Recursive functions

# Lesson Notes

## What is a recursive function?

* A function that calls itself:
  + Either directly or indirectly
    - A calls A (directly)
    - A calls B calls A (indirectly)
* Another way to repeat a process
* Typically used for “divide and conquer” type of problems
* This requires that:
  + Your algorithm divides the problem into parts
  + The parts are solvable using the same algorithm as the whole
  + You know when to stop dividing
  + When you stop dividing (base case), you know how to compute the result
* You can have infinite recursion
  + Just like infinite loop
  + Example function f() { f(); }
  + What happens: stack overflow
  + For example, if you don’t have a base case
  + (try it)

## Find someone in a phonebook

* Show iterative algorithm:
  + Search one by one
  + When I find the person
  + Then I “break” out of my loop
* Show recursive algorithm:
  + Open phonebook to middle
  + Rip it in half
  + Search again
  + (phonebook demonstration)
  + How did I divide?
  + What was my “base case”?

## Nested circles

* Show picture of result
* How can we draw this?
* Show geometry behind the result
* Algorithm:
  + Given two points
  + Find the mid-point
  + Draw a circle between the two points
    - Centered at midpoint
    - Radius of half the distance
  + Determine first two points, recurse
  + Determine second two points, recurse
  + Stop when distance between the two points is small
* Need diagram that illustrates the geometry

## Distance between two points

* Given two coordinates (30, 40), how do we find the distance (10):
  + Subtract one from the other
  + Take absolute value (in case they are switched)
* So:
  + function distance(x0, x1) {
  + return Math.abs(x1 – x0);
  + }

## Finding the midpoint

* Given two coordinates (30, 40), how do we find the midpoint (35)?
  + 30 35 40
  + + ------------------+----------------- +
  + x0 x1
* One way:
  + Midpoint is half way from the first coordinate to the second coordinate
    - Find the distance between the coordinates
    - Divide it in half
    - Add it to the first coordinate
  + So:
    - function midpoint(x0, x1) {
    - var distance = x1 - x0;
    - Return x0 + distance / 2;
    - }
* Another way:
  + Find the “average” of coordinates
    - Sum both points
    - Divide by the number of points (2)
  + So:
    - function midpoint(x0, x1) {
    - return (x1 + x0) / 2;
    - }
* This works for any number of dimensions
  + Just do it for each coordinate (x, y, z)

## Exercise: nested circles

* 500 x 500 canvas
* Y coordinate is 250
* Starting coordinates are 0, 499
* Keep recursing until distance between points is less than 4

## Mountain Ranges

* Show result
* How are we going to do this!
* Need diagram:
  + Left point
  + Right point
  + Middle point (both X, and Y coordinate)
  + Random shift (Y)
* Algorithm
  + Find midpoint (x, y)
  + Generate random delta [0..N)
  + Adjust Y by delta
  + If X distance is less than 5 stop

## Finding the midpoint

* Our midpoint algorithm works for both X and Y
  + var midpointX = midpoint(x0, x1);
  + var midpointY = midpoint(y0, y1);

## Generating the delta

* Need a value from –delta/2 to delta/2
* Use the technique from random number discussion
  + (Math.random() – 0.5) \* maxDelta

## Exercise: Mountain Ranges

* Mountain:
  + Canvas: Width:1000 Height:500
  + Starting point (X:0, Y:250)
  + Ending point (X:999, Y:250)
  + Starting delta: 400
* Mountains:
  + Draw the mountain range 5 times

## Pitfalls of recursion (the “bad news”)

* May be slower due to time it takes to call a function and return
  + Unless compiler can turn recursion into a loop
* May require more memory
  + Each calls requires some space on the stack
* Stack may be limited
  + Especially limited in javascript

## Why learn about recursion?

* Some problems are much easier and much more clear when written recursively
  + Think about the two examples we just did
  + Those would be difficult to think about using an iterative approach
* The stack problem is not always a problem
  + The input may be small
* Thinking recursively is a great problem solving tool
  + Let’s you break down the problem into simpler ones
  + The simpler ones might be pretty simple
* You may be able to easily convert your recursive algorithm to use iteration
* Recursion will reappear later in the course