362. Design Hit Counter

Medium 1575 138 Add to List Share

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:

- $1 <= timestamp <= 2 * 10^9$
- 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`.

$$\begin{tikzcd} \boxed{1} \end{tikzcd}$$

$$\begin{tikzcd} \boxed{1} & \boxed{2} \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} \boxed{1} & \boxed{2} \arrow[l] & \boxed{3} \arrow[l] \end{tikzcd}$$

$$\begin{tikzcd} \boxed{1} & \boxed{2} \arrow[l] & \boxed{3} \arrow[l] \end{tikzcd}$$

getHits(4) —> : 3

Time Complexity :
hit(int) -> O(1)
getHits(int) -> Avg O(1)

$$\begin{tikzcd} \boxed{1} & \boxed{2} \arrow[l] & \boxed{3} \arrow[l] & \boxed{300} \arrow[l] \end{tikzcd}$$

getHits(300) —> : 4

Old request to be removed because
the cache bound is 5 minutes = 300sec

$$\begin{tikzcd} \boxed{1} & \boxed{2} \arrow[l] & \boxed{3} \arrow[l] & \boxed{300} \arrow[l] \end{tikzcd}$$

getHits(301) —> : 3

239. Sliding Window Maximum

Hard 10509 356 Add to List Share

You are given an array of integers `nums`, there is a sliding window of size `k` which is moving from the very left of the array to the very right. You can only see the `k` numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window*.

Example 1:

```
Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Example 2:

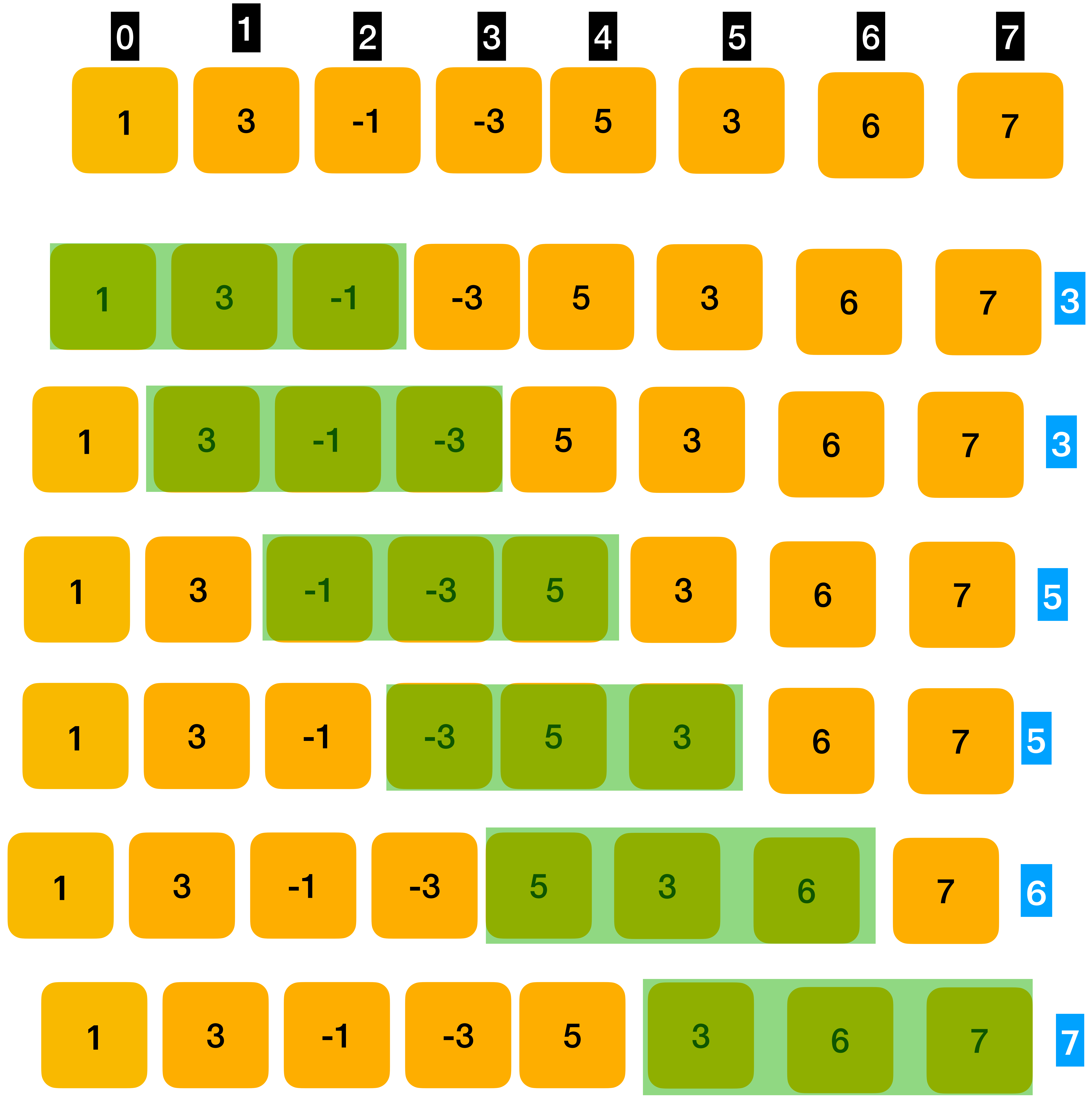
```
Input: nums = [1], k = 1
Output: [1]
```

Constraints:

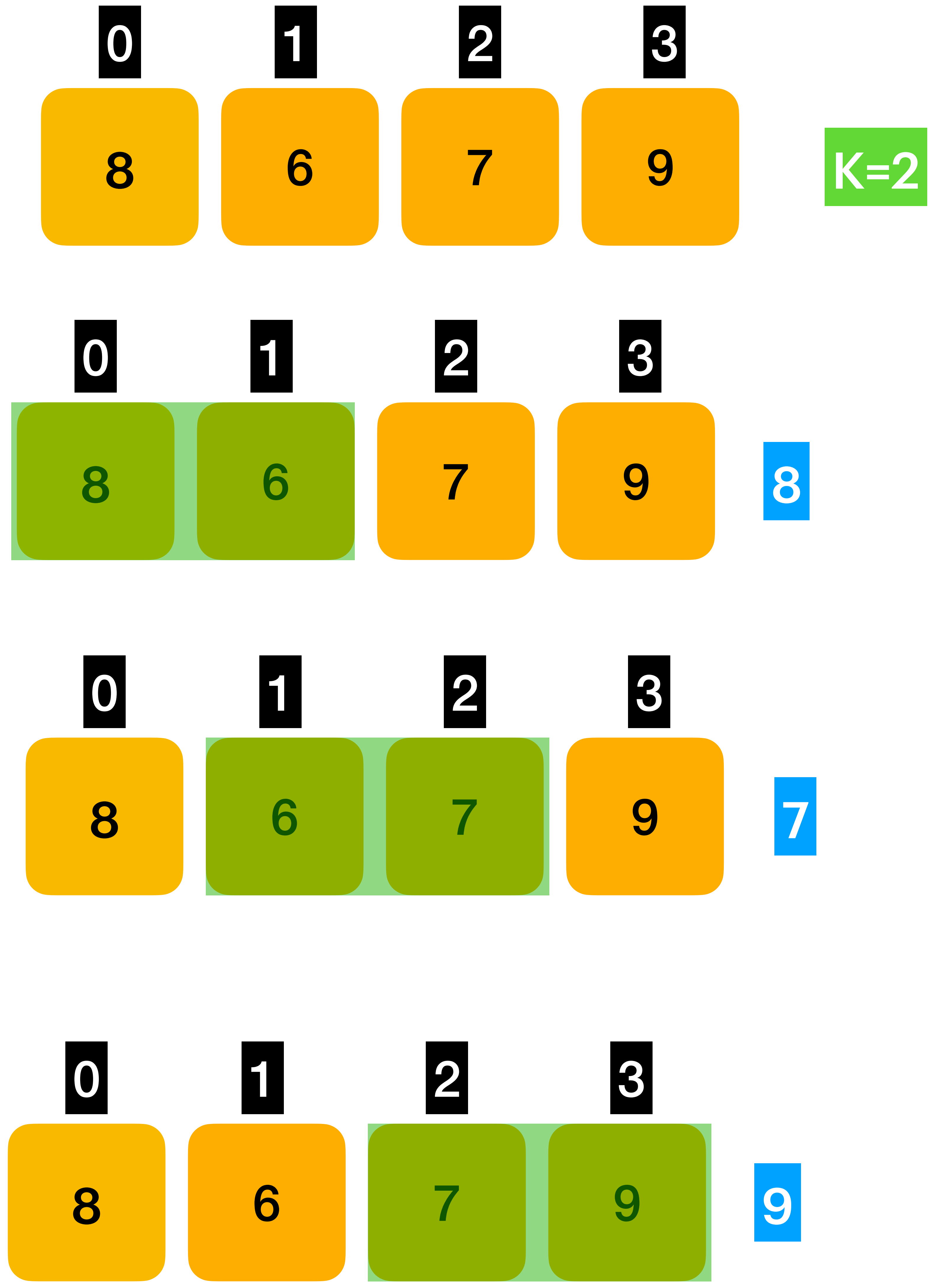
- `1 <= nums.length <= `10^5
- -10^4` <= nums[i] <= `10^4
- `1 <= k <= nums.length`

Return the max value in Each window

Window Size
K = 3

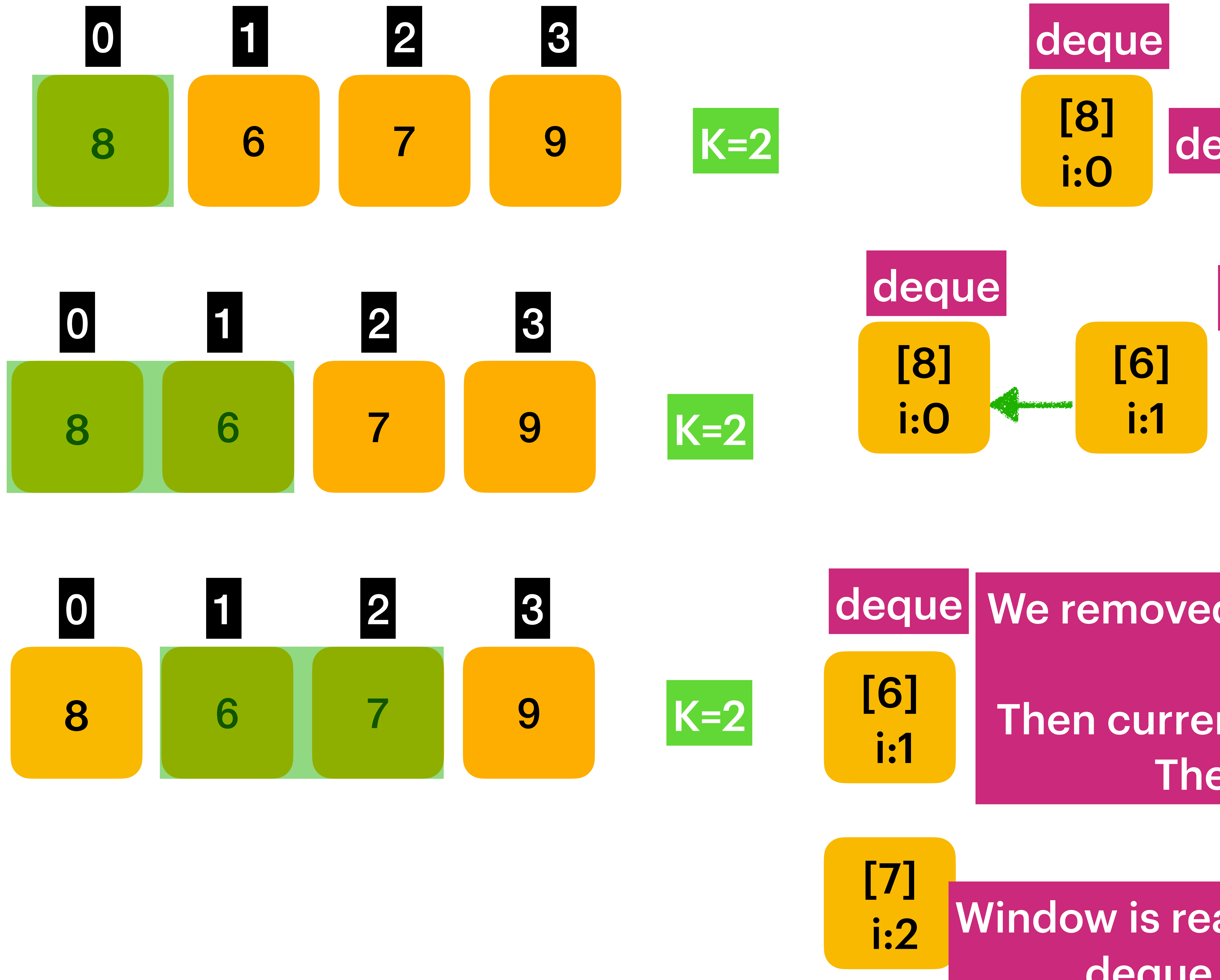


Expected Result : {3,3,5,5,6,7}



Expected Result : {8,7,9}

Algo :
We maintain the max element with in the window as a first element
In the Deque.
When ever we try to add element's we verify
if the current is > top of the deque then we remove iteratively.
Or
If the current < deque.top() then we just add.



deque is Empty so add to deque.

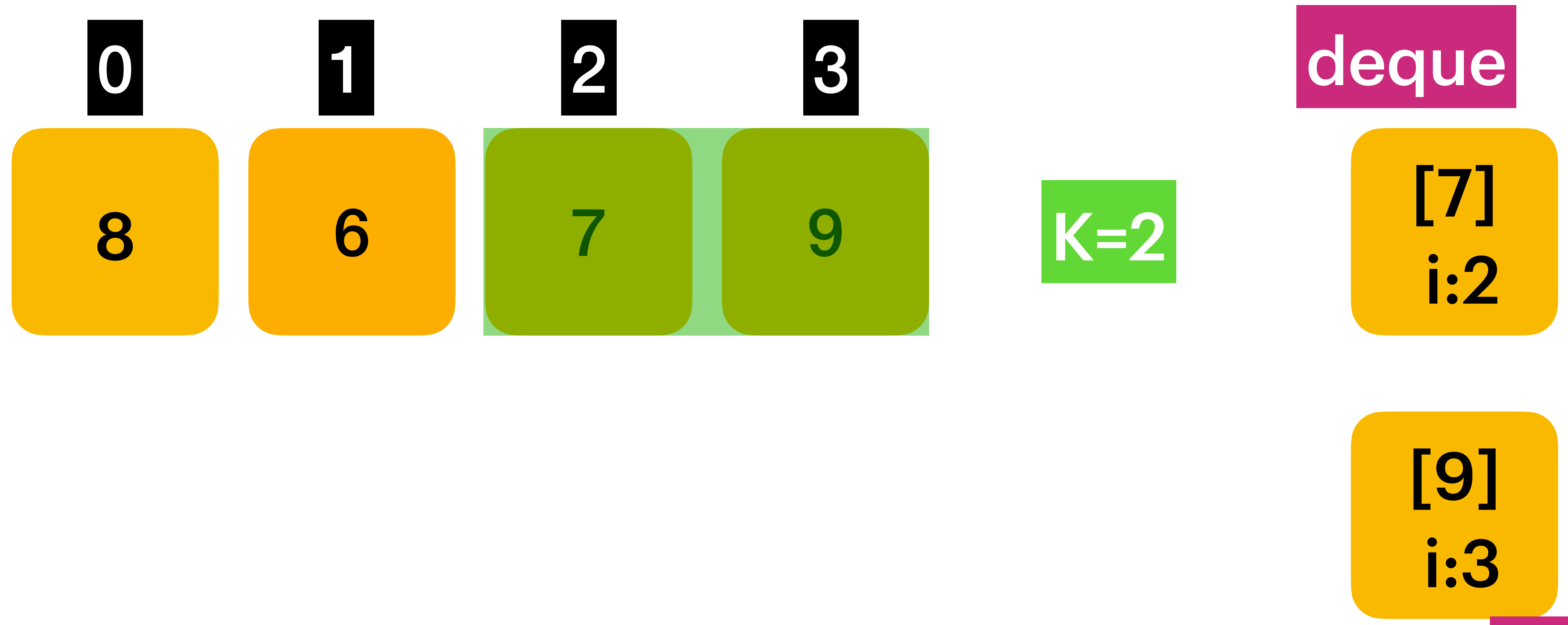
current[6] < top [8] so add to deque

Window is reached so return
deque.getFirst()
Max: 8

We removed 8 because its out of window
Limit :
Then current [7] > top [6] so remove it.[]
Then add the current [7]

Window is reached so return
deque.getFirst()
Max: 7

Algo :
We maintain the max element with in the window as a first element
In the Deque.
When ever we try to add element's we verify
if the current is > top of the deque then we remove iteratively.
Or
If the current < deque.top() then we just add.



Then current [9] > top [7] so remove it.

Then add the current [9]

Window is reached so return
deque.getFirst()
Max: 9

Time Complexity : O(n)
Each element will be visited twice:
Space Complexity : O(n) ~ O(1)