Recursion - Level-1 Revision

- Print Increasing $\{\overset{\text{Faith}}{\boxed{1, 2, 3, 4}}\ \textcircled{5}\}$ → Postorder
- Print Decreasing Preorder $\{\textcircled{5}\ \boxed{4, 3, 2, 1}\}$ → faith
- Print Increasing Decreasing

$\{\textcircled{5}\ \overset{\text{Faith}}{\boxed{4, 3, 2, 1, 1, 2, 3, 4}}\ \textcircled{5}\}$

$$\nearrow (1)^N + \{k + k\} * N = O(N)$$

HW

① Expectation :→ $pI(N)$:→ $1, 2, \ldots N$

Faith :→ $pI(N-1)$:→ $1, 2, 3, \ldots N-1$

Meeting Expectatn :→ Syso(N)

Preorder	Postorder
Syso(N)	pI(N-1)

Preorder

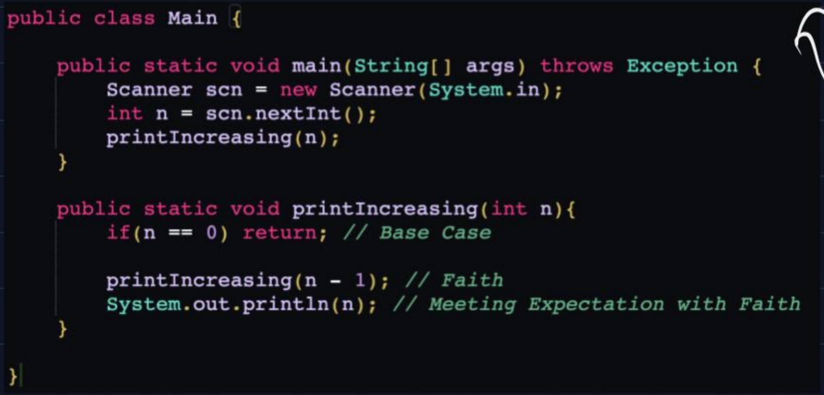
Syso(N)

pI(N-1)

Postorder

pI(N-1)

Syso(N)



```
public class Main {

    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        printIncreasing(n);
    }

    public static void printIncreasing(int n){
        if(n == 0) return; // Base Case

        printIncreasing(n - 1); // Faith
        System.out.println(n); // Meeting Expectation with Faith
    }

}
```

Call → ①

height → Ⓝ

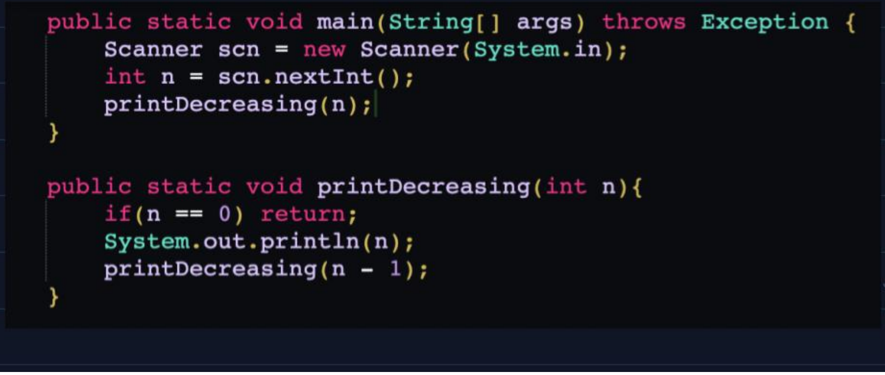
$(1)^N + \{0+k\}*N \Rightarrow \underline{O(N)}$

5 4-3-2-1

② Expectation :→ pD(N) :→ $\boxed{N}$, $\boxed{N-1, N-2, ---2, 1}$

Faith :→ pD(N-1) :→ N-1, N-2, ---1

Meeting Expectatn :→ Preorder : Syso(N)



```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    printDecreasing(n);
}

public static void printDecreasing(int n){
    if(n == 0) return;
    System.out.println(n);
    printDecreasing(n - 1);
}
```

calls = 1, height = N

$(1)^N + \{k+0\}*N$

$= \underline{O(N)}$

Power Function

↳ Expectation :→ x^n ← pow(x,n)

$x = 2.0$, $n = 5$; $2^5 = 32$

Faith → $x^{n-1} * x$

pow(x,n-1) ↙ x^{n-1}

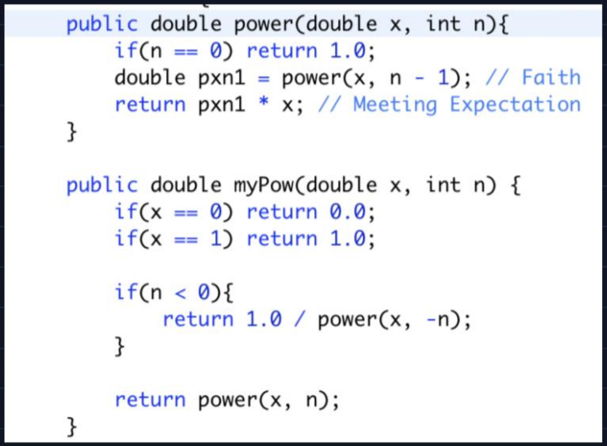
x ↳ Meeting Expectation

Generic Time Complexity of Recursion

↳ $$\boxed{(\text{Calls})^{\text{height}} + \{\text{Preorder} + \text{Postorder}\} * \text{height}}$$

Generic Time Complexity of Recursion

$$\boxed{(\text{Calls})^{\text{height}} + \{\text{Preorder} + \text{Postorder}\} * \text{height}}$$



```java
public double power(double x, int n){
    if(n == 0) return 1.0;
    double pxn1 = power(x, n - 1); // Faith
    return pxn1 * x; // Meeting Expectation
}

public double myPow(double x, int n) {
    if(x == 0) return 0.0;
    if(x == 1) return 1.0;

    if(n < 0){
        return 1.0 / power(x, -n);
    }

    return power(x, n);
}
```

Brute force

Calls = 1

height = N

$$(1)^N + \{k + k\} * N$$

$$= \boxed{\underline{O(N)}}$$

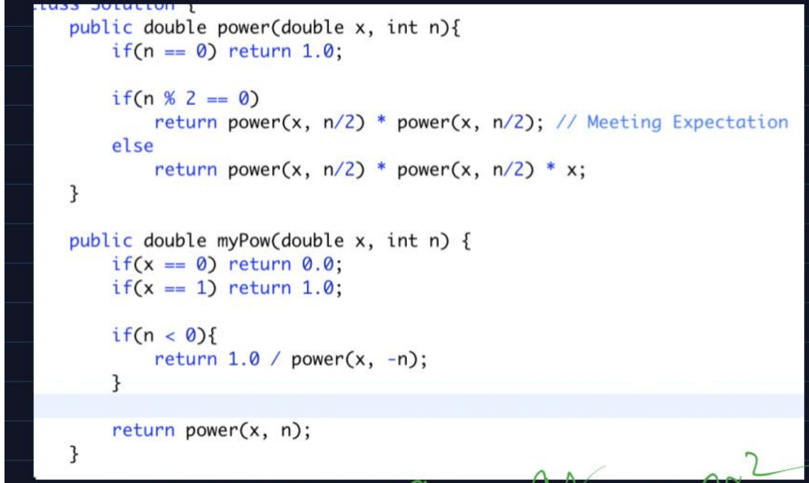
Space

- Input → O(12 bytes) = O(1)
- Output → O(8 bytes) = O(1)
- Auxiliary space (Extra) → O(1)
  - Data Structures
  - Algorithm
- Recursion call space → O(n)
  - → O(height)

Power → Optimized

even
$$\begin{cases} x^n = x^{n/2} * x^{n/2} \\ 2^6 = 2^3 * 2^3 = 2^{3+3} = 2^6 \end{cases}$$

odd
$$\begin{cases} x^n = x^{n/2} * x^{n/2} * x \\ 2^7 = 2^3 * 2^3 * 2 = 2^6 * 2 = 2^7 \end{cases}$$



```
class Solution {
    public double power(double x, int n){
        if(n == 0) return 1.0;

        if(n % 2 == 0)
            return power(x, n/2) * power(x, n/2); // Meeting Expectation
        else
            return power(x, n/2) * power(x, n/2) * x;
    }

    public double myPow(double x, int n) {
        if(x == 0) return 0.0;
        if(x == 1) return 1.0;

        if(n < 0){
            return 1.0 / power(x, -n);
        }

        return power(x, n);
    }
```

calls ⇒ 2 → breadth

height ⇒ $\log_2 n$

↳ depth

$\Rightarrow O\left((2)^{\log_2 n}\right) = \boxed{O(n)}$

Input → O(1)

output → O(1)

Extra → O(1)

Rec. call stack → $O(\log_2 n)$

$a \quad ar \quad ar^2 \quad ar^{h-1}$

$n \to n/2 \to n/4 \to n/8 \to n/16 \text{ ------} \to 1$

h terms

```
    return power(x, n);
}
```

$\Rightarrow O(($

$$\overset{a}{n} \to \overset{ar}{n/2} \to \overset{ar^2}{n/4} \to n/8 \to n/16 \dashrightarrow \overset{ar^{h-1}}{1}$$

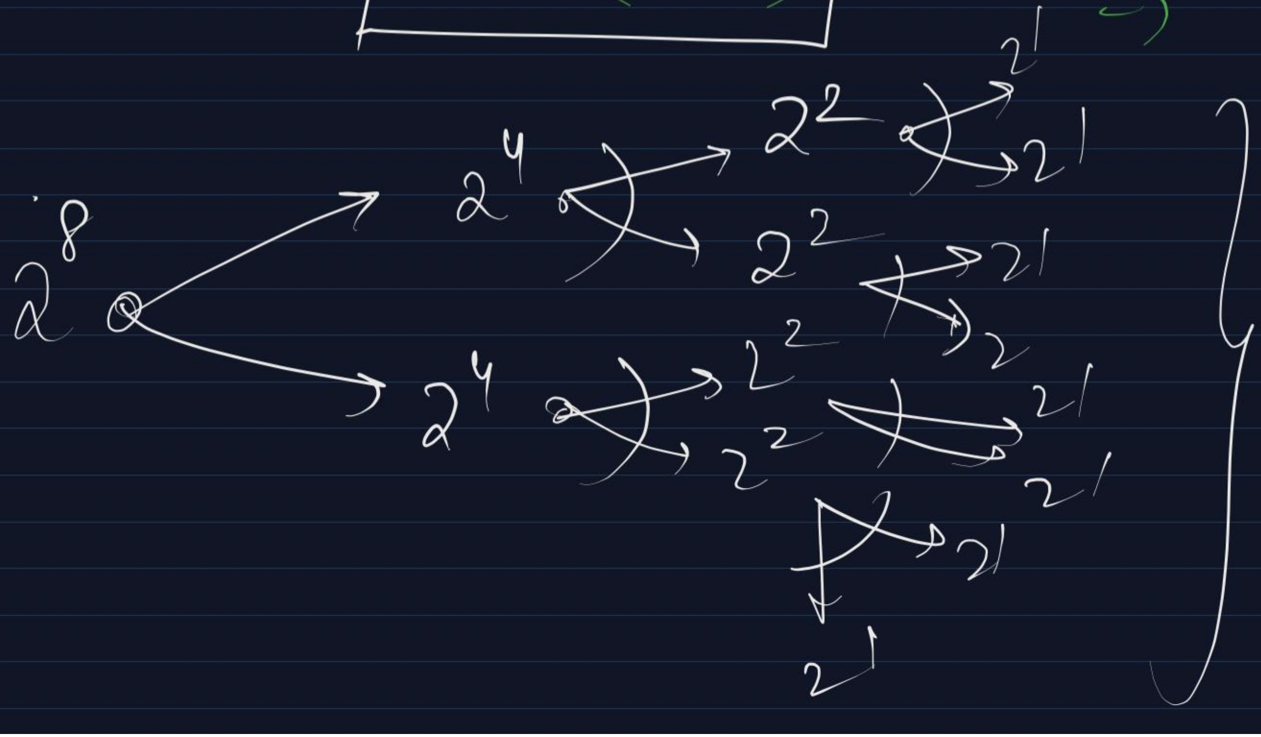
h terms

$$1 = n * (1/2)^{h-1} \Rightarrow 2^{h-1} = n$$

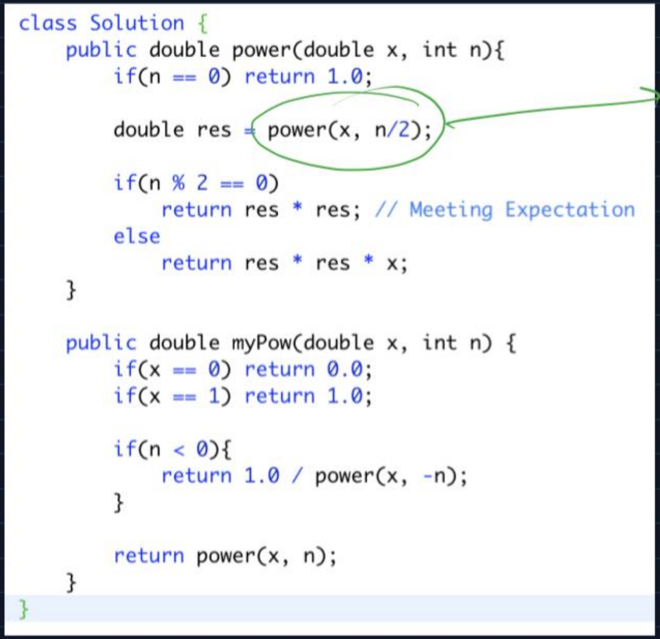
$$\Rightarrow \log_2(2^{h-1}) = \log_2 n$$

$$\Rightarrow h-1 = \log_2 n$$

$$\boxed{h = O(\log_2 n)}$$



Calls will increase



```
class Solution {
    public double power(double x, int n){
        if(n == 0) return 1.0;

        double res = power(x, n/2);

        if(n % 2 == 0)
            return res * res; // Meeting Expectation
        else
            return res * res * x;
    }

    public double myPow(double x, int n) {
        if(x == 0) return 0.0;
        if(x == 1) return 1.0;

        if(n < 0){
            return 1.0 / power(x, -n);
        }

        return power(x, n);
    }
}
```

call = 1

Same S.C as prev

$x^n \to x^{n/2} \to x^{n/4} \to n^{n/8}$

height → $h = O(\log_2 n)$

$O\left((1)^{\log_2 n} + k * \log_2 n \right)$

$= O(\log_2(n)) \approx O(1)$

$\hookrightarrow 2^{31}-1$

$O(\log_2 n) << O(n)$

$N = 2^{16}$

$\log_2 n = 16$

Bit Manipulatn → Modular Exponentiation

→ TC: $O(\log_2 n)$

→ SC: $O(1)$