# Problem 419: Battleships in a Board

## Problem Information

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

## Problem Description

Given an

m x n

matrix

board

where each cell is a battleship

'X'

or empty

'.'

, return

the number of the

battleships

on

board

.

Battleships

can only be placed horizontally or vertically on

board

. In other words, they can only be made of the shape

1 x k

(

1

row,

k

columns) or

k x 1

(

k

rows,

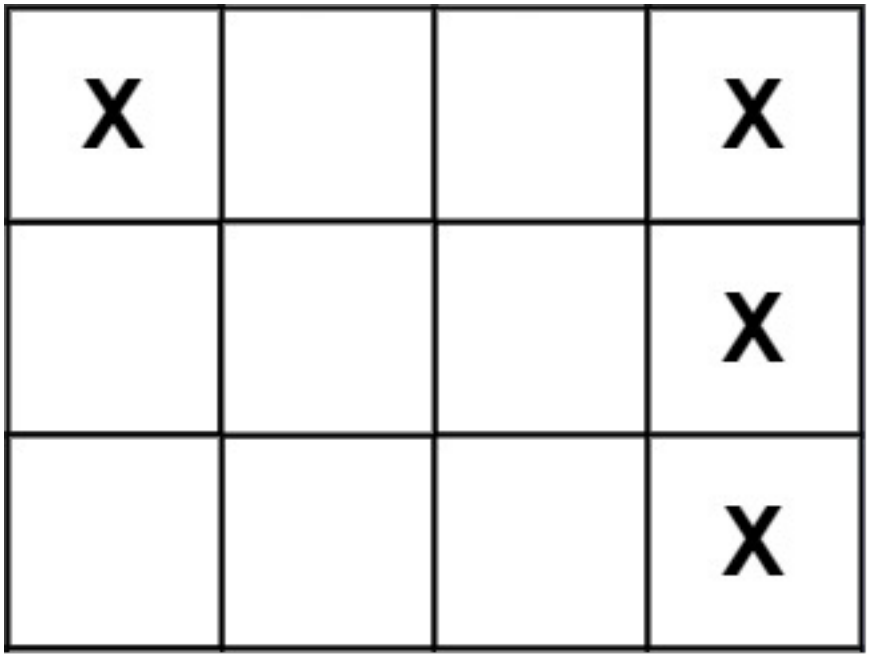
1

column), where

k

can be of any size. At least one horizontal or vertical cell separates between two battleships (i.e., there are no adjacent battleships).

Example 1:




Input:

board = [["X",".",".","X"],[".",".",".","X"],[".",".",".","X"]]

Output:

2

Example 2:

Input:

board = [["."]]

Output:

0

Constraints:

m == board.length

n == board[i].length

1 <= m, n <= 200

board[i][j]

is either

'.'

or

'X'

.

Follow up:

Could you do it in one-pass, using only

O(1)

extra memory and without modifying the values

board

?

## Code Snippets

**C++:**

```
class Solution {
public:
```

```
int countBattleships(vector<vector<char>>& board) {

}
};
```

**Java:**

```
class Solution {
public int countBattleships(char[][] board) {

}
}
```

**Python3:**

```
class Solution:
def countBattleships(self, board: List[List[str]]) -> int:
```

**Python:**

```
class Solution(object):
def countBattleships(self, board):
"""
:type board: List[List[str]]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {character[][]} board
* @return {number}
*/
var countBattleships = function(board) {

};
```

**TypeScript:**

```
function countBattleships(board: string[][]): number {

};
```

**C#:**

```
public class Solution {
public int CountBattleships(char[][] board) {

}
}
```

**C:**

```
int countBattleships(char** board, int boardSize, int* boardColSize) {

}
```

**Go:**

```
func countBattleships(board [][]byte) int {

}
```

**Kotlin:**

```
class Solution {
fun countBattleships(board: Array<CharArray>): Int {

}
}
```

**Swift:**

```
class Solution {
func countBattleships(_ board: [[Character]]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn count_battleships(board: Vec<Vec<char>>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Character[][]} board
# @return {Integer}
def count_battleships(board)

end
```

**PHP:**

```
class Solution {

/**
* @param String[][] $board
* @return Integer
*/
function countBattleships($board) {

}
}
```

**Dart:**

```
class Solution {
int countBattleships(List<List<String>> board) {

}
}
```

**Scala:**

```
object Solution {
def countBattleships(board: Array[Array[Char]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec count_battleships(board :: [[char]]) :: integer
def count_battleships(board) do
```

```
end
end
```

**Erlang:**

```
-spec count_battleships(Board :: [[char()]]) -> integer().
count_battleships(Board) ->
.
```

**Racket:**

```
(define/contract (count-battleships board)
(-> (listof (listof char?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Battleships in a Board
* Difficulty: Medium
* Tags: array, 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:
int countBattleships(vector<vector<char>>& board) {

}
};
```

### Java Solution:

```
/**
* Problem: Battleships in a Board
```

```
* Difficulty: Medium
* Tags: array, 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 int countBattleships(char[][] board) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Battleships in a Board
Difficulty: Medium
Tags: array, 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 countBattleships(self, board: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def countBattleships(self, board):
"""
:type board: List[List[str]]
:rtype: int
"""
```

**JavaScript Solution:**

```
/**
* Problem: Battleships in a Board
* Difficulty: Medium
* Tags: array, 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 {character[][]} board
* @return {number}
*/
var countBattleships = function(board) {

};
```

### TypeScript Solution:

```
/**
* Problem: Battleships in a Board
* Difficulty: Medium
* Tags: array, 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
*/

function countBattleships(board: string[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Battleships in a Board
* Difficulty: Medium
* Tags: array, 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 int CountBattleships(char[][] board) {

}
}
```

### C Solution:

```
/*
* Problem: Battleships in a Board
* Difficulty: Medium
* Tags: array, 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
*/

int countBattleships(char** board, int boardSize, int* boardColSize) {

}
```

### Go Solution:

```
// Problem: Battleships in a Board
// Difficulty: Medium
// Tags: array, 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

func countBattleships(board [][]byte) int {

}
```

### Kotlin Solution:

```
class Solution {
fun countBattleships(board: Array<CharArray>): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func countBattleships(_ board: [[Character]]) -> Int {

}
}
```

**Rust Solution:**

```
// Problem: Battleships in a Board
// Difficulty: Medium
// Tags: array, 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

impl Solution {
pub fn count_battleships(board: Vec<Vec<char>>) -> i32 {

}
}
```

**Ruby Solution:**

```
# @param {Character[][]} board
# @return {Integer}
def count_battleships(board)

end
```

**PHP Solution:**

```
class Solution {
```

```
/**
* @param String[][] $board
* @return Integer
*/
function countBattleships($board) {

}
}
```

**Dart Solution:**

```
class Solution {
int countBattleships(List<List<String>> board) {

}
}
```

**Scala Solution:**

```
object Solution {
def countBattleships(board: Array[Array[Char]]): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec count_battleships(board :: [[char]]) :: integer
def count_battleships(board) do

end
end
```

**Erlang Solution:**

```
-spec count_battleships(Board :: [[char()]]) -> integer().
count_battleships(Board) ->
.
```

**Racket Solution:**

```
(define/contract (count-battleships board)
(-> (listof (listof char?)) exact-integer?)
)
```