1352. Product of the Last K Numbers

Design an algorithm that accepts a stream of integers and retrieves the product of the last `k` integers of the stream.

Implement the `ProductOfNumbers` class:

- `ProductOfNumbers()` Initializes the object with an empty stream.
- `void add(int num)` Appends the integer `num` to the stream.
- `int getProduct(int k)` Returns the product of the last `k` numbers in the current list. You can assume that always the current list has at least `k` numbers.

The test cases are generated so that, at any time, the product of any contiguous sequence of numbers will fit into a single 32-bit integer without overflowing.

Example:

Input
```
["ProductOfNumbers","add","add","add","add","add","getProduct","getProduct","getPro
duct","add","getProduct"]
[[],[3],[0],[2],[5],[4],[2],[3],[4],[8],[2]]
```

Output
```
[null,null,null,null,null,null,20,40,0,null,32]
```

Explanation
```
ProductOfNumbers productOfNumbers = new ProductOfNumbers();
productOfNumbers.add(3);        // [3]
productOfNumbers.add(0);        // [3,0]
productOfNumbers.add(2);        // [3,0,2]
productOfNumbers.add(5);        // [3,0,2,5]
productOfNumbers.add(4);        // [3,0,2,5,4]
productOfNumbers.getProduct(2); // return 20. The product of the last 2 numbers is
5 * 4 = 20
productOfNumbers.getProduct(3); // return 40. The product of the last 3 numbers is
2 * 5 * 4 = 40
productOfNumbers.getProduct(4); // return 0. The product of the last 4 numbers is 0
* 2 * 5 * 4 = 0
productOfNumbers.add(8);        // [3,0,2,5,4,8]
productOfNumbers.getProduct(2); // return 32. The product of the last 2 numbers is
4 * 8 = 32
```

Constraints:

- `0 <= num <= 100`
- `1 <= k <= 4 * 104`
- At most `4 * 104` calls will be made to `add` and `getProduct`.
- The product of the stream at any point in time will fit in a **32-bit** integer.

Follow-up: Can you implement **both** `GetProduct` and `Add` to work in `O(1)` time complexity instead of `O(k)` time complexity?

1352. Product of the Last K Numbers

```
/*
   Prefix product
   Overall time complexity: O(n+m)
   Overall space complexity: O(2n)
   n: #add queries
   m: #getProduct queries
*/
typedef unsigned long long ll;
typedef std::vector<ll> vll;

class ProductOfNumbers{
   private:
      vll prefix_zeros,prefix_prod;

   public:
      // Time complexity: O(1)
      // Space complexity: O(1)
      ProductOfNumbers(){
         prefix_zeros.push_back(0);
         prefix_prod.push_back(1);
      }

      // Time complexity: O(1)
      // Space complexity: O(2n)
      void add(int num){
         ll prev_prod=prefix_prod.back();
         ll prev_zeros=prefix_zeros.back();
         if(num==0){
            prefix_prod.push_back(1);
            prefix_zeros.push_back(prev_zeros+1);
         }
         else{
            prefix_prod.push_back(prev_prod*num*1ll);
            prefix_zeros.push_back(prev_zeros);
         }
      }
```

```
     // Time complexity: O(1)
    // Space complexity: O(1)
    int getProduct(int k){
        int left=prefix_prod.size()-k-1;
        int right=prefix_prod.size()-1;
        return (int)(prefix_zeros[right]-prefix_zeros[left]!=0?0:prefix_prod[right]/prefix_prod[left]);
    }
};
```