Problem 119: Pascal's Triangle II

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an integer

rowIndex

, return the

rowIndex

th

(

0-indexed

) row of the

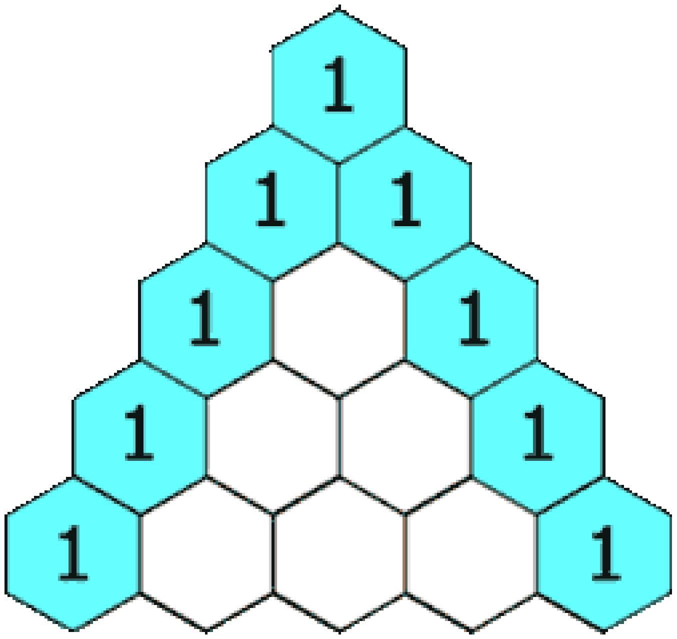
Pascal's triangle

.

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?

Code Snippets

C++:

```
class Solution {
public:
vector<int> getRow(int rowIndex) {

}
};
```

Java:

```
class Solution {
public List<Integer> getRow(int rowIndex) {

}
}
```

Python3:

```
class Solution:
def getRow(self, rowIndex: int) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getRow(rowIndex: number): number[] {

};
```

C#:

```
public class Solution {
public IList<int> GetRow(int rowIndex) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getRow(int rowIndex, int* returnSize) {

}
```

Go:

```
func getRow(rowIndex int) []int {

}
```

Kotlin:

```
class Solution {
fun getRow(rowIndex: Int): List<Int> {

}
}
```

Swift:

```
class Solution {
func getRow(_ rowIndex: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_row(row_index: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} row_index
# @return {Integer[]}
def get_row(row_index)

end
```

PHP:

```
class Solution {

/**
* @param Integer $rowIndex
* @return Integer[]
*/
function getRow($rowIndex) {

}
}
```

Dart:

```
class Solution {
List<int> getRow(int rowIndex) {

}
}
```

Scala:

```
object Solution {
def getRow(rowIndex: Int): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_row(row_index :: integer) :: [integer]
def get_row(row_index) do

end
end
```

Erlang:

```
-spec get_row(RowIndex :: integer()) -> [integer()].
get_row(RowIndex) ->
.
```

Racket:

```
(define/contract (get-row rowIndex)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
vector<int> getRow(int rowIndex) {

}
};
```

Java Solution:

```
/**
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public List<Integer> getRow(int rowIndex) {

}
}
```

Python3 Solution:

```
"""
Problem: Pascal's Triangle II
Difficulty: Easy
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def getRow(self, rowIndex: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function getRow(rowIndex: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public IList<int> GetRow(int rowIndex) {

}
}
```

C Solution:

```
/*
* Problem: Pascal's Triangle II
* Difficulty: Easy
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getRow(int rowIndex, int* returnSize) {

}
```

Go Solution:

```
// Problem: Pascal's Triangle II
// Difficulty: Easy
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func getRow(rowIndex int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun getRow(rowIndex: Int): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func getRow(_ rowIndex: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Pascal's Triangle II
// Difficulty: Easy
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn get_row(row_index: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} row_index
# @return {Integer[]}
def get_row(row_index)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $rowIndex
* @return Integer[]
*/
function getRow($rowIndex) {

}
}
```

Dart Solution:

```
class Solution {
List<int> getRow(int rowIndex) {

}
}
```

Scala Solution:

```
object Solution {
def getRow(rowIndex: Int): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_row(row_index :: integer) :: [integer]
def get_row(row_index) do

end
end
```

Erlang Solution:

```
-spec get_row(RowIndex :: integer()) -> [integer()].
get_row(RowIndex) ->
.
```

Racket Solution:

```
(define/contract (get-row rowIndex)
(-> exact-integer? (listof exact-integer?))
)
```