

# Problem 2647: Color the Triangle Red

## Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given an integer

$n$

. Consider an equilateral triangle of side length

$n$

, broken up into

$n$

$2$

unit equilateral triangles. The triangle has

$n$

1-indexed

rows where the

$i$

th

row has

$2i - 1$

unit equilateral triangles.

The triangles in the

$i$

th

row are also

1-indexed

with coordinates from

$(i, 1)$

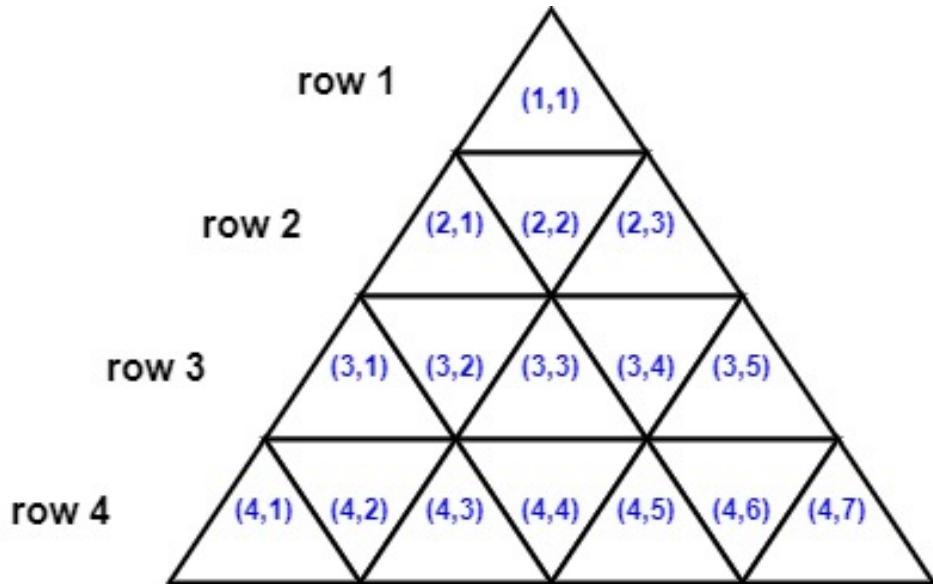
to

$(i, 2i - 1)$

. The following image shows a triangle of side length

4

with the indexing of its triangle.



Two triangles are

neighbors

if they

share a side

. For example:

Triangles

$(1,1)$

and

$(2,2)$

are neighbors

Triangles

$(3,2)$

and

(3,3)

are neighbors.

Triangles

(2,2)

and

(3,3)

are not neighbors because they do not share any side.

Initially, all the unit triangles are

white

. You want to choose

k

triangles and color them

red

. We will then run the following algorithm:

Choose a white triangle that has

at least two

red neighbors.

If there is no such triangle, stop the algorithm.

Color that triangle

red

Go to step 1.

Choose the minimum

$k$

possible and set

$k$

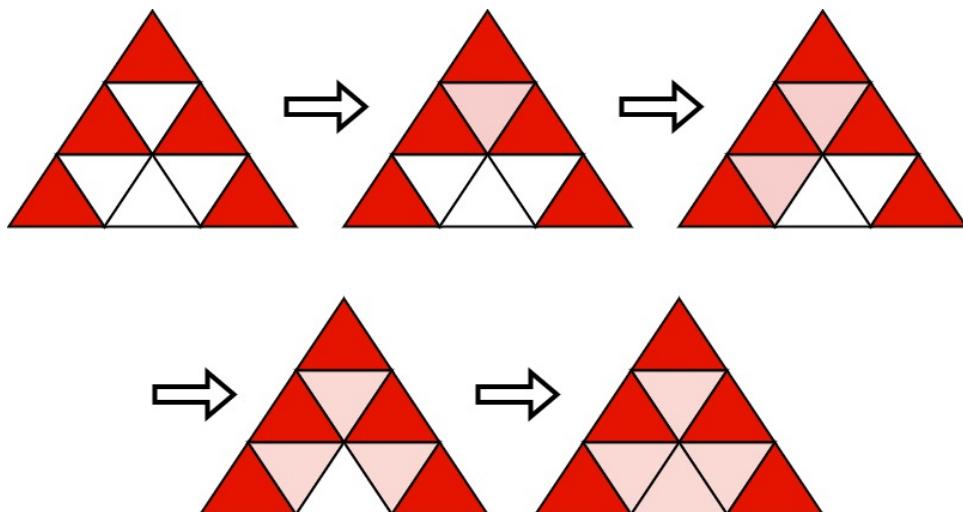
triangles red before running this algorithm such that after the algorithm stops, all unit triangles are colored red.

Return

a 2D list of the coordinates of the triangles that you will color red initially

. The answer has to be of the smallest size possible. If there are multiple valid solutions, return any.

Example 1:



Input:

$n = 3$

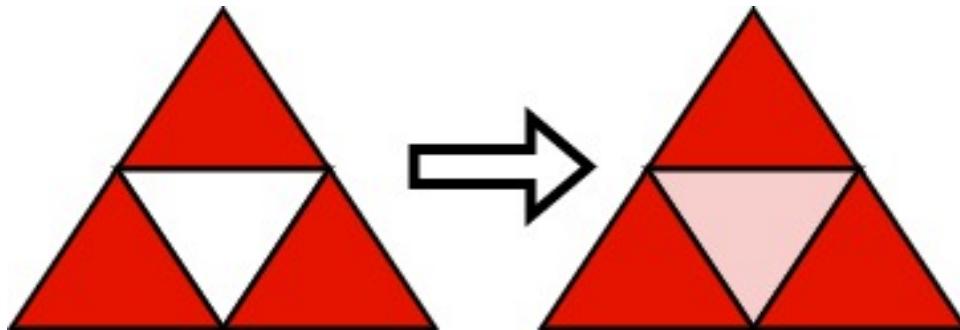
Output:

$[[1,1],[2,1],[2,3],[3,1],[3,5]]$

Explanation:

Initially, we choose the shown 5 triangles to be red. Then, we run the algorithm: - Choose (2,2) that has three red neighbors and color it red. - Choose (3,2) that has two red neighbors and color it red. - Choose (3,4) that has three red neighbors and color it red. - Choose (3,3) that has three red neighbors and color it red. It can be shown that choosing any 4 triangles and running the algorithm will not make all triangles red.

Example 2:



Input:

$n = 2$

Output:

$[[1,1],[2,1],[2,3]]$

Explanation:

Initially, we choose the shown 3 triangles to be red. Then, we run the algorithm: - Choose (2,2) that has three red neighbors and color it red. It can be shown that choosing any 2 triangles and running the algorithm will not make all triangles red.

Constraints:

$1 \leq n \leq 1000$

## Code Snippets

### C++:

```
class Solution {  
public:  
vector<vector<int>> colorRed(int n) {  
  
}  
};
```

### Java:

```
class Solution {  
public int[][][] colorRed(int n) {  
  
}  
}
```

### Python3:

```
class Solution:  
def colorRed(self, n: int) -> List[List[int]]:
```

### Python:

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

### JavaScript:

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

```
};
```

### TypeScript:

```
function colorRed(n: number): number[][] {  
}  
};
```

### C#:

```
public class Solution {  
    public int[][] ColorRed(int n) {  
          
    }  
}
```

### C:

```
/**  
 * Return an array of arrays of size *returnSize.  
 * The sizes of the arrays are returned as *returnColumnSizes array.  
 * Note: Both returned array and *columnSizes array must be malloced, assume  
 caller calls free().  
 */  
int** colorRed(int n, int* returnSize, int** returnColumnSizes) {  
  
}
```

### Go:

```
func colorRed(n int) [][]int {  
}
```

### Kotlin:

```
class Solution {  
    fun colorRed(n: Int): Array<IntArray> {  
          
    }  
}
```

**Swift:**

```
class Solution {  
    func colorRed(_ n: Int) -> [[Int]] {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn color_red(n: i32) -> Vec<Vec<i32>> {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer} n  
# @return {Integer[][]}  
def color_red(n)  
  
end
```

**PHP:**

```
class Solution {  
  
    /**  
     * @param Integer $n  
     * @return Integer[][]  
     */  
    function colorRed($n) {  
  
    }  
}
```

**Dart:**

```
class Solution {  
    List<List<int>> colorRed(int n) {  
  
    }
```

```
}
```

### Scala:

```
object Solution {  
    def colorRed(n: Int): Array[Array[Int]] = {  
          
    }  
}
```

### Elixir:

```
defmodule Solution do  
    @spec color_red(n :: integer) :: [[integer]]  
    def color_red(n) do  
  
    end  
end
```

### Erlang:

```
-spec color_red(N :: integer()) -> [[integer()]].  
color_red(N) ->  
.
```

### Racket:

```
(define/contract (color-red n)  
  (-> exact-integer? (listof (listof exact-integer?)))  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Color the Triangle Red  
 * Difficulty: Hard  
 * Tags: array, math  
 */
```

```

* 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:
vector<vector<int>> colorRed(int n) {

}
};

```

### Java Solution:

```

/**
 * Problem: Color the Triangle Red
 * Difficulty: Hard
 * Tags: array, math
 *
 * 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[][][] colorRed(int n) {

}
}

```

### Python3 Solution:

```

"""
Problem: Color the Triangle Red
Difficulty: Hard
Tags: array, math

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

### Python Solution:

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

### JavaScript Solution:

```
/**  
 * Problem: Color the Triangle Red  
 * Difficulty: Hard  
 * Tags: array, math  
 *  
 * 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} n  
 * @return {number[][][]}  
 */  
var colorRed = function(n) {  
  
};
```

### TypeScript Solution:

```
/**  
 * Problem: Color the Triangle Red  
 * Difficulty: Hard  
 * Tags: array, math
```

```

/*
 * 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 colorRed(n: number): number[][] {
}

```

### C# Solution:

```

/*
 * Problem: Color the Triangle Red
 * Difficulty: Hard
 * Tags: array, math
 *
 * 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[][] ColorRed(int n) {
        return new int[n][];
    }
}

```

### C Solution:

```

/*
 * Problem: Color the Triangle Red
 * Difficulty: Hard
 * Tags: array, math
 *
 * 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
 */

/***

```

```

* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** colorRed(int n, int* returnSize, int** returnColumnSizes) {

}

```

### Go Solution:

```

// Problem: Color the Triangle Red
// Difficulty: Hard
// Tags: array, math
//
// 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 colorRed(n int) [][]int {
}

```

### Kotlin Solution:

```

class Solution {
    fun colorRed(n: Int): Array<IntArray> {
        ...
    }
}

```

### Swift Solution:

```

class Solution {
    func colorRed(_ n: Int) -> [[Int]] {
        ...
    }
}

```

### Rust Solution:

```

// Problem: Color the Triangle Red
// Difficulty: Hard
// Tags: array, math
//
// 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 color_red(n: i32) -> Vec<Vec<i32>> {

}
}

```

### Ruby Solution:

```

# @param {Integer} n
# @return {Integer[][]}
def color_red(n)

end

```

### PHP Solution:

```

class Solution {

/**
 * @param Integer $n
 * @return Integer[][]
 */
function colorRed($n) {

}
}

```

### Dart Solution:

```

class Solution {
List<List<int>> colorRed(int n) {

}
}

```

### **Scala Solution:**

```
object Solution {  
    def colorRed(n: Int): Array[Array[Int]] = {  
  
    }  
}
```

### **Elixir Solution:**

```
defmodule Solution do  
  @spec color_red(n :: integer) :: [[integer]]  
  def color_red(n) do  
  
  end  
end
```

### **Erlang Solution:**

```
-spec color_red(N :: integer()) -> [[integer()]].  
color_red(N) ->  
.
```

### **Racket Solution:**

```
(define/contract (color-red n)  
  (-> exact-integer? (listof (listof exact-integer?)))  
)
```