9-dec-2021

09 December 2021 21:01

Factorial =)

Power =

log Power

Tower of hanoi

Factorial :-

$5! = 5 \times (4 \times 3 \times 2 \times 1) = 120$

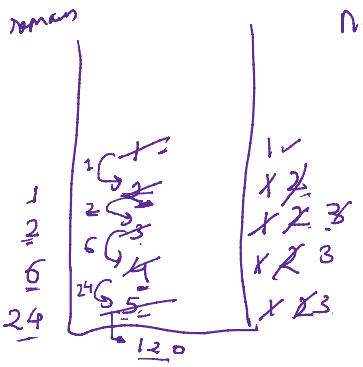
$4! = 4 \times (3 \times (2 \times 1)) = 24$

$1! = 1$

$n = 5$

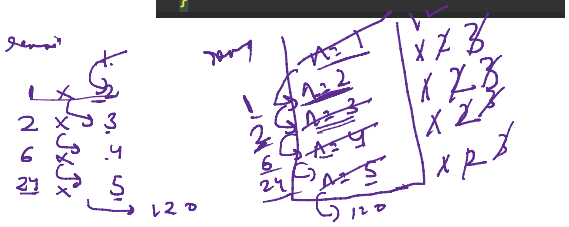
```
int factorial (n) {
if (n == 1) return 1;

int remaining = factorial (n-1)
return n x remaining;
}
```



$n = 5$

```
public static int factorial(int n) {
  1 if (n == 1) return 1;
  2 int remaining = factorial( n - 1);
  3 return remaining * n;
}
```



Power :-

$2^5 = 2 \times 2 \times 2 \times 2 \times 2 = 32$

x, n

$x^{(n)} = x \times x^{(n-1)}$

$2^5 = 2 \times 2^{(4)}$

$(n == 1)$

$2^1 = 2$

$1 \le n \le 10^4$

```
Pow (x, n) {
if(n == 1) return x;
int rem = Pow(x, n-1);
return x x rem;
}
```

$2^5 = 2 \times 2 \times 2 \times 2 \times 2$

$= 2^2 \times 2^2 \times 2$

$2^4 = 2^2 \times 2^2$

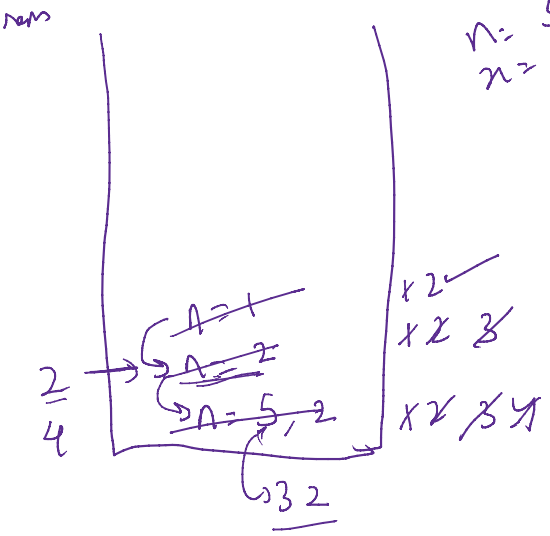
$5/2 = 2$

```
PowOptimize(x, n){
if (n == 1) return x;
(n == 0) return 1;
```

$5/2 = 2$

if (n==1) return x;
(n==0) return 1;
divP = n/2;
int remain q = powOptim(x, divP);
if (n%2 == 1) {
return remain q × remain q × x;
} else {
return remain q × remain q;
}

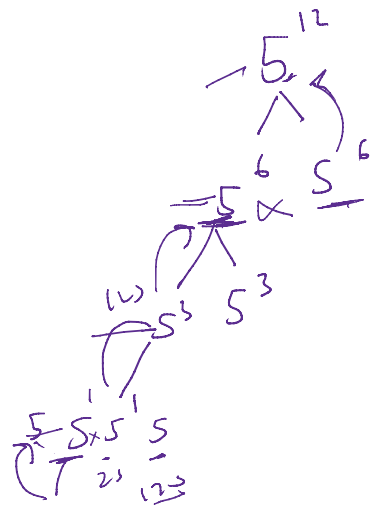
n = 5
x = 2




```
public static int powOptimize(int x, int n) {
    if (n == 0) return 1;
    if (n == 1) return x;
    int divN = n / 2;
    int remaining = powOptimize(x, divN);
    if (n % 2 == 1) {
        return remaining * remaining * x;
    } else {
        return remaining * remaining;
    }
}
```

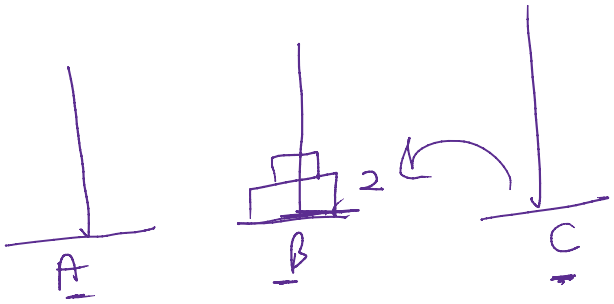
$5/2 = 2$

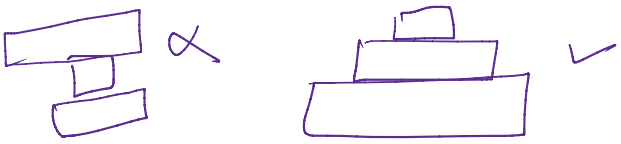
$2^2 \times 2^2 \times 2$



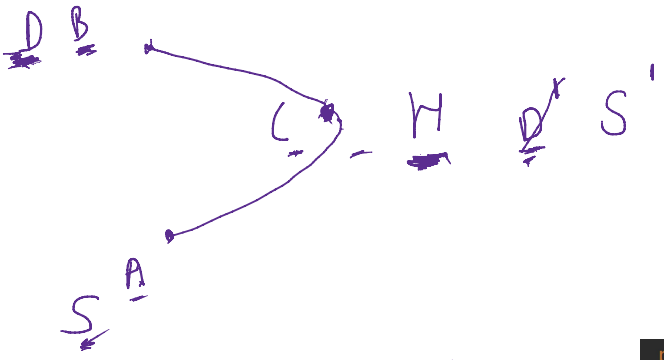
Tower of Hanoi :-

n = 2





1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No larger disk may be placed on top of a smaller disk.




```
public static void toh(int n, char s, char d, char h) {
    if (n == 0) return;
    toh(n - 1, s, h, d);
    System.out.println(n + ":" + s + "->" + d);
    toh(n - 1, h, d, s);
}
```

	s	h	d	
n=0	A	B	C	1
n=0	B	C	A	1
n=1	B	A	C	1 2 3 4
n=0	C	A	B	1
n=0	A	B	C	1
n=1	A	C	B	1 2 3 4
n=2	A	B	C	1 2 3 4

1 : A -> B
2 : A -> C
1 : B -> C

n	s	d	h	
n=1	A	C	B	1 2 3 4 (n=0, n=0)
n=1	B	A	C	1 2 3 4 (n=0, n=0)
n=2	B	C	A	1 2 3 4
n=0				
n=0				
n=1	C	B	A	1 2 3 4
n=0				
n=0				
	A	C	B	1 2 3 4

```
public static void toh(int n, char s, char d, char h) {
    if (n == 0) return;
    toh(n - 1, s, h, d);
    System.out.println(n + ":" + s + "->" + d);
    toh(n - 1, h, d, s);
}
```

1 : A -> C
2 : A -> B
1 : C -> B
2 : A -> C

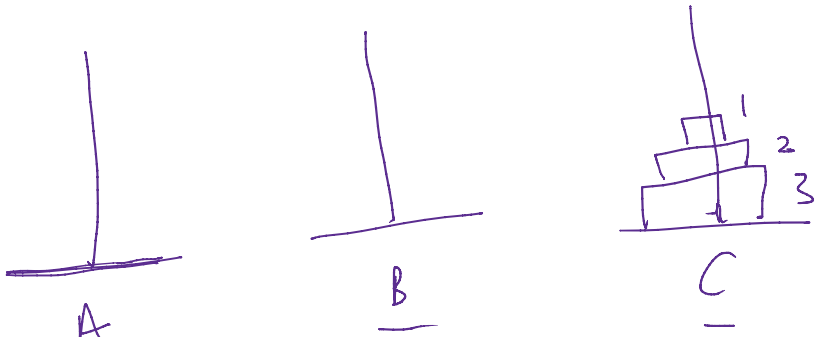
```
1:A->C
2:A->B
1:C->B
3:A->C
1:B->A
2:B->C
```

~~n=1~~	A	C	B
~~n=2~~	A	B	C
~~n=3~~	A	C	B

1: C → B
3: A → C
1: B → A
2: B → C
1: A → C

```
3:A->C
1:B->A
2:B->C
1:A->C
```

```
1:A->C
2:A->B
1:C->B
3:A->C
1:B->A
2:B->C
1:A->C
```



$O(2^{(n)})$

$A[i] = A[A[i]]$

$A[\ ] = A[i]$

4 2 3 0 1
0 1 2 3 4