LAB 3 REPORT

The brief:

Javascript programs that implement the following algorithms from first principles (computing values using functions you write yourself, you may use libraries for ancillary functions like graphics, guis, data loading, etc)

>A recursive drawing technique

>A dynamic programming implementation of the same recursive technique

A writeup describing the algorithms and data structures used in each code example. Discuss why you used the techniques and algorithms you did. Is there a way to do it more efficiently? Were you able to derive a formula for the last program or did you do it iteratively? In the first program how did you handle non numbers in the array?

#1

My idea is to create a recursive grid that keeps draw a new grid within the cells of the grid. It draws not only with rectangles but also ellipses that bind to the grid themselves. I use a recursive algorithm that after each it start drawing a new grid, the starting points and the size of the new grid is depended on the cells of the grid it just draw.

In order to generate the cells of the grid, I use a for loop to the program to draw on each start point of each cells that is formulated by new (x,y) = (x,y) + (width,height) of the new cell of each new grid. The grid itself is split evenly either by 2, 3, 4, 5. And to add a further layer of randomness that avoid the bias of draw the grid mostly on the left cells (which I had noticed after the first time).

```
// Splitting the width and height of the grid into even cells
  // let split = random([1,2,3,4,5])
  // let w2 = (w / random([1,2,3,4,5]));
  // let h2 = (h / random([1,2,3,4,5]));
  // there i noticed that however some bias in this ^
  // because the grid mostly filled in the top left of the canvas

  // Adding another layer of randomness to avoid the problem above
  // The grid then can only choose to either split (split) or not (1)
  // Limitting the chances of the left side to always be split.
  let split = random([2,3,4,5]);
  let w2 = (w / random([1,split]));
  let h2 = (h / random([1,split]));
```

Next I add a base condition for the recursion and decide on how deep the recursion will continue making the new grid. As each recursion is invoke, the depth number is added into until it meets the maxSteps for recursion stack.

```
for(let x1=0; x1 < w; x1 += w2){
    for(let y1=0; y1 < h; y1 += h2){
      if(depth < maxSteps){
        drawGrid(x+x1, y+y1, w2,h2, ++depth, int(random(150,250)));
      }
      else {
        drawCell(x+x1, y+y1, w2, h2, c);
      }
    }
  }
```

#2
To reiterate the code as a dynamic programming implementation, I use stacks with arrays and class objects.

At first, I naively the for-loop in the recursive version to update the parameter of each cells and push it into the array. This method didnt work the same because the supposed recursed grid's position and size were still bound to the initial width and height of the canvas while it should be updated depended on the width and height of the new cells in the new grid.

I was not sure how to fix it so I just start anew. I did not really understand what dynamic programming was specifically so that's why it was kind of hard for me to redo the recursive one into a dynamic prog implementation. I do find this definition the (1) is more comprehensible. "Dynamic programming amounts to breaking down an optimization problem into simpler sub-problems, and storing the solution to each sub-problem so that each sub-problem is only solved once.

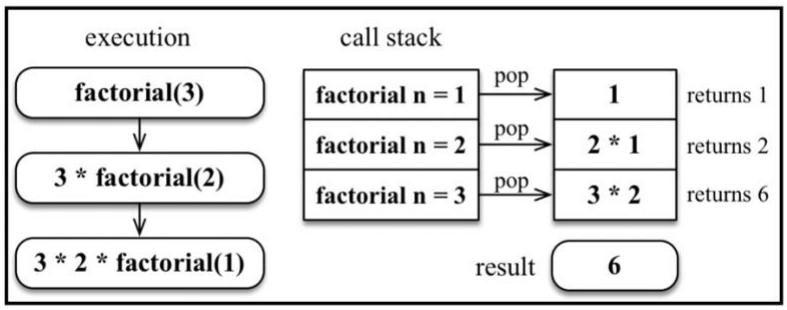
The sub-problems that I identified for any one cell of the grid: Split the cell [a] + Update the size, position of each cells [b] + Push the new parameters as new cells into the array [c] → Draw the object (the grid cells). In order to solve the problem of creating a new parameter for the cells within the new cells, I use stacks structure temporarily storing the cells parameter. I based the new parameter on that of the top cell of the stack to draw it and also to split it into smaller cells and push it onto the top of the stack. After it finished with all the subproblem for the top cell, the stack will pop the top cell, continue with implementing the sub-problem for new top cell.

```
console.log(Shapes.length);
    let x = Shapes[Shapes.length -1].x; // Store the last parameter of the top cell
    let y = Shapes[Shapes.length -1].y;
```

```
    let w = Shapes[Shapes.length -1].w;
    let h = Shapes[Shapes.length -1].h;
    let step = Shapes[Shapes.length -1].step;

    let split = random([2,3,4,5]); // Split the top cells
    let splitX= random([1,split]);
    let splitY= random([1,split]);

    let w2 = (w / splitX); // Splitting
    let h2 = (h / splitY);

    Shapes[Shapes.length -1].drawCell(); // Draw the top cell
    Shapes.pop(); // Pop it out
```

The program then iteratively conitnue pushing in new cells, draw the cells and popping them out until the stacks of cells are emptied. In order to control how many times the new cells are formed, I used a base condition like that of the recursive program. Here I called it as “step”.

```
if(step < n){
        let c = random(150,250);
        for(let i= splitX-1; i >= 0; i--){
            for(let j = splitY-1; j>= 0; j--){
                let newX = x + i*w2;
                let newY = y + j*h2;
                let obj_shape = new shapeObject(newX, newY, w2,h2, c,step+1);
                Shapes.push(obj_shape);


            }
        }

    }
```



"When using recursion, each function call will be stacked on top of each other, due to the possibility of one call depending on the result of the previous invocation itself" (Textbook pg.221)

'

Reference

> Recursion
Textbook: Chapter 9: Recursion

> Dynamic Programming
Textbook: Dynamic Programming pg. 353 – 367
https://www.freecodecamp.org/news/demystifying-dynamic-programming-3efafb8d4296/

> Grids
https://generativeartistry.com/tutorials/piet-mondrian/
https://gorillasun.de/blog/an-algorithm-for-irregular-grids
https://generativeartistry.com/tutorials/hypnotic-squares/

>To read further:
- Top-down and bottom-up dynamic programming
- https://www.geeksforgeeks.org/stack-data-structure/