CSCI 151 Fall 2024
Problem Set #1
Due: October 13, 2024 (11:59 PM)

Name:

Honor Code:

This problem set provides practice identifying the worst- and best-case time complexity of algorithms. Please answer the following questions and submit your solutions through Gradescope. You are welcome to edit this PDF with your answers as a digital file. You could also print out a copy, hand write your answers, then either scan your answers or take pictures with your phone. Either approach will result in a solution that can be uploaded to Gradescope.

1 Identifying Complexity

For the following questions, you should answer by writing down:

1. The **worst-case time complexity** of the algorithm (in $\mathcal{O}$ notation)
2. The **best-case time complexity** of the algorithm (in Ω notation)
3. A **1-2 sentence explanation** of why those complexities are correct. Make sure to describe the meaning of any variables used in your answer, e.g., n = list.size()

1.1 Question 1

```
1  /**
2   * Adds up all of the even numbers from 2 to n.
3   *
4   * @param n The number to add up to
5   * @return 2 + 4 + ... + n
6   */
7  public int addEvens(int n) {
8     int sum = 0;
9     int nextEven = 2;
10
11    while (nextEven <= n) {
12       sum = sum + nextEven;
13       nextEven = nextEven + 2;
14    }
15
16    return sum;
17 }
```

1.2 Question 2

```
1  /**
2   * Calculates the difference of the max and min of an array of integers.
3   *
4   * @param array An array of integers
5   */
6  public int diffMaxMin(int[] array) {
7     // find the maximum value in array
8     int max = Integer.MIN_VALUE;
9     for (int i = 0; i < array.length; i++) {
10       int val = array[i];
11       if (val > max) {
12          max = val;
13       }
14    }
15
16    // find the minimum value in array
17    int min = Integer.MAX_VALUE;
18    for (int i = 0; i < array.length; i++) {
19       int val = array[i];
20       if (val < min) {
21          min = val;
22       }
23    }
24
25    // calculate the difference
26    return max - min;
27 }
```

1.3 Question 3

```
1  /**
2   * Performs BubbleSort on a given ArrayList of Doubles.
3   *
4   * @param list The ArrayList to sort
5   */
6  public void bubbleSort(ArrayList<Double> list) {
7     for (int i = 0; i < list.size() - 1; i++) {
8        int swaps = 0;
9
10       // loop over all pairs of neighbors
11       for (int j = 0; j < list.size() - i - 1; j++) {
12          // swap a pair of neighbors if needed
13          if (list.get(j) > list.get(j+1)) {
14             int temp = list.get(j+1);
15             list.set(j+1, list.get(j));
16             list.set(j, temp);
17             swaps++;
18          }
19       }
20
21       // did we make any swaps on this pass?
22       if (swaps == 0) {
23          return;
24       }
25    }
26 }
```

1.4 Question 4

```
1  /**
2   * Checks whether a given ArrayList of Doubles is sorted in ascending order.
3   *
4   * @param list The ArrayList to check
5   * @return true if list is sorted in ascending order, else false
6   */
7  public boolean checkSorted(ArrayList<Double> list) {
8     for (int i = 0; i < list.size() - 1; i++) {
9        int prev = list.get(i);
10       int next = list.get(i+1);
11
12       if (prev > next) {
13          return false;
14       }
15    }
16
17    return true;
18 }
```

1.5 Question 5

```
1  /**
2   * Draws a right triangle (aligned to the right).
3   *
4   * @param rows The number of rows in the triangle
5   */
6  public void drawRightTriangle(int rows) {
7     for (int r = 1; r <= rows; r++) {
8        int spaces = rows - r;
9        int stars = r;
10
11       // draw the leading spaces
12       for (int sp = 0; sp < spaces; sp++) {
13          System.out.print("␣");
14       }
15
16       // draw the stars
17       for (int st = 0; st < stars; st++) {
18          System.out.println("*");
19       }
20
21       // add a blank line
22       System.out.println();
23    }
24 }
```

Note: assume that calling System.out.print() and System.out.println() can be performed in constant time $\mathcal{O}(1)$.

1.6 Question 6

```
1  /**
2   * Converts an image to grayscale.
3   *
4   * Here, the image is represented by a 3D array of pixel values
5   * with dimensions width * height * 3 colors (red, green, blue).
6   *
7   * @param The array of pixel values making up the image
8   */
9  public void grayscale(int[][][] image) {
10    int width = image.length;
11    int height = image[0].length;
12
13    // loop over the width of the image
14    for (int x = 0; x < width; x++) {
15
16       // loop over the height of the image
17       for (int y = 0; y < height; y++) {
18
19          // calculate the average color value at pixel (x, y)
20          int avg = 0;
21          for (int c = 0; c < 3; c++) {
22             avg = avg + image[x][y][c];
23          }
24          avg = avg / 3;
25
26          // change all 3 color values to avg
27          for (int c = 0; c < 3; c++) {
28             image[x][y][c] = avg;
29          }
30       }
31    }
32 }
```

1.7 Question 7

```
1  /**
2   * Finds the average of all numbers is a nested List of
3   * Lists of Doubles.
4   *
5   * @param allNumbers The nested List of Lists of Doubles
6   * @return The average value of allNumbers
7   */
8  public double findAverageAll(List<List<Double>> allNumbers) {
9     double sum = 0.0;
10    int n = allNumbers.size();
11
12    // average each inner list
13    for (int i = 0; i < n; i++) {
14       List<Double> innerList = allNumbers.get(i);
15       double avg = average(innerList);
16       sum = sum + avg;
17    }
18
19    // return the average of averages
20    return sum / n;
21 }
22
23 /**
24  * Calculates the average of a List of Doubles.
25  *
26  * @param nums A List of Doubles
27  * @return The average of the values in nums
28  */
29 public double average(List<Double> nums) {
30    double sum = 0.0;
31    int n = nums.size();
32
33    // add up all the numbers in nums
34    for (int i = 0; i < n; i++) {
35       double val = nums.get(i);
36       sum = sum + val;
37    }
38
39    // return the average
40    return sum / n;
41 }
```

Question 7 (continued).

For this question, you can assume that there are n total inner lists in $allNumbers$, and each inner list contains exactly n numbers.

Part 1. Assume that the Lists used in the problem are **ArrayList**s. What are the worst- and best-case time complexities of the **findAverageAll** function? Explain your answer.

Part 2. Assume that the Lists used in the problem are **LinkedList**s. What are the worst- and best-case time complexities of the **findAverageAll** function? Explain your answer.

2 Designing Algorithms

For the remaining questions, you should answer by:

1. Writing an **algorithm (in pseudocode or Java)** that accomplishes the requested task
2. Stating the **worst-case time complexity** of the algorithm (in $\mathcal{O}$ notation)
3. Stating the **best-case time complexity** of the algorithm (in Ω notation)
4. Providing a **1-2 sentence explanation** of why those complexities are correct. Make sure to describe the meaning of any variables used in your answer, e.g., n = list.size()

2.1 Question 8

Find and return the middle value of an array of Strings. The middle index should be rounded down to satisfy either even or odd lengths of arrays. For example, calling the function with the input array [”Have”, ”a”, ”great”, ”day”] should return ”a”. Likewise, calling the function with the input array [”Today”, ”is”, ”Sunday”] should return ”is”.

2.2 Question 9

For a provided ArrayList of Integers, find and return the starting index of the pair of neighboring values that are closest together (i.e., have a difference closest to 0). For example, for the input $[1, 3, 5, 7, 3, 4, 9]$, the algorithm should return 4 because the closest pair of neighboring values is $(3, 4)$ (having a difference of only 1), and the first number in that pair (3) is located at index 4 in the given list.

2.3 Question 10

Imagine you have a (singly) LinkedList of Integers that are in sorted order. Provide a method that returns true if there are any repeated numbers in the LinkedList and false if there are no repeated numbers. For example, if the LinkedList is $[0, 1, 2, 3, 4, 5]$, then the method should return false since no numbers in the list are repeated. On the other hand, if the list is $[0, 100, 100, 200, 300, 400]$, then the method should return true since 100 is repeated.

Your algorithm should use the instance variables of the LinkedList class ($head/firstNode$, $tail/lastNode$, and $size$) and the instance variables of the Node class ($item$, $next$). You should **not** call the *get* method of the LinkedList class.

Your answer should also have the fastest worst-case time complexity possible for this problem.