# Your HitCounter object will be instantiated and called as such:
# obj = HitCounter()
# obj.hit(timestamp)
# param_2 = obj.getHits(timestamp)
```