# Problem 1756: Design Most Recently Used Queue

## Problem Information

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

## Problem Description

Design a queue-like data structure that moves the most recently used element to the end of the queue.

Implement the

MRUQueue

class:

MRUQueue(int n)

constructs the

MRUQueue

with

n

elements:

[1,2,3,...,n]

.

int fetch(int k)

moves the

k

th

element

(1-indexed)

to the end of the queue and returns it.

Example 1:

Input:

["MRUQueue", "fetch", "fetch", "fetch", "fetch"] [[8], [3], [5], [2], [8]]

Output:

[null, 3, 6, 2, 2]

Explanation:

MRUQueue mRUQueue = new MRUQueue(8); // Initializes the queue to [1,2,3,4,5,6,7,8].
mRUQueue.fetch(3); // Moves the 3

rd

element (3) to the end of the queue to become [1,2,4,5,6,7,8,3] and returns it.
mRUQueue.fetch(5); // Moves the 5

th

element (6) to the end of the queue to become [1,2,4,5,7,8,3,6] and returns it.
mRUQueue.fetch(2); // Moves the 2

nd

element (2) to the end of the queue to become [1,4,5,7,8,3,6,2] and returns it.
mRUQueue.fetch(8); // The 8

th

element (2) is already at the end of the queue so just return it.

Constraints:

1 <= n <= 2000

1 <= k <= n

At most

2000

calls will be made to

fetch

.

Follow up:

Finding an

O(n)

algorithm per

fetch

is a bit easy. Can you find an algorithm with a better complexity for each

fetch

call?

## Code Snippets

**C++:**

```
class MRUQueue {
public:
MRUQueue(int n) {

}

int fetch(int k) {

}
};

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue* obj = new MRUQueue(n);
* int param_1 = obj->fetch(k);
*/
```

**Java:**

```
class MRUQueue {

public MRUQueue(int n) {

}

public int fetch(int k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = new MRUQueue(n);
* int param_1 = obj.fetch(k);
*/
```

**Python3:**

```
class MRUQueue:

def __init__(self, n: int):


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




# Your MRUQueue object will be instantiated and called as such:
# obj = MRUQueue(n)
# param_1 = obj.fetch(k)
```

**Python:**

```
class MRUQueue(object):

def __init__(self, n):
"""
:type n: int
"""


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




# Your MRUQueue object will be instantiated and called as such:
# obj = MRUQueue(n)
# param_1 = obj.fetch(k)
```

**JavaScript:**

```
/**
* @param {number} n
*/
var MRUQueue = function(n) {
```

```
};

/**
* @param {number} k
* @return {number}
*/
MRUQueue.prototype.fetch = function(k) {

};

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = new MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

**TypeScript:**

```
class MRUQueue {
constructor(n: number) {

}

fetch(k: number): number {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = new MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

**C#:**

```
public class MRUQueue {

public MRUQueue(int n) {

}
```

```
public int Fetch(int k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = new MRUQueue(n);
* int param_1 = obj.Fetch(k);
*/
```

**C:**

```
typedef struct {

} MRUQueue;


MRUQueue* mRUQueueCreate(int n) {

}

int mRUQueueFetch(MRUQueue* obj, int k) {

}

void mRUQueueFree(MRUQueue* obj) {

}

/**
* Your MRUQueue struct will be instantiated and called as such:
* MRUQueue* obj = mRUQueueCreate(n);
* int param_1 = mRUQueueFetch(obj, k);

* mRUQueueFree(obj);
*/
```

**Go:**

```
type MRUQueue struct {

}


func Constructor(n int) MRUQueue {

}


func (this *MRUQueue) Fetch(k int) int {

}


/**
* Your MRUQueue object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Fetch(k);
*/
```

**Kotlin:**

```
class MRUQueue(n: Int) {

fun fetch(k: Int): Int {

}

}

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

**Swift:**

```

class MRUQueue {
```

```
init(_ n: Int) {

}

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

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* let obj = MRUQueue(n)
* let ret_1: Int = obj.fetch(k)
*/
```

**Rust:**

```
struct MRUQueue {

}


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

fn new(n: i32) -> Self {

}

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

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* let obj = MRUQueue::new(n);
* let ret_1: i32 = obj.fetch(k);
```

```
*/
```

**Ruby:**

```
class MRUQueue

=begin
:type n: Integer
=end
def initialize(n)

end


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

end


end

# Your MRUQueue object will be instantiated and called as such:
# obj = MRUQueue.new(n)
# param_1 = obj.fetch(k)
```

**PHP:**

```
class MRUQueue {
/**
* @param Integer $n
*/
function __construct($n) {

}

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

```
*/
function fetch($k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* $obj = MRUQueue($n);
* $ret_1 = $obj->fetch($k);
*/
```

**Dart:**

```
class MRUQueue {

MRUQueue(int n) {

}

int fetch(int k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = MRUQueue(n);
* int param1 = obj.fetch(k);
*/
```

**Scala:**

```
class MRUQueue(_n: Int) {

def fetch(k: Int): Int = {

}

}

/**
```

```
* Your MRUQueue object will be instantiated and called as such:
* val obj = new MRUQueue(n)
* val param_1 = obj.fetch(k)
*/
```

**Elixir:**

```
defmodule MRUQueue do
@spec init_(n :: integer) :: any
def init_(n) do

end

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

end
end

# Your functions will be called as such:
# MRUQueue.init_(n)
# param_1 = MRUQueue.fetch(k)

# MRUQueue.init_ will be called before every test case, in which you can do
some necessary initializations.
```

**Erlang:**

```
-spec mru_queue_init_(N :: integer()) -> any().
mru_queue_init_(N) ->
.

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


%% Your functions will be called as such:
%% mru_queue_init_(N),
%% Param_1 = mru_queue_fetch(K),

%% mru_queue_init_ will be called before every test case, in which you can do
```

```
some necessary initializations.
```

**Racket:**

```
(define mru-queue%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; fetch : exact-integer? -> exact-integer?
(define/public (fetch k)
)))

;; Your mru-queue% object will be instantiated and called as such:
;; (define obj (new mru-queue% [n n]))
;; (define param_1 (send obj fetch k))
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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 MRUQueue {
public:
MRUQueue(int n) {

}

int fetch(int k) {
```

```
}
};

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue* obj = new MRUQueue(n);
* int param_1 = obj->fetch(k);
*/
```

**Java Solution:**

```
/**
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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 MRUQueue {

public MRUQueue(int n) {

}

public int fetch(int k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = new MRUQueue(n);
* int param_1 = obj.fetch(k);
*/
```

**Python3 Solution:**

```
"""
Problem: Design Most Recently Used Queue
Difficulty: Medium
Tags: array, linked_list, queue

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 MRUQueue:

def __init__(self, n: int):


def fetch(self, k: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class MRUQueue(object):

def __init__(self, n):
"""
:type n: int
"""


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




# Your MRUQueue object will be instantiated and called as such:
# obj = MRUQueue(n)
# param_1 = obj.fetch(k)
```

**JavaScript Solution:**

```javascript
/**
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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
*/

/**
* @param {number} n
*/
var MRUQueue = function(n) {

};

/**
* @param {number} k
* @return {number}
*/
MRUQueue.prototype.fetch = function(k) {

};

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = new MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

### TypeScript Solution:

```typescript
/**
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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 MRUQueue {
constructor(n: number) {

}

fetch(k: number): number {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = new MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

**C# Solution:**

```
/*
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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 MRUQueue {

public MRUQueue(int n) {

}

public int Fetch(int k) {

}
}

/**
```

```
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = new MRUQueue(n);
* int param_1 = obj.Fetch(k);
*/
```

### C Solution:

```
/*
* Problem: Design Most Recently Used Queue
* Difficulty: Medium
* Tags: array, linked_list, queue
*
* 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 {

} MRUQueue;


MRUQueue* mRUQueueCreate(int n) {

}

int mRUQueueFetch(MRUQueue* obj, int k) {

}

void mRUQueueFree(MRUQueue* obj) {

}

/**
* Your MRUQueue struct will be instantiated and called as such:
* MRUQueue* obj = mRUQueueCreate(n);
* int param_1 = mRUQueueFetch(obj, k);
```

```
* mRUQueueFree(obj);
*/
```

### Go Solution:

```
// Problem: Design Most Recently Used Queue
// Difficulty: Medium
// Tags: array, linked_list, queue
//
// 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 MRUQueue struct {

}


func Constructor(n int) MRUQueue {

}


func (this *MRUQueue) Fetch(k int) int {

}


/**
* Your MRUQueue object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Fetch(k);
*/
```

### Kotlin Solution:

```
class MRUQueue(n: Int) {

fun fetch(k: Int): Int {
```

```
}

}

/**
* Your MRUQueue object will be instantiated and called as such:
* var obj = MRUQueue(n)
* var param_1 = obj.fetch(k)
*/
```

**Swift Solution:**

```
class MRUQueue {

init(_ n: Int) {

}

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

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* let obj = MRUQueue(n)
* let ret_1: Int = obj.fetch(k)
*/
```

**Rust Solution:**

```
// Problem: Design Most Recently Used Queue
// Difficulty: Medium
// Tags: array, linked_list, queue
//
// 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 MRUQueue {
```

```
}


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

fn new(n: i32) -> Self {

}

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

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* let obj = MRUQueue::new(n);
* let ret_1: i32 = obj.fetch(k);
*/
```

**Ruby Solution:**

```
class MRUQueue

=begin
:type n: Integer
=end
def initialize(n)

end


=begin
:type k: Integer
:rtype: Integer
=end
```

```
def fetch(k)

end


end

# Your MRUQueue object will be instantiated and called as such:
# obj = MRUQueue.new(n)
# param_1 = obj.fetch(k)
```

**PHP Solution:**

```
class MRUQueue {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @param Integer $k
* @return Integer
*/
function fetch($k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* $obj = MRUQueue($n);
* $ret_1 = $obj->fetch($k);
*/
```

**Dart Solution:**

```
class MRUQueue {

MRUQueue(int n) {
```

```
}

int fetch(int k) {

}
}

/**
* Your MRUQueue object will be instantiated and called as such:
* MRUQueue obj = MRUQueue(n);
* int param1 = obj.fetch(k);
*/
```

### Scala Solution:

```
class MRUQueue(_n: Int) {

def fetch(k: Int): Int = {

}

}

/**
* Your MRUQueue object will be instantiated and called as such:
* val obj = new MRUQueue(n)
* val param_1 = obj.fetch(k)
*/
```

### Elixir Solution:

```
defmodule MRUQueue do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec fetch(k :: integer) :: integer
def fetch(k) do
```

```
end
end

# Your functions will be called as such:
# MRUQueue.init_(n)
# param_1 = MRUQueue.fetch(k)

# MRUQueue.init_ will be called before every test case, in which you can do
some necessary initializations.
```

### Erlang Solution:

```
-spec mru_queue_init_(N :: integer()) -> any().
mru_queue_init_(N) ->
.

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


%% Your functions will be called as such:
%% mru_queue_init_(N),
%% Param_1 = mru_queue_fetch(K),

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

### Racket Solution:

```
(define mru-queue%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; fetch : exact-integer? -> exact-integer?
(define/public (fetch k)
)))
```

```
;; Your mru-queue% object will be instantiated and called as such:
;; (define obj (new mru-queue% [n n]))
;; (define param_1 (send obj fetch k))
```