# Problem 1394: Find Lucky Integer in an Array

## Problem Information

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

## Problem Description

Given an array of integers

arr

, a

lucky integer

is an integer that has a frequency in the array equal to its value.

Return

the largest

lucky integer

in the array

. If there is no

lucky integer

return

-1

.

Example 1:

Input:

arr = [2,2,3,4]

Output:

2

Explanation:

The only lucky number in the array is 2 because frequency[2] == 2.

Example 2:

Input:

arr = [1,2,2,3,3,3]

Output:

3

Explanation:

1, 2 and 3 are all lucky numbers, return the largest of them.

Example 3:

Input:

arr = [2,2,2,3,3]

Output:

-1

Explanation:

There are no lucky numbers in the array.

Constraints:

1 <= arr.length <= 500

1 <= arr[i] <= 500

## Code Snippets

### C++:

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

}
};
```

### Java:

```
class Solution {
public int findLucky(int[] arr) {

}
}
```

### Python3:

```
class Solution:
def findLucky(self, arr: List[int]) -> int:
```

### Python:

```
class Solution(object):
def findLucky(self, arr):
"""
:type arr: List[int]
```

```
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} arr
* @return {number}
*/
var findLucky = function(arr) {

};
```

**TypeScript:**

```
function findLucky(arr: number[]): number {

};
```

**C#:**

```
public class Solution {
public int FindLucky(int[] arr) {

}
}
```

**C:**

```
int findLucky(int* arr, int arrSize) {

}
```

**Go:**

```
func findLucky(arr []int) int {

}
```

**Kotlin:**

```
class Solution {
fun findLucky(arr: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func findLucky(_ arr: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn find_lucky(arr: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} arr
# @return {Integer}
def find_lucky(arr)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function findLucky($arr) {

}
}
```

**Dart:**

```
class Solution {
int findLucky(List<int> arr) {

}
}
```

**Scala:**

```
object Solution {
def findLucky(arr: Array[Int]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec find_lucky(arr :: [integer]) :: integer
def find_lucky(arr) do

end
end
```

**Erlang:**

```
-spec find_lucky(Arr :: [integer()]) -> integer().
find_lucky(Arr) ->
.
```

**Racket:**

```
(define/contract (find-lucky arr)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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:
int findLucky(vector<int>& arr) {

}
};
```

### Java Solution:

```
/**
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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 int findLucky(int[] arr) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find Lucky Integer in an Array
Difficulty: Easy
Tags: array, 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 findLucky(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def findLucky(self, arr):
"""
:type arr: List[int]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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[]} arr
* @return {number}
*/
var findLucky = function(arr) {

};
```

### TypeScript Solution:

```
/**
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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
*/

function findLucky(arr: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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 int FindLucky(int[] arr) {

}
}
```

### C Solution:

```
/*
* Problem: Find Lucky Integer in an Array
* Difficulty: Easy
* Tags: array, 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
```

```
*/

int findLucky(int* arr, int arrSize) {

}
```

### Go Solution:

```
// Problem: Find Lucky Integer in an Array
// Difficulty: Easy
// Tags: array, 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

func findLucky(arr []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun findLucky(arr: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func findLucky(_ arr: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Find Lucky Integer in an Array
// Difficulty: Easy
// Tags: array, 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

impl Solution {
pub fn find_lucky(arr: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def find_lucky(arr)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function findLucky($arr) {

}
}
```

### Dart Solution:

```
class Solution {
int findLucky(List<int> arr) {

}
}
```

### Scala Solution:

```
object Solution {
def findLucky(arr: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec find_lucky(arr :: [integer]) :: integer
def find_lucky(arr) do

end
end
```

### Erlang Solution:

```
-spec find_lucky(Arr :: [integer()]) -> integer().
find_lucky(Arr) ->
.
```

### Racket Solution:

```
(define/contract (find-lucky arr)
(-> (listof exact-integer?) exact-integer?)
)
```