Computing Power

Given two integers **x** and **n**, write a function to compute x^n. We may assume that x and n are small and overflow doesn't happen.

Examples :

```
Input : x = 2, n = 3
Output : 8
Input : x = 7, n = 2
Output : 49
```

Naive Approach: To solve the problem follow the below idea:
A simple solution to calculate pow(x, n) would multiply x exactly n times. We can do that by using a simple for loop

Below is the implementation of the above approach:

Python

```python
def power(x, n):

    # initialize result by 1
    pow = 1

    # Multiply x for n times
    for i in range(n):
        pow = pow * x

    return pow


# Driver code
if __name__ == '__main__':

    x = 2
    n = 3
```

Prev
Next

```
    # Function call
    print(power(x, n))
```


Output

```
8
```

Time Complexity: O(n)
Auxiliary Space: O(1)

pow(x, n) using recursion:

We can use the same approach as above but instead of an iterative loop, we can use recursion for the purpose.

Python

```
def power(x, n):

    # If x^0 return 1
    if (n == 0):
        return 1

    # If we need to find of 0^y
    if (x == 0):
        return 0

    # For all other cases
    return x * power(x, n - 1)


# Driver Code
if __name__ == "__main__":
    x = 2
    n = 3

    # Function call
    print(power(x, n))
```

Output

```
8
```

Time Complexity: O(n)
Auxiliary Space: O(n) n is the size of the recursion stack

Program to calculate pow(x, n) using Divide and Conqueror approach:

To solve the problem follow the below idea:

> *The problem can be recursively defined by:*
>
> - *power(x, n) = power(x, n / 2) * power(x, n / 2); // if n is even*
> - *power(x, n) = x * power(x, n / 2) * power(x, n / 2); // if n is odd*

Below is the implementation of the above approach:

Python

```python
# Function to calculate x raised to the power y in O(logn)
def power(x, y):
    temp = 0
    if(y == 0):
        return 1
    temp = power(x, int(y / 2))
    if (y % 2 == 0)
    return temp * temp
    else
    return x * temp * temp
```

Output

```
8
```

Time Complexity: O(log n)
Auxiliary Space: O(log n), for recursive call stack

Mark as Read

Dash

All

Articles

Videos

Problems

Quiz

Report An Issue

If you are facing any issue on this page. Please let us know.