* Sliding Window Maximum

[1, 3, -1, -3, 5, 3, 6, 7] size = 3

3
3
5
5
6
7

use priorityQueue → maxQueue.

- use a double linked List (Deque)
  LinkedList / ArrayDeque

Keep decreasing sequence of values.
If index of top value / first value is out of window, pop it. When add a value, maintain the order by poping from end.

1 3 -1 -3 5 3 6 7

~~0~~

1

1 2

1 2 3

~~1 2 3 4~~

~~4 5~~

~~6~~

7

$O(n)$

$O(k)$

- use 2 arrays & in 2 traversals fill both; left to right & right to left.

	1	3	-1	-3	5	3	6	7
left	1	3	3	-3	5	5	6	7
right	3	3	-1	5	5	3	7	7

if $i \% k = 0$, put the same value. Else take the max of previous & current.

~~right~~ left

if $(i+1) \% k = 0$ put the same value. Else put the max of current & right.

$O(n)$

$O(n)$

for each place take the max of right, left relevant value.