Problem 254: Factor Combinations

Problem Information

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

Problem Description

Numbers can be regarded as the product of their factors.

For example,

8 = 2 x 2 x 2 = 2 x 4

.

Given an integer

n

, return

all possible combinations of its factors

. You may return the answer in

any order

.

Note

that the factors should be in the range

[2, n - 1]

.

Example 1:

Input:

n = 1

Output:

[]

Example 2:

Input:

n = 12

Output:

[[2,6],[3,4],[2,2,3]]

Example 3:

Input:

n = 37

Output:

[]

Constraints:

1 <= n <= 10

7

Code Snippets

C++:

```cpp
class Solution {
public:
vector<vector<int>> getFactors(int n) {

}
};
```

Java:

```java
class Solution {
public List<List<Integer>> getFactors(int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```javascript
/**
* @param {number} n
* @return {number[][]}
*/
var getFactors = function(n) {

};
```

TypeScript:

```
function getFactors(n: number): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> GetFactors(int n) {

}
}
```

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** getFactors(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func getFactors(n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun getFactors(n: Int): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func getFactors(_ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn get_factors(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[][]}
def get_factors(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function getFactors($n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> getFactors(int n) {

}

}
```

Scala:

```
object Solution {
def getFactors(n: Int): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_factors(n :: integer) :: [[integer]]
def get_factors(n) do

end
end
```

Erlang:

```
-spec get_factors(N :: integer()) -> [[integer()]].
get_factors(N) ->
.
```

Racket:

```
(define/contract (get-factors n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Factor 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>> getFactors(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Factor 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>> getFactors(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Factor 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 getFactors(self, n: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Factor 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
* @return {number[][]}
*/
var getFactors = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Factor 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 getFactors(n: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Factor 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>> GetFactors(int n) {

}
}
```

C Solution:

```
/*
* Problem: Factor 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** getFactors(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Factor 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 getFactors(n int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun getFactors(n: Int): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func getFactors(_ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Factor 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 get_factors(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[][]}
def get_factors(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function getFactors($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> getFactors(int n) {

}
}
```

Scala Solution:

```
object Solution {
def getFactors(n: Int): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_factors(n :: integer) :: [[integer]]
def get_factors(n) do

end
end
```

Erlang Solution:

```
-spec get_factors(N :: integer()) -> [[integer()]].
get_factors(N) ->
.
```

Racket Solution:

```
(define/contract (get-factors n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```