

# Problem 2550: Count Collisions of Monkeys on a Polygon

## Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

There is a regular convex polygon with

$n$

vertices. The vertices are labeled from

0

to

$n - 1$

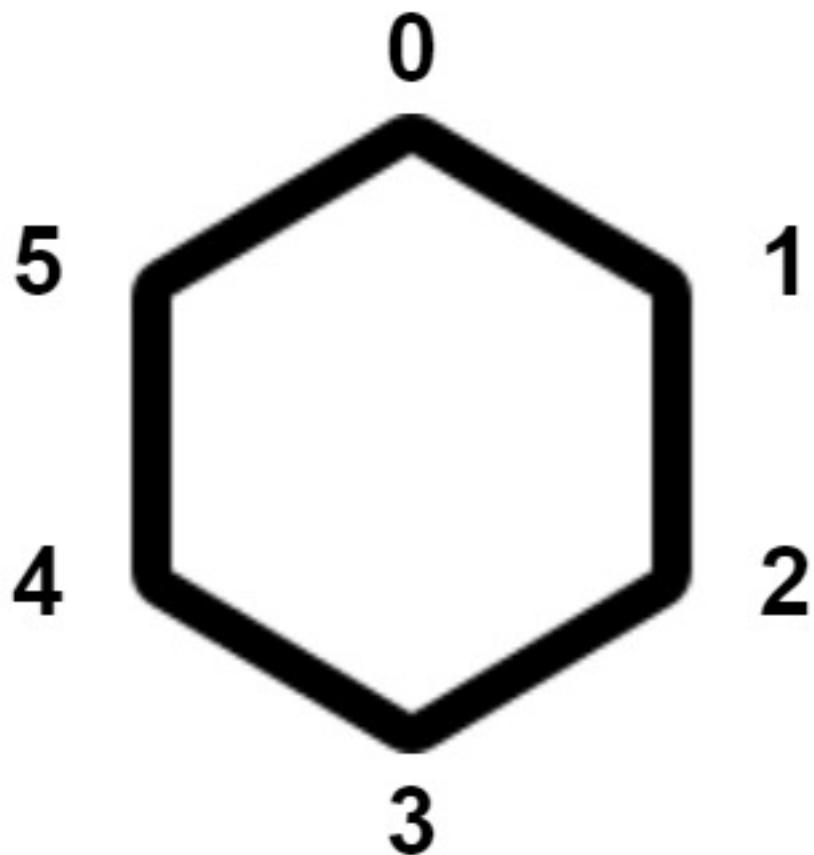
in a clockwise direction, and each vertex has

exactly one monkey

. The following figure shows a convex polygon of

6

vertices.



Simultaneously, each monkey moves to a neighboring vertex. A

collision

happens if at least two monkeys reside on the same vertex after the movement or intersect on an edge.

Return the number of ways the monkeys can move so that at least

one collision

happens. Since the answer may be very large, return it modulo

10

9

+ 7

.

Example 1:

Input:

$n = 3$

Output:

6

Explanation:

There are 8 total possible movements.

Two ways such that they collide at some point are:

Monkey 1 moves in a clockwise direction; monkey 2 moves in an anticlockwise direction; monkey 3 moves in a clockwise direction. Monkeys 1 and 2 collide.

Monkey 1 moves in an anticlockwise direction; monkey 2 moves in an anticlockwise direction; monkey 3 moves in a clockwise direction. Monkeys 1 and 3 collide.

Example 2:

Input:

$n = 4$

Output:

14

Constraints:

$3 \leq n \leq 10$

## Code Snippets

### C++:

```
class Solution {  
public:  
    int monkeyMove(int n) {  
  
    }  
};
```

### Java:

```
class Solution {  
    public int monkeyMove(int n) {  
  
    }  
}
```

### Python3:

```
class Solution:  
    def monkeyMove(self, n: int) -> int:
```

### Python:

```
class Solution(object):  
    def monkeyMove(self, n):  
        """  
        :type n: int  
        :rtype: int  
        """
```

### JavaScript:

```
/**  
 * @param {number} n  
 * @return {number}  
 */  
var monkeyMove = function(n) {
```

```
};
```

### TypeScript:

```
function monkeyMove(n: number): number {  
}  
};
```

### C#:

```
public class Solution {  
    public int MonkeyMove(int n) {  
        }  
    }  
}
```

### C:

```
int monkeyMove(int n) {  
}  
}
```

### Go:

```
func monkeyMove(n int) int {  
}  
}
```

### Kotlin:

```
class Solution {  
    fun monkeyMove(n: Int): Int {  
        }  
    }  
}
```

### Swift:

```
class Solution {  
    func monkeyMove(_ n: Int) -> Int {  
        }  
    }
```

```
}
```

**Rust:**

```
impl Solution {
    pub fn monkey_move(n: i32) -> i32 {
        }
}
```

**Ruby:**

```
# @param {Integer} n
# @return {Integer}
def monkey_move(n)

end
```

**PHP:**

```
class Solution {

    /**
     * @param Integer $n
     * @return Integer
     */
    function monkeyMove($n) {

    }
}
```

**Dart:**

```
class Solution {
    int monkeyMove(int n) {
        }
}
```

**Scala:**

```
object Solution {  
    def monkeyMove(n: Int): Int = {  
        }  
    }  
}
```

### Elixir:

```
defmodule Solution do  
    @spec monkey_move(n :: integer) :: integer  
    def monkey_move(n) do  
  
    end  
    end
```

### Erlang:

```
-spec monkey_move(N :: integer()) -> integer().  
monkey_move(N) ->  
.
```

### Racket:

```
(define/contract (monkey-move n)  
  (-> exact-integer? exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Count Collisions of Monkeys on a Polygon  
 * 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:  
    int monkeyMove(int n) {  
  
    }  
};
```

### Java Solution:

```
/**  
 * Problem: Count Collisions of Monkeys on a Polygon  
 * 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 int monkeyMove(int n) {  
  
}  
}
```

### Python3 Solution:

```
"""  
Problem: Count Collisions of Monkeys on a Polygon  
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 monkeyMove(self, n: int) -> int:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

```
class Solution(object):
    def monkeyMove(self, n):
        """
        :type n: int
        :rtype: int
        """
```

### JavaScript Solution:

```
/**
 * Problem: Count Collisions of Monkeys on a Polygon
 * 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
 */

var monkeyMove = function(n) {

};
```

### TypeScript Solution:

```
/**
 * Problem: Count Collisions of Monkeys on a Polygon
 * 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 monkeyMove(n: number): number {
```

```
};
```

### C# Solution:

```
/*
 * Problem: Count Collisions of Monkeys on a Polygon
 * 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 int MonkeyMove(int n) {
        return 0;
    }
}
```

### C Solution:

```
/*
 * Problem: Count Collisions of Monkeys on a Polygon
 * 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
 */

int monkeyMove(int n) {
    return 0;
}
```

### Go Solution:

```
// Problem: Count Collisions of Monkeys on a Polygon
// 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 monkeyMove(n int) int {

}
```

### Kotlin Solution:

```
class Solution {
    fun monkeyMove(n: Int): Int {
        return n
    }
}
```

### Swift Solution:

```
class Solution {
    func monkeyMove(_ n: Int) -> Int {
        return n
    }
}
```

### Rust Solution:

```
// Problem: Count Collisions of Monkeys on a Polygon
// 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 monkey_move(n: i32) -> i32 {
        return n
    }
}
```

### Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def monkey_move(n)

end
```

### PHP Solution:

```
class Solution {

    /**
     * @param Integer $n
     * @return Integer
     */
    function monkeyMove($n) {

    }
}
```

### Dart Solution:

```
class Solution {
int monkeyMove(int n) {

}
```

### Scala Solution:

```
object Solution {
def monkeyMove(n: Int): Int = {

}
```

### Elixir Solution:

```
defmodule Solution do
@spec monkey_move(n :: integer) :: integer
def monkey_move(n) do
```

```
end  
end
```

### Erlang Solution:

```
-spec monkey_move(N :: integer()) -> integer().  
monkey_move(N) ->  
.
```

### Racket Solution:

```
(define/contract (monkey-move n)  
(-> exact-integer? exact-integer?)  
)
```