Recursion
What's an algorithm again?

- An **algorithm** is a *precise, systematic process for an agent to produce a specified result*
- Programs are algorithms
- Five properties characterize algorithms
  - **Input specified** – tell the form and amount of input required
  - **Output specified** – tell the form and amount of output produced
  - **Definiteness** – explicit steps to do, in order
  - **Effectiveness** – operations are within agent's capabilities
  - **Finiteness** – algorithms stop and give an answer or say "none"
Examples of Algorithms

• Sorting
  – Arrange a stack of exams by student name
  – Create fantasy football standings in spreadsheet

• Searching
  – Find a student's exam in a stack of sorted exams
  – Find a keyword on a webpage
  – Guessing a number 1-100 with clues
Recursion

- **Recursion** is when the solution to a problem depends on solutions to smaller instances of the same problem
  - Algorithms that use recursion to solve problems are called *divide-and-conquer*

- **Recursive calls**
  - A function can call itself within the body of the function
  - A function can also indirectly call itself (*f* calls *g* which calls *f* which calls *g*...)
Many concepts are defined in terms of themselves in mathematics

- **Factorial**
  - $0!$ is 1
  - $n!$ is $n$ times $(n-1)!$ for $n>0$

- **Fibonacci**
  - $\text{fib}(0) = 0$
  - $\text{fib}(1) = 1$
  - $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ for $n > 1$

- **Sorted**
  - A list is sorted if the smallest element appears first and the remaining elements are sorted
How can this possibly work?

YO DAWG I HERD YOU LIKE CARS SO WE PUT A CAR IN YO CAR SO YOU CAN DRIVE WHILE U DRIVE

YO DAWG I HEAR YOU HATE RECURSION

SO I RECURSED MYSELF ON TOP OF MYSELF SO YOU CAN LEARN TO RECURSE WHILE I RECURSE
Recursive Factorial Function

• factorial(4):
  – return 4 * factorial(3)
    • factorial(3):
      – return 3 * factorial(2)
        • factorial(2):
          • return 2 * factorial(1)
            • factorial(1):
              • return 1 * factorial(0)
                • factorial(0):
                  • return 1
                  • factorial(0) is 1
                  • return 1 * 1
                  • factorial(1) is 1
                • return 2 * 1
                • factorial(2) is 2
                  – return 3 * 2
          • factorial(3) is 6
    – return 4 * 6
• factorial(4) is 24

```c
int factorial(int n) {
    if(n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```
Recursive Fibonacci Function

- fib(3):
  - return fib(2) + fib(1)
    - fib(2):
      - return fib(1) + fib(0)
        - fib(1):
          - return 1
        - fib(0):
          - return 0
      - fib(0) is 1
    - fib(1) is 1
    - return 1 + 1
  - fib(2) is 2
  - fib(1):
    - return 1
  - fib(1) is 1
  - return 2 + 1
- fib(3) is 3

```c
int fib(int n) {
    if(n == 0) {
        return 0;
    }
    if(n == 1) {
        return 1;
    }
    return fib(n - 1) + fib(n - 2);
}
```

Double recursion!
Recursive Shield 1 – What does it draw?

void setup() {
  size(400, 400);
  noStroke();
  noLoop();
}

void draw() {
  shield(0, 0, 200, 200);
  shield(200, 200, 200, 200);
}

void mousePressed() {
  redraw();
}

void shield(float xbase, float ybase, float w, float h) {
  float r = random(0, 255);
  float g = random(0, 255);
  float b = random(0, 255);
  fill(color(r, g, b));
  rect(xbase, ybase, w, h);
}

A. a randomly colored square
B. two randomly colored squares
C. two randomly colored rectangles (not square)
D. many randomly colored squares of different sizes
E. there is an error in the program
Recursive Shield (part 2)

Where is the recursive call?

```cpp
void setup() {
  size(400, 400);
  noStroke();
  noLoop();
}

void draw() {
  shield(0, 0, 400, 400);
}

void mousePressed() {
  redraw();
}

void shield(float xbase, float ybase, float w, float h) {
  float r = random(0, 255);
  float g = random(0, 255);
  float b = random(0, 255);
  fill(color(r, g, b));
  rect(xbase, ybase, w, h);
  shield(xbase, ybase, w/2, h/2);
}
```

A

B

C
void setup() {
  size(400, 400);
  noStroke();
  noLoop();
}

void draw() {
  shield(0, 0, 400, 400);
}

void mousePressed() {
  redraw();
}

void shield(float xbase, float ybase, float w, float h) {
  float r = random(0, 255);
  float g = random(0, 255);
  float b = random(0, 255);
  fill(color(r, g, b));
  rect(xbase, ybase, w, h);
  shield(xbase, ybase, w/2, h/2);
}
Oops!

rect(xbase, ybase, w, h);

StackOverflowError: This sketch is attempting too much recursion.

to be called recursively (it's calling itself and going in circles),
or you're intentionally calling a recursive function too much,
and your code should be rewritten in a more efficient manner.
Recursive Shield (part 3)

```c
void shield(float xbase, float ybase, float w, float h) {
    float r = random(0, 255);
    float g = random(0, 255);
    float b = random(0, 255);
    if (w < 2) {
        // Small enough to ignore
        return;
    }
    fill(color(r, g, b));
    rect(xbase, ybase, w, h);
    shield(xbase, ybase, w/2, h/2);
    shield(xbase + w/2, ybase + h/2, w/2, h/2);
}
```

A. a randomly colored square
B. two randomly colored squares
C. two randomly colored rectangles (not square)
D. many randomly colored squares of different sizes
E. there is an error in the program
The Issues

- How are recursive calls executed?
- How do you understand a recursive function?
  - How do you write recursive functions?

Simulating the function and executing its recursive calls can be helpful but is not the only way to fully understand how a recursive function works.
Understanding Recursion

• Step 0
  – Have a precise specification of what the function does

• Step 1
  – Check base cases (no recursive call)

• Step 2
  – Check that arguments are getting smaller or that progress is somehow being made towards base cases

• Step 3
  – Check that recursive cases work properly
**Merge Sort**

- **To sort $n$ elements**
  - If $n$ is 1, copy to the output and you're done
  - Otherwise, split $n$ elements into two equal piles
    - (put one more on the left side if $n$ is odd)
  - Sort the left side
  - Sort the right side
  - Merge the left and right sides into output

- **To merge two sorted piles into one output:**
  - If both piles are empty, you're done
  - If only one pile has elements, pick the first one in that pile
  - If both piles have elements
    - Look at the first element of both piles, pick smaller one
  - Move picked element to output
  - Merge remaining sorted piles into the output
Merge Sort

- http://en.wikipedia.org/wiki/Merge_sort
Does merge sort work?

- Specification of `merge_sort` function
  - Input: one list of elements
  - Output: one list of elements, in sorted order
- Base cases
  - What if input list is just one element?
- Check progress
  - What is getting smaller each time? Will it end?
- Recursive case
  - Assume all recursive calls work
  - Assume all other functions are working
  - Does function work when we do recursive calls?
    - `merge_sort` generates sorted left side, sorted right side
      - We know they are sorted because we assume recursive calls work
    - After merge, output is sorted and contains all elements from left and right
      - Because we assume other functions like merge work
    - Is our output correctly sorted?
How many steps does it take?

- Number of steps depends on:
  - $n$ (number of items in input)
- How long to merge $n$ and $m$ elements in two lists?
  - After some analysis, it takes $n+m$ steps
- Let's call the number of steps $f(n)$ (worst case)
- We know:
  - $f(1) = 1$
  - $f(n) = f(n/2) + f(n/2) + (n/2 + n/2)$
    - $= 2 \cdot f(n/2) + n$
- What function works for $f$?
  - This is a tricky fun math problem!
Wolfram Alpha to the rescue

\[ f(n) = 2 f\left(\frac{n}{2}\right) + n \]

Recurrence equation solution:
\[ f(n) = \frac{c_1 n}{2} + \frac{n \log(n)}{\log(2)} \]

\[ O(n \log(n)) \]
Binary Search

- Binary search
  - Input: sorted list of $n$ elements, element $x$
  - Output: yes/no, answer yes if $x$ is in list, no otherwise

- Steps
  - If list is empty, return no
  - Pick element $y$ in middle of list to divide list into left/right
    - If $n$ is even, choose earlier of two middle elements
    - If $x == y$ then return yes
  - If $x < y$ then use binary search to find $x$ in left list
  - Otherwise use binary search to find $x$ in right list
How many steps?

- Number of steps depends on size of list, \( n \)
- Let \( f(n) \) be number of steps (worst case)
- Must satisfy:
  - \( f(0) = 1 \)
  - \( f(n) = 2 + f(n / 2) \)
- What works?
Wolfram Alpha

Input:

\[ f(n) = 2 + f\left(\frac{n}{2}\right) \]

Recurrence equation solution:

\[ f(n) = c_1 + \frac{2 \log(n)}{\log(2)} \]

O(log(n))
Another Search Variation

- BogoSearch
  - Input: sorted list of \( n \) elements, element \( x \)
  - Output: yes/no, answer yes if \( x \) is in list, no otherwise

- Steps
  - If list is empty, return no
  - Pick element \( y \) at start of list
  - If \( x == y \) then return yes
  - Otherwise remove \( y \) and use bogosearch to find \( x \) in remaining list

- What is the equation for \( f(n) \)?
  - A. \( f(n) = 2 \cdot f(n / 2) + 1 \)
  - B. \( f(n) = 1 + f(n / 2) \)
  - C. \( f(n) = 1 + f(n - 1) \)
  - D. \( f(n) = 1 + f(n) \)
fill(color(r, g, b));
rect(xbase, ybase, w, h);
shield(xbase + 0 * w/3, ybase + 0 * h / 3, w/3, h/3);
// shield(xbase + 1 * w/3, ybase + 0 * h / 3, w/3, h/3);
shield(xbase + 2 * w/3, ybase + 0 * h / 3, w/3, h/3);
// shield(xbase + 0 * w/3, ybase + 1 * h / 3, w/3, h/3);
shield(xbase + 1 * w/3, ybase + 1 * h / 3, w/3, h/3);
// shield(xbase + 2 * w/3, ybase + 1 * h / 3, w/3, h/3);
// shield(xbase + 0 * w/3, ybase + 2 * h / 3, w/3, h/3);
// shield(xbase + 1 * w/3, ybase + 2 * h / 3, w/3, h/3);
shield(xbase + 2 * w/3, ybase + 2 * h / 3, w/3, h/3);
}
http://www.youtube.com/watch?v=3zgBBXWVYWU
Robots (and other physical programs)
Arduino

- http://www.youtube.com/watch?v=UoBUXOOdLXY
Programming Arduino

```cpp
void setup() {  
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {  
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // there is a pull up resistor on the button so the unpressed
  // state is HIGH
  // check to see if the buttonState is LOW:
  if (buttonState == HIGH) {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  } else {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
}
```

Done compiling.

Binary sketch size: 752 bytes (of a 4096 byte maximum)

ATtiny44 (external 20 MHz clock) on COM1
DIY Underwater Robot

- [Link to YouTube video](http://www.youtube.com/watch?v=5riPKdpuoil)
Welcome to the community center for hackers + thinkers.
Robots that Fly

- http://www.youtube.com/watch?v=4ErEBkj_3PY
Are Droids Taking our Jobs?

- http://www.youtube.com/watch?v=WMF-Z74C1QE
End