★ Decrease print Increase print

ip: n = 3

op: 3 2 1 1 2 3

#1:

```
function printDec(n) {
    if(n==0) {
        return;
    }
    process.stdout.write(n + " ");  // 5
    printDec(n-1) // 4 3 2 1
}  // Recursion
```

```
function printInc(n) {
    if(n==0) {
        return;
    }
    printInc(n-1) // 1 2 3 4
    process.stdout.write(n + " "); // 5
}  // Recursion

function printDI(n) {
    printDec(n); // 5 4 3 2 1
    printInc(n); // 1 2 3 4 5
}  // not Recursion
   (not calling itself)
```

#2:

1. faith ⇒

printDI(n)

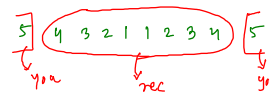
⇒ will print n→1 and 1→n

2. printDI(5)

⇒ 5 4 3 2 1 1 2 3 4 5

printDI(4) ⇕

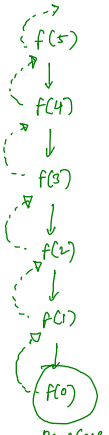
⇒ 4 3 2 1 1 2 3 4



3. function printDI(n) {

```
if(n==0) {
    return;
}
console.log(n); // 5
printDI(n-1); // 4 3 2 1 1 2 3 4
console.log(n); // 5
}
```

4.



op: 5, 4, 3, 2, 1, 1, 2, 3, 4, 5

★ Factorial :

ip: n = 5, op: 5! = 5 x 4 x 3 x 2 x 1
= 120

0! = 1

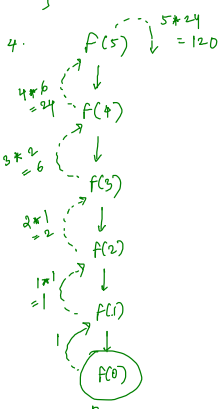
1. faith
⇒ factorial(n)
⇒ n x n-1 x n-2 x ... x 2 x 1

2. factorial(5)
⇒ 5 x 4 x 3 x 2 x 1

factorial(4)
⇒ 4 x 3 x 2 x 1

5! = 5 x 4!

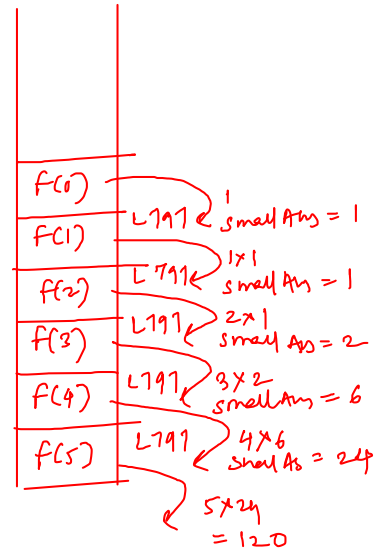
3.
```
function factorial(n) {
    if(n == 0) {
        return 1;
    }
    const smallAns = factorial(n-1); // 4!
    return n * smallAns;
}
```

4.


```
792  function factorial_recursive(n) {
793    if (n == 0) {
794      return 1;
795    }
796
797    const smallAns = factorial_recursive(n - 1);
798    return n * smallAns;
799  }
```

(post-way)

(getting the answer)



pre-order way :

```
function factorial (curr_num, curr_fact, n) {
    if (curr_num == n) {
        return curr_fact
    }

    return factorial (curr_num + 1, curr_fact * (curr_num + 1), n)
}
```

* Keep building your answer as recursion progress using some additional variables in the function parameters.

1. factorial (curr_num, curr_fact, n)

   ⇒ curr_fact will store the (curr_num)!

2. factorial (1, 1, 5)

   factorial (2, 2, 5)

   factorial (3, 6, 5)

   factorial (4, 24, 5)

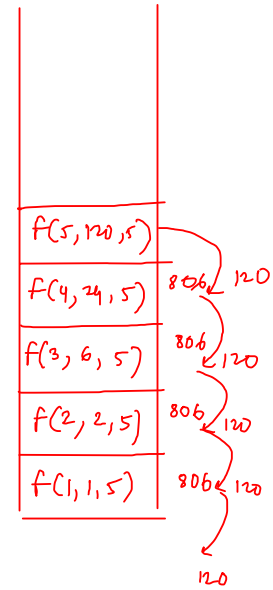
   factorial (5, 120, 5)

```
801  function fact(curr_num, curr_fact, n) {
802    if (curr_num == n) {
803      return curr_fact;
804    }
805
806    return fact(curr_num + 1, curr_fact * (curr_num + 1), n);
807  }
```

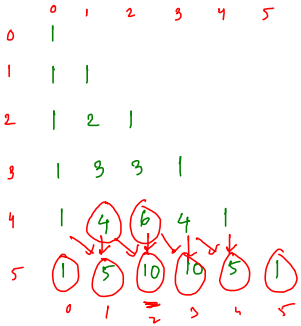
fact (1, 1, 5)

9:30 - 9:45 pm

BREAK



Doubt pascal's triangle



arr[5][2] = arr[4][1]
+ arr[4][2]
= 4 + 6 = 10

no. of cols = row + 1

row = 0, no. of cols = 1
row = 1, no. of cols = 2
row = 2, no. of cols = 3

arr[row][0] = 1

arr[row][no. of cols - 1] = 1

arr[row][col] = arr[row-1][col-1]
+ arr[row-1][col]

```
for( row = 0; row < n; row++ ) {
   for( col = 0; col <= row; col++ ) {
      if ( col == 0 || col == row ) {
         arr[row][col] = 1;
      } else {
         arr[row][col] = arr[row-1][col-1]
                       + arr[row-1][col];
      }
   }
}
```

* x^n :

ip: x = 2 , n = 4 op: $2^4 = 16$

1. faith
   ⇒ power (x, n)
   ⇒ x^n

2. power (2, 4) ⇒ 2^4 ⇒ 16
   power (2, 3) ⇒ 2^3 ⇒ 8

$2^4 = 2 * 2^3$ — you / see

3.
```
function power (x, n) {
    if (n == 0) {
        return 1;
    }

    const smallAns = power(x, n-1) // x^3

    return x * smallAns
}
```

4. f(2, 4) = 16 (2 × 8)

f(2, 4) → n = 4 → f(2, 3)
f(2, 3) → n = 3 → f(2, 2)
f(2, 2) → n = 2 → f(2, 1)
f(2, 1) → n = 1 → f(2, 0)
f(2, 0) — Base Case

pre-order way :

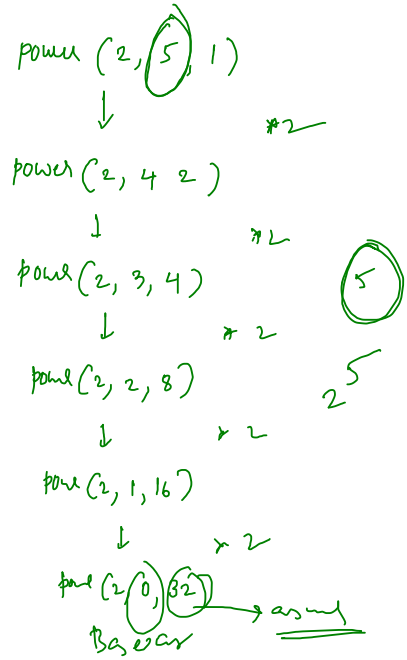
```
function power (x, n, curr_power) {
    if (n == 0) {
        return curr_power;
    }
    return power (x, n-1, curr_power * x);
}
```

5 → 0

5 times

n - calls

⇒ O(n)



* optimized x^n :

```
function power (x, n) {
    if (n == 0) {
        return 1;
    }
    const smallAns = power(x, n/2);
    if (n%2 == 1) {
        return smallAns * smallAns * x;
    }
    return smallAns * smallAns;
}
```

$2^{15} = 2^7 * 2^7 * 2$ ← extra when n is odd

$= 2^{14} * 2$

32 * 32 = 1024

f(2, 10)

4*4*2 = 32 — f(2, 5)

4*1*1 = 4 — f(2, 2)

1*1*2 = 2 — f(2, 1)

1 — f(2, 0) Base case

1. faith
   - ⇒ power(x, n)
   - ⇒ x^n

2. power(2, 10) ⇒ 1024

   power(2, 9) ⇒ 512

$2^{10} = 2^9 * 2$ — 2^9 → rec, 2 → yaa

$2^5 = 2^4 * 2$ — 2^4 → rec, 2 → yaa

$2^{10} = 2^5 * 2^5$ — 2^5 → rec, 2^5 → yaa

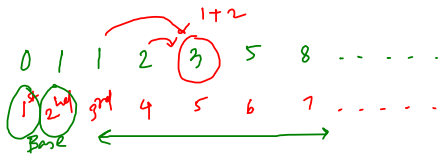
= 2^5

* 10 calls (power) ⟹ O(n)

vs

4 calls (optimized power) ⟹ O(log n)

★ Fibonacci :



n^{th} fib = $n-1^{th}$ fib + $n-2^{th}$ fib

6^{th} fib = 5^{th} fib + 4^{th} fib

= 3 + 2 = 5

1. faith

fib(n) ⇒ n^{th} fibonacci number

2. logic

fib(6) = 5

fib(5) = 3

fib(4) = 2

⇒ fib(6) = fib(5) + fib(4)

rec you rec

```
function fib(n) {
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        return 1;
    }
    return fib(n-1) + fib(n-2);
}
```

★ fib(2)

= fib(1) + fib(0)

= 0 + fib(-1) + fib(-2)

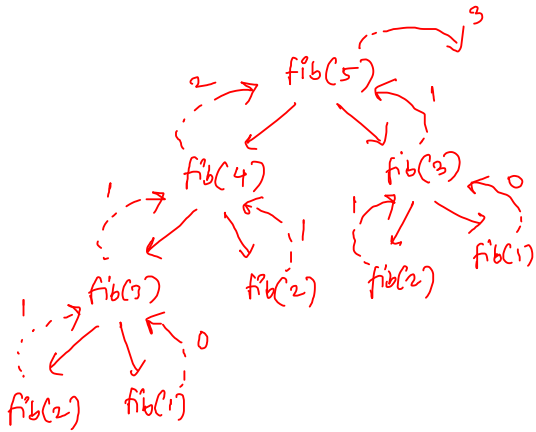
= 0 + fib(-2) + fib(-3)

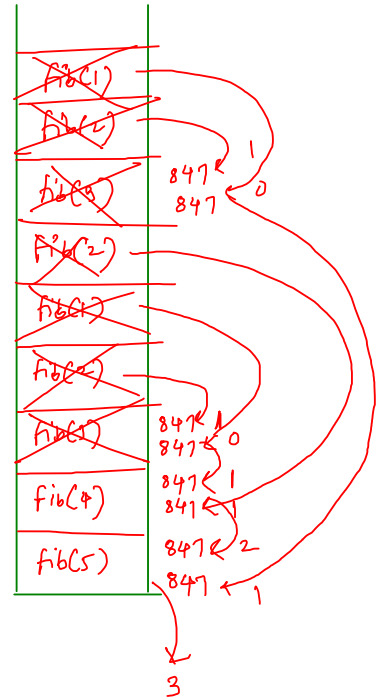
+ fib(-3) + fib(-4)

⋮

```
function fib(n) {
  if (n == 1) return 0;
  if (n == 2) return 1;

  return fib(n - 1) + fib(n - 2);
}
```





```
843  function fib(n) {
844    if (n == 1) return 0;
845    if (n == 2) return 1;
846
847    return fib(n - 1) + fib(n - 2);
848  }
```