CSSE 230 In-class exercise Day 1 Names ____________________ ____________________ Section _______

Work with a partner. If an odd number of students are present, one group may have three people.

1. This question deals with the overhead of having a "growable" array in the **`ReadStrings`** program (Weiss, section 2.4.2, also printed on the back of this page). Let N be the total number of Strings that are actually input when the program is run (N is called *itemsRead* in the program).

   Which statement from the code is responsible for most of the "overhead" time associated with growing the array when it is too small to hold the next String? [Note that all of the array growth originates from calls to `resize()` in line 23.] Write the line

   number of that statement here: ________. If you are in doubt, check with someone else before going on.

Let E_N be the number of times (altogether, during **all of the executions** of the *while* loop in `getStrings`) that your selected line of code is executed for a given value of N. Fill in the E_N column of the following table:

N	**E_N**	**Answers for problem 2**
4	0	0
5	0	0
6	5	5
7		
10		35
11	15	
20		
21		
40		
41		

In the worst case, find a formula for E_N as a function of N. This worst case happens when $N=5*2^k + 1$ (Why? Calculate N for k=0, 1, 2, … to find out), so you should only calculate the formula E_N for this case. (Hint: write E_N as a summation involving k. Simplify the sum in terms of k, and then use the formula relating k and N to express it in terms of N. The final expression should be a simple one.)

2. Suppose that in the line 23 call to `resize()`, we replace **`length * 2`** by **`length + 1`**. In the right column of the table, fill in the numbers that would be the answers to the same question (how many times the "largest overhead" statement executes) in this case. You only need to do this for the first 6 rows of the table (up through N=11).

3. Write a formula (a summation in terms of N) for the number of times the same "maximum execution" line of code from the top of this page is executed in the length $\leftarrow$ length+1 case. Then evaluate and simplify to get a simple closed-form expression involving N.

4. Which would you recommend: the *doubling* or the *increase by one* approach? Explain.

```java
import java.util.Scanner;

public class ReadStrings
{
    // Read an unlimited number of String; return a String [ ]
    // The minimal I/O details used here are not important for
    // this example and are discussed in Section 2.6.
    public static String [ ] getStrings( )
    {
        Scanner in = new Scanner( System.in );
        String [ ] array = new String[ 5 ];
        int itemsRead = 0;

        System.out.println( "Enter strings, one per line; " );
        System.out.println( "Terminate with empty line: " );

        while( in.hasNextLine( ) )
        {
            String oneLine = in.nextLine( );
            if( oneLine.equals( "" ) )
                break;
            if( itemsRead == array.length )
                array = resize( array, array.length * 2 );
            array[ itemsRead++ ] = oneLine;
        }

        return resize( array, itemsRead );
    }
    // Resize a String[ ] array; return new array
    public static String [ ] resize( String [ ] array,
                                     int newSize )
    {
        String [ ] original = array;
        int numToCopy = Math.min( original.length, newSize );

        array = new String[ newSize ];
        for( int i = 0; i < numToCopy; i++ )
            array[ i ] = original[ i ];
        return array;
    }

    public static void main( String [ ] args )
    {
        String [ ] array = getStrings( );
        for( int i = 0; i < array.length; i++ )
            System.out.println( array[ i ] );
    }
}
```