# Problem 849: Maximize Distance to Closest Person

## Problem Information

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

## Problem Description

You are given an array representing a row of

seats

where

seats[i] = 1

represents a person sitting in the

i

th

seat, and

seats[i] = 0

represents that the

i

th

seat is empty

(0-indexed)

.

There is at least one empty seat, and at least one person sitting.

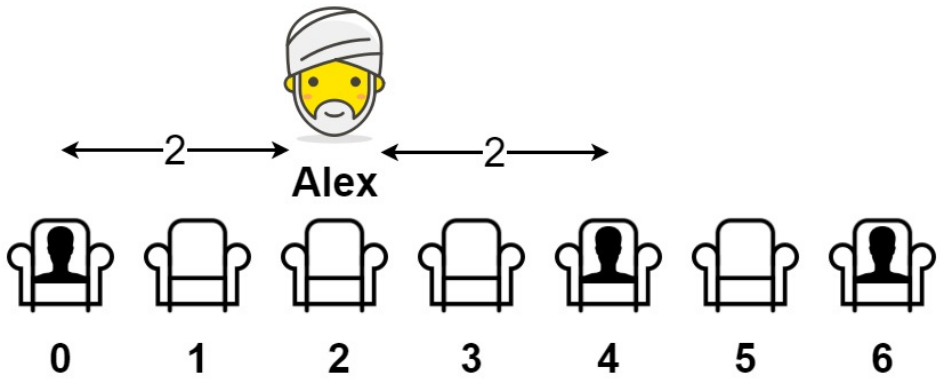
Alex wants to sit in the seat such that the distance between him and the closest person to him is maximized.

Return

that maximum distance to the closest person

.

Example 1:




Input:

seats = [1,0,0,0,1,0,1]

Output:

2

Explanation:

If Alex sits in the second open seat (i.e. seats[2]), then the closest person has distance 2. If Alex sits in any other open seat, the closest person has distance 1. Thus, the maximum distance to the closest person is 2.

Example 2:

Input:

seats = [1,0,0,0]

Output:

3

Explanation:

If Alex sits in the last seat (i.e. seats[3]), the closest person is 3 seats away. This is the maximum distance possible, so the answer is 3.

Example 3:

Input:

seats = [0,1]

Output:

1

Constraints:

2 <= seats.length <= 2 * 10

4

seats[i]

is

0

or

1

.

At least one seat is

empty

.

At least one seat is

occupied

.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

```
class Solution {
public int maxDistToClosest(int[] seats) {

}
}
```

**Python3:**

```
class Solution:
def maxDistToClosest(self, seats: List[int]) -> int:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function maxDistToClosest(seats: number[]): number {

};
```

**C#:**

```
public class Solution {
public int MaxDistToClosest(int[] seats) {

}
}
```

**C:**

```
int maxDistToClosest(int* seats, int seatsSize) {

}
```

**Go:**

```
func maxDistToClosest(seats []int) int {
```

```
}
```

**Kotlin:**

```
class Solution {
fun maxDistToClosest(seats: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func maxDistToClosest(_ seats: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_dist_to_closest(seats: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} seats
# @return {Integer}
def max_dist_to_closest(seats)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $seats
* @return Integer
*/
```

```
function maxDistToClosest($seats) {

}
}
```

**Dart:**

```
class Solution {
int maxDistToClosest(List<int> seats) {

}
}
```

**Scala:**

```
object Solution {
def maxDistToClosest(seats: Array[Int]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_dist_to_closest(seats :: [integer]) :: integer
def max_dist_to_closest(seats) do

end
end
```

**Erlang:**

```
-spec max_dist_to_closest(Seats :: [integer()]) -> integer().
max_dist_to_closest(Seats) ->
.
```

**Racket:**

```
(define/contract (max-dist-to-closest seats)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
* Tags: array
*
* 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:
int maxDistToClosest(vector<int>& seats) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
* Tags: array
*
* 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 int maxDistToClosest(int[] seats) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximize Distance to Closest Person
Difficulty: Medium
Tags: array

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 maxDistToClosest(self, seats: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
* Tags: array
*
* 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[]} seats
* @return {number}
*/
var maxDistToClosest = function(seats) {
```

```
};
```

### TypeScript Solution:

```
/**
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
* Tags: array
*
* 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
*/

function maxDistToClosest(seats: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
* Tags: array
*
* 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 int MaxDistToClosest(int[] seats) {

}
}
```

### C Solution:

```
/*
* Problem: Maximize Distance to Closest Person
* Difficulty: Medium
```

```
* Tags: array
*
* 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
*/

int maxDistToClosest(int* seats, int seatsSize) {

}
```

### Go Solution:

```
// Problem: Maximize Distance to Closest Person
// Difficulty: Medium
// Tags: array
//
// 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

func maxDistToClosest(seats []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxDistToClosest(seats: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func maxDistToClosest(_ seats: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Maximize Distance to Closest Person
// Difficulty: Medium
// Tags: array
//
// 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

impl Solution {
pub fn max_dist_to_closest(seats: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} seats
# @return {Integer}
def max_dist_to_closest(seats)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $seats
* @return Integer
*/
function maxDistToClosest($seats) {

}
}
```

### Dart Solution:

```
class Solution {
int maxDistToClosest(List<int> seats) {
```

```
}
}
```

### Scala Solution:

```
object Solution {
def maxDistToClosest(seats: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec max_dist_to_closest(seats :: [integer]) :: integer
def max_dist_to_closest(seats) do

end
end
```

### Erlang Solution:

```
-spec max_dist_to_closest(Seats :: [integer()]) -> integer().
max_dist_to_closest(Seats) ->
.
```

### Racket Solution:

```
(define/contract (max-dist-to-closest seats)
(-> (listof exact-integer?) exact-integer?)
)
```