Trishanth Naidu

Software Developer | Founder of Peppy UI | Author of Rootz JS

Educator at Relevel

Innovator | Fitness freak | Painter | Chef

You can Follow me on

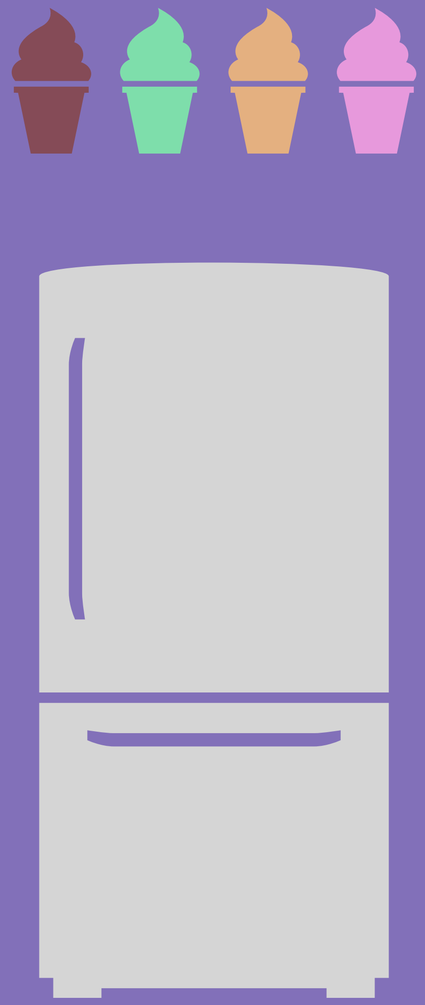
Recursion

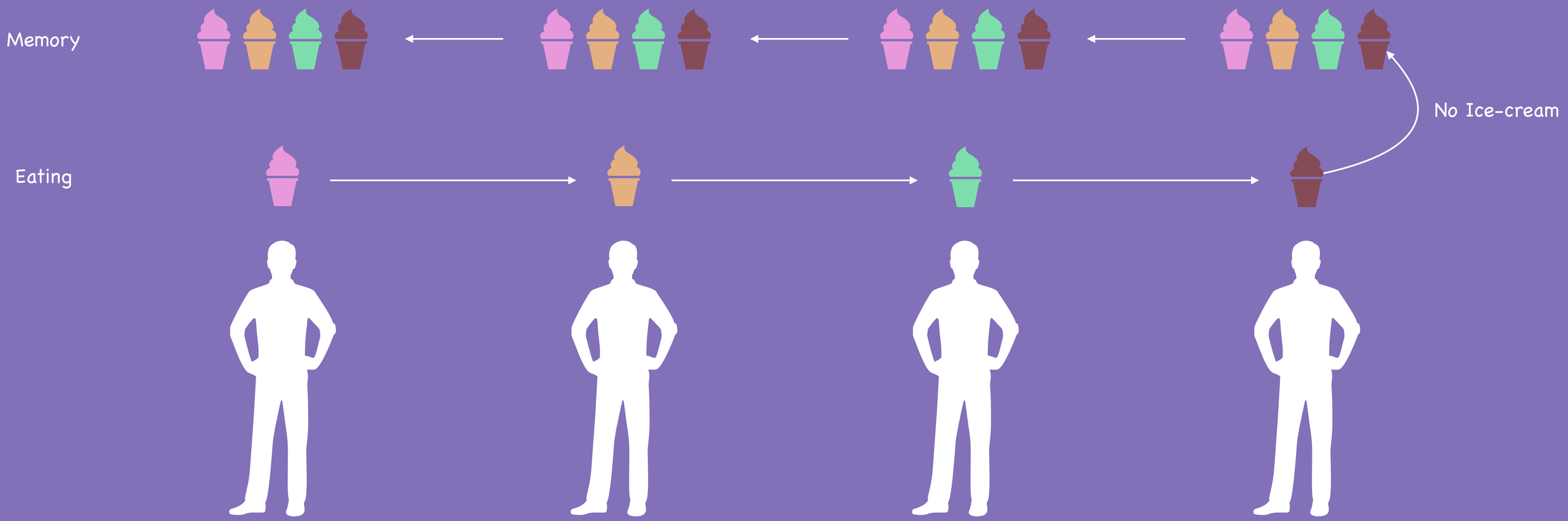
Anything that re-occurs is called recursion

Any process calling itself is called a recursive process

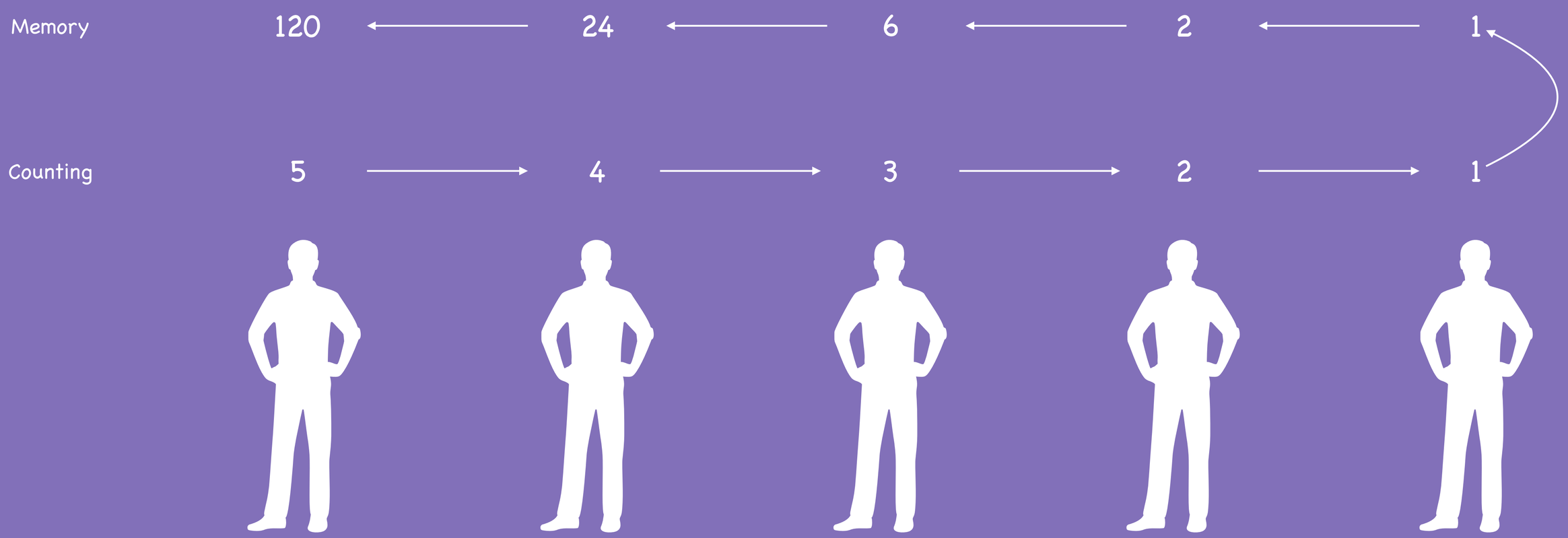
Any function calling itself is called a recursive function

Echo

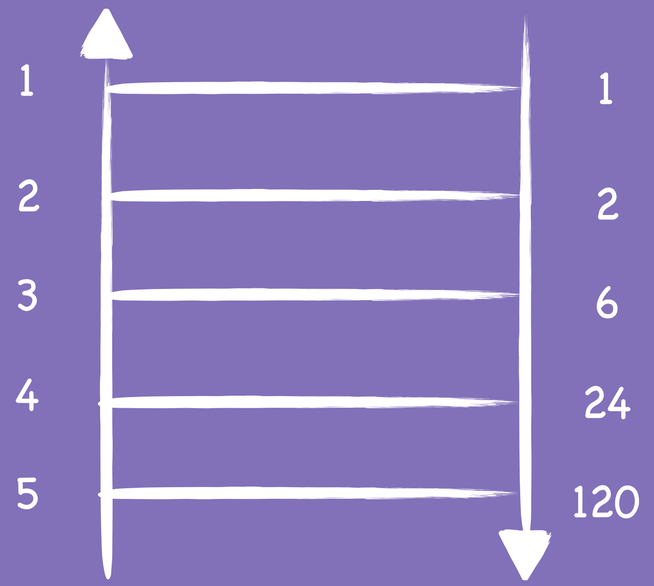



Memory
Eating
No Ice-cream

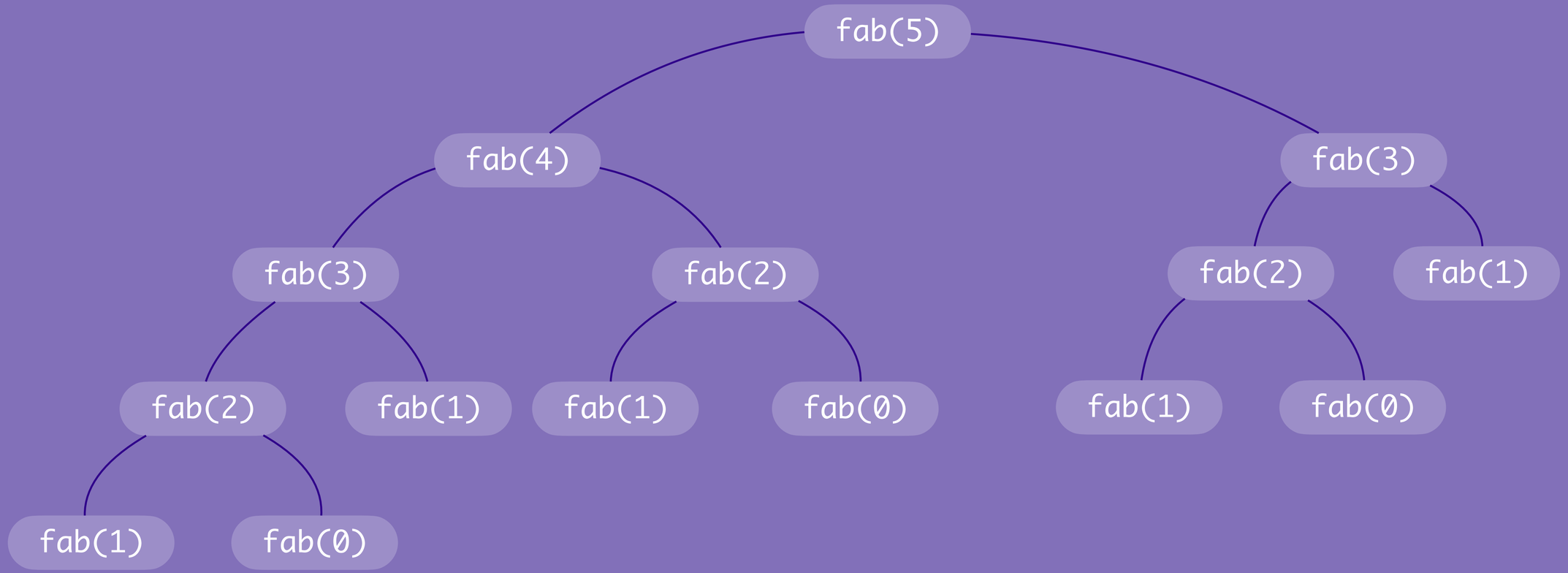
Factorial - passing the parcel



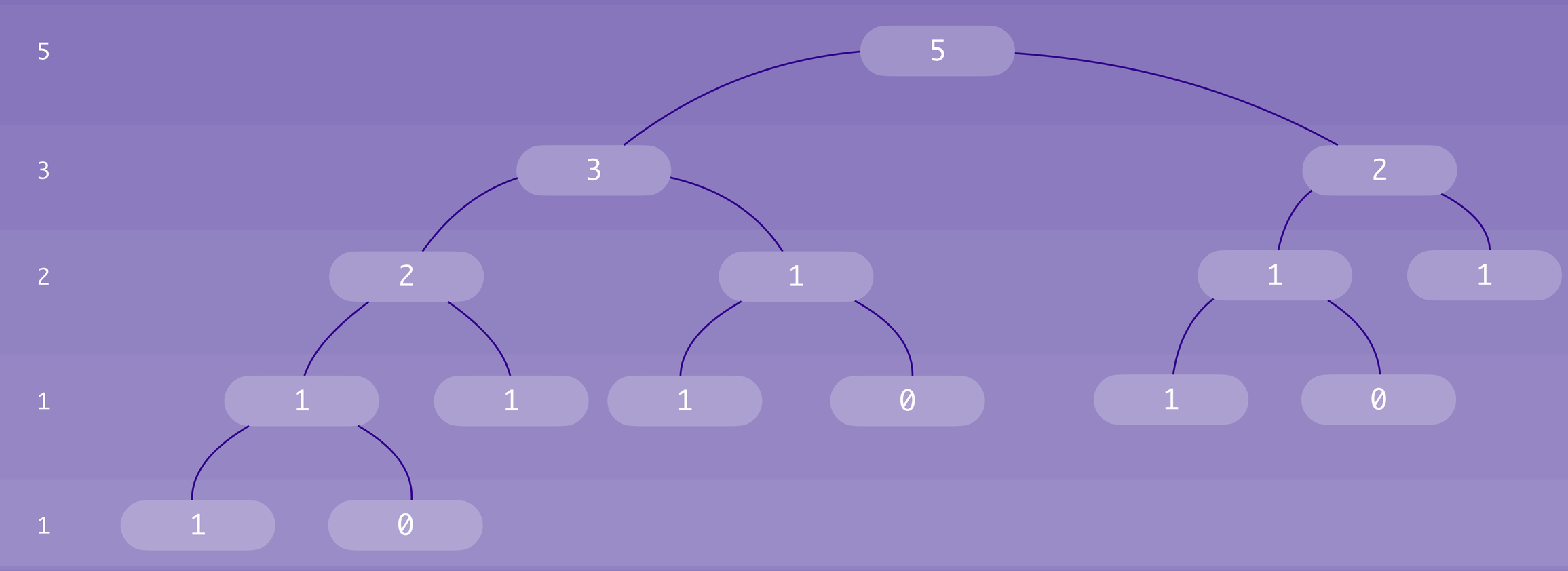
Factorial - using ladder



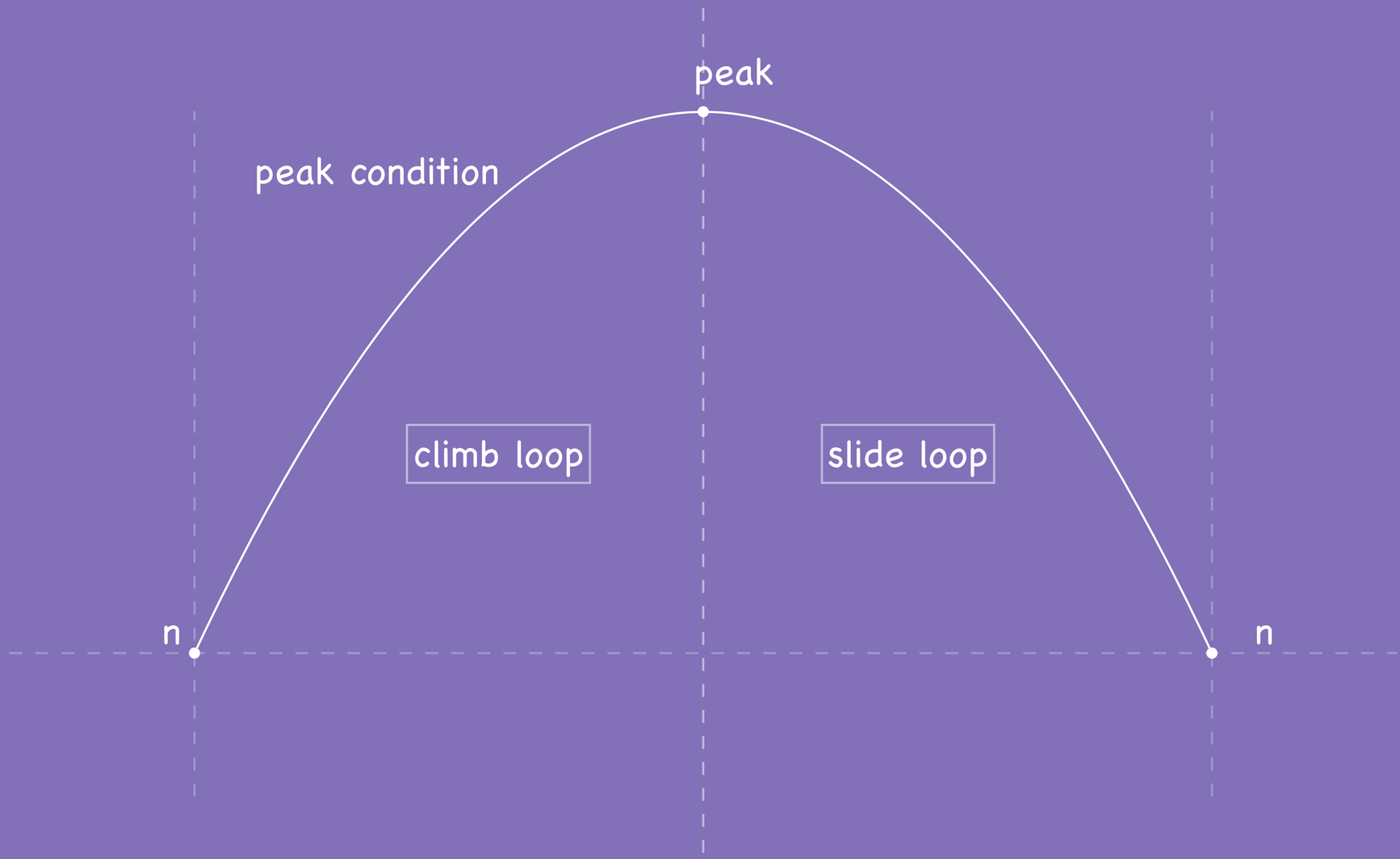
Fibonacci Series



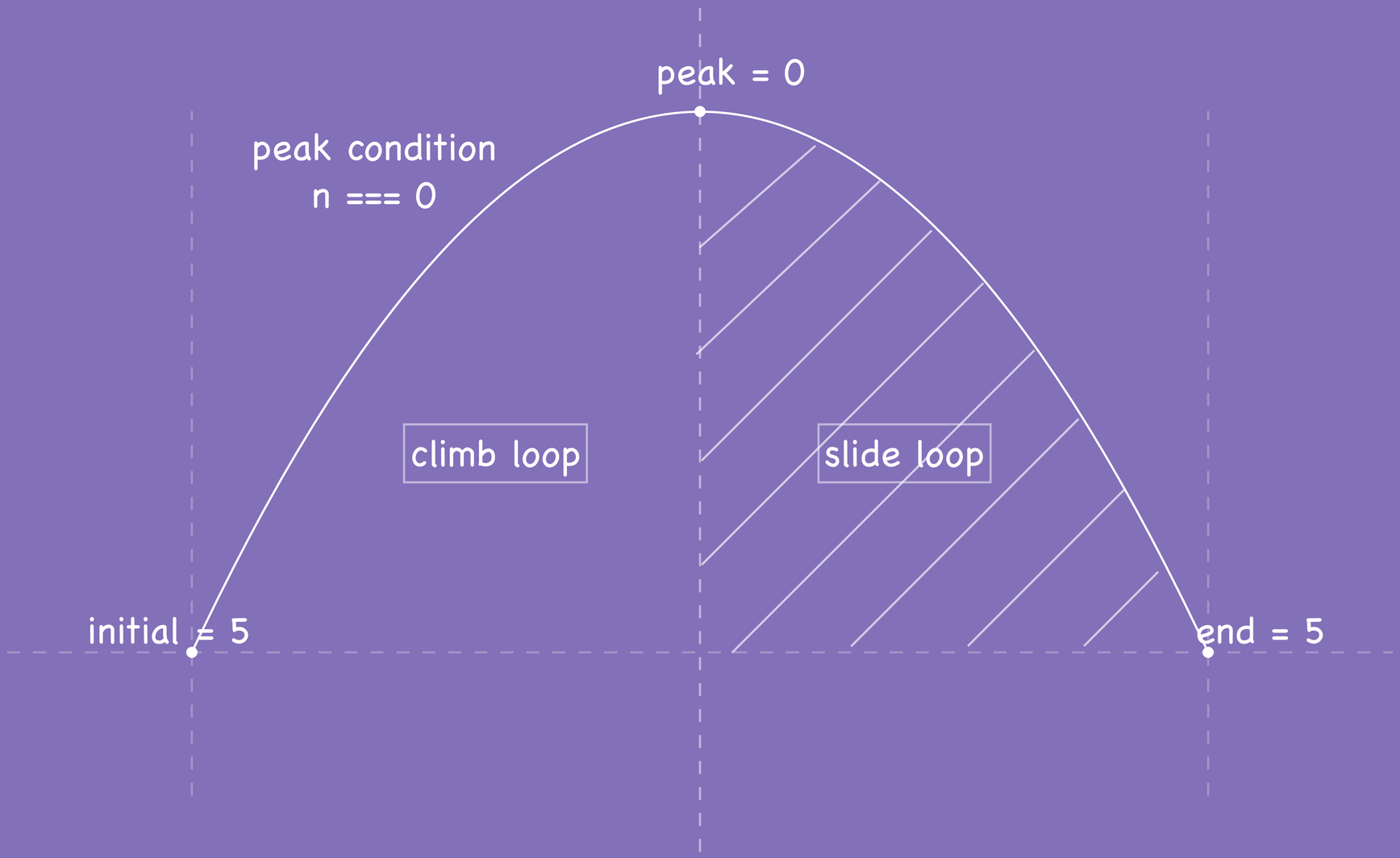
Fibonacci Series



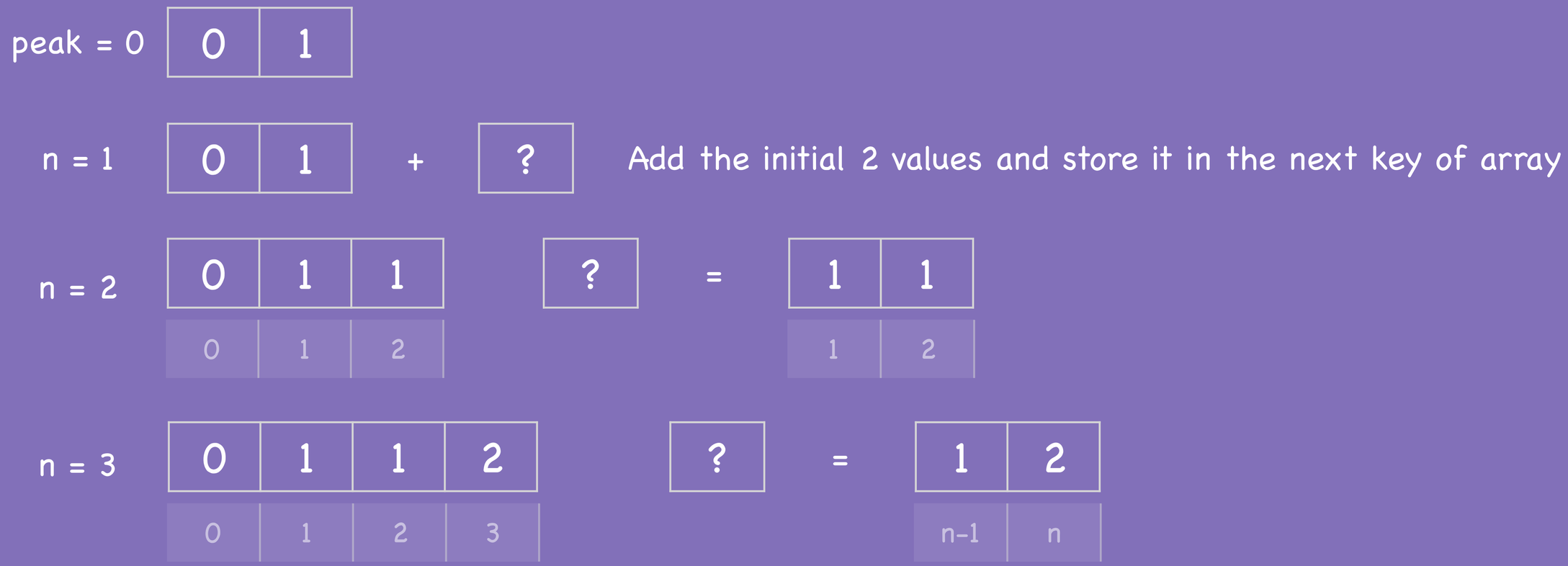
Understanding Recursive functions work



Understanding Recursive functions work for Fibonacci



Understanding Recursive functions work for Fibonacci



Understanding Recursive functions work for Fibonacci

```
function fib(n) {
    // peak
    let peak = 0;

    // peak condition
    if(n === peak) return [0, 1];

    // recursive call
    let result = fib(n-1);

    // slide loop
    result.push(result[n] + result[n-1]);

    return result;
}
```

climb loop

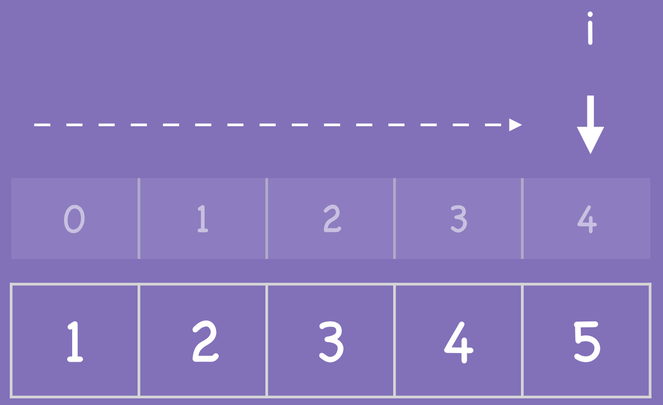
slide loop

Sum Triangle from Array

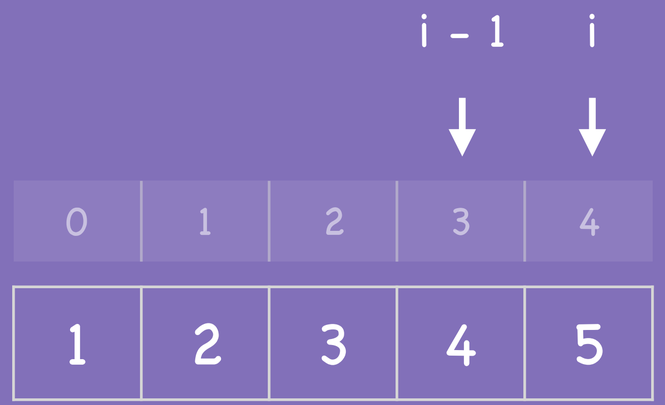
Sum Triangle from Array

```
function pt(arr) {
    // peak
    let peak = 1;
    let compute = [];

    // peak condition
    if(arr.length === peak) {
        return arr;
    }

    // climb loop / logic
    for(i=0;i<arr.length - 1;i++) {
        compute.push(arr[i] + arr[i+1]);
    }

    let result = pt(compute);
    console.log(result);

    return arr;
}

pt([1,2,3,4,5])
```

Sorted Array check

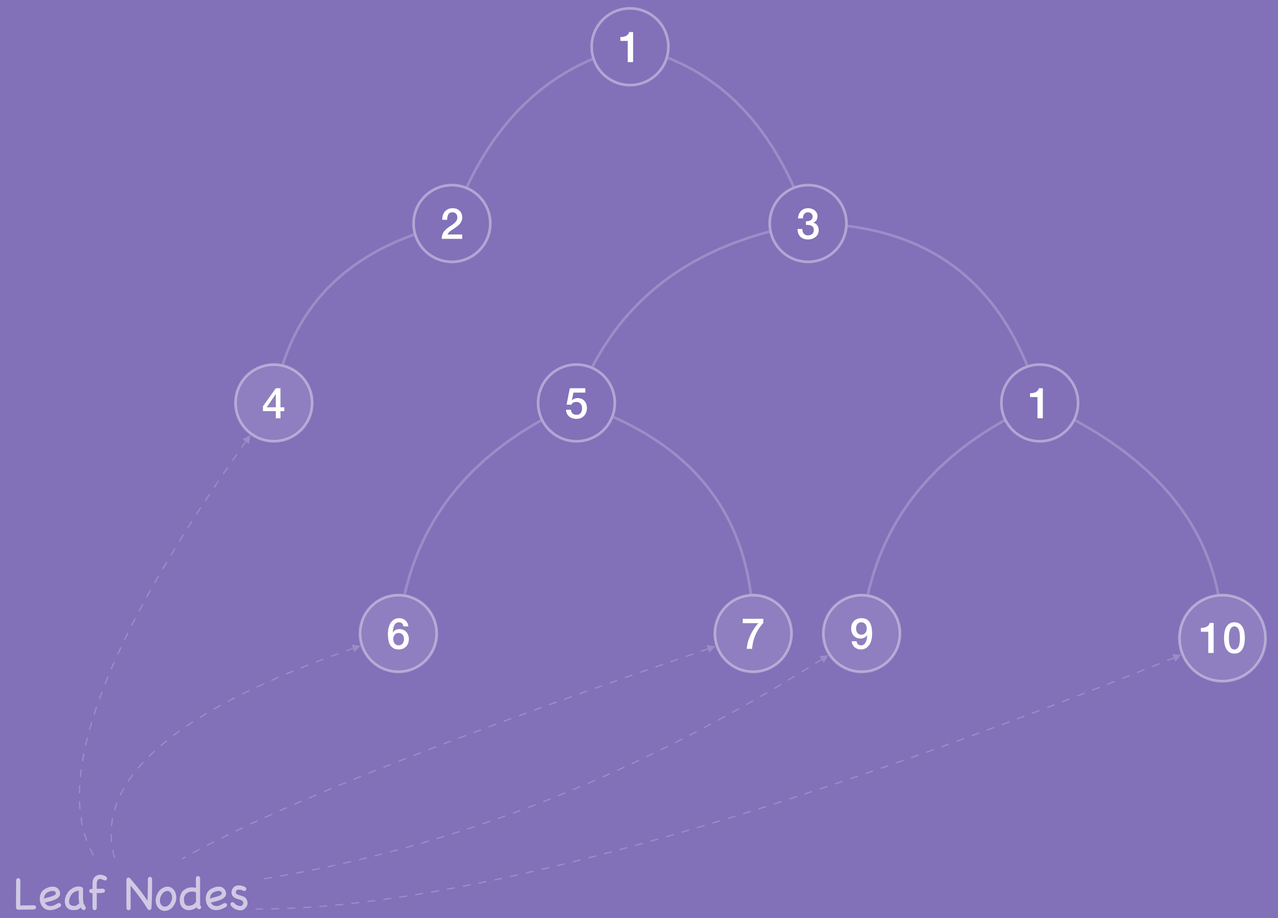


Base condition



If i is greater than equal to i - 1,
then return true,
else false

Traversing a Binary Tree



Build a Binary Tree

```
// constructor to build the binary tree
function Node(val) {
    this.value = val;
    this.left = null;
    this.right = null;
}

// add values to the tree
var root = new Node(1);
root.left = new Node(2);
root.right = new Node(3);
root.left.left = new Node(4);
root.right.left = new Node(5);
root.right.right = new Node(8);
root.right.left.left = new Node(6);
root.right.left.right = new Node(7);
root.right.right.left = new Node(9);
root.right.right.right = new Node(10);
```

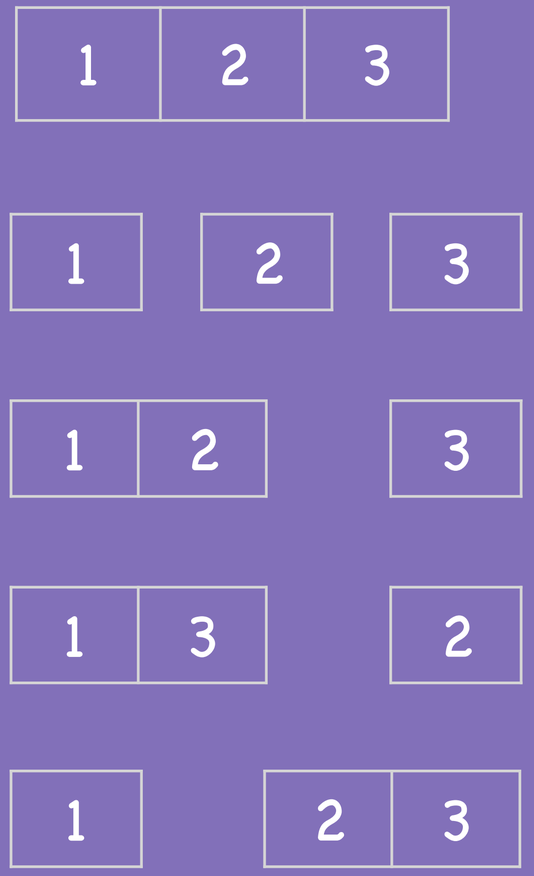
Print all leaf nodes from left right

```
let result = [];
function ltr(node) {
    // safety check
    if(node === null) return;

    // peak
    let peak = null;

    // peak condition
    if(node.left === peak && node.right === peak) {
        result.push(node.value);
    }

    // climb loop
    if(node.right !== null) {
        ltr(node.right);
    }

    if(node.left !== null) {
        ltr(node.left);
    }
}
ltr(root)
console.log(result);
```

Palindrome

```
let isPalindrome = true;
function pal(str) {
    // peak
    let peak = 0;

    // peak condition
    if(str.length === peak) return str;

    // climb loop
    let charAt1 = str.substr(0, 1);
    let charAtN = str.substr(str.length - 1);

    if(isPalindrome && charAt1 !== charAtN) {
        isPalindrome = false
    }

    let newStr = str.substr(1, str.length-2);


    // recursive call
    let result = pal(newStr);

    return str;
}

pal("malayalam1");
console.log(isPalindrome)
```

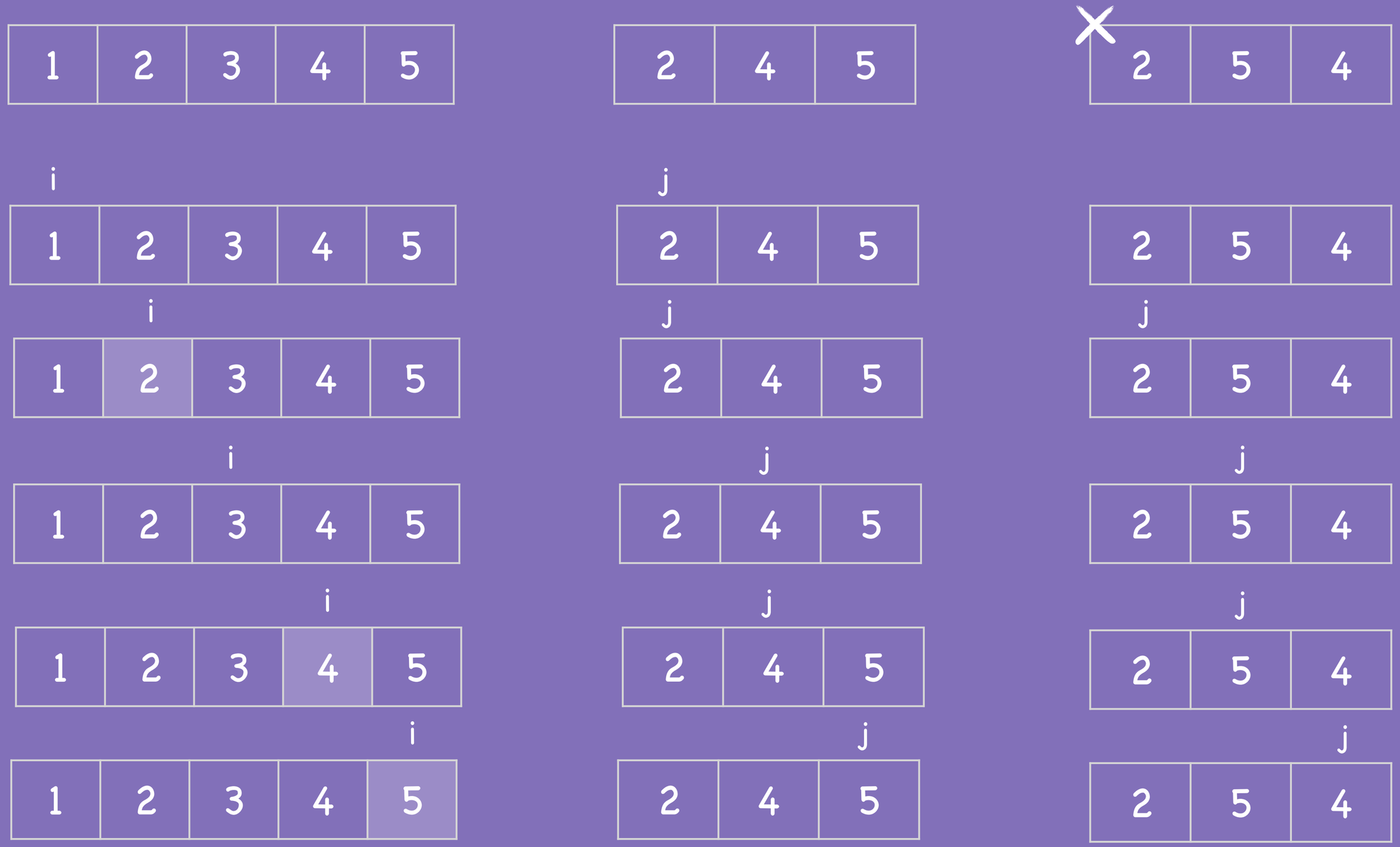
Friends Pairing Problem



If I separate 1 from n it would give a combination of 1 and (n-1) ways

If I separate 1 from the rest n-1 it would give a combination of 1*(n-1) and (n-2) ways. Considering borrowing 1 from n-1 folks anyone from n-1 would volunteer

Subsequence of an Array



Combination of all parenthesis

n = 1

n = 2

Count of the Opening brackets should
be less than the total count

n = 2

Count of the Opening brackets should
be greater than the count of the closing brackets

n = 2

Stop when both open and
close count are equal to total count

n = 2

Combination of all parenthesis

```
let result = [];
let len = 3;
function checkBalPar(open, close, s) {
    // peak
    let peak = len;

    // peak condition
    if(open === len && close === len) {
        result.push(s);
        return;
    }

    // climb loop
    if(open < len) {
        checkBalPar(open + 1, close, s + "{");
    }

    if(close < open) {
        checkBalPar(open, close + 1, s + "}");
    }
}

checkBalPar(0, 0, "");
```

Exponent of a number n

n = 2 e = 4

2 * (2 * (2 * (2 * 1)))

2 * (2 * (2 * (2 * 2))

2 * (2 * (2 * 4))

2 * (2 * (8))

2 * (16)

32

Keep reducing exponent and multiply the result of each iteration with the base

n = pow(n, e-1); pow being the rec func

Binary Search

Binary Search

```
let x = 10;
let arr = [2, 3, 4, 10, 40, 50, 60, 70, 80, 90];

function binarySearch(l, r) {
        // peak
        let mid = Math.floor((r+l)/2);

        // peak condition
        if(arr[mid] === x) {
            return mid
        }

        if(arr[mid] > x) {
            return binarySearch(l, mid-1);
        }

        return binarySearch(mid+1, r);
}

binarySearch(0, arr.length-1)
```