**Jump Game II**

Given an array of non-negative integers arr, a monkey is initially positioned at the first index of the array. Each element in the array arr[i] represents the maximum length of a forward jump from index i. The monkey wants to reach the last index in the minimum number of jumps. Help the monkey to find that. It is guaranteed that the monkey can always reach the last index.

**Input Format**

* The first line takes an integer N (size of the array) as input.
* The second line contains N integers, initializing each index of the array respectively.

**Constraints**

* 1 <= N <= 100
* 0 <= arr[i] <= 1000

**Output Format**

Print the minimum number of jumps the monkey needs to reach the last index.

**Sample Input**

5

2 3 1 1 4

**Sample Output**

2

**Explanation**

The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Test Cases**

**Test Case 1**

**Input:**

5

2 3 1 1 4

**Output:**

2

**Test Case 2**

**Input:**

4

4 3 1 2

**Output:**

1

**Test Case 3**

**Input:**

10

1 2 3 4 1 0 4 3 1 9

**Output:**

4

**Test Case 4**

**Input:**

12

1 6 1 2 3 4 1 0 4 3 1 9

**Output:**

4

**Test Case 5**

**Input:**

19

2 6 1 2 3 4 1 0 4 3 1 9 2 1 3 4 5 6 7

**Output:**

5

**Solution**

To solve this problem efficiently, we can use a greedy approach. The idea is to keep track of the farthest position that can be reached with the current number of jumps, and update the number of jumps whenever the current position exceeds the farthest reach of the previous jump.

Here's the solution:

python

def jump\_game(arr):

n = len(arr)

if n <= 1:

return 0

jumps = 0

current\_end = 0

farthest = 0

for i in range(n-1):

farthest = max(farthest, i + arr[i])

if i == current\_end:

jumps += 1

current\_end = farthest

if current\_end >= n-1:

break

return jumps

# Reading input

N = int(input())

arr = list(map(int, input().split()))

# Calculating the minimum number of jumps

result = jump\_game(arr)

# Printing the result

print(result)

**Explanation of the Solution**

1. **Initialization**: Initialize jumps to 0, current\_end to 0, and farthest to 0.
2. **Iterate through the array**: Loop from the first index to the second-to-last index.
   * Update farthest to the maximum of the current farthest and i + arr[i].
   * If the current index i is equal to current\_end, it means we have to make a jump.
   * Increase the jumps by 1 and set current\_end to farthest.
   * If current\_end is greater than or equal to the last index, break out of the loop.
3. **Return the number of jumps**.