Problem 77: Combinations

Problem Information

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

Problem Description

Given two integers

n

and

k

, return

all possible combinations of

k

numbers chosen from the range

[1, n]

.

You may return the answer in

any order

.

Example 1:

Input:

n = 4, k = 2

Output:

[[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]

Explanation:

There are 4 choose 2 = 6 total combinations. Note that combinations are unordered, i.e., [1,2] and [2,1] are considered to be the same combination.

Example 2:

Input:

n = 1, k = 1

Output:

[[1]]

Explanation:

There is 1 choose 1 = 1 total combination.

Constraints:

1 <= n <= 20

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> combine(int n, int k) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> combine(int n, int k) {

}
}
```

Python3:

```
class Solution:
def combine(self, n: int, k: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def combine(self, n, k):
"""
:type n: int
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number[][]}
*/
var combine = function(n, k) {

};
```

TypeScript:

```
function combine(n: number, k: number): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> Combine(int n, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** combine(int n, int k, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func combine(n int, k int) [][]int {

}
```

Kotlin:

```
class Solution {
fun combine(n: Int, k: Int): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func combine(_ n: Int, _ k: Int) -> [[Int]] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn combine(n: i32, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[][]}
def combine(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[][]
*/
function combine($n, $k) {

}
}
```

Dart:

```
class Solution {
List<List<int>> combine(int n, int k) {

}
}
```

Scala:

```
object Solution {
def combine(n: Int, k: Int): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec combine(n :: integer, k :: integer) :: [[integer]]
def combine(n, k) do

end
end
```

Erlang:

```
-spec combine(N :: integer(), K :: integer()) -> [[integer()]].
combine(N, K) ->
.
```

Racket:

```
(define/contract (combine n k)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
vector<vector<int>> combine(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public List<List<Integer>> combine(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Combinations
Difficulty: Medium
Tags: general

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def combine(self, n: int, k: int) -> List[List[int]]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def combine(self, n, k):
"""
:type n: int
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number} k
* @return {number[][]}
*/
var combine = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
```

```
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function combine(n: number, k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public IList<IList<int>> Combine(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Combinations
* Difficulty: Medium
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
```

```
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** combine(int n, int k, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Combinations
// Difficulty: Medium
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func combine(n int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun combine(n: Int, k: Int): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func combine(_ n: Int, _ k: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Combinations
// Difficulty: Medium
```

```
// Tags: general
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn combine(n: i32, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer[][]}
def combine(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer[][]
*/
function combine($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> combine(int n, int k) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def combine(n: Int, k: Int): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec combine(n :: integer, k :: integer) :: [[integer]]
def combine(n, k) do

end
end
```

Erlang Solution:

```
-spec combine(N :: integer(), K :: integer()) -> [[integer()]].
combine(N, K) ->
.
```

Racket Solution:

```
(define/contract (combine n k)
(-> exact-integer? exact-integer? (listof (listof exact-integer?)))
)
```