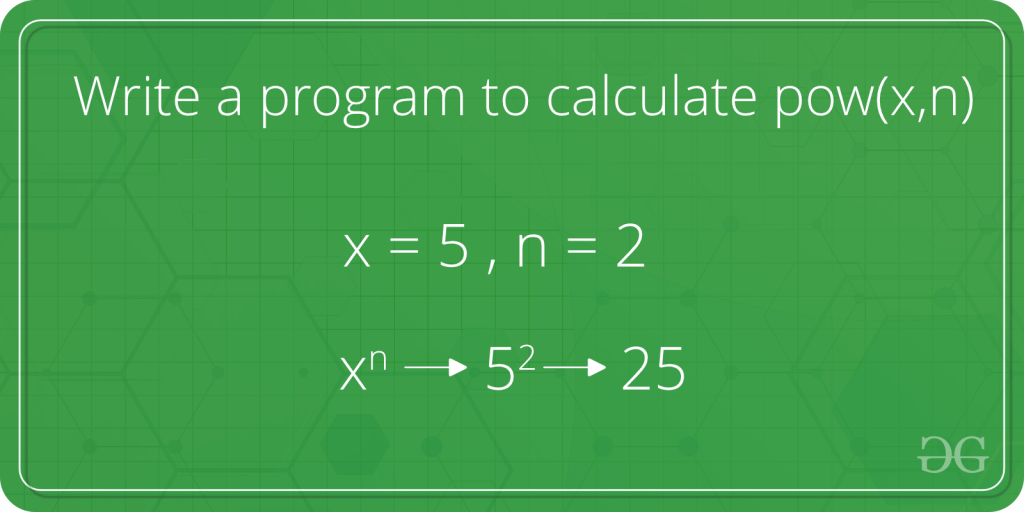
***Computing Power***

Given two integers**x**and **n**, write a function to compute **xn**. 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:

C++Java

// Java program for the above approach

import java.io.\*;

class Gfg {

// Naive iterative solution to calculate pow(x, n)

public static long power(int x, int n)

{

// Initialize result by 1

long pow = 1L;

// Multiply x for n times

for (int i = 0; i < n; i++) {

pow = pow \* x;

}

return pow;

}

// Driver code

public static void main(String[] args)

{

int x = 2;

int n = 3;

System.out.println(power(x, n));

}

};

**Output**

8

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

**An Optimized Divide and Conquer Solution:**

*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*

However there is a problem with the above solution, the same subproblem is computed twice for each recursive call. We can optimize the above function by computing the solution of the subproblem once only.

Below is the implementation of the above approach:

C++Java

/\* Function to calculate x raised to the power y in

\* O(logn)\*/

class GFG {

/\* Function to calculate x raised to the power y \*/

static int power(int x, int y)

{

int temp;

if (y == 0)

return 1;

temp = power(x, y / 2);

if (y % 2 == 0)

return temp \* temp;

else

return x \* temp \* temp;

}

// Driver code

public static void main(String[] args)

{

int x = 2;

int y = 3;

// Function call

System.out.printf("%d", power(x, y));

}

}

**Output**

8

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