2.3 Recursion
The factorial of a positive integer $N$ is computed as the product of $N$ with all positive integers less than or equal to $N$.

4! = $4 \times 3 \times 2 \times 1 = 24$

30! = $30 \times 29 \times ... \times 2 \times 1 = 2652528598121910586363084800000000$
1. int b = factorial(5);

2. static int factorial(int n) {
3.   int f = 1;
4.   for (int i=n; i>=1; i--) {
5.     f = f * i;
6.   }
7.   return f;
8. }

Trace it.
5! = 5 \times 4 \times 3 \times 2 \times 1
4! = 4 \times 3 \times 2 \times 1
5! = 5 \times 4!
N! = N \times (N-1)!

Factorial can be defined in terms of itself
Recursion
Recursion
Overview

What is recursion? When one function calls itself directly or indirectly.

Why learn recursion?
- New mode of thinking
- Powerful programming paradigm

Many computations are naturally self-referential:
- Mergesort, FFT, gcd, depth-first search
- Linked data structures
- A folder contains files and other folders

Closely related to mathematical induction
Factorial – Recursive Implementation

1. int b = factorial(5);

2. static int factorial(int n) {
3.   if (n == 1) {
4.     return 1;
5.   } else {
6.     int f = n * factorial(n-1);
7.     return f;
8.   }
9. }

Trace it.
Last In First Out (LIFO) Stack of Plates
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

public static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}

void main(String[] args){
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}
```java
1. void main(String[] args) {
2.   int a = 10;
3.   int b = factorial(5);
4.   System.out.println(b);
5. }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * factorial(n-1);
6.     return f;
7.   }
8. }
```
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

Call Stack:
```
main()
  a=10, Line=3
```
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}
```

```java
static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

**Compiled Code**

**Executing Function**

**Call Stack**

```
main()
a=10, Line=3
```
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
1. public static void main (String[] args) {
2.     int a = 10;
3.     int b = factorial(5);
4.     System.out.println(b);
5. }

1. static int factorial(int n) {
2.     if (n == 1) {
3.         return 1;
4.     } else {
5.         int f = n * factorial(n-1);
6.         return f;
7.     }
8. }

1. int factorial(int n=5) {
2.     if (n == 1) {
3.         return 1;
4.     } else {
5.         int f = n * factorial(n-1);
6.         return f;
7.     }
8. }
1. public static void main(String[] args) {
   2. int a = 10;
   3. int b = factorial(5);
   4. System.out.println(b);
   5. }

1. static int factorial(int n) {
   2. if (n == 1) {
      3. return 1;
   4. } else {
      5. int f = n *
         6. factorial(n-1);
      7. return f;
   8. }

```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}
```

```java
public static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

```
1. int factorial(int n=4) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * factorial(n-1);
6.     return f;
7.   }
8. }
```

Call Stack
- factorial() n=5, Line=5
- main() a=10, Line=3

Compiled Code
- `public static void main(String[] args) {`
  - `int a = 10;`
  - `int b = factorial(5);`
  - `System.out.println(b);`
- `}`

- `public static int factorial(int n) {`
  - `if (n == 1) {
  - `return 1;
  - `} else {
  - `int f = n * factorial(n-1);
  - `return f;
  - `}
- `}`
1. int factorial(int n=4) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * 
6.     factorial(n-1);
7.     return f;
8.   }
9. }

1. public static void main
2.   (String[] args) {
3.     int a = 10;
4.     int b = factorial(5);
5.     System.out.println(b);
6.   }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * 
6.     factorial(n-1);
7.     return f;
8.   }
9. }
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}

int factorial(int n=3) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```java
1. public static void main(String[] args) {
2.   int a = 10;
3.   int b = factorial(5);
4.   System.out.println(b);
5. }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * factorial(n-1);
6.     return f;
7.   }
8. }
```

```
Call Stack
factorial()
n=4, Line=5
factorial()
n=5, Line=5
main()
a=10, Line=3
```
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

Call Stack:
- `main()`
  - a=10, Line=3
- `factorial()`
  - n=5, Line=5
- `factorial()`
  - n=4, Line=5
- `factorial()`
  - n=3, Line=5
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

The call stack shows the execution of the `factorial` function with different values of `n`. The values on the call stack are:

- `n=3, Line=5`
- `n=4, Line=5`
- `n=5, Line=5`
- `main(a=10, Line=3)`

The main function calls `factorial` with `a=10`, and the call stack shows the recursive calls with decreasing values of `n` until it reaches `n=1`, returning `1` and building up the result for `b`.
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```

Call Stack:
- main()
a=10, Line=3
- factorial(n=5, Line=5)
- factorial(n=4, Line=5)
- factorial(n=3, Line=5)
- factorial(n=2)
1. public static void main
   (String[] args) {
2.   int a = 10;
3.   int b = factorial(5);
4.   System.out.println(b);
5. }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n *
       factorial(n-1);
6.     return f;
7.   } 
8. }

1. int factorial(int n=2) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int  f = n *
         factorial(n-1);
6.     return f;
7.   }
8. }
int factorial(int n=1) {
  if (n == 1) {
    return 1;
  } else {
    int f = n * factorial(n-1);
    return f;
  }
}

public static void main(String[] args) {
  int a = 10;
  int b = factorial(5);
  System.out.println(b);
}

int factorial(int n) {
  if (n == 1) {
    return 1;
  } else {
    int f = n * factorial(n-1);
    return f;
  }
}

Call Stack:
- main()
a=10, Line=3
- factorial()
  n=5, Line=5
- factorial()
  n=4, Line=5
- factorial()
  n=3, Line=5
- factorial()
  n=2, Line=5
1. int factorial(int n=1) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * factorial(n-1);
6.     return f;
7.   }
8. }

1. public static void main
   (String[] args) {
2.   int a = 10;
3.   int b = factorial(5);
4.   System.out.println(b);
5. }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n *
       factorial(n-1);
6.     return f;
7.   }
8. }
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```
```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}

int factorial(int n=3) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * 2;
        return f;
    }
}
```

```java
public static void main(String[] args) {
    int a = 10;
    int b = factorial(5);
    System.out.println(b);
}

static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        int f = n * factorial(n-1);
        return f;
    }
}
```
1. int factorial(int n=5) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * 24;
6.     return f;
7.   }
8. }

main()
   a=10, Line=3
```java
1. void main(String[] args) {
2.   int a = 10;
3.   int b = factorial(5);
4.   System.out.println(b);
5. }

1. static int factorial(int n) {
2.   if (n == 1) {
3.     return 1;
4.   } else {
5.     int f = n * factorial(n-1);
6.     return f;
7.   }
8. }
```
The Call Stack keeps track of ...

1. all functions that are suspended, in order
2. the point in the function where execution should resume after the invoked subordinate function returns
3. a snapshot of all variables and values within the scope of the suspended function so these can be restored upon continuing execution
Recursive Graphics
L-systems
L-systems
L-systems

void drawTree(int x, int y, double angle, int depth) {
    // base case
    // compute end coordinate (x2, y2)
    //     (with length = depth)
    // draw tree recursively
    (x,y)
    (x + cos(angle) * length, y + sin(angle) * length)
}

(x,y)

(x,y)
void drawTree(int x, int y, double angle, int depth) {
    // base case

    // compute end coordinate (x2, y2)
    // (with length = depth )

    // draw tree recursively
    StdDraw.line (x, y, x2, y2)
    drawTree(x2, y2, angle + leftangle, depth – 0.01);
    drawTree(x2, y2, angle - rightangle, depth – 0.01);
}

At each stage we want two ‘sub-trees’ to be created.
One grows to the left and one grows to the right.
Also, the depth controls the length of the branch.
public static void drawTree(int x, int y, double angle, int depth) {
    // base case
    if (depth == 0) return;

    // compute end coordinate
    double angleRadians = Math.toRadians(angle);
    double x2 = x + (Math.cos(angleRadians) * depth);
    double y2 = y + (Math.sin(angleRadians) * depth);

    // draw tree recursively
    StdDraw.line(x, y, x2, y2);
    drawTree(x2, y2, angle - leftangle, depth - 1);
    drawTree(x2, y2, angle + rightangle, depth - 1);
}

For the complete program refer to the cis110 website
Creating a maze, recursively

1. Start with a rectangular region defined by its upper left and lower right corners
2. Divide the region at a random location through its more narrow dimension
3. Add an opening at a random location
4. Repeat on two rectangular subregions

Towers of Hanoi

Towers of Hanoi

Move all the discs from the leftmost peg to the rightmost one.

- Only one disc may be moved at a time.
- A disc can be placed either on empty peg or on top of a larger disc.
Towers of Hanoi Legend

Q. Is world going to end (according to legend)?
   - 64 golden discs on 3 diamond pegs.
   - World ends when certain group of monks accomplish task.

Q. Will computer algorithms help?
Towers of Hanoi: Recursive Solution

Move n-1 smallest discs right.

Move n-1 smallest discs right. Move largest disc left.

cyclic wrap-around

Move n-1 smallest discs right.
public class TowersOfHanoi {

   public static void moves(int n, boolean left) {
      if (n == 0) return;
      moves(n-1, !left);
      if (left) System.out.println(n + " left");
      else System.out.println(n + " right");
      moves(n-1, !left);
   }

   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      moves(N, true);
   }
}

moves(n, true) : move discs 1 to n one pole to the left
moves(n, false): move discs 1 to n one pole to the right

smallest disc
Towers of Hanoi: Recursive Solution

% java TowersOfHanoi 3
1 left
2 right
1 left
3 left
1 left
2 right
1 left

% java TowersOfHanoi 4
1 right
2 left
1 right
3 right
1 right
2 left
1 right
4 left
1 right
2 left
1 right
3 right
1 right
2 left
1 right

every other move is smallest disc

subdivisions of ruler
Towers of Hanoi: Recursion Tree

$n$, left

3, true

2, false

1, true

1, true

2, false

1, true

1 left 2 right 1 left 1 left

$n$, left

1 14

2 7

3 4 6 5 9 10 12 11 17 18 20 19 23 24 26 25

13 8 16 21 27 22

28 15
Remarkable properties of recursive solution.

- Takes $2^n - 1$ moves to solve $n$ disc problem.
- Sequence of discs is same as subdivisions of ruler.
- Every other move involves smallest disc.

Recursive algorithm yields non-recursive solution!

- Alternate between two moves:
  - move smallest disc to right if $n$ is even
  - make only legal move not involving smallest disc

Recursive algorithm may reveal fate of world.

- Takes 585 billion years for $n = 64$ (at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long!
Fruits of Design, Certified Organic

The heyday of design was the Art Deco era, when the premiere decorators, artists and sculptors created such iconic designs as the Chrysler Building, the Empire State Building and the Rockefeller Center. The Bauhaus school in Germany, the school of design and architecture, was the birthplace of modernism, and its influence can be seen in every aspect of design, from furniture to fashion. The Bauhaus also gave rise to the concept of design as a tool for social change, and the school's graduates went on to design everything from celluloid to plastic. The Bauhaus was dissolved by the Nazis, but its legacy lives on in the works of such designers as Dieter Rams, who designed the Braun line of kitchen appliances, and Dieter Rams, who designed the Braun line of kitchen appliances.


The Gifts to Open Again and Again

This magazine is packed with articles on design, art and architecture, and features interviews with leading designers, architects and artists. It is a great resource for anyone interested in these fields, and it is also a beautiful object in its own right. The cover of this issue features a piece of art by the renowned artist, Vincent van Gogh, and the pages are filled with images of beautiful buildings and objects designed by some of the world's greatest minds.

Black, White and Read All Over Over

By RANDY JENKINS

In case you were wondering, the current "New York Times Magazine" cover is "The New Book of Droste". It is a special issue that focuses on design, art and architecture, and features interviews with leading designers, architects and artists. It also includes a piece on the history of design and architecture, and a section on contemporary design. The cover of this issue features a piece of art by the renowned artist, Vincent van Gogh, and the pages are filled with images of beautiful buildings and objects designed by some of the world's greatest minds. The issue also includes a section on the history of design and architecture, and a section on contemporary design. The cover of this issue features a piece of art by the renowned artist, Vincent van Gogh, and the pages are filled with images of beautiful buildings and objects designed by some of the world's greatest minds. The issue also includes a section on the history of design and architecture, and a section on contemporary design.

Divine and Devotee Meet Across Hinges

In case you were wondering, the current "New York Times Magazine" cover is "The New Book of Droste". It is a special issue that focuses on design, art and architecture, and features interviews with leading designers, architects and artists. It also includes a piece on the history of design and architecture, and a section on contemporary design. The cover of this issue features a piece of art by the renowned artist, Vincent van Gogh, and the pages are filled with images of beautiful buildings and objects designed by some of the world's greatest minds. The issue also includes a section on the history of design and architecture, and a section on contemporary design. The cover of this issue features a piece of art by the renowned artist, Vincent van Gogh, and the pages are filled with images of beautiful buildings and objects designed by some of the world's greatest minds. The issue also includes a section on the history of design and architecture, and a section on contemporary design.
H-tree of order n.

- Draw an H.
- Recursively draw 4 H-trees of order n-1, one connected to each tip.
Animated H-tree

Animated H-tree. Pause for 1 second after drawing each H.
public class Htree {
    public static void draw(int n, double sz, double x, double y) {
        if (n == 0) return;
        double x0 = x - sz/2, x1 = x + sz/2;
        double y0 = y - sz/2, y1 = y + sz/2;
        StdDraw.line(x0, y, x1, y);
        StdDraw.line(x0, y0, x0, y1);
        StdDraw.line(x1, y0, x1, y1);
        draw(n-1, sz/2, x0, y0);
        draw(n-1, sz/2, x0, y1);
        draw(n-1, sz/2, x1, y0);
        draw(n-1, sz/2, x1, y1);
    }
}

public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    draw(n, .5, .5, .5);
}
Recursive subdivision

• Divide your original area into smaller segments

• Recurse inside each of the segments

• With each recursive step add a little bit of randomness

• Patterns like Brownian motion, Plasma Clouds etc
Plasma Cloud

Plasma cloud centered at \((x, y)\) of size \(s\).

- Each corner labeled with some grayscale value.
- Divide square into four quadrants.
- The grayscale of each new corner is the average of others.
  - center: average of the four corners + random displacement
  - others: average of two original corners
- Recurse on the four quadrants.

\[
\begin{align*}
&c_1 + c_4 \\
&c_2 + c_3 \\
&c_1 + c_2 \\
&(c_1 + c_2 + c_3 + c_4) + \delta
\end{align*}
\]
Plasma Cloud