Mastery II — Data Struct. & Algo. (T. III/18–19)

Name: ____________________________

ID: ______________________________

Directions:

- You have 170 minutes (i.e., 2 hour and 50 minutes) to complete the following examination.
- There are 4 problems. The maximum possible score is 35. We will grade you out of $T \leqslant 30$, where T is yet to be decided. Anything above T is extra credit. You should think of this as *three* real problems plus *one* 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 person 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. **Importantly:** your implementation must not be part of a package.
- We're providing a starter package, which you can download at

  `https://cs.muic.mahidol.ac.th/courses/ds/yoyo.zip`

  The password is "grape".
  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.

Problem 1: Count: One, Two, Three (10 points)

Nonny is given a sorted array of integers. This array can be very large. She is tasked to determine the count of an item in this collection. To help her, you'll implement a fast algorithm that takes **a sorted array** `xs` and a number `k`, and returns the number of times that `k` appears in `xs`. In particular, inside the class `Count`, you will implement a method

```
public static int count(int[] xs, int k)
```

that meets the above specification. For example, if `xs` $= [1, 20, 34, 34, 34, 34, 47]$ and `k` $= 34$, then `count(xs, k)` should return the count of 34 in `xs`, which is 4. For the same `xs` but with `k` $= 11$, `count(xs, k)` should return 0 because 11 doesn't appear in `xs` at all.

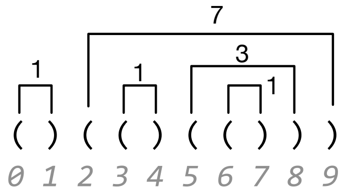
Performance Expectation: The largest test case we'll use contains up to 10,000,000 numbers. You should aim for an $O(\log n)$-time solution. Partial credit will be given to correct solutions that run in $O(n)$ time.

Hint: `xs` is sorted. Can you determine the first index of k quickly? How about the last index of k?

Problem 2: Parenthesis Matching (10 points)

The language of parentheses (aka. the paren language) has only two characters in the alphabet: `(` and `)`. You learn early on 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 to each line is a number showing how far apart the matching pair is.

Inside a class named `ParenMatcher`, you will implement a function **public static int**`[] match(String ex)` that takes a parenthesis expression string and returns an **int** array of the same length as `ex` with the following property: If `d = match(ex)`, then the paren at index `i + d[i]` is the matching pair for the paren at index `i`. In words, `d[i]` is how far we'll walk from position `i` to find its matching pair with a positive value denoting walking to the right and a negative value denoting walking to the left.

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

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 10,000.

Problem 3: Game of k Stacks (10 points)

Gift has neatly arranged k stacks $S_1, S_2, \ldots, S_k$. Each S_i is a stack whose values are sorted from small (top) to large (bottom). She challenges K2 to play the following game: At the beginning, K2 is given a number x.

- In each move, K2 can remove one integer from the top of one of the stacks.
- Gift keeps a running sum of the integers K2 removed from the stacks. K2 looses if at any point, this running sum becomes greater ($>$) than a value x given at the beginning.
- K2's *final score* is the total number of integers he manages to remove.
- His 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 K2 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 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: Manhattan Distance (5 points)

A robot can be instructed to walk in one of the following directions: N, S, E, and W. Each instruction causes the robot to move one step in the specified direction.

Inside Manhattan.java, you will implement a method

```
public static int distanceFromStart(String moves)
```

that takes a string of commands (i.e., consisting of only N, S, E, and W) and computes how far the robot is from the starting point. The robot is initially at coordinate $(0,0)$. If it ends up at coordinate (x,y), your function will report abs$(x)+$ abs(y), where abs is the absolute function. This distance is known Manhattan distance.

Example: distanceFromStart("NNSEENWWWWN")==5 as the robot will end up 3 units north and 2 units west of where it started.

Promises, Constraints, and Grading

- The input string will be at most $1,000,000$ characters long.
- Your code must run within 0.5 seconds.