Problem 398: Random Pick Index

Problem Information

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

Problem Description

Given an integer array

nums

with possible

duplicates

, randomly output the index of a given

target

number. You can assume that the given target number must exist in the array.

Implement the

Solution

class:

Solution(int[] nums)

Initializes the object with the array

nums

.

int pick(int target)

Picks a random index

i

from

nums

where

nums[i] == target

. If there are multiple valid i's, then each index should have an equal probability of returning.

Example 1:

Input

["Solution", "pick", "pick", "pick"] [[[1, 2, 3, 3, 3]], [3], [1], [3]]

Output

[null, 4, 0, 2]

Explanation

Solution solution = new Solution([1, 2, 3, 3, 3]); solution.pick(3); // It should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning. solution.pick(1); // It should return 0. Since in the array only nums[0] is equal to 1. solution.pick(3); // It should return either index 2, 3, or 4 randomly. Each index should have equal probability of returning.

Constraints:

1 <= nums.length <= 2 * 10

4

-2

31

<= nums[i] <= 2

31

- 1

target

is an integer from

nums

.

At most

10

4

calls will be made to

pick

.

Code Snippets

C++:

```
class Solution {
public:
```

```
Solution(vector<int>& nums) {

}

int pick(int target) {

}
};

/**
* Your Solution object will be instantiated and called as such:
* Solution* obj = new Solution(nums);
* int param_1 = obj->pick(target);
*/
```

Java:

```
class Solution {

public Solution(int[] nums) {

}

public int pick(int target) {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(nums);
* int param_1 = obj.pick(target);
*/
```

Python3:

```
class Solution:

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


def pick(self, target: int) -> int:
```

```
# Your Solution object will be instantiated and called as such:
# obj = Solution(nums)
# param_1 = obj.pick(target)
```

Python:

```
class Solution(object):

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


def pick(self, target):
"""
:type target: int
:rtype: int
"""



# Your Solution object will be instantiated and called as such:
# obj = Solution(nums)
# param_1 = obj.pick(target)
```

JavaScript:

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

};

/**
* @param {number} target
* @return {number}
*/
```

```
Solution.prototype.pick = function(target) {

};

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(nums)
* var param_1 = obj.pick(target)
*/
```

TypeScript:

```
class Solution {
constructor(nums: number[]) {

}

pick(target: number): number {

}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(nums)
* var param_1 = obj.pick(target)
*/
```

C#:

```
public class Solution {

public Solution(int[] nums) {

}

public int Pick(int target) {

}
}

/**
```

```
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(nums);
* int param_1 = obj.Pick(target);
*/
```

C:

```
typedef struct {

} Solution;


Solution* solutionCreate(int* nums, int numsSize) {

}

int solutionPick(Solution* obj, int target) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(nums, numsSize);
* int param_1 = solutionPick(obj, target);

* solutionFree(obj);
*/
```

Go:

```
type Solution struct {

}
```

```
func Constructor(nums []int) Solution {

}


func (this *Solution) Pick(target int) int {

}


/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.Pick(target);
*/
```

Kotlin:

```
class Solution(nums: IntArray) {

fun pick(target: Int): Int {

}

}

/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(nums)
* var param_1 = obj.pick(target)
*/
```

Swift:

```
class Solution {

init(_ nums: [Int]) {

}

func pick(_ target: Int) -> Int {
```

```
}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution(nums)
* let ret_1: Int = obj.pick(target)
*/
```

Rust:

```
struct Solution {

}


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

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

}

fn pick(&self, target: i32) -> i32 {

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(nums);
* let ret_1: i32 = obj.pick(target);
*/
```

Ruby:

```
class Solution
```

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

end


=begin
:type target: Integer
:rtype: Integer
=end
def pick(target)

end



end

# Your Solution object will be instantiated and called as such:
# obj = Solution.new(nums)
# param_1 = obj.pick(target)
```

PHP:

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

}

/**
* @param Integer $target
* @return Integer
*/
function pick($target) {

}
}
```

```
/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($nums);
* $ret_1 = $obj->pick($target);
*/
```

Dart:

```
class Solution {

Solution(List<int> nums) {

}

int pick(int target) {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(nums);
* int param1 = obj.pick(target);
*/
```

Scala:

```
class Solution(_nums: Array[Int]) {

def pick(target: Int): Int = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(nums)
* val param_1 = obj.pick(target)
*/
```

Elixir:

```
defmodule Solution do
@spec init_(nums :: [integer]) :: any
def init_(nums) do

end

@spec pick(target :: integer) :: integer
def pick(target) do

end
end

# Your functions will be called as such:
# Solution.init_(nums)
# param_1 = Solution.pick(target)

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

Erlang:

```
-spec solution_init_(Nums :: [integer()]) -> any().
solution_init_(Nums) ->
.

-spec solution_pick(Target :: integer()) -> integer().
solution_pick(Target) ->
.


%% Your functions will be called as such:
%% solution_init_(Nums),
%% Param_1 = solution_pick(Target),

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

Racket:

```
(define solution%
(class object%
(super-new)
```

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

; pick : exact-integer? -> exact-integer?
(define/public (pick target)
)))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [nums nums]))
;; (define param_1 (send obj pick target))
```

Solutions

C++ Solution:

```
/*
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
Solution(vector<int>& nums) {

}

int pick(int target) {

}
};

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

```
* int param_1 = obj->pick(target);
*/
```

Java Solution:

```
/**
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {

public Solution(int[] nums) {

}

public int pick(int target) {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(nums);
* int param_1 = obj.pick(target);
*/
```

Python3 Solution:

```
"""
Problem: Random Pick Index
Difficulty: Medium
Tags: array, math, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) for hash map
"""

class Solution:

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


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

Python Solution:

```
class Solution(object):

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


def pick(self, target):
"""
:type target: int
:rtype: int
"""



# Your Solution object will be instantiated and called as such:
# obj = Solution(nums)
# param_1 = obj.pick(target)
```

JavaScript Solution:

```
/**
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
*/
var Solution = function(nums) {

};

/**
* @param {number} target
* @return {number}
*/
Solution.prototype.pick = function(target) {

};

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(nums)
* var param_1 = obj.pick(target)
*/
```

TypeScript Solution:

```
/**
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
constructor(nums: number[]) {
```

```
}

pick(target: number): number {

}
}

/**
* Your Solution object will be instantiated and called as such:
* var obj = new Solution(nums)
* var param_1 = obj.pick(target)
*/
```

C# Solution:

```
/*
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {

public Solution(int[] nums) {

}

public int Pick(int target) {

}
}

/**
* Your Solution object will be instantiated and called as such:
* Solution obj = new Solution(nums);
* int param_1 = obj.Pick(target);
*/
```

C Solution:

```
/*
* Problem: Random Pick Index
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} Solution;


Solution* solutionCreate(int* nums, int numsSize) {

}

int solutionPick(Solution* obj, int target) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(nums, numsSize);
* int param_1 = solutionPick(obj, target);

* solutionFree(obj);
*/
```

Go Solution:

```
// Problem: Random Pick Index
// Difficulty: Medium
// Tags: array, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type Solution struct {

}


func Constructor(nums []int) Solution {

}


func (this *Solution) Pick(target int) int {

}


/**
* Your Solution object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.Pick(target);
*/
```

Kotlin Solution:

```
class Solution(nums: IntArray) {

fun pick(target: Int): Int {

}

}

/**
* Your Solution object will be instantiated and called as such:
* var obj = Solution(nums)
```

```
* var param_1 = obj.pick(target)
*/
```

Swift Solution:

```
class Solution {

init(_ nums: [Int]) {

}

func pick(_ target: Int) -> Int {

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution(nums)
* let ret_1: Int = obj.pick(target)
*/
```

Rust Solution:

```
// Problem: Random Pick Index
// Difficulty: Medium
// Tags: array, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct Solution {

}



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

```
*/
impl Solution {

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

}

fn pick(&self, target: i32) -> i32 {

}
}

/**
* Your Solution object will be instantiated and called as such:
* let obj = Solution::new(nums);
* let ret_1: i32 = obj.pick(target);
*/
```

Ruby Solution:

```
class Solution

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

end


=begin
:type target: Integer
:rtype: Integer
=end
def pick(target)

end


end
```

```
# Your Solution object will be instantiated and called as such:
# obj = Solution.new(nums)
# param_1 = obj.pick(target)
```

PHP Solution:

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

}

/**
* @param Integer $target
* @return Integer
*/
function pick($target) {

}
}

/**
* Your Solution object will be instantiated and called as such:
* $obj = Solution($nums);
* $ret_1 = $obj->pick($target);
*/
```

Dart Solution:

```
class Solution {

Solution(List<int> nums) {

}

int pick(int target) {

}
}
```

```
/**
* Your Solution object will be instantiated and called as such:
* Solution obj = Solution(nums);
* int param1 = obj.pick(target);
*/
```

Scala Solution:

```
class Solution(_nums: Array[Int]) {

def pick(target: Int): Int = {

}

}

/**
* Your Solution object will be instantiated and called as such:
* val obj = new Solution(nums)
* val param_1 = obj.pick(target)
*/
```

Elixir Solution:

```
defmodule Solution do
@spec init_(nums :: [integer]) :: any
def init_(nums) do

end

@spec pick(target :: integer) :: integer
def pick(target) do

end
end

# Your functions will be called as such:
# Solution.init_(nums)
# param_1 = Solution.pick(target)
```

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

Erlang Solution:

```
-spec solution_init_(Nums :: [integer()]) -> any().
solution_init_(Nums) ->
.

-spec solution_pick(Target :: integer()) -> integer().
solution_pick(Target) ->
.


%% Your functions will be called as such:
%% solution_init_(Nums),
%% Param_1 = solution_pick(Target),

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

Racket Solution:

```
(define solution%
(class object%
(super-new)

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

; pick : exact-integer? -> exact-integer?
(define/public (pick target)
)))

;; Your solution% object will be instantiated and called as such:
;; (define obj (new solution% [nums nums]))
;; (define param_1 (send obj pick target))
```