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

Name: _____

ID: _____

Directions:

- You have 110 minutes (i.e., 1 hour and 50 minutes) to complete the following *three* problems. There is one extra credit problem at the end.
- 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.
- We're providing a starter package, which you can download at

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

  The password is "bezoar".
  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 `mastery1.zip` and upload it to Canvas.

Problem 1: Closest Squares (10 points)

You are given an array `a[]` of arbitrarily-ordered integers. It is guaranteed that the integers are distinct. Your task is to find the smallest absolute difference between the squares of two different array elements. Mathematically, you're to find the smallest $|(a[i])^2 - (a[j])^2|$ with $i \neq j$.

Inside `ClosestSquares.java`, you will implement a function

```
public static long closestSquares(int[] a)
```

that takes in an array $a[]$ of integers and returns an **long** representing the smallest absolute difference between the squares of any two different array elements.

Sample Input:

- `closestSquares([3, 5, 2, 7, 1])` should return 3. This is attained by $2^2 - 1^2 = 4 - 1 = 3$.
- `closestSquares([5, 9, 1, 11, 2, -9])` should return 0, obtained from $(9)^2 - (-9)^2 = 0$.
- `closestSquares([-20,-3916237, -357920, -3620601, 7374819, -7330761, 30, 6246457, -6461594, 266854])` should return 500, obtained from $(30)^2 - (-20)^2 = 500$.

Constraints & Grading:

- The input array contains at least 2 elements and at most 200,000 elements.
- Each `a[i]` satisfies $-10^7 \leqslant$ `a[i]` $\leqslant 10^7$. This means $(a[i])^2$ **might be too large to keep in an int**. Consider using the **long** data type.
- Your solution must finish within 1 second per test. The desired solution must run in $O(n \log n)$ time or faster.
- If your solution runs in n^2 time, you will receive some partial credit.

Problem 2: Recursively Palindromic, Revisited (10 points)

Remember the recursively palindromic problem from Assignment 2? Here's a twist on that problem.

Let $N > 0$ be an integer. We say that a list X of positive integers is a *partition* of N if the elements of X add up to exactly N. For example, each of $[1,2,4]$ and $[2,3,2]$ is a partition of 7.

As you might know already, a list is *palindromic* if it reads the same forward and backward. Of the above example partitions, $[1,2,4]$ is not palindromic, but $[2,3,2]$ is palindromic. What's more, we know that if X is palindromic, then the *first half* (precisely the first `X.length/2` numbers) is the reverse of the last half (precisely the last `X.length/2` numbers).

In this task, we're interested in partitions that are palindromic recursively. A partition is *recursively palindromic* if it is palindromic itself and its first half is recursively palindromic or empty. For example, there are 6 recursively palindromic partitions of 7:

```
[7], [1,5,1], [2,3,2], [1,1,3,1,1], [3,1,3], [1,1,1,1,1,1,1]
```

The twist is that this time *we're only interested in recursively palindromic partitions where none of the numbers are* 1. As an example, out of the partitions of 7 above, we are interested in:

[7], ~~[1,5,1]~~, [2,3,2], ~~[1,1,3,1,1]~~, ~~[3,1,3]~~, ~~[1,1,1,1,1,1,1]~~

Your Task: Inside `RPal.java`, you will implement a function

```
public static int numberOfProperRPal(int n)
```

that takes a number N and returns the number of palindromic partitions of N that contain no 1 in the partition.

Let's look at a few examples:

- `numberOfProperRPal(6)` should return 3.
- `numberOfProperRPal(7)` should return 2.
- `numberOfProperRPal(14)` should return 9.

Constraints & Grading:

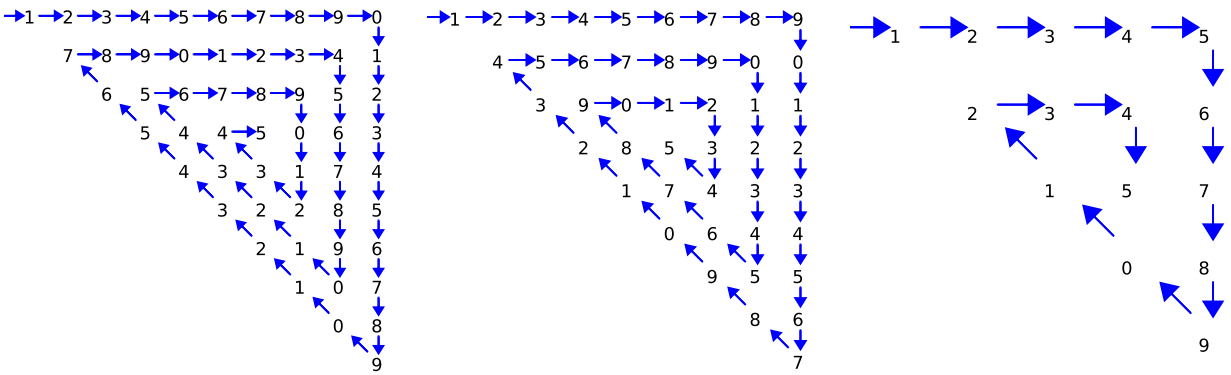
- The input value N is such that $1 \leqslant N \leqslant 256$.
- On this range of input, your solution must finish within 1 second per test.
- There will be partial credits for slower solutions.

Problem 3: Ply's Gadget (10 points)

Ply just made a special present for Gift: a collection of triangular-shaped gadgets filled with numbers.

A size-N gadget is an N-by-N array where only the upper triangular area is used, so it has the shape of a right triangle, whose legs have length N. Example gadgets of sizes $N = 10$, $N = 9$, and $N = 5$ are shown below.

In such an array, the columns are numbered 0 through $N - 1$ (from left to right), and the rows are numbered 0 through $N - 1$ (from top to bottom). The numbers in the right triangle are filled counterclockwise using Ply's signature algorithm. Starting from the horizontal leg on row 0, the algorithm continues down the vertical leg and moves up diagonally to the left to then begin row 1, repeatedly filling the triangle in this circular fashion until the area is full. To fill the area, the algorithm starts counting from 1. When the number reaches 9, the next number is reset to 0. (It helps to look at the examples below.) As a result of this process, all the numbers are between 0 and 9 (inclusive).



Example: Ply's gadgets with $N = 10$, $N = 9$, and $N = 5$ (left to right).

Ply doesn't just want to give Gift a present; he has a challenge for her. Given a size N, a row parameter `row`, and a column parameter `col`, determine the number that appears there in the gadget of size N.

Your Task: Inside `PlysGadget.java`, you'll implement a static method

```
public static int getPlysNumber(int N, int row, int col)
```

that takes as input three parameters `N`, `row`, and `col` and returns an integer equal to the number appearing in Ply's size-N gadget on row `row` and column `col`.

Sample Input/Output:

- `getPlysNumber(10, 1, 2)== 8`
- `getPlysNumber(12, 7, 8)== 2`
- `getPlysNumber(1000000, 8775, 57412)== 3`

Constraints & Grading:

- $1 \leqslant N \leqslant 1,000,000$ and it is guaranteed that `row` and `col` are a location inside the upper triangular part with respect to the given gadget size.
- The desired solution must run in $O(N)$ time and uses no more than $O(N)$ space. Although faster solutions exist, you can already earn a perfect score with an $O(N)$ solution.
- Slower solutions (e.g., $O(N^2)$) will receive partial credits. If your function runs within 1 second for $N \leqslant 1000$, you'll get at least 6 points.

Problem 4: EXTRA: 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.