Section 3 Recursion

Recursive functions should have a

1) base case(s) – usually a trivial case
2) recursive step – breaks the problem up into smaller problems that are recursive

Summing up number 0 through k.
S(k) = 1 + 2 + 3+ … + k

Could do it iteratively

```
int sum =0;
for(int i=1; i< =k; i++)
{
  sum +=  i;
}
```

or rescursively

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

Examples:
1) Reverse a string recursively

```
   public String reverseString(String word)
   {
       if(word == null || word.equals(“”))
          return “”;
       else
          return reverseString(word.substring(1, word.length())) + word.substring(0,1);
    }
```

2) Find the modulus of a function (the remainder) without using %

```
      public int modulus(int val, int divisor)
      {
         if( val < divisor)
            return val;
         else
            return modulus(val - divisor, divisor);
      }
```

3) What is the result of mystery (2, 25)? mystery(3, 35)? Given positive integers a and b, describe what mystery(a, b) does.

mystery(2,25) = 50
mystery(3,35) = 105
mystery (a,b) multiplies a and b

```
public int mystery(int a, int b)
{
  if (b == 0)
    return 0;
  else if (b % 2 == 0)
     return mystery(a + a, b / 2);
  else
     return mystery(a + a, b / 2) + a;
}
```

4) Find the gcd (greatest common divisor) of 2 numbers

```
public int gcd(int a, int b)
{
  int remainder = a % b;

  if (remainder == 0)
    return b;
  else
    return gcd(b, remainder);
}
```

5) Write print out the binary representation of a given number

```
public void convertBinary( int n )
{
  if(n==0)
    return;
  convertBinary(n/2);
  System.out.println(n % 2);
}
```

6) What is f(0)?

f(0) = 997

```
public static int f(int x)
{
  if( x > 1000)
     return x-4;
```

```
    else
        return f(f(x+5));
  }
```

7) Merge two strings that are ordered alphabetically. The result should also be alphabetical.

```
  public String merge(String first, String second)
  {
    if(first ==null || first.equals(“”))
      return second;
    else if (second == null || second.equals(“”))
      return first;
    else if(first.charAt(0) < second.charAt(0))
      return first.charAt(0) + merge( first.substring(1, first.length()), second);
    else
      return second.charAt(0) + merge(first, second.substring(1, second.length()));
  }
```

8) Given an array of characters and the length of the array, print out all the permutations of the array. (hint you might need a helper method to do this)

This solution is different then what we were developing in section. It fixes the last element of the array instead of the first. It could similarly be done to fix the first letter.

```
  public void permute(char[] a, int length)
  {
     if (length == 1)
    {
        System.out.println(a);
        return;
    }

    for (int i = 0; i < length; i++)
   {
        swap(a, i, length-1);
        permute(a, length-1);
        swap(a, i, length-1);
     }
  }

  public void swap(char[] a, int i, int j)
  {
    char c;
    c = a[i];
    a[i] = a[j];
    a[j] = c;
  }
```