Problem 50: Pow(x, n)

Problem Information

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

Problem Description

Implement

pow(x, n)

, which calculates

x

raised to the power

n

(i.e.,

x

n

).

Example 1:

Input:

x = 2.00000, n = 10

Output:

1024.00000

Example 2:

Input:

x = 2.10000, n = 3

Output:

9.26100

Example 3:

Input:

x = 2.00000, n = -2

Output:

0.25000

Explanation:

2

-2

= 1/2

2

= 1/4 = 0.25

Constraints:

-100.0 < x < 100.0

-2

31

<= n <= 2

31

-1

n

is an integer.

Either

x

is not zero or

n > 0

.

-10

4

<= x

n

<= 10

4

Code Snippets

C++:

```
class Solution {
public:
double myPow(double x, int n) {

}
};
```

Java:

```
class Solution {
public double myPow(double x, int n) {

}
}
```

Python3:

```
class Solution:
def myPow(self, x: float, n: int) -> float:
```

Python:

```
class Solution(object):
def myPow(self, x, n):
"""
:type x: float
:type n: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number} x
* @param {number} n
* @return {number}
*/
var myPow = function(x, n) {

};
```

}

TypeScript:

```
function myPow(x: number, n: number): number {

};
```

C#:

```
public class Solution {
public double MyPow(double x, int n) {

}
}
```

C:

```
double myPow(double x, int n) {

}
```

Go:

```
func myPow(x float64, n int) float64 {

}
```

Kotlin:

```
class Solution {
fun myPow(x: Double, n: Int): Double {

}
}
```

Swift:

```
class Solution {
func myPow(_ x: Double, _ n: Int) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn my_pow(x: f64, n: i32) -> f64 {

}
}
```

Ruby:

```
# @param {Float} x
# @param {Integer} n
# @return {Float}
def my_pow(x, n)

end
```

PHP:

```
class Solution {

/**
* @param Float $x
* @param Integer $n
* @return Float
*/
function myPow($x, $n) {

}
}
```

Dart:

```
class Solution {
double myPow(double x, int n) {

}
}
```

Scala:

```
object Solution {
def myPow(x: Double, n: Int): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec my_pow(x :: float, n :: integer) :: float
def my_pow(x, n) do

end
end
```

Erlang:

```
-spec my_pow(X :: float(), N :: integer()) -> float().
my_pow(X, N) ->
.
```

Racket:

```
(define/contract (my-pow x n)
(-> flonum? exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pow(x, n)
* 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:
double myPow(double x, int n) {
}
};
```

Java Solution:

```
/**
* Problem: Pow(x, n)
* 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 double myPow(double x, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Pow(x, n)
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 myPow(self, x: float, n: int) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def myPow(self, x, n):
"""
:type x: float
:type n: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Pow(x, n)
* 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} x
* @param {number} n
* @return {number}
*/
var myPow = function(x, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Pow(x, n)
* 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 myPow(x: number, n: number): number {
};
```

C# Solution:

```
/*
* Problem: Pow(x, n)
* 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 double MyPow(double x, int n) {

}
}
```

C Solution:

```
/*
* Problem: Pow(x, n)
* 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
*/

double myPow(double x, int n) {

}
```

Go Solution:

```
// Problem: Pow(x, n)
// 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 myPow(x float64, n int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun myPow(x: Double, n: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func myPow(_ x: Double, _ n: Int) -> Double {

}
}
```

Rust Solution:

```
// Problem: Pow(x, n)
// 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 my_pow(x: f64, n: i32) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Float} x
# @param {Integer} n
# @return {Float}
def my_pow(x, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Float $x
* @param Integer $n
* @return Float
*/
function myPow($x, $n) {

}
}
```

Dart Solution:

```
class Solution {
double myPow(double x, int n) {

}
}
```

Scala Solution:

```
object Solution {
def myPow(x: Double, n: Int): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec my_pow(x :: float, n :: integer) :: float
def my_pow(x, n) do

end
end
```

Erlang Solution:

```
-spec my_pow(X :: float(), N :: integer()) -> float().
my_pow(X, N) ->
.
```

Racket Solution:

```
(define/contract (my-pow x n)
(-> flonum? exact-integer? flonum?)
)
```