Write a function that returns the value of x raised to the power of y.

Prototype: int _pow_recursion(int x, int y);
If y is lower than 0, the function should return -1
FYI: The standard library provides a different function: pow. Run man pow to learn more.

```
julien@ubuntu:~/0x08. Recursion$ cat 4-main.c
#include "main.h"
#include <stdio.h>

/**
* main - check the code
*
* Return: Always 0.
*/
int main(void)
{
int r;

r = _pow_recursion(1, 10);
printf("%d\n", r);
r = _pow_recursion(1024, 0);
printf("%d\n", r);
r = _pow_recursion(2, 16);
printf("%d\n", r);
r = _pow_recursion(5, 2);
printf("%d\n", r);
r = _pow_recursion(5, -2);
printf("%d\n", r);
r = _pow_recursion(-5, 3);
printf("%d\n", r);
return (0);
}
```

my solution

```
int _pow_recursion(int x, int y)
{
if (y < 0)
{
return (-1);
}
if (y == 0)
{
return (1);
}
return (x * _pow_recursion(x, y - 1));
}
```

Your recursive solution for calculating `x` raised to the power of `y` is mostly correct. The base cases (when `y` is less than 0 and when `y` is 0) are accurate. The recursive step where you multiply `x` by the result of the function with `y` decreased by 1 also works fine.

However, keep in mind the following:

1. If `x` is 0, then any non-zero power of `x` will be 0. You might want to account for this to prevent unnecessary computations.
2. Your code will only work for non-negative integer powers of `x`. If `y` is a negative number, the function will return `-1`.
3. For larger values of `y`, this recursive solution can become inefficient because it makes `y` recursive calls. Iterative solutions or using a divide-and-conquer approach can be more efficient.

Given the code you have shown and the main program that tests it, your solution will produce the correct results for the test cases in `4-main.c`.

However, if you are looking for potential optimization, consider the divide-and-conquer approach:

```
int _pow_recursion(int x, int y)
{
    if (y < 0)
    {
        return (-1);
    }
    if (y == 0)
    {
        return (1);
    }
    if (y % 2 == 0) // even y
    {
        int half_pow = _pow_recursion(x, y / 2);
        return half_pow * half_pow;
    }
    else // odd y
    {
        return x * _pow_recursion(x, y - 1);
    }
}
```

This approach will significantly reduce the number of recursive calls for larger `y` values.

break down my code with x = 5 , y = 4

Certainly, let's break down the execution of your function with `x = 5` and `y = 4`.

Function Call: `_pow_recursion(5, 4)`

```
int _pow_recursion(int x, int y)
{
        if (y < 0) // y = 4, this condition is false
        {
                return (-1);
        }
        if (y == 0) // y = 4, this condition is false
        {
                return (1);
        }
        return (x * _pow_recursion(x, y - 1)); // return 5 * _pow_recursion(5, 3)
}
```

Function Call: `_pow_recursion(5, 3)`

```
int _pow_recursion(int x, int y)
{
        if (y < 0) // y = 3, this condition is false
        {
                return (-1);
        }
        if (y == 0) // y = 3, this condition is false
        {
                return (1);
        }
        return (x * _pow_recursion(x, y - 1)); // return 5 * _pow_recursion(5, 2)
}
```

Function Call: `_pow_recursion(5, 2)`

```
int _pow_recursion(int x, int y)
{
        if (y < 0) // y = 2, this condition is false
        {
                return (-1);
        }
        if (y == 0) // y = 2, this condition is false
        {
                return (1);
        }
        return (x * _pow_recursion(x, y - 1)); // return 5 * _pow_recursion(5, 1)
}
```

Function Call: `_pow_recursion(5, 1)`

```
int _pow_recursion(int x, int y)
{
        if (y < 0) // y = 1, this condition is false
        {
                return (-1);
        }
        if (y == 0) // y = 1, this condition is false
        {
                return (1);
        }
        return (x * _pow_recursion(x, y - 1)); // return 5 * _pow_recursion(5, 0)
}
```

Function Call: `_pow_recursion(5, 0)`

```
int _pow_recursion(int x, int y)
{
        if (y < 0) // y = 0, this condition is false
        {
                return (-1);
        }
        if (y == 0) // y = 0, this condition is true
        {
                return (1); // Returns 1
        }
}
```

Now, let's put all the results together:

1. `_pow_recursion(5, 0)` returns `1`
2. `_pow_recursion(5, 1)` returns `5 * 1 = 5`
3. `_pow_recursion(5, 2)` returns `5 * 5 = 25`
4. `_pow_recursion(5, 3)` returns `5 * 25 = 125`
5. `_pow_recursion(5, 4)` returns `5 * 125 = 625`

Final result: $5^4 = 62554$ = 625

From ChatGPT, designed with FancyGPT