More on Recursive

- Recursion vs. Iteration

- Why Recursion?

- Common Errors in Writing Recursive Methods:
Recursion vs. Iteration

- In general, an iterative version of a method will execute more efficiently in terms of time and space than a recursive version.

- This is because the overhead involved in entering and exiting a function in terms of stack I/O is avoided in iterative version.

- Sometimes we are forced to use iteration because stack cannot handle enough activation records - Example: power(2, 5000))
Why Recursion?

• Usually recursive algorithms have less code, therefore algorithms can be easier to write and understand - e.g. Towers of Hanoi. However, avoid using excessively recursive algorithms even if the code is simple.

• Sometimes recursion provides a much simpler solution. Obtaining the same result using iteration requires complicated coding - e.g. Quicksort, Towers of Hanoi, etc.

• Recursive methods provide a very natural mechanism for processing recursive data structures. A recursive data structure is a data structure that is defined recursively – e.g. Tree.

• Functional programming languages such as Clean, FP, Haskell, Miranda, and SML do not have explicit loop constructs. In these languages looping is achieved by recursion.
Why Recursion?

- Some recursive algorithms are more efficient than equivalent iterative algorithms.
- Example:

```java
public static long power1 (int x, int n) {
    long product = 1;
    for (int i = 1; i <= n; i++)
        product *= x;
    return product;
}

public static long power2 (int x, int n) {
    if (n == 1) return x;
    else if (n == 0)return 1;
    else {
        long t = power2(x , n / 2);
        if (((n % 2) == 0) return t * t;
        else return x * t * t;
    }
}
```
Common Errors in Writing Recursive Methods

- The method does not call itself directly or indirectly.

- Non-terminating Recursive Methods (Infinite recursion):
  a) No base case.

```java
int badFactorial(int x) {
    return x * badFactorial(x-1);
}
```

b) The base case is never reached for some parameter values.

```java
int anotherBadFactorial(int x) {
    if(x == 0)
        return 1;
    else
        return x*(x-1)*anotherBadFactorial(x -2);
    // When x is odd, we never reach the base case!!
}
```
Common Errors in Writing Recursive Methods

- Post increment and decrement operators must not be used since the update will not occur until AFTER the method call - infinite recursion.

```java
public static int sumArray (int[] x, int index) {
    if (index == x.length) return 0;
    else
        return x[index] + sumArray (x, index++);
}
```

- Local variables must not be used to accumulate the result of a recursive method. Each recursive call has its own copy of local variables.

```java
public static int sumArray (int[] x, int index) {
    int sum = 0;
    if (index == x.length) return sum;
    else {
        sum += x[index];
        return sumArray(x,index + 1);
    }
}
```
Common Errors in Writing Recursive Methods

- Wrong placement of `return` statement.
- Consider the following method that is supposed to calculate the sum of the first $n$ integers:

```java
public static int sum (int n, int result) {
    if (n >= 0)
        sum(n - 1, n + result);
    return result;
}
```

- When `result` is initialized to 0, the method returns 0 for whatever value of the parameter $n$. The result returned is that of the final `return` statement to be executed. Example: A trace of the call `sum(3, 0)` is:
Common Errors in Writing Recursive Methods

• A correct version of the method is:

```java
public static int sum(int n, int result){
    if (n == 0)
        return result;
    else
        return sum(n-1, n + result);
}
```

• Example: A trace of the call `sum(3, 0)` is:
Common Errors in Writing Recursive Methods

• The use of instance or static variables in recursive methods should be avoided.

• Although it is not an error, it is bad programming practice. These variables may be modified by code outside the method and cause the recursive method to return wrong result.

```java
public class Sum{
    private int sum;

    public int sumArray(int[] x, int index){
        if(index == x.length)
            return sum;
        else {
            sum += x[index];
            return sumArray(x,index + 1);
        }
    }
}
```