Problem 372: Super Pow

Problem Information

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

Problem Description

Your task is to calculate

a

b

mod

1337

where

a

is a positive integer and

b

is an extremely large positive integer given in the form of an array.

Example 1:

Input:

a = 2, b = [3]

Output:

8

Example 2:

Input:

a = 2, b = [1,0]

Output:

1024

Example 3:

Input:

a = 1, b = [4,3,3,8,5,2]

Output:

1

Constraints:

1 <= a <= 2

31

- 1

1 <= b.length <= 2000

0 <= b[i] <= 9

b

does not contain leading zeros.

Code Snippets

C++:

```
class Solution {
public:
int superPow(int a, vector<int>& b) {

}
};
```

Java:

```
class Solution {
public int superPow(int a, int[] b) {

}
}
```

Python3:

```
class Solution:
def superPow(self, a: int, b: List[int]) -> int:
```

Python:

```
class Solution(object):
def superPow(self, a, b):
"""
:type a: int
:type b: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} a
* @param {number[]} b
* @return {number}
*/
```

```
var superPow = function(a, b) {

};
```

TypeScript:

```
function superPow(a: number, b: number[]): number {

};
```

C#:

```
public class Solution {
public int SuperPow(int a, int[] b) {

}
}
```

C:

```
int superPow(int a, int* b, int bSize) {

}
```

Go:

```
func superPow(a int, b []int) int {

}
```

Kotlin:

```
class Solution {
fun superPow(a: Int, b: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func superPow(_ a: Int, _ b: [Int]) -> Int {


}
}
```

Rust:

```
impl Solution {
pub fn super_pow(a: i32, b: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} a
# @param {Integer[]} b
# @return {Integer}
def super_pow(a, b)

end
```

PHP:

```
class Solution {

/**
* @param Integer $a
* @param Integer[] $b
* @return Integer
*/
function superPow($a, $b) {

}
}
```

Dart:

```
class Solution {
int superPow(int a, List<int> b) {

}
}
```

Scala:

```
object Solution {
def superPow(a: Int, b: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec super_pow(a :: integer, b :: [integer]) :: integer
def super_pow(a, b) do

end
end
```

Erlang:

```
-spec super_pow(A :: integer(), B :: [integer()]) -> integer().
super_pow(A, B) ->
.
```

Racket:

```
(define/contract (super-pow a b)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Solution {
public:
int superPow(int a, vector<int>& b) {

}
};
```

Java Solution:

```
/**
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int superPow(int a, int[] b) {

}
}
```

Python3 Solution:

```
"""
Problem: Super Pow
Difficulty: Medium
Tags: array, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def superPow(self, a: int, b: List[int]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def superPow(self, a, b):
"""
:type a: int
:type b: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} a
* @param {number[]} b
* @return {number}
*/
var superPow = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function superPow(a: number, b: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SuperPow(int a, int[] b) {

}
}
```

C Solution:

```
/*
* Problem: Super Pow
* Difficulty: Medium
* Tags: array, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int superPow(int a, int* b, int bSize) {
}
```

Go Solution:

```
// Problem: Super Pow
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func superPow(a int, b []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun superPow(a: Int, b: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func superPow(_ a: Int, _ b: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Super Pow
// Difficulty: Medium
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
impl Solution {
pub fn super_pow(a: i32, b: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} a
# @param {Integer[]} b
# @return {Integer}
def super_pow(a, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $a
* @param Integer[] $b
* @return Integer
*/
function superPow($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
int superPow(int a, List<int> b) {

}
}
```

Scala Solution:

```
object Solution {
def superPow(a: Int, b: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec super_pow(a :: integer, b :: [integer]) :: integer
def super_pow(a, b) do

end
end
```

Erlang Solution:

```
-spec super_pow(A :: integer(), B :: [integer()]) -> integer().
super_pow(A, B) ->
.
```

Racket Solution:

```
(define/contract (super-pow a b)
(-> exact-integer? (listof exact-integer?) exact-integer?)
)
```