# Problem 528: Random Pick with Weight

## Problem Information

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

## Problem Description

You are given a

0-indexed

array of positive integers

w

where

w[i]

describes the

weight

of the

i

th

index.

You need to implement the function

pickIndex()

, which

randomly

picks an index in the range

[0, w.length - 1]

(

inclusive

) and returns it. The

probability

of picking an index

i

is

w[i] / sum(w)

.

For example, if

w = [1, 3]

, the probability of picking index

0

is

1 / (1 + 3) = 0.25

(i.e.,

25%

), and the probability of picking index

1

is

3 / (1 + 3) = 0.75

(i.e.,

75%

).

Example 1:

Input

["Solution","pickIndex"] [[[1]],[]]

Output

[null,0]

Explanation

Solution solution = new Solution([1]); solution.pickIndex(); // return 0. The only option is to return 0 since there is only one element in w.

Example 2:

Input

["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"] [[[1,3]],[],[],[],[],[]]

Output

[null,1,1,1,1,0]

Explanation

Solution solution = new Solution([1, 3]); solution.pickIndex(); // return 1. It is returning the second element (index = 1) that has a probability of 3/4. solution.pickIndex(); // return 1 solution.pickIndex(); // return 1 solution.pickIndex(); // return 1 solution.pickIndex(); // return 0. It is returning the first element (index = 0) that has a probability of 1/4.

Since this is a randomization problem, multiple answers are allowed. All of the following outputs can be considered correct: [null,1,1,1,1,0] [null,1,1,1,1,1] [null,1,1,1,0,0] [null,1,1,1,0,1] [null,1,0,1,0,0] ...... and so on.

Constraints:

1 <= w.length <= 10

4

1 <= w[i] <= 10

5

pickIndex

will be called at most

10

4

times.

## Code Snippets

**C++:**

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

}

int pickIndex() {

}
};

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

**Java:**

```
class Solution {

public Solution(int[] w) {

}

public int pickIndex() {

}
}

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

**Python3:**

```
class Solution:

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

```

```
def pickIndex(self) -> int:



# Your Solution object will be instantiated and called as such:
# obj = Solution(w)
# param_1 = obj.pickIndex()
```

### Python:

```
class Solution(object):

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


def pickIndex(self):
"""
:rtype: int
"""



# Your Solution object will be instantiated and called as such:
# obj = Solution(w)
# param_1 = obj.pickIndex()
```

### JavaScript:

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

};

/**
* @return {number}
*/
Solution.prototype.pickIndex = function() {
};

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

### TypeScript:

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

}

pickIndex(): number {

}
}

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

### C#:

```
public class Solution {

public Solution(int[] w) {

}

public int PickIndex() {

}
}

/**
* Your Solution object will be instantiated and called as such:
```

```
* Solution obj = new Solution(w);
* int param_1 = obj.PickIndex();
*/
```

**C:**

```
typedef struct {

} Solution;


Solution* solutionCreate(int* w, int wSize) {

}

int solutionPickIndex(Solution* obj) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(w, wSize);
* int param_1 = solutionPickIndex(obj);

* solutionFree(obj);
*/
```

**Go:**

```
type Solution struct {

}


func Constructor(w []int) Solution {
}


func (this *Solution) PickIndex() int {

}


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

### Kotlin:

```
class Solution(w: IntArray) {

fun pickIndex(): Int {

}

}

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

### Swift:

```

class Solution {

init(_ w: [Int]) {

}

func pickIndex() -> Int {
}
}

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

### 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(w: Vec<i32>) -> Self {

}

fn pick_index(&self) -> i32 {

}
}

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

### Ruby:

```
class Solution

=begin
```

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

end



=begin
:rtype: Integer
=end
def pick_index()

end



end

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

### PHP:

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

}

/**
* @return Integer
*/
function pickIndex() {

}
}

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

### Dart:

```
class Solution {

Solution(List<int> w) {

}

int pickIndex() {

}
}

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

### Scala:

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

def pickIndex(): Int = {

}

}

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

### Elixir:

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

```
def init_(w) do

end

@spec pick_index() :: integer
def pick_index() do

end
end

# Your functions will be called as such:
# Solution.init_(w)
# param_1 = Solution.pick_index()

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

### Erlang:

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

-spec solution_pick_index() -> integer().
solution_pick_index() ->
.



%% Your functions will be called as such:
%% solution_init_(W),
%% Param_1 = solution_pick_index(),

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

### Racket:

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

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

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

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

## Solutions

### C++ Solution:

```
/*
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution {
public:
Solution(vector<int>& w) {

}

int pickIndex() {

}
};

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

```
*/
```

### Java Solution:

```
/**
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution {

public Solution(int[] w) {

}

public int pickIndex() {

}
}

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

### Python3 Solution:

```
"""
Problem: Random Pick with Weight
Difficulty: Medium
Tags: array, math, search

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 Solution:

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



def pickIndex(self) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):

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



def pickIndex(self):
"""
:rtype: int
"""





# Your Solution object will be instantiated and called as such:
# obj = Solution(w)
# param_1 = obj.pickIndex()
```

### JavaScript Solution:

```
/**
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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[]} w
*/
var Solution = function(w) {

};

/**
* @return {number}
*/
Solution.prototype.pickIndex = function() {

};

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

### TypeScript Solution:

```
/**
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution {
constructor(w: number[]) {

}

pickIndex(): number {
}
}

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

### C# Solution:

```
/*
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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 Solution {

public Solution(int[] w) {

}

public int PickIndex() {

}
}

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

### C Solution:

```
/*
* Problem: Random Pick with Weight
* Difficulty: Medium
* Tags: array, math, search
*
* 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 {

} Solution;


Solution* solutionCreate(int* w, int wSize) {

}

int solutionPickIndex(Solution* obj) {

}

void solutionFree(Solution* obj) {

}

/**
* Your Solution struct will be instantiated and called as such:
* Solution* obj = solutionCreate(w, wSize);
* int param_1 = solutionPickIndex(obj);

* solutionFree(obj);
*/
```

### Go Solution:

```
// Problem: Random Pick with Weight
// Difficulty: Medium
// Tags: array, math, search
//
// 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 Solution struct {

}



func Constructor(w []int) Solution {

}



func (this *Solution) PickIndex() int {

}



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

### Kotlin Solution:

```
class Solution(w: IntArray) {

fun pickIndex(): Int {

}

}

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

```
*/
```

## Swift Solution:

```
class Solution {

init(_ w: [Int]) {

}

func pickIndex() -> Int {

}
}

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

## Rust Solution:

```
// Problem: Random Pick with Weight
// Difficulty: Medium
// Tags: array, math, search
//
// 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 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(w: Vec<i32>) -> Self {

}

fn pick_index(&self) -> i32 {

}
}

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

### Ruby Solution:

```
class Solution

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

end



=begin
:rtype: Integer
=end
def pick_index()

end



end

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

### PHP Solution:

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

}

/**
* @return Integer
*/
function pickIndex() {

}
}

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

### Dart Solution:

```
class Solution {

Solution(List<int> w) {

}

int pickIndex() {

}
}

/**
* Your Solution object will be instantiated and called as such:
```

```
* Solution obj = Solution(w);
* int param1 = obj.pickIndex();
*/
```

### Scala Solution:

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

def pickIndex(): Int = {

}

}

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

### Elixir Solution:

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

end

@spec pick_index() :: integer
def pick_index() do

end
end

# Your functions will be called as such:
# Solution.init_(w)
# param_1 = Solution.pick_index()

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

### Erlang Solution:

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

-spec solution_pick_index() -> integer().
solution_pick_index() ->
.


%% Your functions will be called as such:
%% solution_init_(W),
%% Param_1 = solution_pick_index(),

%% 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)

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

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

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