Name: ______________________________
Login: ____________

Honor code acknowledgment (signature) ________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>value</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>22 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 2</td>
<td>12 pts.</td>
<td></td>
</tr>
<tr>
<td>Problem 3</td>
<td>20 pts.</td>
<td></td>
</tr>
<tr>
<td>TOTAL:</td>
<td>54 pts.</td>
<td></td>
</tr>
</tbody>
</table>
PROBLEM 1: (Being Regular with Regex (22 points))

Part A (4 points)
We used a program in class that processed regular expressions and listed English words from a dictionary that matched the regular expression.

The regular expression \( \ldots \)\1 generates exactly one match, the word \textit{beriberi}. A small modification, using the regular expression \( \ldots .\)\1 generates two matches: \textit{bandstands} and \textit{hodgepodge}. Explain why \textit{beriberi} does not match the second regular expression and why the second expression has the two matches it does.

Part B (4 points)
If the regular expression is changed to \( \ldots .\ast \)\1 then 13 matches are found as shown below.

\begin{itemize}
  \item atherosclerosis
  \item bandstands
  \item beriberi
  \item hodgepodge
  \item kinnikinnic
  \item knickerbocker
  \item knickerbockers
  \item lightweight
  \item misunderstander
  \item misunderstanders
  \item nationalization
  \item rationalization
  \item rationalizations
\end{itemize}

Explain broadly why there are more matches, including all three described in part A. Explain specifically why \textit{atherosclerosis} matches. Finally, circle the five out of the thirteen that match \( \ldots .\ast \)\1 (note: one extra dot in the parentheses).
Part C (2 points)
The regular expression `sp[a1]s` generates twelve matches as follows. Explain why `despise` and `spasm` match this regular expression.

despise
despised
despises
despising
dispamionate
spasm
spastic	
tresspass
tresspassed
tresspasser
tresspassers
tresspasses

Part D (4 points)
This regular expression `^(.{2,4})\1$` generates a list of 10 words as follows. Based on your knowledge of regular expressions and your ability to analyze data, provide an explanation of how the regular expression works in generating this list. Note that we did not discuss the `{2,4}` part of the expression, you have to offer an explanation of this based on what you know and what the data show.

beriberi
booboo
coco
dada
isis
mama
mimi
murmur
papa
toto

Part E (4 points)
To find seven-letter palindromes, someone enters this regular expression `(.)(.)(.)\3\2\1$`. This generates seven matches, but only the last two are seven letter palindromes. Explain why `precipice` matches this regular expression and how to fix the regex so that only palindromes match it.

analyticity
interpret
precipice
recognizing
reinterpre
reviver
rotator
Part F (4 points)

Recall that a start codon is \textit{ATG} and that a stop codon is any of the three \textit{TAG}, \textit{TGA}, or \textit{TAA}. The Java code below is an attempt to find start/stop codon pairs in a strand of DNA. A run is shown for the strand indicated in the program.

Here’s the code

```java
import java.util.regex.*;

public class Restrict {
    static String dna = "ATGxxxTAG...ATGyyyyzzzTGA...ATGwwwwTAA...ATGaaaaaaa";
    // 012345678901234567890123456789012345678901234567890123

    public static void main(String[] args) {
        Pattern starter = Pattern.compile("(ATG).*(TAG|TGA|TAA)");
        Matcher match = starter.matcher(dna);
        while (match.find()) {
            System.out.println(match.start() + " " + match.end());
            System.out.println(match.group());
            System.out.println("--");
        }
    }
}
```

When this code is run it generates the three matches shown below on the left. However, when the regular expression is changed so that the question mark is removed, that is it becomes "(ATG).*(TAG|TGA|TAA)"}, then the output generated shows only one match as displayed on the right below (this is a separate run of the code).

<table>
<thead>
<tr>
<th>Run/executed with ? in regex</th>
<th>Run/executed without ? in regex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 9  ATGxxxTAG</td>
<td>0 38  ATGxxxTAG...ATGyyyyzzzTGA...ATGwwwwTAA</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>12 25 ATGyyyyzzzTGA</td>
<td>---</td>
</tr>
<tr>
<td>28 38 ATGwwwwTAA</td>
<td>---</td>
</tr>
</tbody>
</table>

Provide an explanation of the different behavior based on your knowledge of regular expressions and your ability to reason. We did not discuss the question mark as part of regular expressions. When it’s used, the regular expression matches are called \textit{reluctant}. When the question mark is not used, a match is called \textit{greedy}. These terms may help in your explanation.
The code below is similar to code we’ve studied in class. This code finds all 6mers (sequences of six nucleotides in a dna strand) in a strand and stores unique 6mers in the ArrayList `allmers`. The code below uses the ArrayList `indexOf` method, which is similar to the String method of the same name (but the method below searches in an ArrayList for a specific value).

```java
ArrayList<String> allmers = new ArrayList<String>();
for(int k=0; k < dna.length(); k+=6){
    String sub = dna.substring(k,k+6);
    if (allmers.indexOf(sub) == -1){
        allmers.add(sub);
    }
}
```

If a dna strand consists of exactly the $4^6 = 4096$ different 6mers the code above takes 0.213 seconds to run on the Linux computer on my desk. Assume that the time for forming substrings is negligible, so all the time the computer uses in executing the code above is in searching the ArrayList `allmers` to see if a string is already stored before adding it.

**Part A (2 points)**

Why is an ArrayList used in the code above rather than an array?

**Part B (10 points)**

Based on the assumption that all the time for the code is based on searching the ArrayList to see if a String is stored in it, either calculate how long you think the code will take to run when finding all 8mers in a strand of dna that contains exactly the $4^8 = 65,536$ different 8mers or explain how you would calculate how long it takes. In either case you must clearly explain your reasoning and equations that you use. You don’t need to show a single number as an answer, you can describe the reasoning and equations you would use to answer this question.
PROBLEM 3: (The Vicissitudes of Life (20 points))

In some competitions such as gymnastics and figure skating several judges score a competitor’s effort. The score assigned is based on the average of all the judge’s scores after removing the high and low score from those from which the average is calculated.

The method `computeScore` computes a total score by determining the average judge-score after removing the high and low score and then multiplying this average by a difficulty factor. You’re given two implementations of this method below and asked to comment on them.

For example, if the scores 5.0, 5.5, 4.5, 5.0, and 5.5 are stored in the array `scores` as indicated:

-5.0
-5.5
-4.5
-5.0
-5.5

for a dive with difficulty factor 2.3, then the expression `calculateScore(scores, 2.3)` should evaluate to 5.1667*2.3 = 11.88. The average of 5.0, 5.5, and 5.0 is 5.1667—note that one high score of 5.5 and the low score of 4.5 do not contribute to the average.

You’re given two implementations of `computeScore` and asked to comment about features they have. Given identical arrays and difficulty factor values, each implementation returns the same results, i.e., the only differences in the methods are in the style/code, not in whether the methods are correct. The implementations are on the next page.

**Part A (4 points)**

Explain, briefly, why and how the methods fail when passed an array of two elements.

**Part B (4 points)**

Briefly, why is the initialization `Double.MAX_VALUE` used for `low` in implementation I?

**Part C (4 points)**

Briefly, why is the array sorted in Implementation II?
Implementation I

/**
 * Calculate score by averaging data in scores without
 * high and low and multiplying average by difficulty factor
 * @param scores represents judge-scores
 * @param df is the difficulty factor
 * @return the calculated score
 */
public double computeScore(int[] scores, double df){
    double low = Double.MAX_VALUE;
    double high = Double.MIN_VALUE;
    double total = 0.0;
    for(int k=0; k < scores.length; k++){
        total += scores[k];
        if (scores[k] < low) low = scores[k];
        if (scores[k] > high) high = scores[k];
    }
    int count = scores.length-2;
    return df*(total - low - high)/count;
}

Implementation II

/**
 * Calculate score by averaging data in scores without
 * high and low and multiplying average by difficulty factor
 * @param scores represents judge-scores
 * @param df is the difficulty factor
 * @return the calculated score
 */
public double computeScore(int[] scores, double df){
    Arrays.sort(scores);
    double total = 0.0;
    for(int k=0; k < scores.length; k++){
        total += scores[k];
    }
    int count = scores.length-2;
    return df*(total - scores[0] - scores[scores.length-1])/count;
}
Part D (8 points)

You are to write a new implementation of `computeScore` in which *all scores* equal to the high score and *all scores* equal to the low score are thrown out, i.e., they do not contribute to the average which is then multiplied by the difficulty factor. For example, for the scores (5, 3.5, 4, 3, 4, 5, 2, 3.5, 5, 2) the average would be $(3.5 + 4 + 3 + 4 + 3.5) = 18/5 = 3.6$ since each score of 5 (the high score) and each score of 2 (the low score) do not contribute to the final average.

```java
public double computeScore(int[] scores, double df){
    // Your implementation here
}
```