COS126 Solutions to Scientific Computing Questions

1. Imagine you can only store and use integers with 10 digits. What results when you add 9999999999 + 0000000001?

   (0000000000)

2. Using a factorial method which returns N! as an int, you get the following results:

   ```
   12! =  479001600
   13! = 1932053504
   14! = 1278945280
   15! = 2004310016
   16! = 2004189184
   17! = -288522240
   ```

   What happened? Where did things start going wrong?

   (Overflow. Data type int can only store from -2147483648 to +2147483647. 13! is over six billion, so it is too big for an int, the leftmost bits fall off and the 32 bits left in memory are used as the answer.)

3. What will the following java fragment print?

   ```
       double x1 = 0.3;
       double x2 = 0.1 + 0.1 + 0.1;
       StdOut.println(x1 == x2);

       double z1 = 0.5;
       double z2 = 0.1 + 0.1 + 0.1 + 0.1 + 0.1;
       StdOut.println(z1 == z2);
   ```

   (First prints false, second prints true. But the point is that this issue is inherent in floating point representation of number, so don't test float and double for strict equality.)

4. Is the previous result a consequence of Round Off Error or Catastrophic Cancellation?

   (round off error, but this will lead to "What is catastrophic cancellation?")

5. Will the following java fragment print 0.0?

   ```
   System.out.println( (.3 -.1 -.1-.1)*1e15);
   ```

   (No, it prints -0.027755575615628914)

6. Is the previous result a consequence of Round Off Error or Catastrophic Cancellation?

   (Both. The result that (.3-.1-.1-.1) is not 0.0 is a result of round off error. When we multiply that erroneous result by a large number, or repeatedly use it in a calculation, we get catastrophic cancellation - something that should have cancelled out and been zero is now big enough to cause a problem.)

7. Give an example when java will give you NaN.

   (Math.sqrt(-1))

8. Give an example when java will give you Infinity.

   (1.0/0.0)

9. What will the following java fragment print?

   ```
   System.out.println( 1/0 );
   ```

   (Trick question. It won't print any answer. This is integer division and will cause a runtime error: ArithmeticException / 0)

10. What will the following java fragment print?

    ```
    System.out.println( 1000000000000. + .00001);
    ```

    (1.0E12 Another round off error. The lesson here is don't add or subtract very differently sized floating point numbers.)

11. Why is an ill-conditioned problem worse than an unstable algorithm?

    (With an ill-conditioned problem, computing a reasonable answer is doomed from the start. Any slight variation in the starting condition results in a huge deviation from the desired answer. No algorithm can compensate for this.

    With an unstable algorithm, you might be able to rearrange the terms or steps so that you can avoid the problem area and be able to compute a close enough answer, as long as the problem is well-conditioned.)