Problem 1:

- This code has 2 for loops, 1 nested for loop and a while loop that would run through its entirety. For loops have a linear time complexity so the 2 single *for loops* are linear, O(N). The *nested for loop* is a double loop so the time complexity is quadratic or O(N^2). The *while loop* has a time complexity that is logarithmic, *log(N)*. In the best case each *for loop* would complete the entire conditional statement and the while loop would only run once. In the worst case each for loop would run to its entirety of the *for loops* but the *while loop* would run more than once. And with each time the for loop runs there is a reassignment of the variables.

```
        for (int i = 0; i < arraySize; i++) {
            numbers = String.valueOf(a[i]).length(); // assigr
                if (numbers > max) {
                    max = numbers;
                }
        } // finds the maximum of digits in the entire array

        for (int power = 1; power < max; power++) {
            int p = (int) Math.pow(10, power );

            for(int j = 0; j < arraySize; j++) {
                x[j] = a[j] % p; // sort by the certain place
            }
            for (int k = 0; k < arraySize; k++) {
                int big = a[k];
                int small = x[k];
                int m = k - 1;

                while (m >= 0 && x[m] > small) {
                    a [m+1] = a[m];
                    x[m+1] = x[m];
                    m--;
                }
                a[m+1] = big;
                x[m+1] = small;
            }

            System.out.println("[");
            for (int b = 0; b < arraySize; b++) {
                System.out.println("" + a[b]);
            }
            System.out.println("]");
        }
    }
```

- The bottleneck, shown by the yellow highlighted part, occurs where there is an increment because the frequency for any increment is N^2. There are 5 for loops so the increment happens in 5 different places for each time each of these loops run.
- This is a stable sort because it compares the place of each element which means that each element can go in only one sorted order every time. This order isn't interchangeable since it compares the digits to each other.
- The algorithm is not using constant extra space, shown by the green highlighted region. Since it's a *while loop* and it creates a list of size *m* which is not constant and keeps changing. If the algorithm used constant extra space, O(1), then we would be able to know exactly where an array begins that stores the values for the loops. This way the memory would be constantly stored and finding the index of a certain element would be much easier.