1. Consider the code below which recursively counts and returns the number of “7”’s that appear in a multi-digit number. For example, a call of `countSevens(7564747)` would return 3 since there are 3 7’s in the input number.

```java
public static int countSevens(int num) {
    if (num == 0) {
        return 0;
    } else {
        int digit = num % 10;
        if (digit == 7)
            return 1 + countSevens(num / 10);
        else
            return countSevens(num / 10);
    }
}
```

As we know, a recursive solution to a problem must have 3 things: 1) a base case or stopping point, 2) a recursive call to a smaller problem, and 3) a small bit of work in order to build up a solution to a larger problem using the solution to the smaller problem.

In your own words, describe how the code above implements these three things. It is not enough to restate the code; you must demonstrate understanding of what the code does.

(a) (2 points) What is the stopping point of this code?

**Solution:** We stop when num is 0. Obviously, when num is 0 there are no 7’s in the number, so we can return 0. The code will eventually stop when we make a recursive call with any single-digit number divided by 10 (eg `countSevens(9 / 10)`) which will be the same as calling `countSevens(0)`.

(b) (2 points) What is the smaller problem that we use when making the recursive call? How does this make the problem smaller?

**Solution:** We divide num by 10. This essentially removes a digit from the number each time we make the recursive call. For example, calling —`countSevens(532 / 10)`— is the same thing as calling `countSevens(53)`.

(c) (2 points) What is the “small bit of work” we combine with the solution to the smaller (recursive) problem in order to arrive at the solution to the problem?

**Solution:** We examine the last (right-most) digit using the mod operator. Any multi-digit number mod 10 will yield the last digit only. If this digit is a 7, we add one to the solution to the smaller problem. If it is NOT a seven, we just return the solution to the smaller problem.
2. Consider the `Date.java` file below:

```java
public class Date {
    private int month;
    private int day;
    private int year;

    public Date(int month, int day, int year) {
        //part a
    }
    public Date() {} 

    public String toString() {
        return month + "-" + day + "-" + year;
    }
}
```

(a) (2 points) Complete the constructor for the `Date` class, (labeled “part a” above) by assigning the values of the parameter variables to the corresponding instance variables.

Solution:
```java
public Date(int month, int day, int year) {
    this.month = month;
    this.day = day;
    this.year = year;
}
```

(b) (2 points) What is the output of this code or the error generated by this code fragment?
```java
Date d = new Date();
System.out.println(d);
```

Solution: 0-0-0

3. (5 points) String methods: Give the output of the code below. (You may assume it is part of a class/program which compiles and runs.)

```java
String s1 = "Hello World";
String s2 = "Hello ";
s2 += "World";
System.out.println(s1);
System.out.println(s2);
System.out.println(s1 == s2);
s1.toUpperCase();
s2 = s2.substring(6);
System.out.println(s1);
System.out.println(s2);
```
Solution: Hello World
Hello World
false
Hello World
World