# Problem 1352: Product of the Last K Numbers

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

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","getProduct","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 * 10

4

At most

4 * 10

4

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?

## Code Snippets

**C++:**

```
class ProductOfNumbers {
public:
ProductOfNumbers() {

}

void add(int num) {

}

int getProduct(int k) {

}
};

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers* obj = new ProductOfNumbers();
* obj->add(num);
* int param_2 = obj->getProduct(k);
*/
```

**Java:**

```
class ProductOfNumbers {

public ProductOfNumbers() {

}
```

```
public void add(int num) {

}

public int getProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = new ProductOfNumbers();
* obj.add(num);
* int param_2 = obj.getProduct(k);
*/
```

**Python3:**

```
class ProductOfNumbers:

def __init__(self):


def add(self, num: int) -> None:


def getProduct(self, k: int) -> int:




# Your ProductOfNumbers object will be instantiated and called as such:
# obj = ProductOfNumbers()
# obj.add(num)
# param_2 = obj.getProduct(k)
```

**Python:**

```
class ProductOfNumbers(object):

def __init__(self):
```

```
def add(self, num):
"""
:type num: int
:rtype: None
"""


def getProduct(self, k):
"""
:type k: int
:rtype: int
"""



# Your ProductOfNumbers object will be instantiated and called as such:
# obj = ProductOfNumbers()
# obj.add(num)
# param_2 = obj.getProduct(k)
```

**JavaScript:**

```
var ProductOfNumbers = function() {

};

/**
* @param {number} num
* @return {void}
*/
ProductOfNumbers.prototype.add = function(num) {

};

/**
* @param {number} k
* @return {number}
*/
ProductOfNumbers.prototype.getProduct = function(k) {
```

```
};

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = new ProductOfNumbers()
* obj.add(num)
* var param_2 = obj.getProduct(k)
*/
```

**TypeScript:**

```
class ProductOfNumbers {
constructor() {

}

add(num: number): void {

}

getProduct(k: number): number {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = new ProductOfNumbers()
* obj.add(num)
* var param_2 = obj.getProduct(k)
*/
```

**C#:**

```
public class ProductOfNumbers {

public ProductOfNumbers() {

}

public void Add(int num) {
```

```
}

public int GetProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = new ProductOfNumbers();
* obj.Add(num);
* int param_2 = obj.GetProduct(k);
*/
```

**C:**

```
typedef struct {

} ProductOfNumbers;


ProductOfNumbers* productOfNumbersCreate() {

}

void productOfNumbersAdd(ProductOfNumbers* obj, int num) {

}

int productOfNumbersGetProduct(ProductOfNumbers* obj, int k) {

}

void productOfNumbersFree(ProductOfNumbers* obj) {

}

/**
* Your ProductOfNumbers struct will be instantiated and called as such:
```

```
* ProductOfNumbers* obj = productOfNumbersCreate();
* productOfNumbersAdd(obj, num);

* int param_2 = productOfNumbersGetProduct(obj, k);

* productOfNumbersFree(obj);
*/
```

**Go:**

```
type ProductOfNumbers struct {

}


func Constructor() ProductOfNumbers {

}


func (this *ProductOfNumbers) Add(num int) {

}


func (this *ProductOfNumbers) GetProduct(k int) int {

}


/**
* Your ProductOfNumbers object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(num);
* param_2 := obj.GetProduct(k);
*/
```

**Kotlin:**

```
class ProductOfNumbers() {

fun add(num: Int) {
```

```
}

fun getProduct(k: Int): Int {

}

}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = ProductOfNumbers()
* obj.add(num)
* var param_2 = obj.getProduct(k)
*/
```

**Swift:**

```
class ProductOfNumbers {

init() {

}

func add(_ num: Int) {

}

func getProduct(_ k: Int) -> Int {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* let obj = ProductOfNumbers()
* obj.add(num)
* let ret_2: Int = obj.getProduct(k)
*/
```

**Rust:**

```
struct ProductOfNumbers {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl ProductOfNumbers {

fn new() -> Self {

}

fn add(&self, num: i32) {

}

fn get_product(&self, k: i32) -> i32 {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* let obj = ProductOfNumbers::new();
* obj.add(num);
* let ret_2: i32 = obj.get_product(k);
*/
```

**Ruby:**

```
class ProductOfNumbers
def initialize()

end


=begin
:type num: Integer
```

```
:rtype: Void
=end
def add(num)

end


=begin
:type k: Integer
:rtype: Integer
=end
def get_product(k)

end



end

# Your ProductOfNumbers object will be instantiated and called as such:
# obj = ProductOfNumbers.new()
# obj.add(num)
# param_2 = obj.get_product(k)
```

**PHP:**

```
class ProductOfNumbers {
/**
*/
function __construct() {

}

/**
* @param Integer $num
* @return NULL
*/
function add($num) {

}

/**
* @param Integer $k
```

```
* @return Integer
*/
function getProduct($k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* $obj = ProductOfNumbers();
* $obj->add($num);
* $ret_2 = $obj->getProduct($k);
*/
```

**Dart:**

```
class ProductOfNumbers {

ProductOfNumbers() {

}

void add(int num) {

}

int getProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = ProductOfNumbers();
* obj.add(num);
* int param2 = obj.getProduct(k);
*/
```

**Scala:**

```
class ProductOfNumbers() {
```

```
def add(num: Int): Unit = {

}

def getProduct(k: Int): Int = {

}

}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* val obj = new ProductOfNumbers()
* obj.add(num)
* val param_2 = obj.getProduct(k)
*/
```

**Elixir:**

```
defmodule ProductOfNumbers do
@spec init_() :: any
def init_() do

end

@spec add(num :: integer) :: any
def add(num) do

end

@spec get_product(k :: integer) :: integer
def get_product(k) do

end
end

# Your functions will be called as such:
# ProductOfNumbers.init_()
# ProductOfNumbers.add(num)
# param_2 = ProductOfNumbers.get_product(k)

# ProductOfNumbers.init_ will be called before every test case, in which you
```

```
can do some necessary initializations.
```

**Erlang:**

```
-spec product_of_numbers_init_() -> any().
product_of_numbers_init_() ->
.

-spec product_of_numbers_add(Num :: integer()) -> any().
product_of_numbers_add(Num) ->
.

-spec product_of_numbers_get_product(K :: integer()) -> integer().
product_of_numbers_get_product(K) ->
.


%% Your functions will be called as such:
%% product_of_numbers_init_(),
%% product_of_numbers_add(Num),
%% Param_2 = product_of_numbers_get_product(K),

%% product_of_numbers_init_ will be called before every test case, in which
you can do some necessary initializations.
```

**Racket:**

```
(define product-of-numbers%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add num)
)
; get-product : exact-integer? -> exact-integer?
(define/public (get-product k)
)))

;; Your product-of-numbers% object will be instantiated and called as such:
;; (define obj (new product-of-numbers%))
;; (send obj add num)
```

```
;; (define param_2 (send obj get-product k))
```

## Solutions

### C++ Solution:

```
/*
* Problem: Product of the Last K Numbers
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class ProductOfNumbers {
public:
ProductOfNumbers() {

}

void add(int num) {

}

int getProduct(int k) {

}
};

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers* obj = new ProductOfNumbers();
* obj->add(num);
* int param_2 = obj->getProduct(k);
*/
```

### Java Solution:

```
/**
* Problem: Product of the Last K Numbers
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class ProductOfNumbers {

public ProductOfNumbers() {

}

public void add(int num) {

}

public int getProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = new ProductOfNumbers();
* obj.add(num);
* int param_2 = obj.getProduct(k);
*/
```

### Python3 Solution:

```
"""
Problem: Product of the Last K Numbers
Difficulty: Medium
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
```

```
"""

class ProductOfNumbers:

def __init__(self):


def add(self, num: int) -> None:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class ProductOfNumbers(object):

def __init__(self):


def add(self, num):
"""
:type num: int
:rtype: None
"""


def getProduct(self, k):
"""
:type k: int
:rtype: int
"""



# Your ProductOfNumbers object will be instantiated and called as such:
# obj = ProductOfNumbers()
# obj.add(num)
# param_2 = obj.getProduct(k)
```

**JavaScript Solution:**

```
/**
* Problem: Product of the Last K Numbers
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/


var ProductOfNumbers = function() {

};

/**
* @param {number} num
* @return {void}
*/
ProductOfNumbers.prototype.add = function(num) {

};

/**
* @param {number} k
* @return {number}
*/
ProductOfNumbers.prototype.getProduct = function(k) {

};

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = new ProductOfNumbers()
* obj.add(num)
* var param_2 = obj.getProduct(k)
*/
```

### TypeScript Solution:

```
/**
* Problem: Product of the Last K Numbers
```

```
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class ProductOfNumbers {
constructor() {

}

add(num: number): void {

}

getProduct(k: number): number {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = new ProductOfNumbers()
* obj.add(num)
* var param_2 = obj.getProduct(k)
*/
```

**C# Solution:**

```
/*
* Problem: Product of the Last K Numbers
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class ProductOfNumbers {

public ProductOfNumbers() {

}

public void Add(int num) {

}

public int GetProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = new ProductOfNumbers();
* obj.Add(num);
* int param_2 = obj.GetProduct(k);
*/
```

### C Solution:

```
/*
* Problem: Product of the Last K Numbers
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} ProductOfNumbers;
```

```
ProductOfNumbers* productOfNumbersCreate() {

}

void productOfNumbersAdd(ProductOfNumbers* obj, int num) {

}

int productOfNumbersGetProduct(ProductOfNumbers* obj, int k) {

}

void productOfNumbersFree(ProductOfNumbers* obj) {

}

/**
* Your ProductOfNumbers struct will be instantiated and called as such:
* ProductOfNumbers* obj = productOfNumbersCreate();
* productOfNumbersAdd(obj, num);

* int param_2 = productOfNumbersGetProduct(obj, k);

* productOfNumbersFree(obj);
*/
```

**Go Solution:**

```
// Problem: Product of the Last K Numbers
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type ProductOfNumbers struct {

}
```

```
func Constructor() ProductOfNumbers {

}


func (this *ProductOfNumbers) Add(num int) {

}


func (this *ProductOfNumbers) GetProduct(k int) int {

}


/**
* Your ProductOfNumbers object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(num);
* param_2 := obj.GetProduct(k);
*/
```

**Kotlin Solution:**

```
class ProductOfNumbers() {

fun add(num: Int) {

}

fun getProduct(k: Int): Int {

}

}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* var obj = ProductOfNumbers()
* obj.add(num)
```

```
* var param_2 = obj.getProduct(k)
*/
```

**Swift Solution:**

```
class ProductOfNumbers {

init() {

}

func add(_ num: Int) {

}

func getProduct(_ k: Int) -> Int {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* let obj = ProductOfNumbers()
* obj.add(num)
* let ret_2: Int = obj.getProduct(k)
*/
```

**Rust Solution:**

```
// Problem: Product of the Last K Numbers
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct ProductOfNumbers {

}
```

```
/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl ProductOfNumbers {

fn new() -> Self {

}

fn add(&self, num: i32) {

}

fn get_product(&self, k: i32) -> i32 {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* let obj = ProductOfNumbers::new();
* obj.add(num);
* let ret_2: i32 = obj.get_product(k);
*/
```

**Ruby Solution:**

```
class ProductOfNumbers
def initialize()

end


=begin
:type num: Integer
:rtype: Void
=end
def add(num)
```

```
end


=begin
:type k: Integer
:rtype: Integer
=end
def get_product(k)

end



end

# Your ProductOfNumbers object will be instantiated and called as such:
# obj = ProductOfNumbers.new()
# obj.add(num)
# param_2 = obj.get_product(k)
```

**PHP Solution:**

```
class ProductOfNumbers {
/**
*/
function __construct() {

}

/**
* @param Integer $num
* @return NULL
*/
function add($num) {

}

/**
* @param Integer $k
* @return Integer
*/
```

```
function getProduct($k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* $obj = ProductOfNumbers();
* $obj->add($num);
* $ret_2 = $obj->getProduct($k);
*/
```

**Dart Solution:**

```
class ProductOfNumbers {

ProductOfNumbers() {

}

void add(int num) {

}

int getProduct(int k) {

}
}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* ProductOfNumbers obj = ProductOfNumbers();
* obj.add(num);
* int param2 = obj.getProduct(k);
*/
```

**Scala Solution:**

```
class ProductOfNumbers() {

def add(num: Int): Unit = {
```

```
}

def getProduct(k: Int): Int = {

}

}

/**
* Your ProductOfNumbers object will be instantiated and called as such:
* val obj = new ProductOfNumbers()
* obj.add(num)
* val param_2 = obj.getProduct(k)
*/
```

**Elixir Solution:**

```
defmodule ProductOfNumbers do
@spec init_() :: any
def init_() do

end

@spec add(num :: integer) :: any
def add(num) do

end

@spec get_product(k :: integer) :: integer
def get_product(k) do

end
end

# Your functions will be called as such:
# ProductOfNumbers.init_()
# ProductOfNumbers.add(num)
# param_2 = ProductOfNumbers.get_product(k)

# ProductOfNumbers.init_ will be called before every test case, in which you
```

```
can do some necessary initializations.
```

### Erlang Solution:

```
-spec product_of_numbers_init_() -> any().
product_of_numbers_init_() ->
.

-spec product_of_numbers_add(Num :: integer()) -> any().
product_of_numbers_add(Num) ->
.

-spec product_of_numbers_get_product(K :: integer()) -> integer().
product_of_numbers_get_product(K) ->
.


%% Your functions will be called as such:
%% product_of_numbers_init_(),
%% product_of_numbers_add(Num),
%% Param_2 = product_of_numbers_get_product(K),

%% product_of_numbers_init_ will be called before every test case, in which
you can do some necessary initializations.
```

### Racket Solution:

```
(define product-of-numbers%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add num)
)
; get-product : exact-integer? -> exact-integer?
(define/public (get-product k)
)))

;; Your product-of-numbers% object will be instantiated and called as such:
;; (define obj (new product-of-numbers%))
```

```
;; (send obj add num)
;; (define param_2 (send obj get-product k))
```