119. Pascal's Triangle II

Solved

Easy | Topics | Companies

Given an integer `rowIndex`, return the `rowIndex`th (**0-indexed**) row of the **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:

Example 1:

```
Input: rowIndex = 3
Output: [1,3,3,1]
```

Example 2:

```
Input: rowIndex = 0
Output: [1]
```

Example 3:

```
Input: rowIndex = 1
Output: [1,1]
```

Constraints:

- `0 <= rowIndex <= 33`

Follow up: Could you optimize your algorithm to use only `O(rowIndex)` extra space?

Python:

```
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        row = [1] * (rowIndex + 1)   # initialize with 1s

        # Build row in place
        for i in range(2, rowIndex + 1):  # start from row 2
            for j in range(i - 1, 0, -1):  # update backwards
                row[j] += row[j - 1]

        return row
```

JavaScript:

```
/**
 * @param {number} rowIndex
 * @return {number[]}
 */
var getRow = function(rowIndex) {
    let row = [1];  // start with the first row

    for (let i = 1; i <= rowIndex; i++) {
        row.push(1);  // add new element at the end for the new row
```

```
        // update from right to left
        for (let j = i - 1; j > 0; j--) {
            row[j] = row[j] + row[j - 1];
        }
    }

    return row;
};
```

Java:

```
import java.util.*;

class Solution {
    public List<Integer> getRow(int rowIndex) {
        List<Integer> row = new ArrayList<>();
        long val = 1; // use long to avoid overflow during calculation
        for (int i = 0; i <= rowIndex; i++) {
            row.add((int) val);
            val = val * (rowIndex - i) / (i + 1);
        }
        return row;
    }
}
```