Get started Open in app

Danielle McCarthy

57 Followers · About Follow

Constructing an H-tree with JavaScript

Danielle McCarthy Mar 4, 2019 · 5 min read

I recently came across this Algorithm while participating in a mock interview on Pramp. At first glance, it's an intimidating problem, I'm here to break it down.

What is an H-tree? A geometric shape that consists of repeating letter Hs.

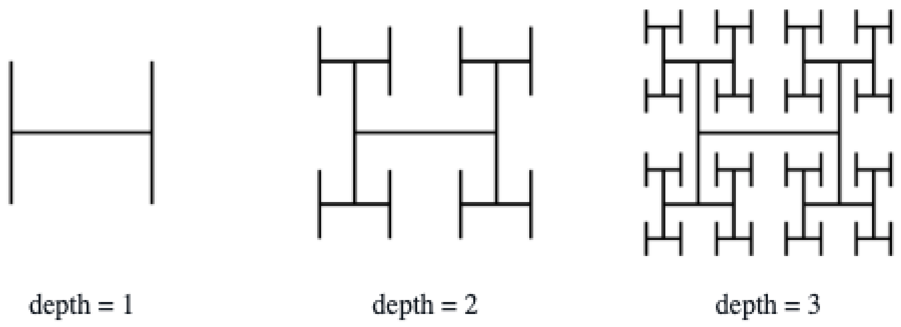


Image from Pramp.

According to Wikipedia, the H-tree is a fractal tree structure constructed from perpendicular line segments, each smaller by a factor of the square root of 2 from the next large adjacent segment.

The prompt: Write a function `drawHTree` that constructs an H-tree, given its center (x and y coordinates), a starting length, and depth. Assume that the starting line is parallel to the X-axis. Use the function `drawLine` provided to implement your algorithm. Since this is not production code, `drawLine` should just print its arguments.

SPOILER ALERT: THE ANSWER IS BELOW*

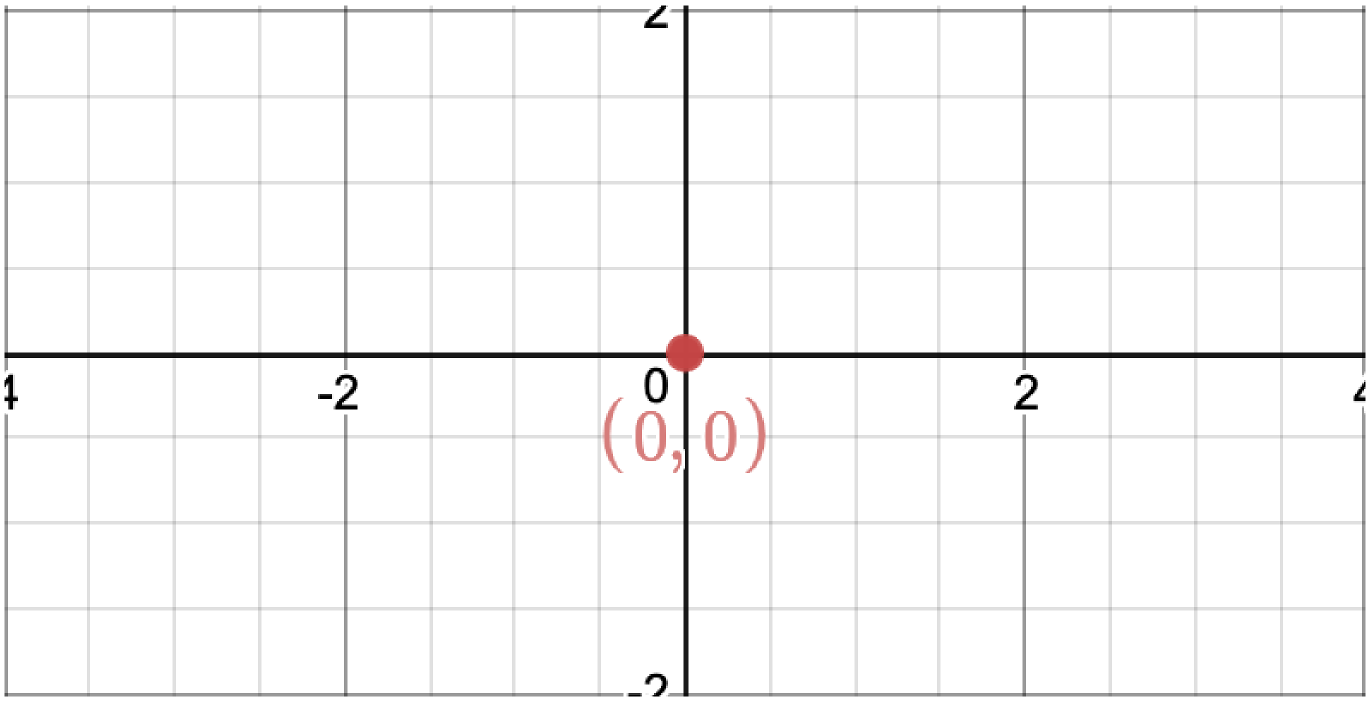
```
const drawLine = (coords) => {
  console.log(`starting point: x: ${coords[0]} y: ${coords[1]}`)
  console.log(`ending point: x: ${coords[2]} y: ${coords[3]}`)
}

const drawHTree = (x, y, length, depth) => {

  let leftMiddle = [x - length / 2, y]
  let rightMiddle = [x + length / 2, y]

  let topLeft = [x - length / 2, y + length / 2]
  let bottomLeft = [x - length / 2, y - length / 2]
  let topRight = [x + length / 2, y + length / 2]
  let bottomRight = [x + length / 2, y - length / 2]

  drawLine([...leftMiddle, ...rightMiddle]) //drawing horizontal line
  drawLine([...topLeft, ...bottomLeft]) // drawing left vertical line
  drawLine([...topRight, ...bottomRight]) // drawing right vertical line

  let newLength = length / Math.sqrt(2)

  if (depth - 1 === 0) {
    return;
  } else {
    let newDepth = depth - 1;

    drawHTree(topLeft[0], topLeft[1], newLength, newDepth)
    drawHTree(bottomLeft[0], bottomLeft[1], newLength, newDepth)
    drawHTree(topRight[0], topRight[1], newLength, newDepth)
    drawHTree(bottomRight[0], bottomRight[1], newLength, newDepth)
  }

}
```

gistfile1.txt hosted with ❤ by GitHub view raw

Say we invoke `drawHTree(0,0,2,2)` . What happens?

Breaking down the algorithm

We are given the H's center point, x and y, as shown in the graph below.



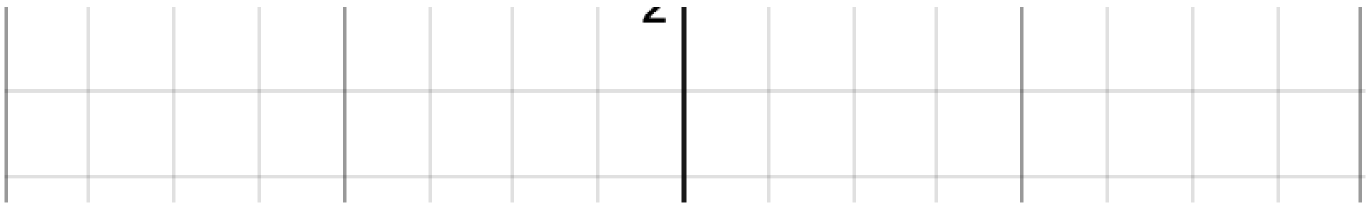
We invoked the function with 0,0 as the coordinates of the center point.

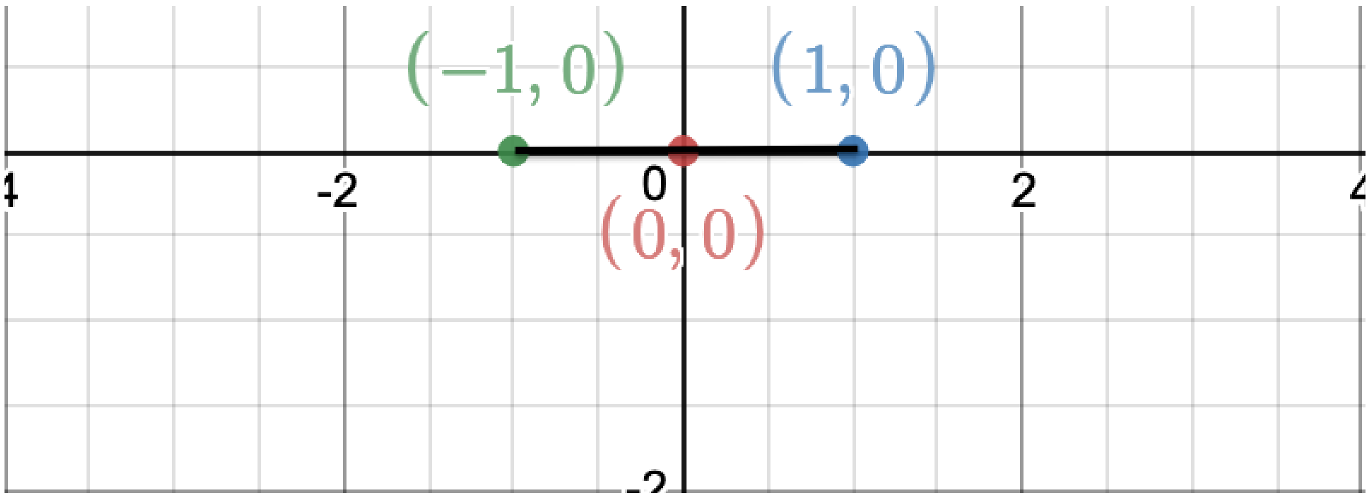
Next, we need to figure out where the horizontal line begins and ends. Since we're given the length of each line we can use this to figure out where to plot each point.

In my code I use `leftMiddle` and `rightMiddle` to denote these points

```
let leftMiddle = [x–length/2, y] // this equals [–1, 0]
let rightMiddle = [x+length/2, y] // this equals [1, 0]
```

We know that the horizontal line will be on the same y-axis as our center point and that each side of the horizontal line is half of the given `length` from the center point.





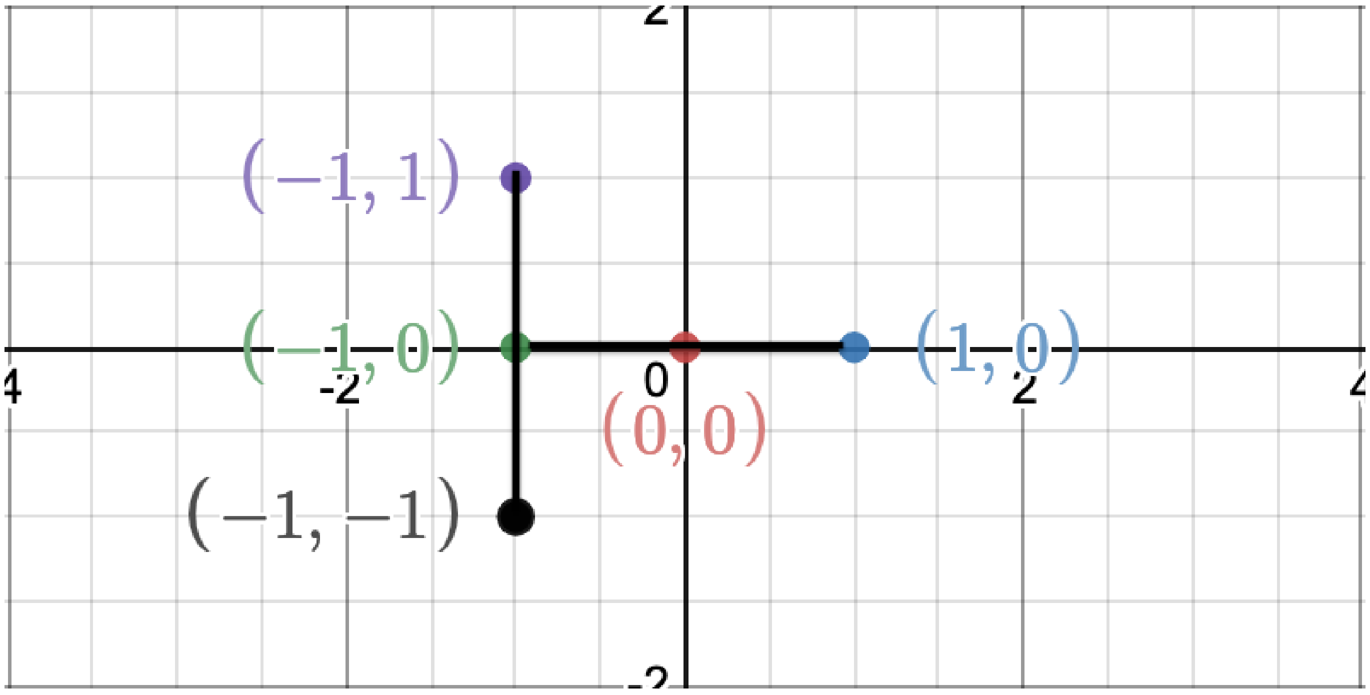
Now we have the horizontal line of our H!

Next, we need to figure out where the beginning and end of our left vertical line is. Using similar logic as above we know where the center point of the left vertical line is, and we can then use half of the given `length` to find out where each point is.

In my code, I use `topLeft` and `bottomleft` to denote these points.

```
let topLeft = [x-length/2, y + length /2] //this equals [-1, 1]
let bottomleft = [x-length/2, y-length/2] //this equals [-1, -1]
```

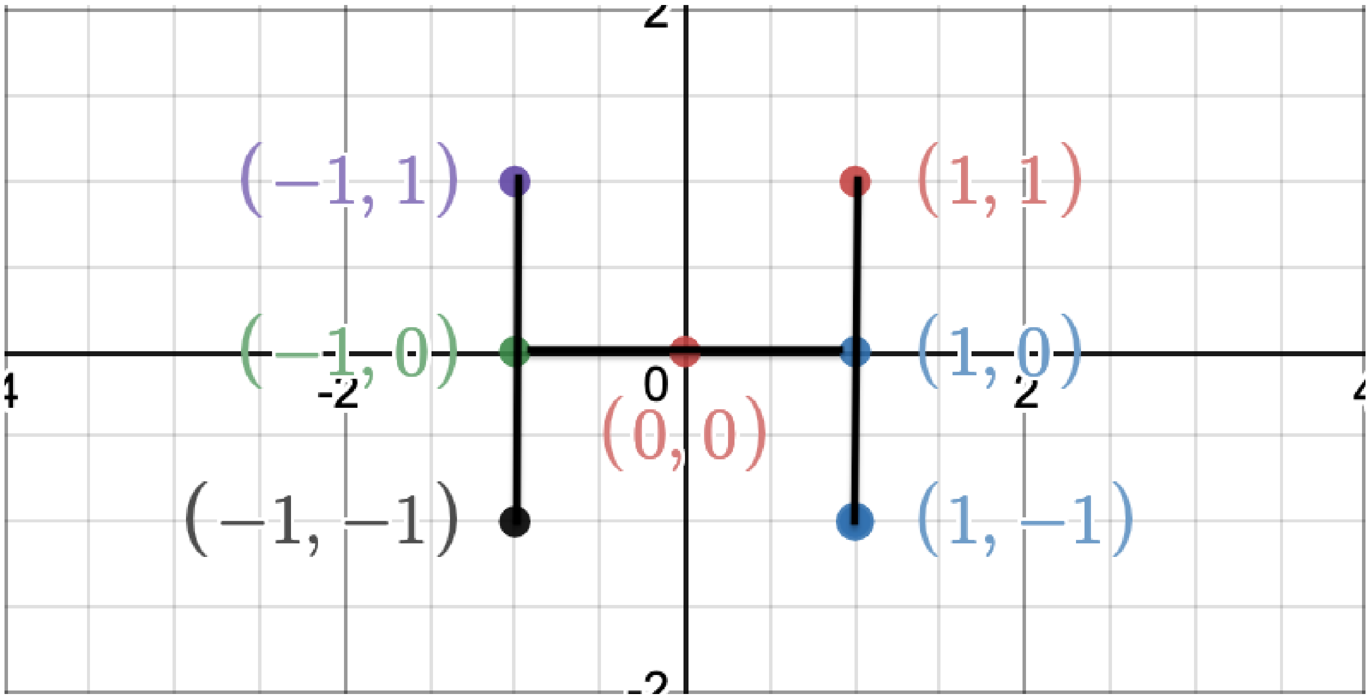
Ok, great! Now we have the left half of our H.



We're almost done with our first H.

We use the same logic to find the plot points of the right vertical line and call `drawLine` 3 times with the beginning and end points of each line, in this code `drawLine` just logs out the plot points as seen below.

```
drawLine([...leftMiddle, ...rightMiddle]) //drawing horizontal line
drawLine([...topLeft, ...bottomLeft]) // drawing left vertical line
drawLine([...topRight, ...bottomRight]) // drawing right vertical
line

// From my log:
starting point: x: −1 y: 0
ending point: x: 1 y: 0
starting point: x: −1 y: 1
ending point: x: −1 y: −1
starting point: x: 1 y: 1
ending point: x: 1 y: −1
```



Yay! We have our H!

Ok, great, we have our H, but our H-tree only has a depth of 1 and we input a depth of 2 when we invoked `drawHTree`. We want our H-tree to look similar to this…

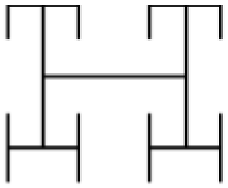


Image from Pramp.

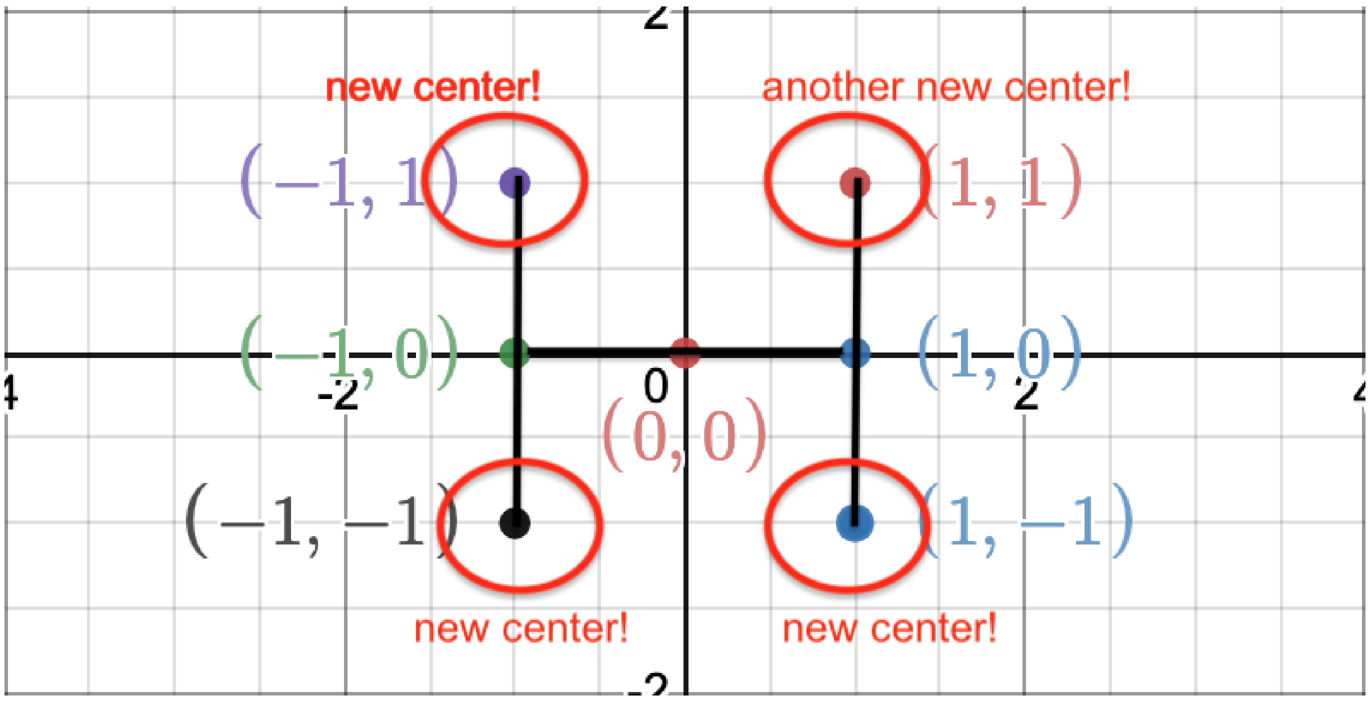
That's where recursion comes in. We check whether `depth` would equal 0 if we subtracted 1, if it does not equal 0 then we must make more H's!

Depth by depth the H gets smaller, remember the prompt told us that the length of the line segments drawn at each stage is decreased by `√2` .

So before our recursive calls, we set a `newDepth` and `newLength` .

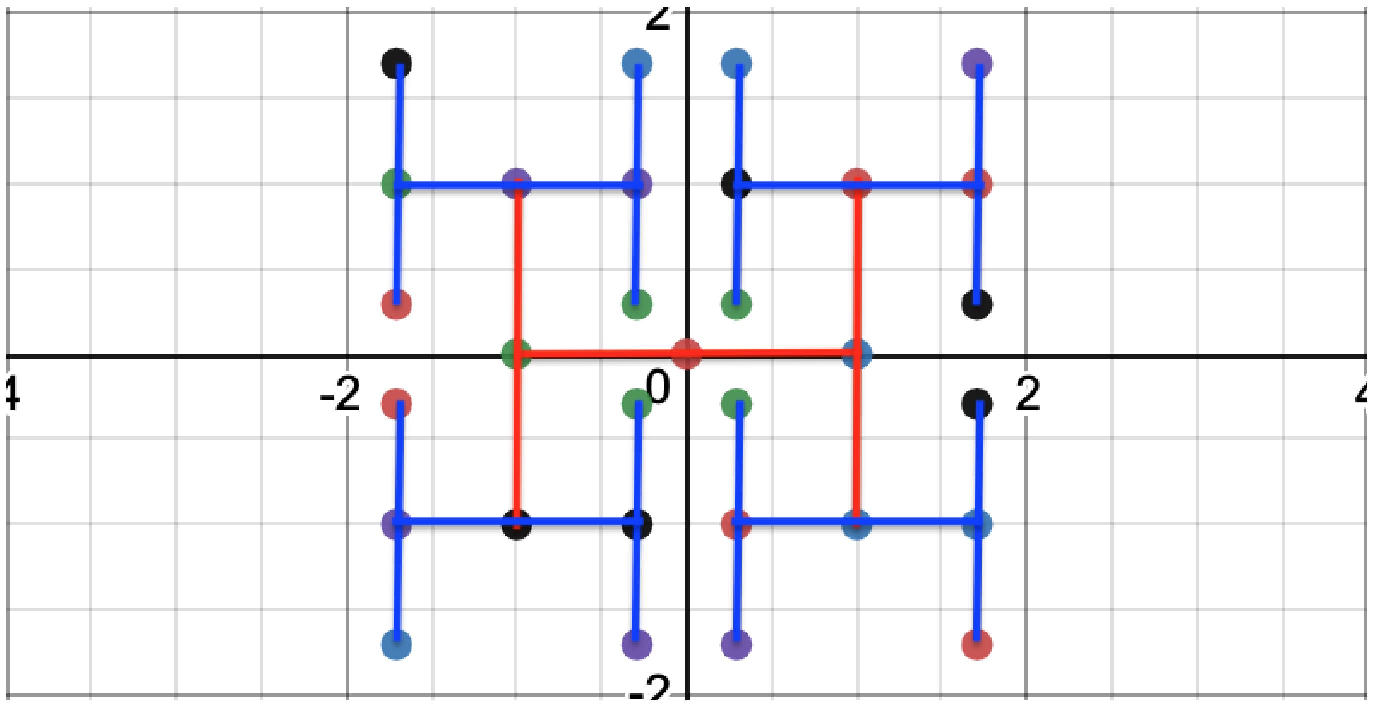
```
let newLength = length / Math.sqrt(2)
let newDepth = depth - 1
```

Each new H will begin at each start and end point of the vertical lines so we need to recursively call `drawHTree` 4 times with the `x` and `y` arguments, or the new center, as the start and end points of the vertical lines.



```
  drawHTree(topLeft[0],topLeft[1],newLength,newDepth)
  drawHTree(bottomLeft[0],bottomLeft[1], newLength, newDepth)
  drawHTree(topRight[0],topRight[1], newLength, newDepth)
  drawHTree(bottomRight[0],bottomRight[1], newLength, newDepth)
```

More and more Hs will be plotted until `depth` equals 0. Let's take a look at our final H-tree.



Our final H-tree!

If you're interested check out my repl to see the algorithm in action.

Time Complexity

The solution on Pramp tells us that every call of `drawhTree` invokes 9 expressions whose time complexity is `O(1)` and 4 calls of `drawhTree` until `depth` reaches `0`. Therefore: `T(D) = 9 + 4 * T(D-1)`, where `T` is the time complexity function and `D` is the depth of the H-tree. Now, if we expand `T(D-1)` recursively all the way to `T(0)`, we can see that `T(D) = O(4^D)`.

Space Complexity

Since this is a recursive solution, each call is stored in the stack taking up space. The space occupied in the stack is `O(D)`.

Thank you for reading, feel free to leave a comment, shoot me an email, or connect with me on LinkedIn.

JavaScript Algorithms