CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

EXPERIMENT 4

Name: Aaditya Maheshwari	**UID: 22BET10094**
Branch: IT	**Section/Group: 22BET_IOT-702(A)**
Semester: 6th	**Date of Performance: 13/02/25**
Subject Name: Advance Programming	**Subject Code: 22ITP-351**

Aim: Longest Nice Substring

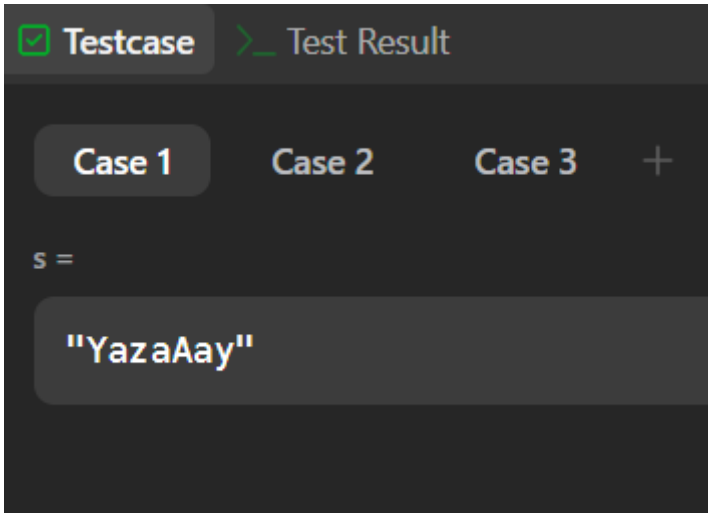
Objective: A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not.

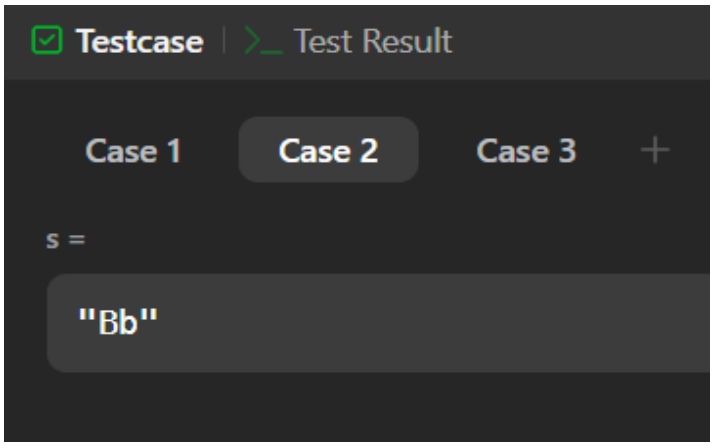
Code:

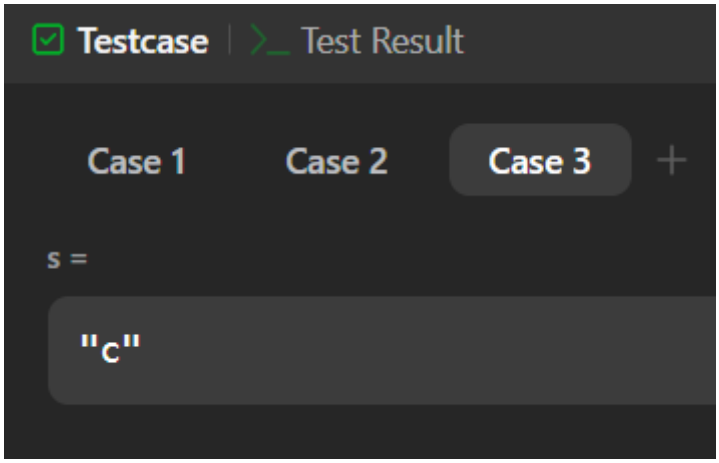
```
class Solution:
    def longestNiceSubstring(self, s):
        def is_nice(sub):
            return all(c.lower() in sub and c.upper() in sub for c in sub)

        n = len(s)
        max_nice = ""

        for i in range(n):
            for j in range(i + 1, n + 1):
                substring = s[i:j]
```

```
            if is_nice(substring) and len(substring) > len(max_nice):
                max_nice = substring

        return max_nice

s = "YazaAay"
print(Solution().longestNiceSubstring(s))
```

Output:







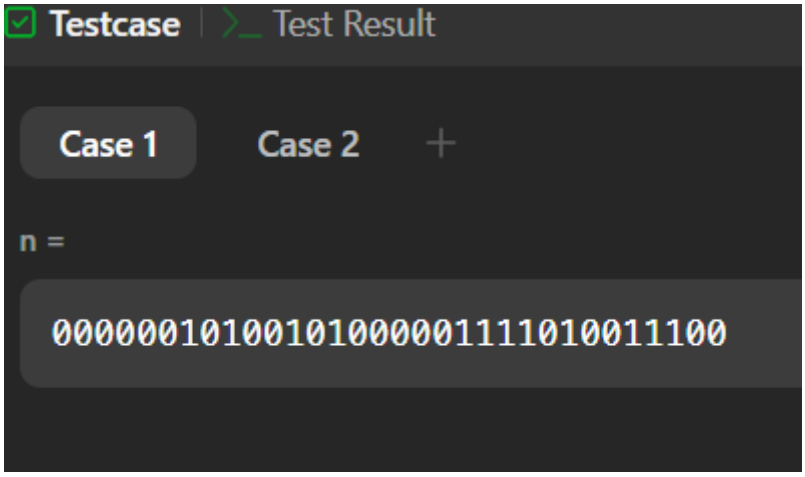
Aim: Reverse Bits

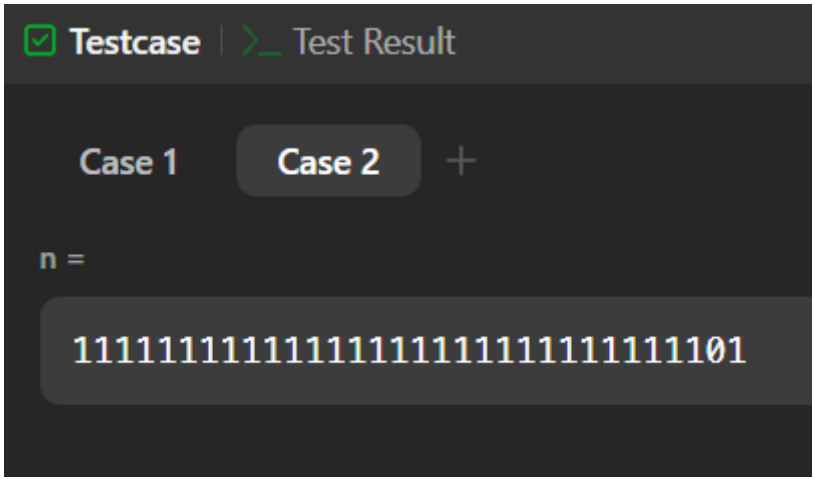
Objective: Reverse bits of a given 32 bits unsigned integer.

Code:

```
class Solution:
    def reverseBits(self, n):
        result = 0
        for i in range(32):
            result = (result << 1) | (n & 1)
            n >>= 1
        return result

solution = Solution()
n = 0b00000010100101000001111010011100
print(solution.reverseBits(n))
```

Output:





Aim: Number of 1 bits

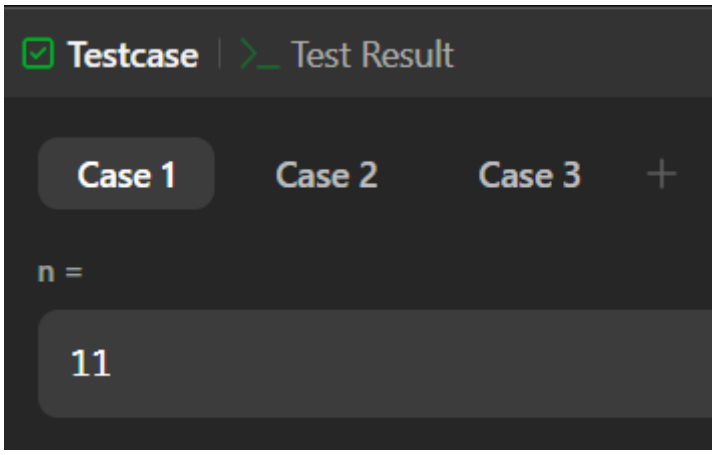
Objective: Given a positive integer n, write a function that returns the number of set bits in its binary representation (also known as the Hamming weight).

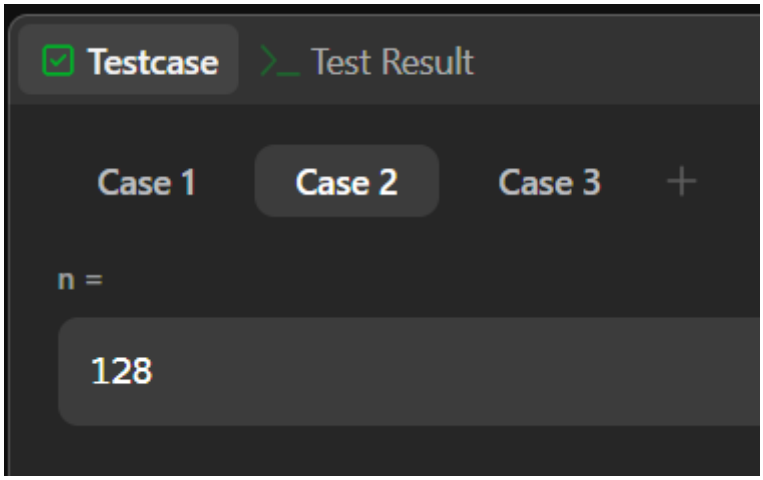
Code:

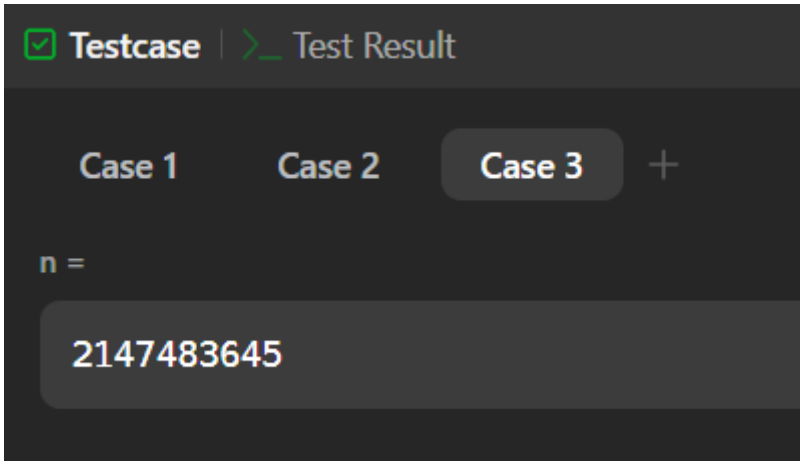
```
class Solution:
    def hammingWeight(self, n):
        count = 0
        while n:
```

```
    count += n & 1
    n >>= 1
return count
```

Output:







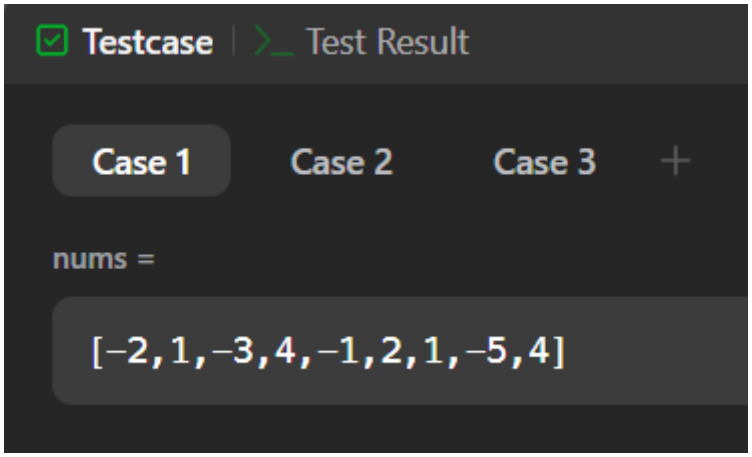
Aim: Max Subarray

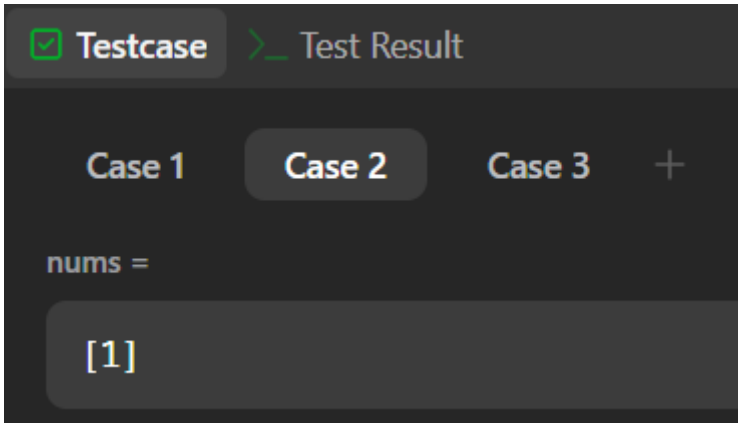
Objective: Given an integer array nums, find the subarray with the largest sum, and return *its sum*.

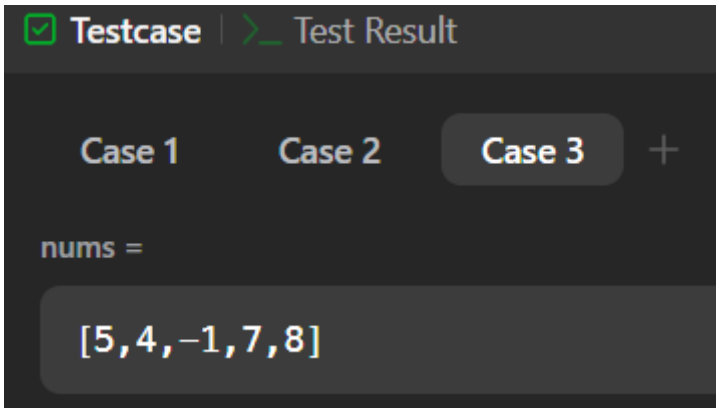
Code:

```
class Solution:
    def maxSubArray(self, nums):
        max_sum = float('-inf')
        current_sum = 0

        for num in nums:
            current_sum = max(num, current_sum + num)
            max_sum = max(max_sum, current_sum)

        return max_sum
```

Output:







Aim: Search 2d matrix 2

Objective: Write an efficient algorithm that searches for a value target in an m x n integer matrix matrix. This matrix has the following properties:

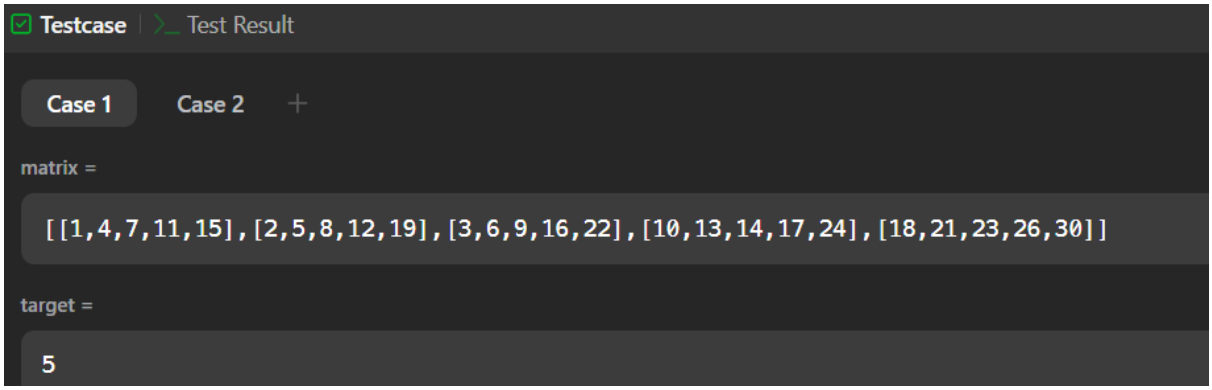
- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

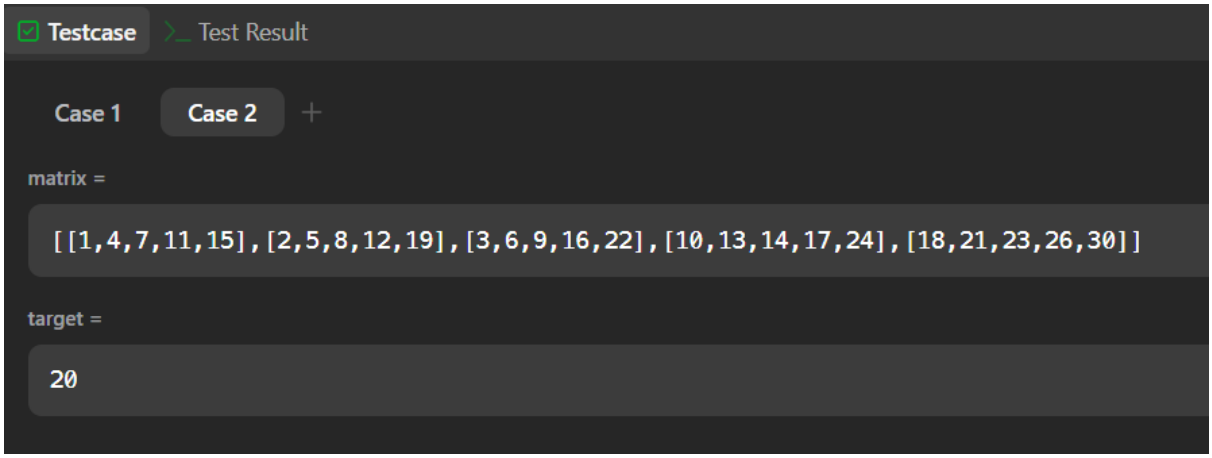
Code:

```
class Solution:
    def searchMatrix(self, matrix, target):
        if not matrix or not matrix[0]:
            return False
```

```
rows, cols = len(matrix), len(matrix[0])
row, col = 0, cols - 1

while row < rows and col >= 0:
    if matrix[row][col] == target:
        return True
    elif matrix[row][col] > target:
        col -= 1
    else:
        row += 1

return False
```

Output:





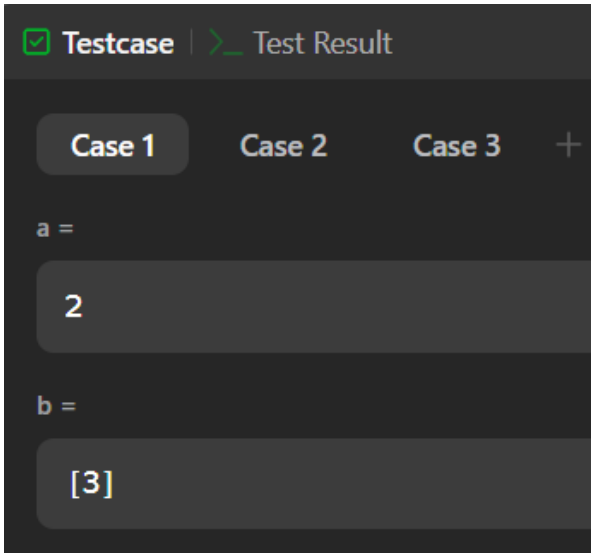
Aim: Super Pow

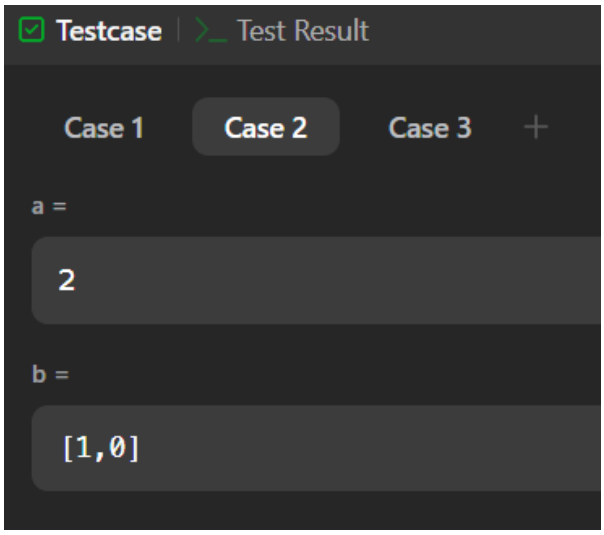
Objective: Your task is to calculate a^b mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

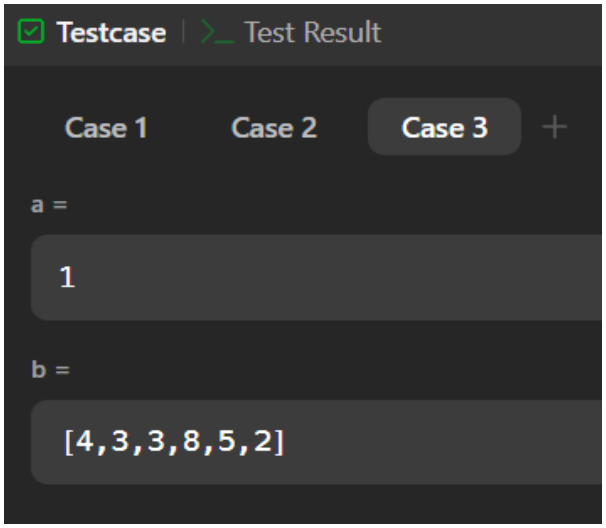
Code:

```
class Solution:
    def superPow(self, a, b):
        def power_mod(x, y, mod):
            result = 1
            for digit in y:
                result = (pow(result, 10, mod) * pow(x, digit, mod)) % mod
            return result

        return power_mod(a, b, 1337)
```

Output:


Testcase
Test Result
Case 1
Case 2
Case 3
a =
2
b =
[3]


Testcase
Test Result
Case 1
Case 2
Case 3
a =
2
b =
[1,0]


Testcase
Test Result
Case 1
Case 2
Case 3
a =
1
b =
[4,3,3,8,5,2]

Aim: Beautiful Array

Objective: An array nums of length n is **beautiful** if:

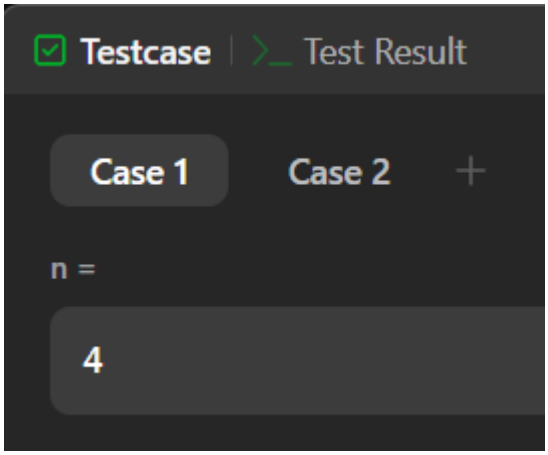
- nums is a permutation of the integers in the range [1, n].
- For every 0 <= i < j < n, there is no index k with i < k < j where 2 * nums[k] == nums[i] + nums[j].

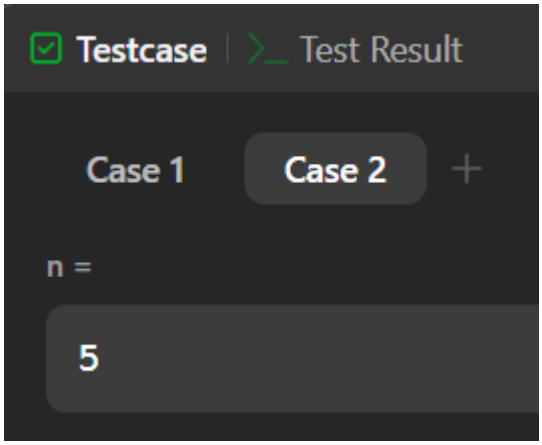
Given the integer n, return *any **beautiful** array* nums *of length* n. There will be at least one valid answer for the given n.

Code:

```
class Solution:
    def beautifulArray(self, n):
        if n == 1:
            return [1]

        odd = self.beautifulArray((n + 1) // 2)
        even = self.beautifulArray(n // 2)

        return [2 * x - 1 for x in odd] + [2 * x for x in even]

solution = Solution()
n = 5
print(solution.beautifulArray(n))
```

Output:


Testcase
Test Result
Case 1
Case 2
n =
4


Testcase
Test Result
Case 1
Case 2
n =
5