i {} 5 ⊙ □

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).

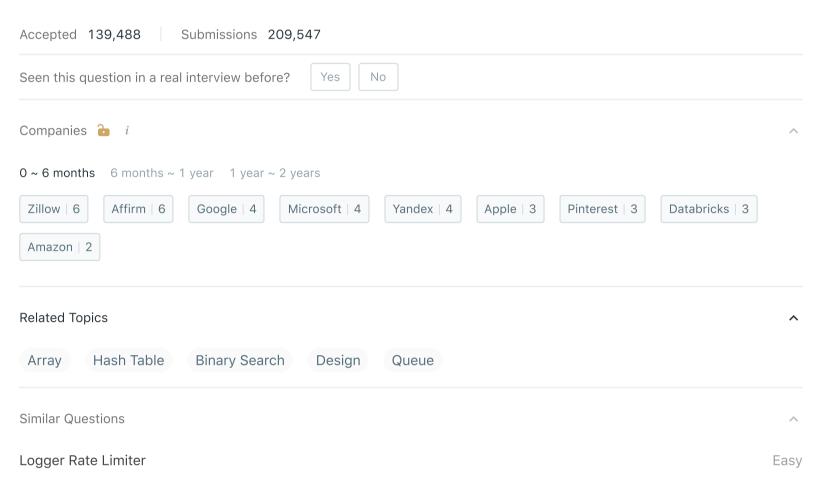
Example 1:

```
["HitCounter", "hit", "hit", "getHits", "hit", "getHits"]
[[], [1], [2], [3], [4], [300], [300], [301]]
Output
[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 <= $2 * 10^9$
- 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?



```
1 ▼ class HitCounter {
         private int total;
         private Deque<Pair<Integer, Integer>> hits;
         /** Initialize your data structure here. */
6 ▼
         public HitCounter() {
             // Initialize total to 0
             this.total = 0;
10
             this.hits = new LinkedList<Pair<Integer, Integer>>();
11
12
13 ▼
         /** Record a hit.
14
             @param timestamp - The current timestamp (in seconds granularity). */
15 ▼
         public void hit(int timestamp) {
16 ▼
             if (this.hits.isEmpty() || this.hits.getLast().getKey() != timestamp) {
17
                 // Insert the new timestamp with count = 1
18
                 this.hits.add(new Pair<Integer, Integer>(timestamp, 1));
19 ▼
             } else {
20
                 // Update the count of latest timestamp by incrementing the count by 1
21
22
                 // Obtain the current count of the latest timestamp
23
                 int prevCount = this.hits.getLast().getValue();
24
                 // Remove the last pair of (timestamp, count) from the deque
25
                 this.hits.removeLast();
26
                 // Insert a new pair of (timestamp, updated count) in the deque
27
                 this.hits.add(new Pair<Integer, Integer>(timestamp, prevCount + 1));
28
             // Increment total
29
30
             this.total++;
31
32
33 ▼
         /** Return the number of hits in the past 5 minutes.
34
             @param timestamp - The current timestamp (in seconds granularity). */
35 ▼
         public int getHits(int timestamp) {
36 ▼
             while (!this.hits.isEmpty()) {
37
                 int diff = timestamp - this.hits.getFirst().getKey();
38 ▼
                 if (diff >= 300) {
39
                     // Decrement total by the count of the oldest timestamp
40
                     this.total -= this.hits.getFirst().getValue();
41
                     this.hits.removeFirst();
42
43
                 else break;
44
45
             return this.total;
46
47
```

Autocomplete

i Java