Student-Info

Student ID: 100586212

Name: Kaushal Patel

Pseudo-Code

```
**Longest-Runway**(N)
for i = 0 to N.length - 1
    (x1, y1) ← points(i)
    for j = i+1 to N.length
        (x2, y2) ← points(j)
        //Initialize tracking variables, ex. maxLength, change, initial etc.
        //Calculate parametric equation for line formed from points(i), point(j)
        tLine ← {$(x_a + x_b t), (y_a + y_b t)$}
        for k = 0 to N.length
            //Using Parametric equations find s and t where the two lines intersect
            (s,t) ← tLine = lines[k]
            //Break conditions
            //if intersection outside of line[k] endpoints, ∴ outside polygon
            if $t < 0$ or $t > 1$ → continue
            //if intersection in between tLine endpoints, ∴ line too short
            if $0 < s < 1$ → break
            //Set s bounds
            if s > maxS
                maxS ← s
            if s < minS
                minS ← s
        //Check if line length is greater than current maxLine, store
```

```
if (minS, maxS).length > maxLine.length
maxLine ← (minS, maxS)
```

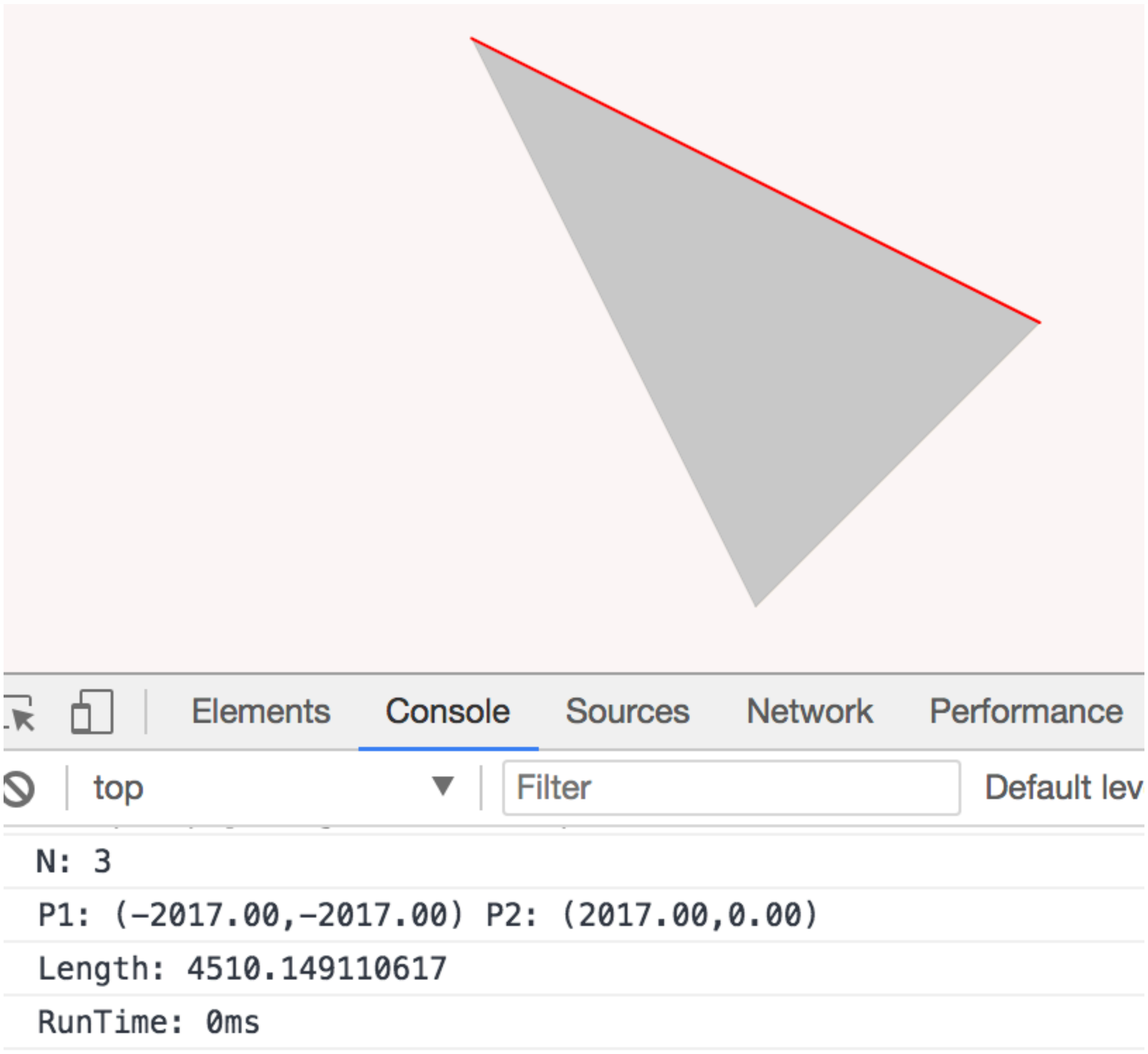
Time Complexity

- 3 **for** loops
- **for** i = 0 **to** N.length - 1 : n
- **for** j = i+1 **to** N.length : $\sum_{i+1}^{n} t_i - 1 \rightarrow \sum_{k=1}^{n-1} k \rightarrow 1 + 2 + 3 \ldots + n - 2 + n - 1 \rightarrow \frac{n(n+1)}{2} - n$
- **for** k = 0 **to** N.length : $\sum_{i+1}^{n} t_i - 1 * n \rightarrow n + 2n + 3n \ldots + n^2 - 2 + n^2 - 1$

$\therefore$

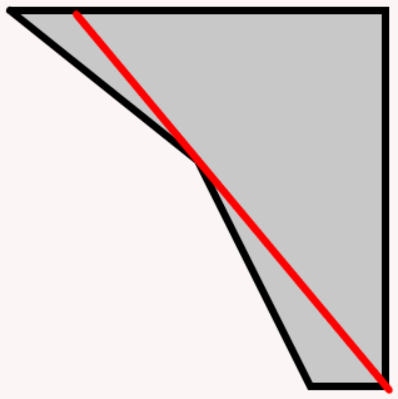
O(n) = n^2

Output

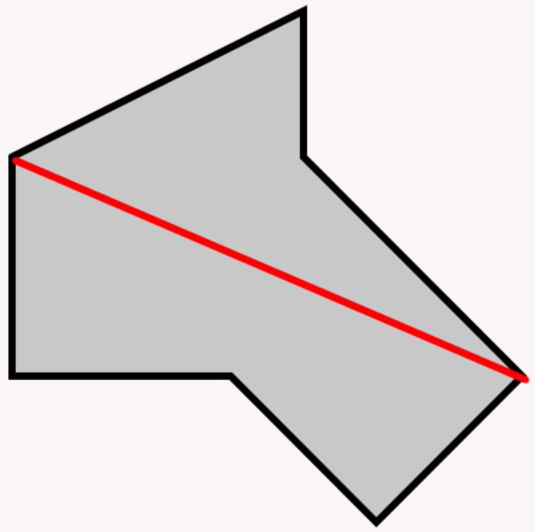


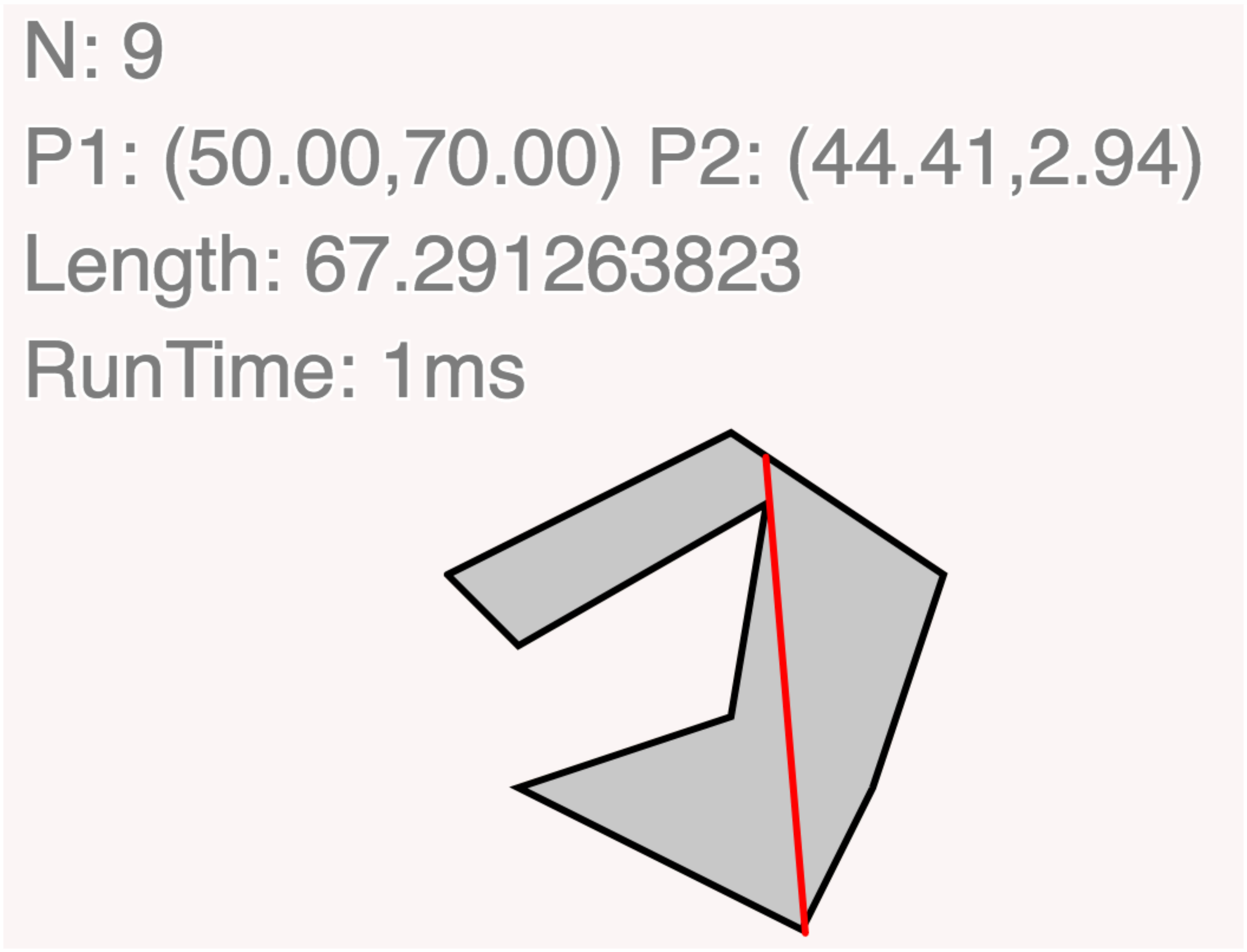
```
N: 3
P1: (-2017.00,-2017.00) P2: (2017.00,0.00)
Length: 4510.149110617
RunTime: 0ms
```

```
N: 5
P1: (50.00,50.00) P2: (8.33,0.00)
Length: 65.085413966
RunTime: 1ms
```

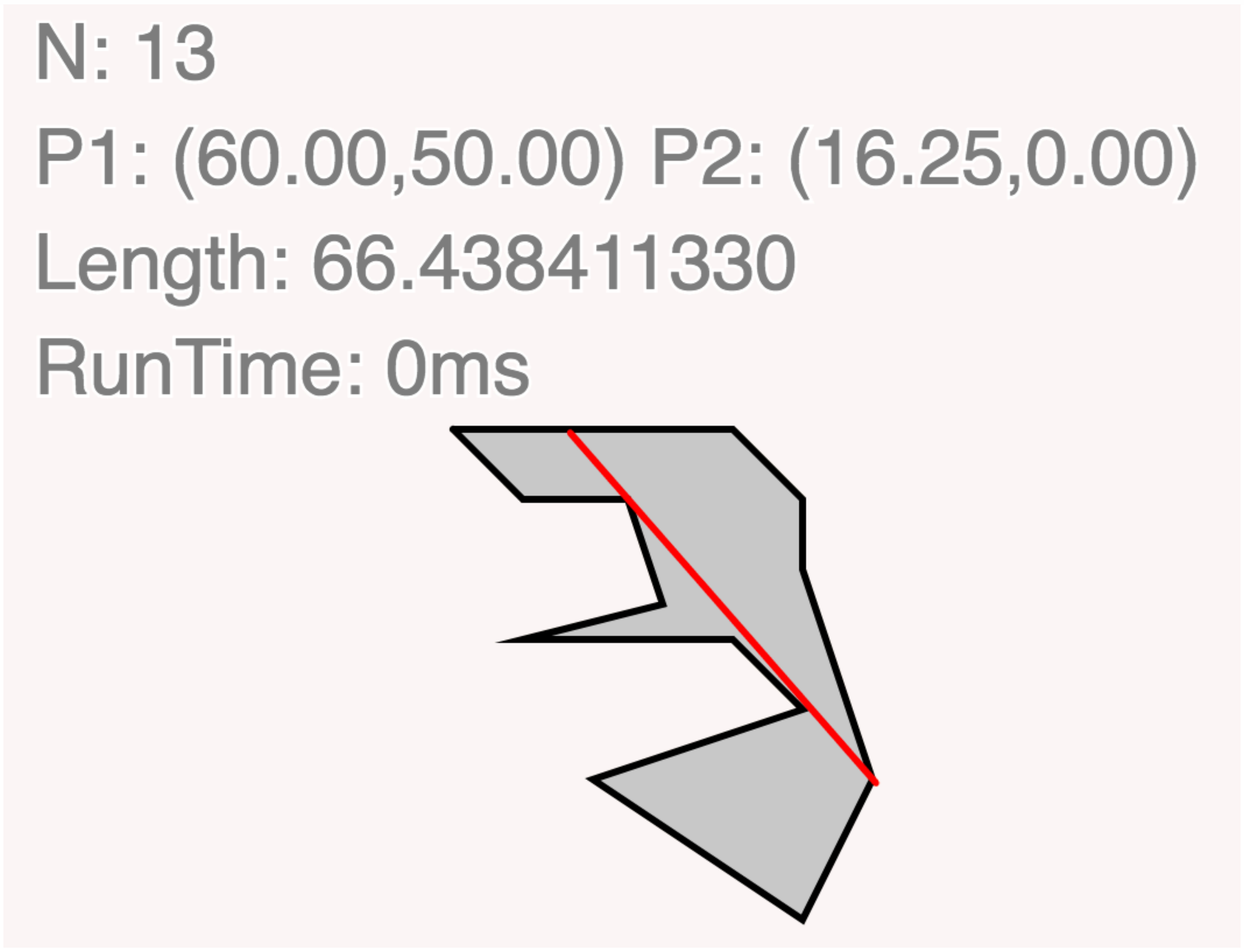


```
N: 7
P1: (0.00,20.00) P2: (70.00,50.00)
Length: 76.157731059
RunTime: 1ms
```





```
N: 9
P1: (50.00,70.00) P2: (44.41,2.94)
Length: 67.291263823
RunTime: 1ms
```



```
N: 13
P1: (60.00,50.00) P2: (16.25,0.00)
Length: 66.438411330
RunTime: 0ms
```

Code

`/*var points = [

```
[0, 2017],
[-2017, -2017],
[2017, 0]
```

];/* var points = [

```
[0, 0],
[50, 0],
[50, 50],
[40, 50],
[25, 20],
```

];*/ /* var points = [

```
[0, 20],
[40, 0],
[40, 20],
[70, 50],
[50, 70],
[30, 50],
[0, 50]
```

]*/ /* var points = [

```
[0, 20],
[40, 0],
[70, 20],
[60, 50],
[50, 70],
[10, 50],
[40, 40],
[45, 10],
[10, 30]
```

]*/ var points = [

```
[0, 0],
[40, 0],
[50, 10],
[50, 20],
[60, 50],
[50, 70],
[20, 50],
[50, 40],
[40, 30],
[10, 30],
[30, 25],
[25, 10],
[10, 10]
```

]

var startTime = 0;

var n = points.length; var m;

var lines = []; var maxLine = {x1 : 0, y1 : 0, x2 : 0, y2 : 0, length : 0};

function setup(){

```
createCanvas(640, 480);
noLoop();
```

}

function draw(){

```
fill('#ccc'); //Add background color to polygon
beginShape(); //Begin polygon shape

for(i=0; i<n; i++){ //Add vertices to shape from points array

    vertex(points[i][0], points[i][1]);

    if(i >= 1){ //Start adding lines once more than 1 vertex is read
        x1 = points[i-1][0];
        y1 = points[i-1][1];
        x2 = points[i][0];
        y2 = points[i][1];

        //Lines are added as parametric equations allowing for vector
manipulation
        //Store as: v = a + tb for (x,y) = (xa, ya) + t(xb, yb)
        lines.push({x : {a : x1, b : x2-x1}, y : {a : y1, b : y2-y1}});
    }
}

//Initiate lines counter, used later in CheckIntersection
m = lines.length;

//Extra case for last element in vectors array, need to accomadate for the last
and first vectors
x1 = points[n-1][0];
y1 = points[n-1][1];
x2 = points[0][0];
y2 = points[0][1];

lines.push({x : {a : x1, b : x2-x1}, y : {a : y1, b : y2-y1}});

translate(width/2 - 50, height/2 - 50);
scale(3);
endShape(CLOSE); //End polygon and close last line
console.log(lines); //Checking lines array

startTime = Date.now();
```

```
//Check all lines for max length using the points array
//Initiate a new line and cross reference with the lines array for
intersections
for(i=0; i<n-1; i++){
    //Initialize the first vertex coordinates
    x1 = points[i][0];
    y1 = points[i][1];

    for(j=i+1; j<n; j++){
        //Initialize the second vertex coordinates
        x2 = points[j][0];
        y2 = points[j][1];

        //Set maxS, minS, and tempLine
        var maxS = 1;
        var minS = 0;
        //var tempLine = {x1 : 0, y1 : 0, x2 : 0, y2 : 0, length : 0};

        //Set escaped
        var escaped = false;
        var change = false;
        var initial = true;

        //Create temporary line object for current set of vertices
        tLine = {x : {a : x1, b : x2-x1}, y : {a : y1, b : y2-y1}};

        //CheckIntersection
        //Find all intersections with current vector created with all vectors
of polygon
        for(k=0; k<m; k++){

            //Store (a,b) values for both lines to solve for t
            //    (x1,y1) = tLine, (x2, y2) = lines[k]

            //Only need 3 variables after equating the two parametric equations
            //ex. (x1,y1) = (x1a,y1a) + s(x1b,y1b), and (x2,y2) = (x2a,y2a) +
t(x2b,y2b)
            //    (x1 = x2) =>
            //    (x1a + s*x1b = x2a + t*x2b) =>
            //    (s*x1b - t*x2b = x2a - x1a) =>
            //    (px - qx = rx)

            var px = tLine['x']['b'];
            var qx = lines[k]['x']['b'];
            var rx = lines[k]['x']['a'] - tLine['x']['a'];
            //Same with y
            //    (py - qy = ry)

            var py = tLine['y']['b'];
            var qy = lines[k]['y']['b'];
            var ry = lines[k]['y']['a'] - tLine['y']['a'];

            //Find Intersection values
            //Solve for s and t, they give the coordinates of intersection
respective to their vectors
            //s => (x1,y1) tLine
            //t => (x2,y2) lines[k]

            //Using matrices to find formulas to solve for s and t
            //s = (px*ry - rx*py / px*qy - qx*py)
            //t = (qx*ry - rx*qy / px*qy - qx*py)
            //Store (px*qy - qx*py), common in both and need to ensure it is
not 0

            var d = (qx*py) - (px*qy);

            //If lines are parallel, continue to next iteration of loop
            if(d == 0){continue;}

            var s = (qx*ry - rx*qy) / d;
            var t = (px*ry - rx*py) / d;

            //Check is t is in bounds of lines[k], 0 < t < 1
            //If not, intersection is outside of polygon, and should be skipped
            if((t < 0)||(t > 1)){
                continue;
            }

            //Check if s is in bounds of tLine, 0 < s < 1
            //If so, intersection is between tLine vertices (too short), should
line[k]
            if((s > 0)&&(s < 1)){
                escaped = true;
                break;
            }

            //If loop has not been terminated yet, check if s is greater on
either ends
            if(initial){
                if(s <= 0){
                    minS = s;
                    change = true;
                }

                if(s >= 1){
                    maxS = s;
                    change = true;
                }

                initial = false;
            }else{

                if((s > minS)&&(s < 0)){
                    minS = s;
                }

                if((s < maxS)&&(s > 1)){
                    maxS = s;
                }
            }
        }

        //Check if new points create longer line than maxLine if loop was not
broken
        if((!escaped) && (change==true)){
            //Calculate coordinates and length using tLine parametric equation
            tempX1 = tLine['x']['a'] + minS*tLine['x']['b'];
            tempY1 = tLine['y']['a'] + minS*tLine['y']['b'];
            tempX2 = tLine['x']['a'] + maxS*tLine['x']['b'];
            tempY2 = tLine['y']['a'] + maxS*tLine['y']['b'];

            length = Math.sqrt(Math.pow(tempX2 - tempX1, 2) + Math.pow(tempY2 -
tempY1, 2));

            //Store if longer than maxLine
            if(length > maxLine['length']){
                maxLine['x1'] = tempX1.toFixed(2);
                maxLine['y1'] = tempY1.toFixed(2);
                maxLine['x2'] = tempX2.toFixed(2);
                maxLine['y2'] = tempY2.toFixed(2);
                maxLine['length'] = length.toFixed(9);
            }
        }
    }
}
```

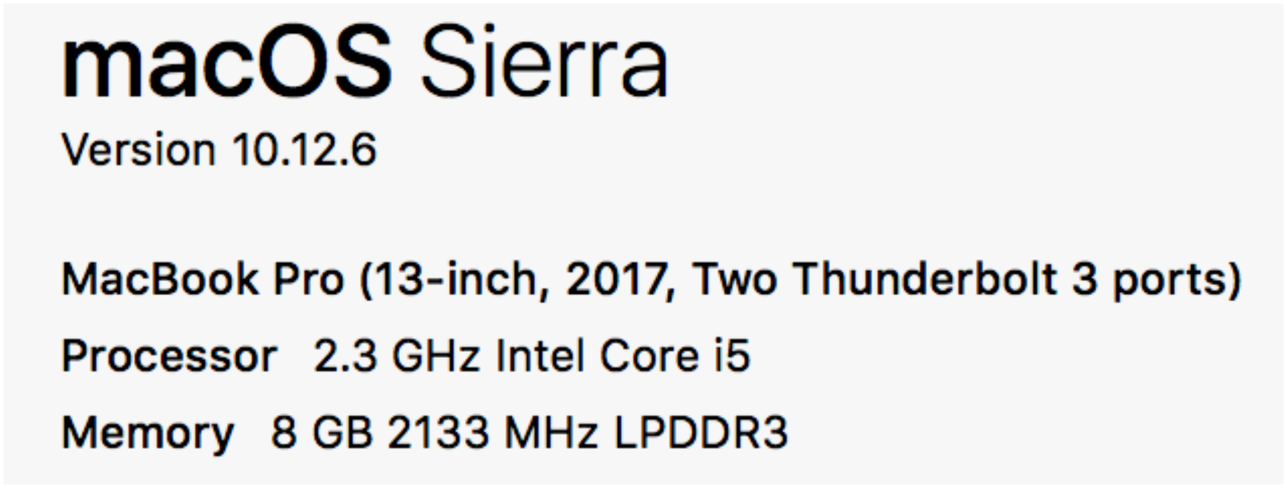
```
var RunTime = Date.now() - startTime;

stroke('red');
line(maxLine['x1'], maxLine['y1'], maxLine['x2'], maxLine['y2']);
stroke('333');
fill('grey');
textSize(11);
text('N: '+n, -60, -50);
text('P1: ('+maxLine['x1']+','+maxLine['y1']+') P2:
('+maxLine['x2']+','+maxLine['y2']+')', -60, -35);
text('Length: '+maxLine['length'], -60, -20);
text('RunTime: '+RunTime+'ms', -60, -5);

console.log('N: '+n);
console.log('P1: ('+maxLine['x1']+','+maxLine['y1']+') P2:
('+maxLine['x2']+','+maxLine['y2']+')');
console.log('Length: '+maxLine['length']);
console.log('RunTime: '+RunTime+'ms');
```

}`

Machine Specs


macOS Sierra
Version 10.12.6

MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports)
Processor 2.3 GHz Intel Core i5
Memory 8 GB 2133 MHz LPDDR3

Libraries

Language: JavaScript

Libraries: p5.js