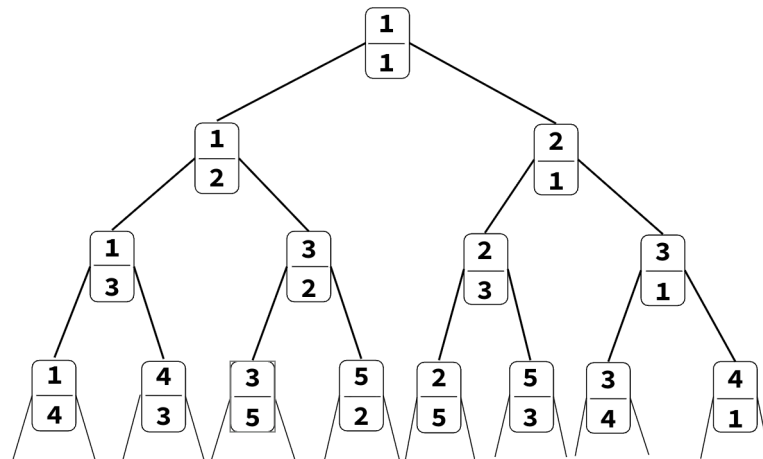


🌴 IG Problem 1A: Tree Climbing 🌴

Written by Ben Yan

This exercise is all about **Calkin-Wilf trees**, which rely on the following construction below, where each vertex is a positive fraction. It's similar in heart and spirit to the Stern-Brocot trees you saw—and may have loved or hated—on Assignment 1, but different enough in ways that I hope you find interesting! ☺



We start with a root fraction of $1 / 1$, and from that point forward, each vertex n / d has 2 children:

- The fraction $n / (n + d)$ on the left.
- The fraction $(n + d) / d$ on the right.

Notice the left child is always smaller than 1 in value, and the right child is always greater than 1. That way, you can support fractions where the numerator exceeds the denominator, sometimes rather insultingly called “*improper fractions*” ☺. **It turns out that every positive fraction has a vertex they call home in this tree**, whereas residency in our Stern-Brocot tree was limited to fractions between 0 and 1.

Your task? Implement the function **cws** (short for **Calkin-Wilf Sequence**), which accepts the numerator and denominator of a positive fraction as input, and **returns a string that corresponds to a sequence of left ("L") and right ("R") moves from the root of 1/1 to reach that fraction in the tree**. For instance,

- **cws(4, 3)** returns "LLR"
- **cws(1, 1)** returns ""
- **cws(3, 1)** returns "RR"
- **cws(1, 4)** returns "LLL"

But instead of descending the tree top-to-bottom as you did in Stern-Brocot, **you must traverse this tree bottom-up**. That is, start from the target fraction, and find its parent fraction, and that fraction's parent, and so forth, until you reach the root, which traces the left and right moves followed in reverse order.

From a vertex n / d , you can ascend to its parent as follows (by rearranging the equations above):

- If $n < d$, it's a left child, and its parent vertex is $n / (d - n)$.
- If $n > d$, it's a right child, and its parent vertex is $(n - d) / d$.

Present your own implementation on this page – you got this! **Note:** For a vertex / fraction in the tree, you are highly encouraged to store the numerator and denominator as separate integer variables.

```
/* Returns a string of Ls and Rs corresponding to left and right moves
 * from the root 1 / 1 to the fraction num / denom on the Calkin-Wilf tree
 */
function cws(num, denom) {
```

```
}
```

IG Problem 1B: Pastiche to Matisse

Written by Ben Yan

If you're more interested in fine arts, visual flair, and drawing with your **JSGraphics** paintbrushes and easels than traversing trees of fractions, by all means, try this problem with your section leader instead!

One of the most striking exhibits at the Tate Modern in London is *L'escargot* (*The Snail*) by French artist Henry Matisse. The 1953 painting, created over the span of a year from spring to late winter, is a collage of pigmented paper sliced up with shears and scattered like fairy dust over a white canvas, and is alternatively titled *La Composition Chromatique*. I'm not French, but this probably translates to *The Chromatic Composition* ☺—a fittingly evocative name for a work celebrated for its color, vibrancy, and abstraction of life.

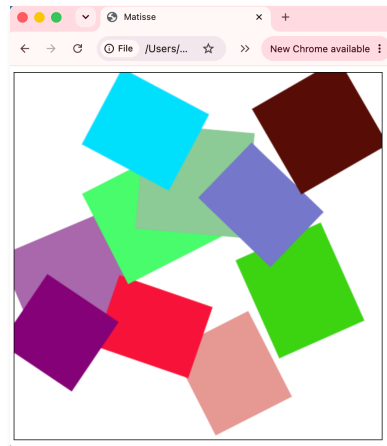


As a homage to Matisse's masterpiece, **implement a program that adds 10 rectangles of varying height, width, color, placement, and later orientation to a 500 x 500 graphics window**. Each rectangle's height and width should be randomly and independently chosen between **100** and **150** pixels, inclusive, using the function **randomInteger(min, max)**. Once the height and width are chosen, you should give the rectangle a random position, ensuring that its shape stays entirely within screen without extending past the edge.

Also, use the **randomColor** method to pigment each rectangle, and **setColor**, **setFilled** to ensure the rectangle—and its boundary—are filled with that color. Lastly, assume the **GRect** class supports a method **rotate(angle)**, which accepts an **integer angle** in degrees, and rotates the rectangle object by that angle counterclockwise about its center. You'll want to rotate each rectangle by a randomly selected angle between **0** and **359**, inclusive on both ends, leveraging **randomInteger** to select one.

While you should ensure that each rectangle, prior to being rotated, fits entirely within the window, it's perfectly fine if part of the rectangle leaks off screen after rotation. A few examples are displayed below.

Use the canvas on the next page to write out your implementation and impress your imagination.



My "attempt" at recreating Matisse's painting, at an arts event organized by Eugene Francisco.

```
/* This program adds rectangular strips / boxes of varying size, color,  
 * and orientation to a canvas, as a homage to Matisse's Snail painting.  
 */
```



```
const GWINDOW_HEIGHT = 500; /* in pixels */  
const GWINDOW_WIDTH = 500;  
  
const NUM_BOXES = 10;  
const BOX_MIN_WIDTH = 100; /* in pixels */  
const BOX_MAX_WIDTH = 150;  
const BOX_MIN_HEIGHT = 100;  
const BOX_MAX_HEIGHT = 150;  
  
function Matisse() {  
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);  
    // fill this in along with any helper functions your heart desires  
  
}
```