Find nth root

Using binary search :

```
double multiply(double number, int n) {
    double ans = 1.0;
    for(int i = 1;i<=n;i++) {
        ans = ans * number;
    }
    return ans;
}
int NthRoot(int n, int m) {
  // Write your code here.
    double low = 1;
    double high = m;
    double eps = 0.1;

    while((high - low) > eps) {
        double mid = (low + high) / 2.0;
        if(multiply(mid, n) < m) {
            low = mid;
        }
        else {
            high = mid;
        }
    }

  int ans=(int)high;
  if(multiply(ans,n)!=m)
    return -1;
  return ans;
}
```

```
/*

    Time Complexity : O(N), because we call the recursion until we multiply the base exponent times. Thus the
    time complexity is linear.

    Space Complexity : O(N), Recursion stack space.

*/


/************************************** Approach 1 ***************************************/

class Solution {
private:
    double power(double x, int n){
        if(n==0){
            return 1;
        }
        return x * power(x, n-1);
    }
public:
    double myPow(double x, int n) {
        if (n == INT_MAX) return (x == 1) ? 1 : (x == -1) ? -1 : 0;
        if (n == INT_MIN) return (x == 1 || x == -1) ? 1 : 0;
```

```
        double num = 1;
        if(n>=0){
            num = power(x, n);
        }
        else{
            n = -n;
            num = power(x, n);
            num = 1.0/num;
        }
        return num;
    }
};




/*

    Time Complexity : O(N), because we loop until we multiply the base exponent times. Thus the time complexity
    is linear.

    Space Complexity : O(1), Constant space.

*/


/************************************* Approach 2 ***************************************/

class Solution {
public:
    double myPow(double x, int n) {
        if (n == INT_MAX) return (x == 1) ? 1 : (x == -1) ? -1 : 0;
        if (n == INT_MIN) return (x == 1 || x == -1) ? 1 : 0;
        double num = 1;
        if(n>=0){
            while(n>0){
                num *= x;
                n--;
            }
        }
        else{
            n = -n;
            while(n>0){
                num *= x;
                n--;
            }
            num = 1.0/num;
        }
        return num;
    }
};






/*

    Time Complexity : O(logN).

    Space Complexity : O(logN), Recursion stack space.
```

```
*/


/************************************** Approach 3 **************************************/

class Solution {
public:
    double myPow(double x, int n) {
        if(n==0) return 1;
        if(n<0) {
            n = abs(n);
            x = 1/x;
        }
        if(n%2==0){
            return myPow(x*x, n/2);
        }
        else{
            return x*myPow(x, n-1);
        }
    }
};







/*

    Time Complexity : O(logN).

    Space Complexity : O(1), Constant space.

*/


/************************************** Approach 4 **************************************/

class Solution {
public:
    double myPow(double x, int n) {
        double num = 1;
        long long nn = n;
        if(nn < 0) nn = -nn;
        while(nn>0){
            if(nn%2==1){
                num = num * x;
                nn--;
            }
            else{
                x = x*x;
                nn/=2;
            }
        }
        if(n < 0){
            num = 1.0/num;
        }
        return num;
    }
};
```

```
/*

    Time Complexity : O(logN).

    Space Complexity : O(logN), Constant space.

*/


/************************************* Approach 5 *************************************/

class Solution {
public:
    double myPow(double x, int n) {
        return pow(x, n);
    }
};
```