Problem 313: Super Ugly Number

Problem Information

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

Problem Description

A

super ugly number

is a positive integer whose prime factors are in the array

primes

.

Given an integer

n

and an array of integers

primes

, return

the

n

th

super ugly number

.

The

n

th

super ugly number

is

guaranteed

to fit in a

32-bit

signed integer.

Example 1:

Input:

n = 12, primes = [2,7,13,19]

Output:

32

Explanation:

[1,2,4,7,8,13,14,16,19,26,28,32] is the sequence of the first 12 super ugly numbers given primes = [2,7,13,19].

Example 2:

Input:

n = 1, primes = [2,3,5]

Output:

1

Explanation:

1 has no prime factors, therefore all of its prime factors are in the array primes = [2,3,5].

Constraints:

1 <= n <= 10

5

1 <= primes.length <= 100

2 <= primes[i] <= 1000

primes[i]

is

guaranteed

to be a prime number.

All the values of

primes

are

unique

and sorted in

ascending order

.

Code Snippets

C++:

```
class Solution {
public:
int nthSuperUglyNumber(int n, vector<int>& primes) {

}
};
```

Java:

```
class Solution {
public int nthSuperUglyNumber(int n, int[] primes) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function nthSuperUglyNumber(n: number, primes: number[]): number {

};
```

C#:

```
public class Solution {
public int NthSuperUglyNumber(int n, int[] primes) {

}
}
```

C:

```
int nthSuperUglyNumber(int n, int* primes, int primesSize) {

}
```

Go:

```
func nthSuperUglyNumber(n int, primes []int) int {

}
```

Kotlin:

```
class Solution {
fun nthSuperUglyNumber(n: Int, primes: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[]} primes
# @return {Integer}
def nth_super_ugly_number(n, primes)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $primes
* @return Integer
*/
function nthSuperUglyNumber($n, $primes) {

}
}
```

Dart:

```
class Solution {
int nthSuperUglyNumber(int n, List<int> primes) {
```

```
}
}
```

Scala:

```
object Solution {
def nthSuperUglyNumber(n: Int, primes: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec nth_super_ugly_number(n :: integer, primes :: [integer]) :: integer
def nth_super_ugly_number(n, primes) do

end
end
```

Erlang:

```
-spec nth_super_ugly_number(N :: integer(), Primes :: [integer()]) ->
integer().
nth_super_ugly_number(N, Primes) ->
.
```

Racket:

```
(define/contract (nth-super-ugly-number n primes)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Super Ugly Number
```

```
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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:
int nthSuperUglyNumber(int n, vector<int>& primes) {

}
};
```

Java Solution:

```
/**
* Problem: Super Ugly Number
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 int nthSuperUglyNumber(int n, int[] primes) {

}
}
```

Python3 Solution:

```
"""
Problem: Super Ugly Number
Difficulty: Medium
Tags: array, dp, math, sort

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 nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Super Ugly Number
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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} n
* @param {number[]} primes
* @return {number}
*/
var nthSuperUglyNumber = function(n, primes) {

};
```

TypeScript Solution:

```
/**
* Problem: Super Ugly Number
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 nthSuperUglyNumber(n: number, primes: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Super Ugly Number
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 int NthSuperUglyNumber(int n, int[] primes) {

}
}
```

C Solution:

```
/*
* Problem: Super Ugly Number
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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
```

```
*/

int nthSuperUglyNumber(int n, int* primes, int primesSize) {

}
```

Go Solution:

```
// Problem: Super Ugly Number
// Difficulty: Medium
// Tags: array, dp, math, sort
//
// 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 nthSuperUglyNumber(n int, primes []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun nthSuperUglyNumber(n: Int, primes: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Super Ugly Number
// Difficulty: Medium
// Tags: array, dp, math, sort
```

```
//
// 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 nth_super_ugly_number(n: i32, primes: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[]} primes
# @return {Integer}
def nth_super_ugly_number(n, primes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[] $primes
* @return Integer
*/
function nthSuperUglyNumber($n, $primes) {

}
}
```

Dart Solution:

```
class Solution {
int nthSuperUglyNumber(int n, List<int> primes) {

}
}
```

Scala Solution:

```
object Solution {
def nthSuperUglyNumber(n: Int, primes: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec nth_super_ugly_number(n :: integer, primes :: [integer]) :: integer
def nth_super_ugly_number(n, primes) do

end
end
```

Erlang Solution:

```
-spec nth_super_ugly_number(N :: integer(), Primes :: [integer()]) ->
integer().
nth_super_ugly_number(N, Primes) ->
.
```

Racket Solution:

```
(define/contract (nth-super-ugly-number n primes)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```