Java 2 Recursion

GUESS WHOSE BACK! BACK AGAIN!!

- GNU
  - GNU's Not UNIX
  - Recursive Acronym because it is self referential.

- Necessary properties of recursion
  - Recursive are self-referential
  - Recursive things must not be circular
  - Recursive things must have meaning

- Structural Recursion
  - The structure being defined appears as part of the definition.
  - ie Node
  - Ex: A linked list is either empty or it is a single node that points to a linked list.

- Computational Recursion
  - The computation being defined is applied within its own definition
  - Ex: The factorial function

- Recursive Def
  - A recursive definition, whether structural or computational, consists of two parts:
    - Base Case:
      - A simple statement or definition that does not involve recursion.
    - Reduction step:
      - A set of rules that reduce all other cases toward the base case.
  - Ex: A linked list either empty (base case) or it is a single node that points to a linked list (reduction step).

- Writing recursive methods

```
if (base case) {
	return solution;
}
else {
	recursive_call;
}
```

- Calling recursive methods
  - When a method is called - recursive or not - an activations record (or stack frame) for that method is pushed onto the runtime stack (call stack).
  - When a method returns - recursive or not - its activation is popped from the stack.
  - Time complexity? O(N)

- Writing a recursive method to get the length a linked list

```
if (n == null) {
	return 0;
} else {
	return 1 + length(n.getNext());
}
```

Time complexity? O(N)

Linear search:

```
public boolean contains(Node n, Object target) {
	if (n == null) {
		return false;
	} else {
		if (n.getElement().equals(target)) {
			return true;
		} else {
			return contains(n.getNext(), target);
		}
	}
}
```

- Tower of Hanoi
  - How to move disk from A to C in the most optimal way given the following constraints
  - Only the top disk on a stack can be moved
  - Only one disk can be moved at a time
  - No disk can be placed on top of a smaller disk.
  - Base Case
    - One disk
  - Reduction Step //How to view these N disks
    - N - 1 disks
    - Move N-1 disks from A to B
    - Move the 1 disk left to C
    - Move N-1 disks from B to C

- Variation on the Pattern
  - Finding the nth Fibonacci Number
  - Has more than one base case
  - Time comp 2^N

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

- Performance Issues
  - Recursive code can be less efficient than an equivalent iterative version because of overhead costs.
  - Primary costs associated with recursion:
    - (1) Method call/return overhead
    - (2) Call stack space
- Dealing with the cost
  - Rewrite iteratively - new algorithm
  - Imitate recursion with a stack

Factorial can written as:

```
public int factorial(int n) {
	int fact = 1;
	for (int i = 1; i <= n; i++) {
		fact *= i;
	}
```

```
    return fact;
}
```

- Tail recursion
  - a special case of recursion where the last operation of the method is the recursive call
  - Pattern:
    - To write a method in tail recursive form, we will add parameters as necessary so that the computation is performed on the ‘down trip’ via the parameters instead of performing it on the ‘up trip’

```
public in factTR(int n, int fact) {
    if (n == 0)
        return 1;
    else
        return factTR(n - 1, n*fact);
}
```

- Compilers can eliminate tail recursion automatically, and this is a common optimization
  - However, there are no compilers in Java
- GCC eliminates tail recursion in C programs with the -O2 optimization flag.
  - % gcc -S trgactorial.c //observes the assembly code
  - % gcc -S -O2 trfactorial.c //checks for any optimization possibilities
  - GCC can actually optimize regular recursion as well

- There is still a performance benefit in java from using tail recursion
  - Writing the Fibonacci sequence in tail recursion gives O(N)