Problem 703: Kth Largest Element in a Stream

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are part of a university admissions office and need to keep track of the

kth

highest test score from applicants in real-time. This helps to determine cut-off marks for interviews and admissions dynamically as new applicants submit their scores.

You are tasked to implement a class which, for a given integer

k

, maintains a stream of test scores and continuously returns the

k

th highest test score

after

a new score has been submitted. More specifically, we are looking for the

k

th highest score in the sorted list of all scores.

Implement the

KthLargest

class:

KthLargest(int k, int[] nums)

Initializes the object with the integer

k

and the stream of test scores

nums

.

int add(int val)

Adds a new test score

val

to the stream and returns the element representing the

k

th

largest element in the pool of test scores so far.

Example 1:

Input:

["KthLargest", "add", "add", "add", "add", "add"]

[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]

Output:

[null, 4, 5, 5, 8, 8]

Explanation:

KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]);

kthLargest.add(3); // return 4

kthLargest.add(5); // return 5

kthLargest.add(10); // return 5

kthLargest.add(9); // return 8

kthLargest.add(4); // return 8

Example 2:

Input:

["KthLargest", "add", "add", "add", "add"]

[[4, [7, 7, 7, 7, 8, 3]], [2], [10], [9], [9]]

Output:

[null, 7, 7, 7, 8]

Explanation:

KthLargest kthLargest = new KthLargest(4, [7, 7, 7, 7, 8, 3]);

kthLargest.add(2); // return 7

kthLargest.add(10); // return 7

kthLargest.add(9); // return 7

kthLargest.add(9); // return 8

Constraints:

0 <= nums.length <= 10

4

1 <= k <= nums.length + 1

-10

4

<= nums[i] <= 10

4

-10

4

<= val <= 10

4

At most

10

4

calls will be made to

add

.

Code Snippets

C++:

```
class KthLargest {
public:
KthLargest(int k, vector<int>& nums) {

}

int add(int val) {

}
};

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest* obj = new KthLargest(k, nums);
* int param_1 = obj->add(val);
*/
```

Java:

```
class KthLargest {

public KthLargest(int k, int[] nums) {

}

public int add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = new KthLargest(k, nums);
* int param_1 = obj.add(val);
*/
```

Python3:

```
class KthLargest:

def __init__(self, k: int, nums: List[int]):


def add(self, val: int) -> int:



# Your KthLargest object will be instantiated and called as such:
# obj = KthLargest(k, nums)
# param_1 = obj.add(val)
```

Python:

```
class KthLargest(object):

def __init__(self, k, nums):
"""
:type k: int
:type nums: List[int]
"""


def add(self, val):
"""
:type val: int
:rtype: int
"""



# Your KthLargest object will be instantiated and called as such:
# obj = KthLargest(k, nums)
# param_1 = obj.add(val)
```

JavaScript:

```
/**
* @param {number} k
* @param {number[]} nums
*/
var KthLargest = function(k, nums) {
```

```
};

/**
* @param {number} val
* @return {number}
*/
KthLargest.prototype.add = function(val) {

};

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = new KthLargest(k, nums)
* var param_1 = obj.add(val)
*/
```

TypeScript:

```
class KthLargest {
constructor(k: number, nums: number[]) {

}

add(val: number): number {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = new KthLargest(k, nums)
* var param_1 = obj.add(val)
*/
```

C#:

```
public class KthLargest {

public KthLargest(int k, int[] nums) {

}
```

```
public int Add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = new KthLargest(k, nums);
* int param_1 = obj.Add(val);
*/
```

C:

```
typedef struct {

} KthLargest;


KthLargest* kthLargestCreate(int k, int* nums, int numsSize) {

}

int kthLargestAdd(KthLargest* obj, int val) {

}

void kthLargestFree(KthLargest* obj) {

}

/**
* Your KthLargest struct will be instantiated and called as such:
* KthLargest* obj = kthLargestCreate(k, nums, numsSize);
* int param_1 = kthLargestAdd(obj, val);

* kthLargestFree(obj);
*/
```

Go:

```
type KthLargest struct {

}


func Constructor(k int, nums []int) KthLargest {

}


func (this *KthLargest) Add(val int) int {

}


/**
* Your KthLargest object will be instantiated and called as such:
* obj := Constructor(k, nums);
* param_1 := obj.Add(val);
*/
```

Kotlin:

```
class KthLargest(k: Int, nums: IntArray) {

fun add(`val`: Int): Int {

}

}

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = KthLargest(k, nums)
* var param_1 = obj.add(`val`)
*/
```

Swift:

```
class KthLargest {
```

```
init(_ k: Int, _ nums: [Int]) {

}

func add(_ val: Int) -> Int {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* let obj = KthLargest(k, nums)
* let ret_1: Int = obj.add(val)
*/
```

Rust:

```
struct KthLargest {

}


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

fn new(k: i32, nums: Vec<i32>) -> Self {

}

fn add(&self, val: i32) -> i32 {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* let obj = KthLargest::new(k, nums);
* let ret_1: i32 = obj.add(val);
```

```
*/
```

Ruby:

```
class KthLargest

=begin
:type k: Integer
:type nums: Integer[]
=end
def initialize(k, nums)

end


=begin
:type val: Integer
:rtype: Integer
=end
def add(val)

end


end

# Your KthLargest object will be instantiated and called as such:
# obj = KthLargest.new(k, nums)
# param_1 = obj.add(val)
```

PHP:

```
class KthLargest {
/**
* @param Integer $k
* @param Integer[] $nums
*/
function __construct($k, $nums) {

}

/**
```

```
* @param Integer $val
* @return Integer
*/
function add($val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* $obj = KthLargest($k, $nums);
* $ret_1 = $obj->add($val);
*/
```

Dart:

```
class KthLargest {

KthLargest(int k, List<int> nums) {

}

int add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = KthLargest(k, nums);
* int param1 = obj.add(val);
*/
```

Scala:

```
class KthLargest(_k: Int, _nums: Array[Int]) {

def add(`val`: Int): Int = {

}

}
```

```
/**
* Your KthLargest object will be instantiated and called as such:
* val obj = new KthLargest(k, nums)
* val param_1 = obj.add(`val`)
*/
```

Elixir:

```
defmodule KthLargest do
@spec init_(k :: integer, nums :: [integer]) :: any
def init_(k, nums) do

end

@spec add(val :: integer) :: integer
def add(val) do

end
end

# Your functions will be called as such:
# KthLargest.init_(k, nums)
# param_1 = KthLargest.add(val)

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

Erlang:

```
-spec kth_largest_init_(K :: integer(), Nums :: [integer()]) -> any().
kth_largest_init_(K, Nums) ->
.

-spec kth_largest_add(Val :: integer()) -> integer().
kth_largest_add(Val) ->
.


%% Your functions will be called as such:
%% kth_largest_init_(K, Nums),
%% Param_1 = kth_largest_add(Val),
```

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

Racket:

```
(define kth-largest%
(class object%
(super-new)

; k : exact-integer?
; nums : (listof exact-integer?)
(init-field
k
nums)

; add : exact-integer? -> exact-integer?
(define/public (add val)
)))

;; Your kth-largest% object will be instantiated and called as such:
;; (define obj (new kth-largest% [k k] [nums nums]))
;; (define param_1 (send obj add val))
```

Solutions

C++ Solution:

```
/*
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class KthLargest {
public:
```

```
KthLargest(int k, vector<int>& nums) {

}

int add(int val) {

}
};

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest* obj = new KthLargest(k, nums);
* int param_1 = obj->add(val);
*/
```

Java Solution:

```
/**
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class KthLargest {

public KthLargest(int k, int[] nums) {

}

public int add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = new KthLargest(k, nums);
```

```
* int param_1 = obj.add(val);
*/
```

Python3 Solution:

```
"""
Problem: Kth Largest Element in a Stream
Difficulty: Easy
Tags: tree, sort, search, queue, heap

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

class KthLargest:

def __init__(self, k: int, nums: List[int]):


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

Python Solution:

```
class KthLargest(object):

def __init__(self, k, nums):
"""
:type k: int
:type nums: List[int]
"""


def add(self, val):
"""
:type val: int
:rtype: int
"""
```

```
# Your KthLargest object will be instantiated and called as such:
# obj = KthLargest(k, nums)
# param_1 = obj.add(val)
```

JavaScript Solution:

```
/**
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number} k
* @param {number[]} nums
*/
var KthLargest = function(k, nums) {

};

/**
* @param {number} val
* @return {number}
*/
KthLargest.prototype.add = function(val) {

};

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = new KthLargest(k, nums)
* var param_1 = obj.add(val)
*/
```

TypeScript Solution:

```
/**
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

class KthLargest {
constructor(k: number, nums: number[]) {

}

add(val: number): number {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = new KthLargest(k, nums)
* var param_1 = obj.add(val)
*/
```

C# Solution:

```
/*
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

public class KthLargest {

public KthLargest(int k, int[] nums) {
```

```
}

public int Add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = new KthLargest(k, nums);
* int param_1 = obj.Add(val);
*/
```

C Solution:

```
/*
* Problem: Kth Largest Element in a Stream
* Difficulty: Easy
* Tags: tree, sort, search, queue, heap
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/




typedef struct {

} KthLargest;


KthLargest* kthLargestCreate(int k, int* nums, int numsSize) {

}

int kthLargestAdd(KthLargest* obj, int val) {

}
```

```
void kthLargestFree(KthLargest* obj) {

}

/**
* Your KthLargest struct will be instantiated and called as such:
* KthLargest* obj = kthLargestCreate(k, nums, numsSize);
* int param_1 = kthLargestAdd(obj, val);

* kthLargestFree(obj);
*/
```

Go Solution:

```
// Problem: Kth Largest Element in a Stream
// Difficulty: Easy
// Tags: tree, sort, search, queue, heap
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

type KthLargest struct {

}


func Constructor(k int, nums []int) KthLargest {

}


func (this *KthLargest) Add(val int) int {

}


/**
* Your KthLargest object will be instantiated and called as such:
* obj := Constructor(k, nums);
* param_1 := obj.Add(val);
```

```
*/
```

Kotlin Solution:

```
class KthLargest(k: Int, nums: IntArray) {

fun add(`val`: Int): Int {

}

}

/**
* Your KthLargest object will be instantiated and called as such:
* var obj = KthLargest(k, nums)
* var param_1 = obj.add(`val`)
*/
```

Swift Solution:

```
class KthLargest {

init(_ k: Int, _ nums: [Int]) {

}

func add(_ val: Int) -> Int {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* let obj = KthLargest(k, nums)
* let ret_1: Int = obj.add(val)
*/
```

Rust Solution:

```
// Problem: Kth Largest Element in a Stream
// Difficulty: Easy
// Tags: tree, sort, search, queue, heap
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

struct KthLargest {

}



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

fn new(k: i32, nums: Vec<i32>) -> Self {

}

fn add(&self, val: i32) -> i32 {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* let obj = KthLargest::new(k, nums);
* let ret_1: i32 = obj.add(val);
*/
```

Ruby Solution:

```
class KthLargest

=begin
:type k: Integer
:type nums: Integer[]
=end
```

```
def initialize(k, nums)

end


=begin
:type val: Integer
:rtype: Integer
=end
def add(val)

end



end

# Your KthLargest object will be instantiated and called as such:
# obj = KthLargest.new(k, nums)
# param_1 = obj.add(val)
```

PHP Solution:

```
class KthLargest {
/**
* @param Integer $k
* @param Integer[] $nums
*/
function __construct($k, $nums) {

}

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

}
}

/**
```

```
* Your KthLargest object will be instantiated and called as such:
* $obj = KthLargest($k, $nums);
* $ret_1 = $obj->add($val);
*/
```

Dart Solution:

```
class KthLargest {

KthLargest(int k, List<int> nums) {

}

int add(int val) {

}
}

/**
* Your KthLargest object will be instantiated and called as such:
* KthLargest obj = KthLargest(k, nums);
* int param1 = obj.add(val);
*/
```

Scala Solution:

```
class KthLargest(_k: Int, _nums: Array[Int]) {

def add(`val`: Int): Int = {

}

}

/**
* Your KthLargest object will be instantiated and called as such:
* val obj = new KthLargest(k, nums)
* val param_1 = obj.add(`val`)
*/
```

Elixir Solution:

```
defmodule KthLargest do
@spec init_(k :: integer, nums :: [integer]) :: any
def init_(k, nums) do

end

@spec add(val :: integer) :: integer
def add(val) do

end
end

# Your functions will be called as such:
# KthLargest.init_(k, nums)
# param_1 = KthLargest.add(val)

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

Erlang Solution:

```
-spec kth_largest_init_(K :: integer(), Nums :: [integer()]) -> any().
kth_largest_init_(K, Nums) ->
.

-spec kth_largest_add(Val :: integer()) -> integer().
kth_largest_add(Val) ->
.


%% Your functions will be called as such:
%% kth_largest_init_(K, Nums),
%% Param_1 = kth_largest_add(Val),

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

Racket Solution:

```
(define kth-largest%
(class object%
(super-new)
```

```
; k : exact-integer?
; nums : (listof exact-integer?)
(init-field
k
nums)

; add : exact-integer? -> exact-integer?
(define/public (add val)
)))

;; Your kth-largest% object will be instantiated and called as such:
;; (define obj (new kth-largest% [k k] [nums nums]))
;; (define param_1 (send obj add val))
```