CS 284: Homework Assignment 2
Due: September 29, 11:55pm

1 Assignment Policies

Collaboration Policy. Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions or using generative AI, such as ChatGPT, to generate code.

Under absolutely no circumstances code can be exchanged between students. Excerpts of code presented in class can be used.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

Late Policy. There are a total of 7 points available for this assignment. You will lose .5 points for every two hours late you hand in your assignment (rounded down).

2 Assignment (7 pts)

This assignment asks you to implement a number of methods for a class `Complexity`. These methods should be implemented using for loops, as seen in class (except for the extra-credit one). In addition, each of these methods should print out the value of an accumulator that counts the number of "operations" performed. The notion of "operation" should be taken loosely; the idea is that if you are requested to implement a method of time complexity $\mathcal{O}(n)$, then it should print out values from 1 to n (or close enough). For example, the following code implements a method that has time complexity $\mathcal{O}(n)$:

```
public static void method0(int n) {
   int counter=0;
   for (i=0; i<n; i++) {
      System.out.println("Operation "+counter);
      counter++;
   }
}
```

The methods you should implement are:

1. `public static void method1(int n)`: a method that has time complexity $\mathcal{O}(n^2)$. (1 pt)
2. `public static void method2(int n)`: a method that has time complexity $\mathcal{O}(n^3)$. (1 pt)
3. `public static void method3(int n)`: a method that has time complexity $\mathcal{O}(\log n)$. (1 pt)
4. Consider the following method where `a` is assumes sorted in ascending order and `x` is a number larger than any number in `a`:

```
public static boolean bSearch(int[] a, int x) {
  int end = a.length - 1;
  int start = 0;
  while (start <= end) {
    int mid = (end - start) / 2 + start;
    if (a[mid] == x) return true;
    else if (a[mid] > x) end = mid - 1;
    else start = mid + 1;
  }
  return false;
}
```

   Suppose the length of `a` is 32. Fill in the table below with the values that `end` and `start` take in every iteration before the while loop exits. The first row has been completed for you. Note: you must add this table as a Java comment in your source code. (.5 pt)

Iteration	`start`	`end`
1	0	31

   Suppose the length of `a` is 64. Fill in the table below with the values that `end` and `start` take in every iteration before the while loop exits. The first row has been completed for you. (.5 pt)

Iteration	`start`	`end`
1	0	63

5. What is the relation between the size `n` of `a` and the number of iterations? (.5 pt)
6. What is the time complexity of `bSearch`? (.5 pt)
7. `public static void method4(int n)`: a method that has time complexity $\mathcal{O}(n \log n)$. (1 pt)
8. `public static void method5(int n)`: a method that has time complexity $\mathcal{O}(\log \log n)$. (1 pt)
9. (Bonus) `public static int method6(int n)`: a method that has time complexity $\mathcal{O}(2^n)$. For this method you should consider using recursion.

   Please note that solutions of the form

```
public static void method6(int n) {
   counter=0;
   for (i=0; i<Math.log(n); i++) {
      System.out.println("Operation "+counter);
      counter++;
   }
}
```

or similar will not receive credit. This applies for all methods including the bonus question.

3 Submission instructions

Submit a single file named `Complexity.java` through Canvas. No report is required.