Mastery II — Data Struct. & Algo. (T. III/17–18)

Name: _____

ID: _____

Directions:

- You have 170 minutes (i.e., 2 hour and 50 minutes) to complete the following *four* problems. The maximum possible points is 35, but we'll grade out of ____ points. Anything above that is extra credit.
- No collaboration of any kind whatsoever is permitted during the exam.
- **WHAT IS PERMITTED:**
  - Reading the official Java documentation
  - Accessing Canvas for submission.
- **WHAT IS *NOT* PERMITTED:**
  - Browsing (online) tutorials or reading stack overflow threads.
  - Accessing previously-written code on your own machine.
  - Communicating with other people or using any other aid.
- For each problem, the entirety of you solution must live in one file, named according to the instructions in this handout. When grading a problem, the script will only compile that one file for the problem.
- We're providing a starter package, which you can download at

  `https://cs.muic.mahidol.ac.th/courses/ds/puyo-mania.zip`

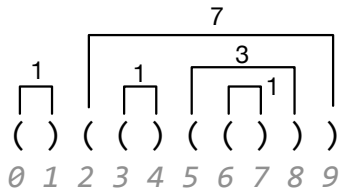
  The password is "tetris".
  When you unpack the package, you'll see one file for each problem.
- To submit your work, zip all your Java files as one zip file called `mastery2.zip` and upload it to Canvas.

1	2	3	4 (EC)	Σ

Problem 1: Parenthesis Pairs (10 points)

The language of parentheses (aka. the paren language) has only two characters in the alphabet: `(` and `)`. It's quite straightforward to recognize well-formed parenthesis expressions. To give some examples, we know `(())()` is well-formed, whereas `())(` is not. As another example, `()(()(()))` is well-formed.

In this problem, you will be given a parenthesis expression. It is guaranteed to be well-formed. This means, every paren has a matching pair—an open paren is matched with a close paren and a close paren is matched with an open one. Our goal is to find the matching pair for every paren in the expression.



The figure here shows an example of a parenthesis expression annotated with lines denoting the matching pairs. For example, the open paren at index 2 is matched with the close paren at index 9. Another pair is 5 and 8. Also, next each line is a number showing how far apart the matching pair is.

Inside a class named `ParenPairs`, you will implement a function

```
public static int[] match(String ex)
```

that takes a parenthesis expression and returns an **int** array of the same length as `ex`. The array the function returns satisfies the following: If `d = match(ex)`, then:

- If `ex[i]` is an open paren, then $d[i]$ contains the index of its matching close paren.
- If `ex[i]` is a close paren, then $d[i]$ is a value such that $i + d[i]$ is the index of its matching open paren. In this case, $d[i]$ will always be negative.

Examples:

- `match("()(()(()))")` should return `[1, -1, 9, 4, -1, 8, 7, -1, -3, -7]`. Explanation: The matching pair of the paren at index 0 is at index 1. The matching pair of the paren at index 2 is at index 9. Also, the matching pair of the paren at index 8 is 3 position to the left.
- `match("(()(()))")` should return `[9, 2, -1, 4, -1, 8, 7, -1, -3, -9]`

Performance Expectation: The largest test case we'll use contains up to 500,000 parens. For every test case, your code should finish within 1 second to receive full credit. You should aim for an $O(n)$-time solution. Partial credit will be given to solutions that correctly solve the problem for n up to 5,000.

Problem 2: Lost Items (10 points)

Dr. Piti keeps a rare collection of numbers. In this collection, a number may be repeated multiple times. He treasures this collection so much that he maintains two identical copies of the collection, a and b (so they are backups of each other).

One day, he brought collection a to a Deadly Math lecture, and some numbers were stolen. Now the collection is rather large, so he asked you to help him identify the missing numbers.

Inside `LostItems.java`, you will write a function

```
public static int[] lostItems(int[] a, int b[])
```

that takes in both a and b, each an array of **int**s, and returns an array of **int**s containing all the lost numbers. He has made the following requests:

- From his perspective, a number x is lost if the number of times x appears in a is less than the number of times it appears in b. For example, if 203 appears 3 times in b but only once in a, the number 203 is lost—two copies have disappeared.
- Each lost number is reported in the output only once, even if multiple copies are lost.
- The output array must be ordered from small to large.

Promises, Constraints, and Grading

- $1 \leqslant$ `a.length` $\leqslant$ `b.length` $\leqslant 100,000$.
- We promise that every item that appears in a also appears in b.
- Each `b[i]` satisfies $0 \leqslant$ `b[i]` $\leqslant 500,000$.
- Your solution must finish without 2 seconds per test. Let n be the length of b. The desired solution must run in $O(n \log n)$ time or faster.
- Partial credit will be given to $O(n^2)$ solutions.

Example:

```
a = {203, 204, 205, 206, 207, 208, 203, 204, 205, 206}
b = {203, 204, 204, 205, 206, 207, 205, 208, 203, 206, 205, 206, 204}
lostItems(a, b) should return {204, 205, 206}.
```

Problem 3: Game of k Stacks (10 points)

(Inspired by the Game of Two Stacks problem from your assignment.)

In return for Ply's special present, Gift is inviting Ply to a specially-designed game. Gift has neatly arranged k stacks $S_1, S_2, \ldots, S_k$, where each S_i is a stack whose values are sorted from small (top) to large (bottom). She challenges Ply to play the following game: At the beginning, Ply is given a number x.

- In each move, Ply can remove one integer from the top of one of the stacks.
- Gift keeps a running sum of the integers Ply's removed from the stacks. Ply looses if at any point, this running sum becomes greater ($>$) than a value x given at the beginning.
- Ply's *final score* is the total number of integers he manages to remove. Ply's goal, of course, is to maximize the final score.

Your Task: Inside a class named `KStacks`, you will implement a function

```
public static int maximizeScore(List<Stack<Integer>> S, int x)
```

that takes as input (i) a list of integer stacks and (ii) an integer x, and returns the largest final score Ply can obtain from this input.

Sample Input: Suppose $x = 9$ and the input stacks are:

```
Stack 1: 6, 3, 1 (with 1 being the top)
Stack 2: 9, 5, 2, 1 (with 1 being the top)
Stack 3: 4, 1 (with 1 being the top)
```

The expected output is 5, achieved by popping Stack 1 twice, Stack 2 twice, and Stack 3 once.

Constraints & Grading:

- There will be at least 1 stack and $x \geqslant 0$. We guarantee that the sum of all the numbers in every stack in S combined will fit in an **int**. A number may be repeated multiple times.
- Your solution must finish within 3 seconds per test. The desired solution must run in $O(N \log k)$ time or faster, where N is the combined length of all the input stacks and k is the number of stacks. All test cases have $N \leqslant 5,000,000$ and $k \leqslant 500$.
- If your solution runs slower than that, you will receive some partial credit.

Problem 4: Nesting Cups (5 points)

You are writing a program to put together cups of different sizes. The cups have been measured using several sensors that can accurately determine the radius and color of a cup. However, there is a glitch in how the measurement is reported:

- if the result of the color arrives after the radius, the reported radius was doubled (i.e., 2x the true radius).
- otherwise, the reported values are accurate.

This means, for instance, a red cup with a radius of 5 units will either turn up "red 5" or "10 red".

Inside the `NestingCups` class, you'll write

```
public static String[] orderNestingCups(String[] measurements)
```

meeting the following spec: Given a list of reporting strings (such as above), each describing a different cup, your function will report the colors in order of the (true) radii of the cups, from the smallest to the largest.

You should know that

- The true radius of each cup is an integer, and the radius, as reported, will always be an integer between 1 and 20,000.
- A color is a single, non-empty word consisting of only lowercase letters in the English alphabet (i.e., `a-z`).
- The true radii of the cups are all unique.
- The color component and the radius component are separated by a single space.
- The number of cups n is at most 5,000.

Example: `orderNestingCups(["red 10", "10 blue", "green 7"])` should return `["blue", "green", "red"]`.

(*Hint:* Want to know if a character is a digit (0 - 9) or a letter? Check out `Character.is_____`. Use autocomplete in your IDE to find out.)