Can Place Flower

# ## Description:

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in **adjacent** plots.

Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true *if* n *new flowers can be planted in the* flowerbed *without violating the no-adjacent-flowers rule and* false *otherwise*.

## Example 1:

**Input:** flowerbed = [1,0,0,0,1], n = 1

**Output:** true

## Example 2:

**Input:** flowerbed = [1,0,0,0,1], n = 2

**Output:** false

## Constraints:

* 1 <= flowerbed.length <= 2 \* 104
* flowerbed[i] is 0 or 1.
* There are no two adjacent flowers in flowerbed.
* 0 <= n <= flowerbed.length

# ## Algorithm:

## Initialize Counter:

* + Create a variable count and set it to 0. This will keep track of the number of flowers that can be planted.

## Iterate Over Flowerbed:

* + Use a for-loop to iterate over each plot in the flowerbed array.

## Check if Current Plot is Empty:

* + For each plot, check if the current plot (flowerbed[i]) is 0 (empty).

## Check Adjacent Plots:

* + If the current plot is empty, check the adjacent plots:
    - **Left Plot**: If i is 0 (first plot), consider it as empty; otherwise, check if flowerbed[i

- 1] is 0.

* + - **Right Plot**: If i is the last plot, consider it as empty; otherwise, check if flowerbed[i + 1] is 0.

## Plant a Flower:

* + If both the left and right adjacent plots are empty (or boundary conditions are met), plant a flower at the current plot by setting flowerbed[i] to 1.
  + Increment the count by 1.

## Check if Requirement is Met:

* + After iterating through the flowerbed, check if count is greater than or equal to n.
  + If true, return true; otherwise, return false.

# ## Pseudocode:

function canPlaceFlowers(flowerbed: array of int, n: int) -> boolean: count = 0

for i from 0 to length of flowerbed - 1:

if flowerbed[i] == 0:

emptyLeft = (i == 0) or (flowerbed[i - 1] == 0)

emptyRight = (i == length of flowerbed - 1) or (flowerbed[i + 1] == 0) if emptyLeft and emptyRight:

flowerbed[i] = 1

count += 1 return count >= n

# ## Code:

class Solution {

public boolean canPlaceFlowers(int[] flowerbed, int n) { int count = 0;

for(int i = 0; i < flowerbed.length; i++){

// Check if the current plot is empty if(flowerbed[i] == 0){

// Check if the left and right plots are empty

boolean emptyLeft = (i == 0) || (flowerbed[i - 1] == 0);

boolean emptyRight = (i == flowerbed.length - 1) || (flowerbed[i + 1] == 0);

// If both plots are empty, we can plant a flower here if(emptyLeft && emptyRight){

flowerbed[i] = 1; count++;

}

}

}

return count >= n;

}

}

# ## Conclusion

The function effectively checks each position in the flowerbed to see if a flower can be planted there while ensuring that the adjacent plots are empty. By incrementing the count each time a flower is planted, it keeps track of how many flowers have been successfully planted. The final comparison of count with n determines the outcome.