# Problem 1401: Circle and Rectangle Overlapping

## Problem Information

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

## Problem Description

You are given a circle represented as

(radius, xCenter, yCenter)

and an axis-aligned rectangle represented as

(x1, y1, x2, y2)

, where

(x1, y1)

are the coordinates of the bottom-left corner, and

(x2, y2)

are the coordinates of the top-right corner of the rectangle.

Return

true

if the circle and rectangle are overlapped otherwise return

false

. In other words, check if there is

any

point

(x

i

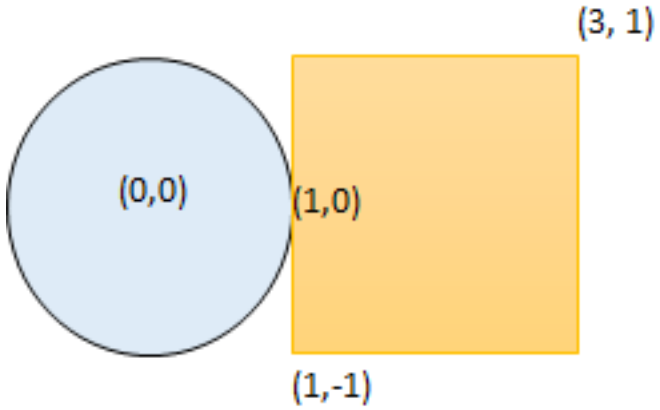
, y

i

)

that belongs to the circle and the rectangle at the same time.

Example 1:




Input:

radius = 1, xCenter = 0, yCenter = 0, x1 = 1, y1 = -1, x2 = 3, y2 = 1

Output:

true

Explanation:

Circle and rectangle share the point (1,0).

Example 2:

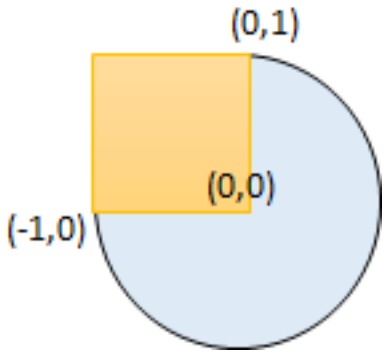
Input:

radius = 1, xCenter = 1, yCenter = 1, x1 = 1, y1 = -3, x2 = 2, y2 = -1

Output:

false

Example 3:




Input:

radius = 1, xCenter = 0, yCenter = 0, x1 = -1, y1 = 0, x2 = 0, y2 = 1

Output:

true

Constraints:

1 <= radius <= 2000

-10

4

<= xCenter, yCenter <= 10

4

-10

4

<= x1 < x2 <= 10

4

-10

4

<= y1 < y2 <= 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int
x2, int y2) {

}
};
```

**Java:**

```
class Solution {
public boolean checkOverlap(int radius, int xCenter, int yCenter, int x1, int
y1, int x2, int y2) {
```

```
}
}
```

**Python3:**

```
class Solution:
def checkOverlap(self, radius: int, xCenter: int, yCenter: int, x1: int, y1:
int, x2: int, y2: int) -> bool:
```

**Python:**

```
class Solution(object):
def checkOverlap(self, radius, xCenter, yCenter, x1, y1, x2, y2):
"""
:type radius: int
:type xCenter: int
:type yCenter: int
:type x1: int
:type y1: int
:type x2: int
:type y2: int
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number} radius
* @param {number} xCenter
* @param {number} yCenter
* @param {number} x1
* @param {number} y1
* @param {number} x2
* @param {number} y2
* @return {boolean}
*/
var checkOverlap = function(radius, xCenter, yCenter, x1, y1, x2, y2) {

};
```

**TypeScript:**

```
function checkOverlap(radius: number, xCenter: number, yCenter: number, x1: number, y1: number, x2: number, y2: number): boolean {

};
```

**C#:**

```
public class Solution {
public bool CheckOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int x2, int y2) {

}
}
```

**C:**

```
bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int x2, int y2) {

}
```

**Go:**

```
func checkOverlap(radius int, xCenter int, yCenter int, x1 int, y1 int, x2 int, y2 int) bool {

}
```

**Kotlin:**

```
class Solution {
fun checkOverlap(radius: Int, xCenter: Int, yCenter: Int, x1: Int, y1: Int, x2: Int, y2: Int): Boolean {

}
}
```

**Swift:**

```
class Solution {
func checkOverlap(_ radius: Int, _ xCenter: Int, _ yCenter: Int, _ x1: Int, _ y1: Int, _ x2: Int, _ y2: Int) -> Bool {
```

```
}
}
```

**Rust:**

```
impl Solution {
pub fn check_overlap(radius: i32, x_center: i32, y_center: i32, x1: i32, y1:
i32, x2: i32, y2: i32) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer} radius
# @param {Integer} x_center
# @param {Integer} y_center
# @param {Integer} x1
# @param {Integer} y1
# @param {Integer} x2
# @param {Integer} y2
# @return {Boolean}
def check_overlap(radius, x_center, y_center, x1, y1, x2, y2)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer $radius
* @param Integer $xCenter
* @param Integer $yCenter
* @param Integer $x1
* @param Integer $y1
* @param Integer $x2
* @param Integer $y2
* @return Boolean
*/
function checkOverlap($radius, $xCenter, $yCenter, $x1, $y1, $x2, $y2) {
```

```
}
}
```

**Dart:**

```
class Solution {
bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int
x2, int y2) {

}
}
```

**Scala:**

```
object Solution {
def checkOverlap(radius: Int, xCenter: Int, yCenter: Int, x1: Int, y1: Int,
x2: Int, y2: Int): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec check_overlap(radius :: integer, x_center :: integer, y_center ::
integer, x1 :: integer, y1 :: integer, x2 :: integer, y2 :: integer) ::
boolean
def check_overlap(radius, x_center, y_center, x1, y1, x2, y2) do

end
end
```

**Erlang:**

```
-spec check_overlap(Radius :: integer(), XCenter :: integer(), YCenter ::
integer(), X1 :: integer(), Y1 :: integer(), X2 :: integer(), Y2 ::
integer()) -> boolean().
check_overlap(Radius, XCenter, YCenter, X1, Y1, X2, Y2) ->
.
```

**Racket:**

```
(define/contract (check-overlap radius xCenter yCenter x1 y1 x2 y2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer? boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Circle and Rectangle Overlapping
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int
x2, int y2) {

}
};
```

### Java Solution:

```
/**
* Problem: Circle and Rectangle Overlapping
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean checkOverlap(int radius, int xCenter, int yCenter, int x1, int
```

```
y1, int x2, int y2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Circle and Rectangle Overlapping
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def checkOverlap(self, radius: int, xCenter: int, yCenter: int, x1: int, y1:
int, x2: int, y2: int) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def checkOverlap(self, radius, xCenter, yCenter, x1, y1, x2, y2):
"""
:type radius: int
:type xCenter: int
:type yCenter: int
:type x1: int
:type y1: int
:type x2: int
:type y2: int
:rtype: bool
"""
```

### JavaScript Solution:

```
/**
* Problem: Circle and Rectangle Overlapping
```

```
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} radius
* @param {number} xCenter
* @param {number} yCenter
* @param {number} x1
* @param {number} y1
* @param {number} x2
* @param {number} y2
* @return {boolean}
*/
var checkOverlap = function(radius, xCenter, yCenter, x1, y1, x2, y2) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Circle and Rectangle Overlapping
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function checkOverlap(radius: number, xCenter: number, yCenter: number, x1:
number, y1: number, x2: number, y2: number): boolean {

};
```

**C# Solution:**

```
/*
* Problem: Circle and Rectangle Overlapping
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CheckOverlap(int radius, int xCenter, int yCenter, int x1, int
y1, int x2, int y2) {

}
}
```

### C Solution:

```
/*
* Problem: Circle and Rectangle Overlapping
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int
x2, int y2) {

}
```

### Go Solution:

```
// Problem: Circle and Rectangle Overlapping
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
```

```
// Space Complexity: O(1) to O(n) depending on approach

func checkOverlap(radius int, xCenter int, yCenter int, x1 int, y1 int, x2
int, y2 int) bool {

}
```

**Kotlin Solution:**

```
class Solution {
fun checkOverlap(radius: Int, xCenter: Int, yCenter: Int, x1: Int, y1: Int,
x2: Int, y2: Int): Boolean {

}
}
```

**Swift Solution:**

```
class Solution {
func checkOverlap(_ radius: Int, _ xCenter: Int, _ yCenter: Int, _ x1: Int, _
y1: Int, _ x2: Int, _ y2: Int) -> Bool {

}
}
```

**Rust Solution:**

```
// Problem: Circle and Rectangle Overlapping
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn check_overlap(radius: i32, x_center: i32, y_center: i32, x1: i32, y1:
i32, x2: i32, y2: i32) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer} radius
# @param {Integer} x_center
# @param {Integer} y_center
# @param {Integer} x1
# @param {Integer} y1
# @param {Integer} x2
# @param {Integer} y2
# @return {Boolean}
def check_overlap(radius, x_center, y_center, x1, y1, x2, y2)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer $radius
* @param Integer $xCenter
* @param Integer $yCenter
* @param Integer $x1
* @param Integer $y1
* @param Integer $x2
* @param Integer $y2
* @return Boolean
*/
function checkOverlap($radius, $xCenter, $yCenter, $x1, $y1, $x2, $y2) {

}
}
```

### Dart Solution:

```
class Solution {
bool checkOverlap(int radius, int xCenter, int yCenter, int x1, int y1, int
x2, int y2) {

}
}
```

### Scala Solution:

```
object Solution {
def checkOverlap(radius: Int, xCenter: Int, yCenter: Int, x1: Int, y1: Int,
x2: Int, y2: Int): Boolean = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec check_overlap(radius :: integer, x_center :: integer, y_center ::
integer, x1 :: integer, y1 :: integer, x2 :: integer, y2 :: integer) ::
boolean
def check_overlap(radius, x_center, y_center, x1, y1, x2, y2) do

end
end
```

### Erlang Solution:

```
-spec check_overlap(Radius :: integer(), XCenter :: integer(), YCenter ::
integer(), X1 :: integer(), Y1 :: integer(), X2 :: integer(), Y2 ::
integer()) -> boolean().
check_overlap(Radius, XCenter, YCenter, X1, Y1, X2, Y2) ->
.
```

### Racket Solution:

```
(define/contract (check-overlap radius xCenter yCenter x1 y1 x2 y2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer? boolean?)
)
```