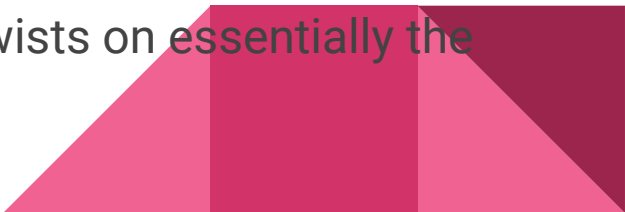


Fractal Trees

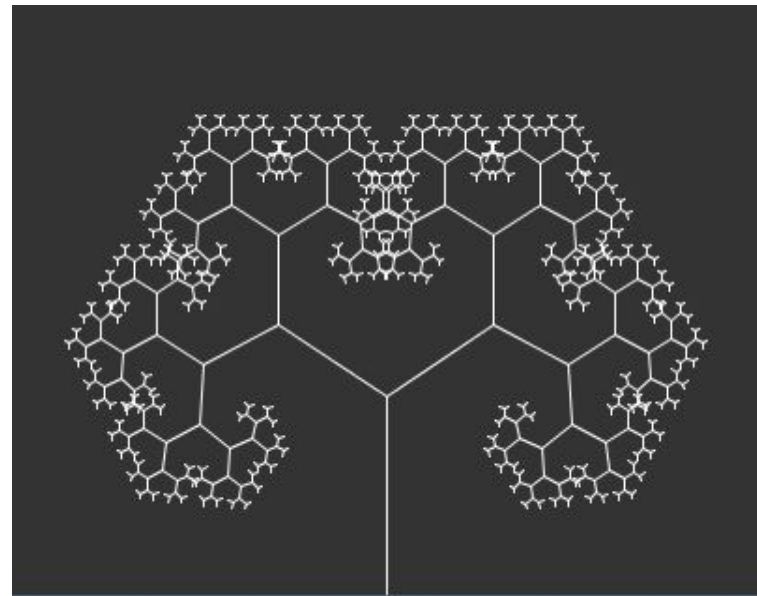
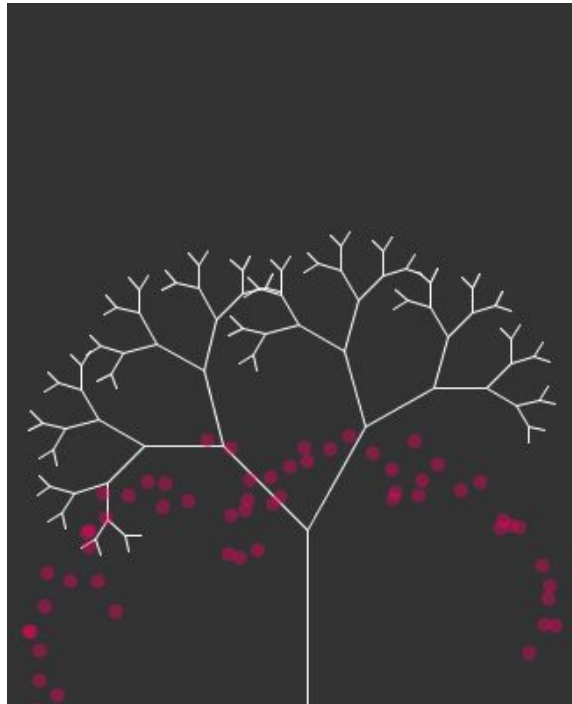
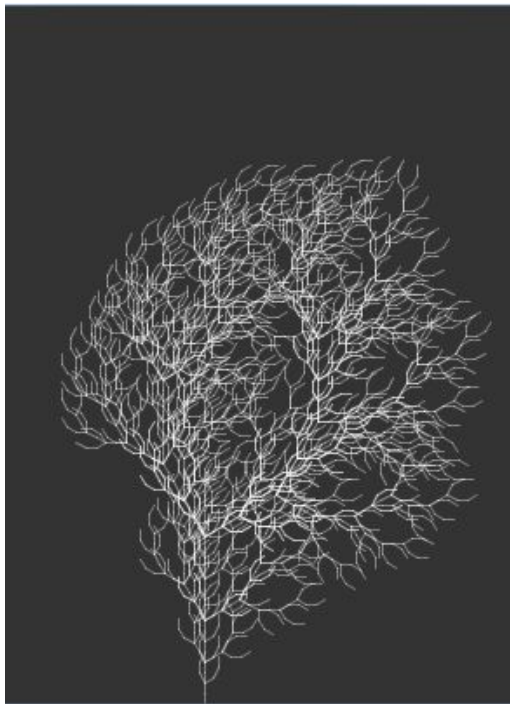
(Object Oriented)

By: Andy, Nick, Hamza

Introduction

- What are Fractal Trees?
 - Fractal: “A curve or geometric figure, each part of which has the same statistical character as the whole.”
 - Can be represented through the use of fractal trees
 - It may seem simple, but there are many different things that you can change with different additions and variations of the concept.
 - Make it algorithmic
 - Make it special
 - Make it recursive
 - The concepts are all similar but all create different twists on essentially the same idea.
- 

Examples

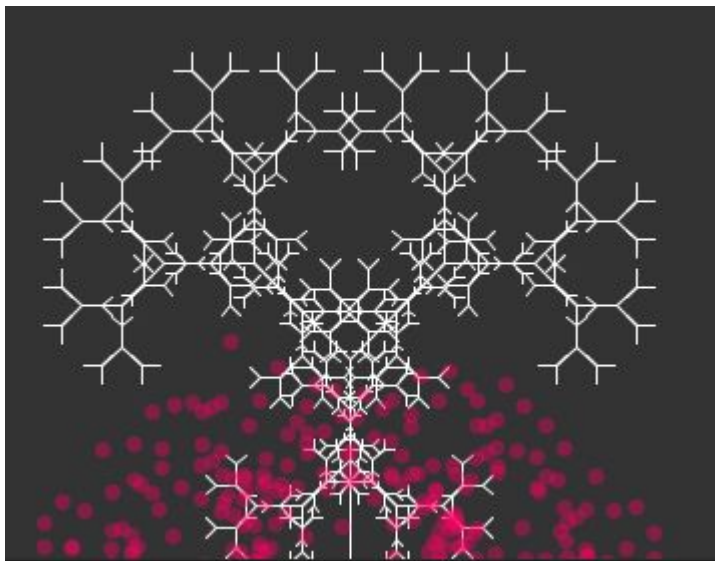


Creative Aspects

- A lot of different takes that you can take on fractal trees
- There are elements of the programs that are:
 - easily replaceable, changeable within the context of the individual types and variations of the program.
 - The programs in itself give the user a lot of leeway to make changes without messing up other parts of the code since
 - the connection between the different parts of the program are easy to distinguish.
- The concept of the 'fractal' trees remains the same,
 - the way they are effected and introduced into the system are changed.
- This allows us to gain a better understanding of the program,
 - exactly what the numbers in the systems are defining in terms of the characteristics of the trees



Spinoff: Andy



```
this.branchC = function() {  
  var dir = p5.Vector.sub(this.end, this.begin);  
  dir.rotate(PI / 4);  
  dir.mult(0.50);  
  var newEnd = p5.Vector.add(this.end, dir);  
  var b = new Branch(this.end, newEnd);  
  return b;  
}
```

```
void mousePressed(){  
  for(int i = tree.size() -1; i >= 0; i--){  
    Branch current = tree.get(i);  
    //if the current Branch has no children: add them  
    if(!current.finished){  
      tree.add(current.branchA());  
      tree.add(current.branchB());  
      tree.add(current.branchC());  
    }  
    //now that Branch has children  
    current.finished = true;  
  }  
  //new Level added  
  count ++;
```

Spinoff: Andy

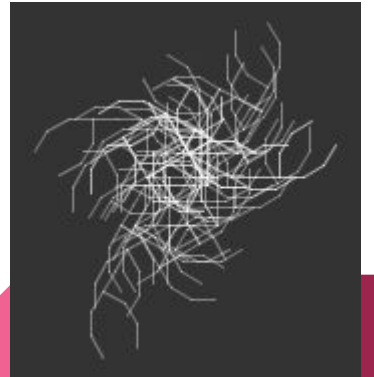
Changed Elements:

- Algorithmically the object oriented code is the most straight forward, since it deals with the concept of arrays and many different elements instead of one integrated code.

```
Branch branchC(){  
    PVector dir = PVector.sub(end, begin);  
    dir.rotate( PI / 4);  
    dir.mult(0.50);  
    PVector newEnd = PVector.add(end, dir);  
    Branch b = new Branch(end, newEnd);  
    return b;  
}
```

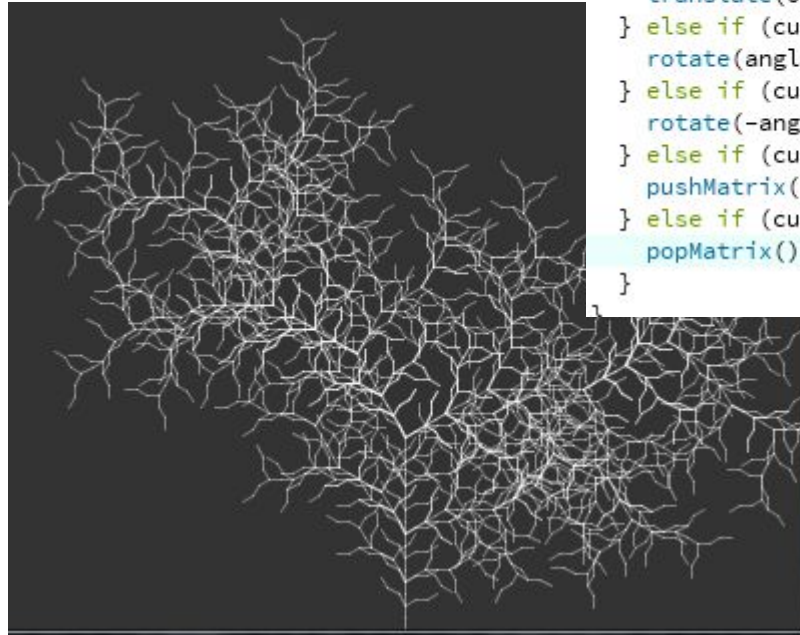
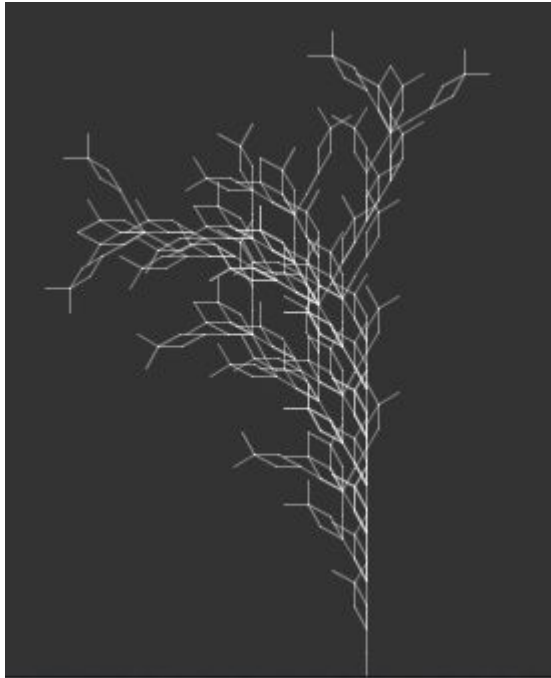
Spinoff: Hamza

```
rules = new Rule[1];  
rules[0] = new Rule('F', "FF-[+F-F-F]-[-F+F+F]");
```



Spinoff: Hamza

```
rules = new Rule[1];  
rules[0] = new Rule('F', "FF-[+F-F-F]+[-F+F+F]"
```



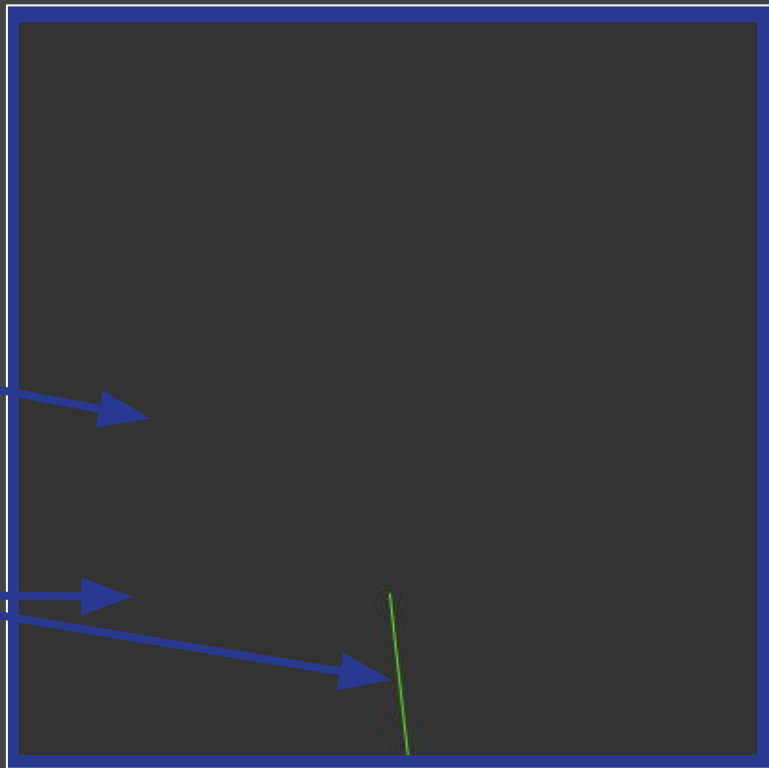
```
stroke(255, 100);  
for (int i = 0; i < sentence.length(); i++) {  
    char current = sentence.charAt(i);  
  
    if (current == 'F') {  
        line(0, 0, 0, -len);  
        translate(0, -len);  
    } else if (current == '+') {  
        rotate(angle);  
    } else if (current == '-') {  
        rotate(-angle);  
    } else if (current == '[') {  
        pushMatrix();  
    } else if (current == ']') {  
        popMatrix();  
    }  
}
```


Spinoff: Nick

Key Differences

```
dir.rotate(- PI / 2);  
dir.mult(0.70);
```

```
void show() {  
  stroke(random(255), random(255), random(255));  
  line(begin.x+10, begin.y, end.x, end.y+10);  
}
```



Algorithm Differentiation/Explanations

[illegible]

```
1 //Rules for the System
2 A = AB
3 B = A
4 //Start with A, every step apply the
5 //rules to the previous step.
6
7 1. A
8 2. AB
9 3. ABA
10 4. ABAAB
11 5. ABAABABA
12 6. ABAABABAABABA
```

What does this represent?

- Concepts of recursion and patterns
- Allows for the tree to construct the points of the next branch based off the newly constructed points on tree.
- The letters allow for the representation of the points and the branches, making the creation of the tree strictly based off the rules stated
 - This can be played around with to create many changes.

Algorithm Differentiation/Explanations

```
void branch(float len){
    line(0,0,0,-len);
    translate(0, -len);
    if (len > 4){
        pushMatrix();
        rotate(angle);
        branch(len * branch_ratio);
        popMatrix();
        pushMatrix();
        rotate(-angle);
        branch(len * branch_ratio);
        popMatrix();
    }
}

void mouseWheel(MouseEvent event){
    angle += event.getCount()/10.0;
}
```

What does this represent?

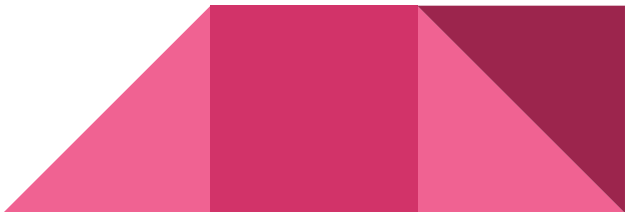
- The relationship of how the shape of the fractal tree
 - determined by the angle
 - ratio of the branches
 - What number in the recursion is occurring at that given moment.
- The next branch will be affected by the same rules, and will result in the same change to itself.
- This results in the shapes that are seen in the fractal trees and cause the creation of the different shapes

Algorithm Differentiation/Explanations

This pattern to draw the trees is probably the most simple, since it calls upon the characteristics of the existing branch that was made.

- Beginning and end point of the branch
- Rotation point based off the end point.
- Creates the array of a branches, introducing the new set every click

```
for(int i = tree.size() -1; i >= 0; i--){  
    Branch current = tree.get(i);  
    //if the current Branch has no children: add them  
    if(!current.finished){  
        tree.add(current.branchA());  
        tree.add(current.branchB());  
        tree.add(current.branchC());  
    }  
    //now that Branch has children  
    current.finished = true;  
}  
//new Level added  
count ++;
```



Coding Challenge

How to Code it



Part 1: The Branch Object

```
1 class Branch{
2
3   public PVector begin;
4   public PVector end;
5   public boolean finished = false;
6
7   Branch(PVector begin, PVector end){
8     this.begin = begin;
9     this.end = end;
10  }
```

This Piece of code defines the begin and end points of each branch object

This Piece of code creates a branch with a beginning and end-point using vectors to determine their 2D orientation

```
12 void jitter(){
13   end.x += random(-1, 1);
14   end.y += random(-1, 1);
15 }
16
17 void show(){
18   stroke(255);
19   line(begin.x, begin.y, end.x, end.y)
20 }
```

This Piece of code is used to create a sort of “jitter” in the x & y values between 1 & -1 of their native values.

This bit is used to display the branch.

Part 1(Cont.)

```
23 Branch branchA(){
24     PVector dir = PVector.sub(end, begin);
25     dir.rotate(PI / 6);
26     dir.mult(0.67);
27     PVector newEnd = PVector.add(end, dir);
28     Branch b = new Branch(end, newEnd);
29     return b;
30 }
31
32 Branch branchB(){
33     PVector dir = PVector.sub(end, begin);
34     dir.rotate(- PI / 4);
35     dir.mult(0.67);
36     PVector newEnd = PVector.add(end, dir);
37     Branch b = new Branch(end, newEnd);
38     return b;
39 }
```

This code is used to generate a branch for the right and left side, A & B respectively.

Part 2: The Tree

The Setup

```
1 ArrayList<Branch> tree = new ArrayList<Branch>();
2 ArrayList<PVector> leaves = new ArrayList<PVector>();
3 int count = 0;
4
5 void setup(){
6   size(400,400);
7   //create root-Branch
8   PVector a = new PVector(width / 2, height);
9   PVector b = new PVector(width / 2, height - 100);
10  Branch root = new Branch(a, b);
11  tree.add(root);
12 }
13
14 void mousePressed(){
15   for(int i = tree.size() - 1; i >= 0; i--){
16     Branch current = tree.get(i);
17     if(!current.finished){
18       tree.add(current.branchA());
19       tree.add(current.branchB());
20     }
21     current.finished = true;
22   }
23 }
```

These lines set up the tree; and as the comment states it also contains the lines to control the “root” Branch.

If the current branch does not have any attached, add one.

Checks current branch to see if other branches have been added.

Part 2(Cont.)

```
38 void draw(){
39     background(51);
40
41     for(int i = 0; i < tree.size(); i++){
42         tree.get(i).show();
43     }
44
45     for(int i = 0; i < leaves.size(); i++){
46         fill(255, 0, 100, 100);
47         noStroke();
48         PVector leave = leaves.get(i);
49         ellipse(leave.x, leave.y, 8, 8);
50         leave.y += random(0, 2);
51     }
52 }
```

New Level added.

On the 6th level it

Draws the Tree &
the Leaves.

Incase You Missed it

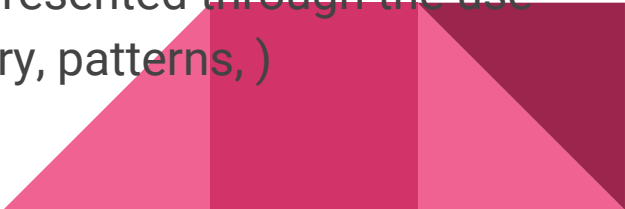
>> <https://pastebin.com/sMuCLHW2> <<

All the code we just talked about can be found at this link

Math Concept - Recursion

Recursion is the basis of this code, and without the the creation the fractal trees is inefficient. Recursion allows for the characteristics of the tree to be defined, before calling them into a function in a way that displays them in a pattern. This repetitive pattern is what we call “recursion”, but it is essentially an algorithm that allows for the creation of the program.

This concept was explored in the algorithms section of khan academy. Mathematically what is being done, is that the results of the previous object, sets the base for the next sequence of events. This can be represented through the use of factorials, as well as other abstract concepts (symmetry, patterns,)



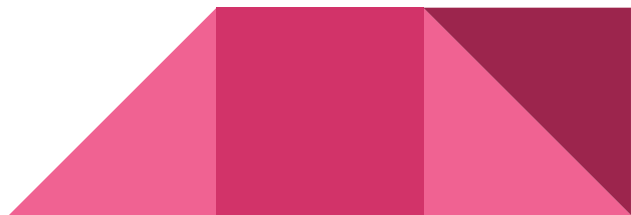
Examples of Recursion

Factorial

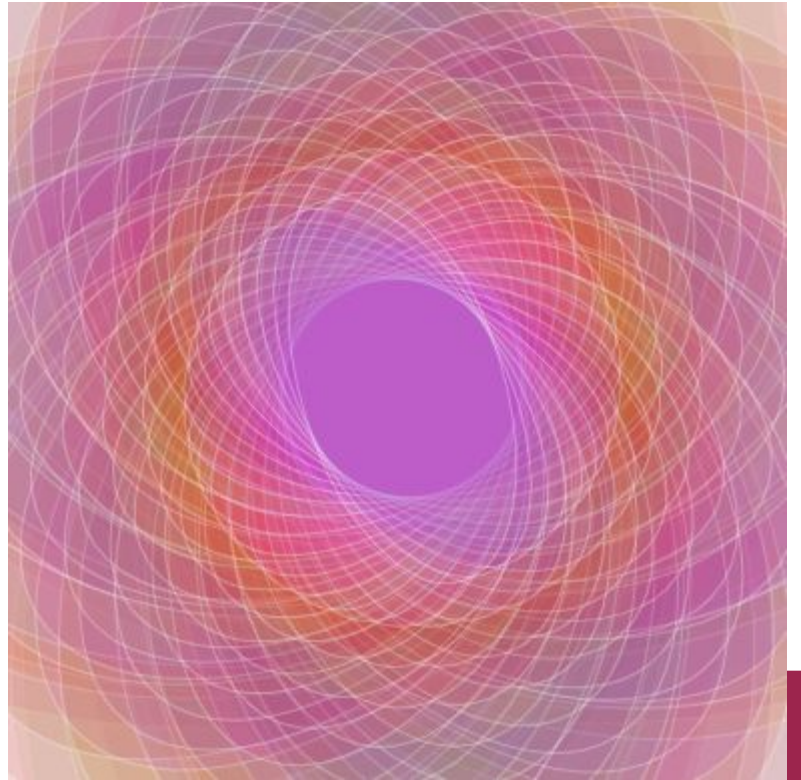
```
var factorial = function(n) {  
  var result = 1;  
  
  for(var i = 1; i <= n; i++) {  
    result *= i;  
  }  
  
  return result;  
};
```

Visual Representation

```
var drawShape = function(x, y, radius) {  
  stroke(255, 255, 255, 80);  
  var newRadius = radius/1.00983;  
  rotate(radius/23);  
  
  if (newRadius >= 200) {  
    fill(random(125, 255), random(0, 200), radius%255, 25);  
    ellipse(x, y, radius, radius/2);  
    drawShape(x, y, newRadius);  
  }  
};
```

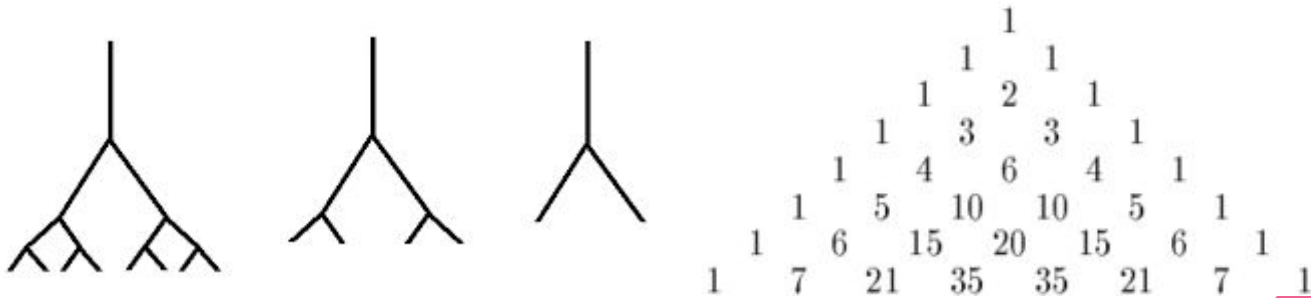


The value of $0!$ is 1.
The value of $5!$ is 120.



Math Concept #2 - Pascals Triangle

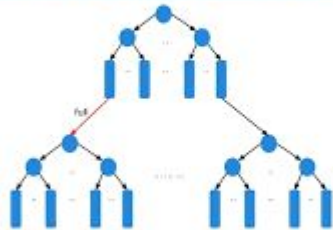
- PT used for binomial expansion, statistics and more
- Resembles the pyramid
- Fractals trees can be used for PT
- Each branch branches outwards (Continuously)
- Every branch out is the next level on the triangle



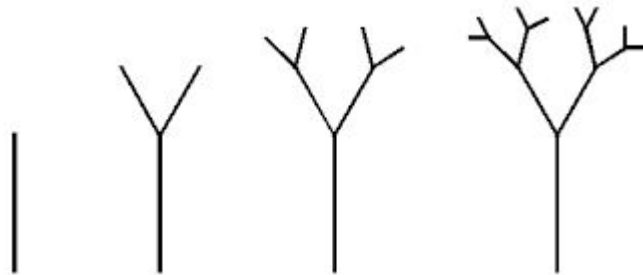
Computer Science Concept- Tree Data Structures

- Type of Data Structure ; Fractal Tree Index
- Storing Data
- Access; insert, delete data
- Faster than another type of DS; B-Tree
- Can continue to grow if needed

Fractal-tree Indexes (Block size)

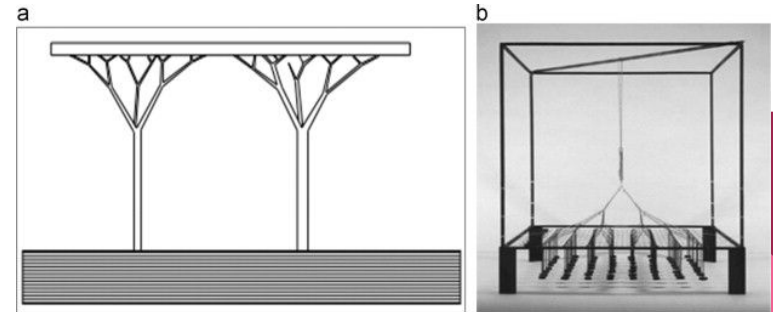
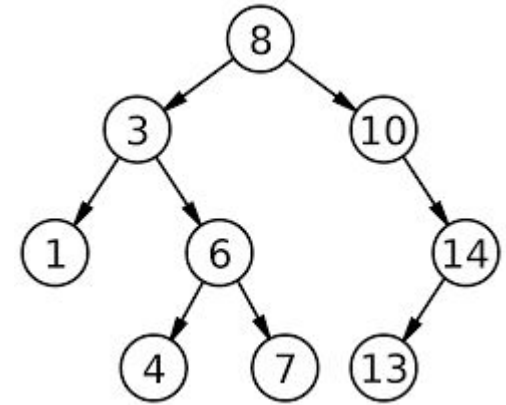


Fractal! 4MB one seek...



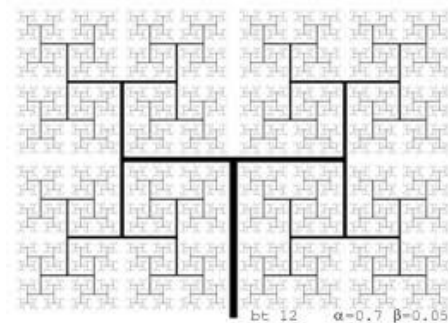
Future Opportunities

- Code in SQL on Khan academy
 - Managing Data & databases
 - Show you how to store data like Fractal Tree index
- Real Life Applications (abstract):
 - Managing Data
 - Statistics
 - Architecture
 - Fractal Image Compression (JPEG)



Emerging Technologies

- Antenna's adopting Fractal Design
 - Used in many devices; certain computers, electronics , aircrafts, etc
- Enhances the bandwidth of incoming signals
 - Multi Banded
- Allow for a compact design
 - Still allows for high performance
- Fractal Design
 - Ability to develop rapid beamforming algorithms based on the recursive nature of fractals



Thanks for
Listening :))

