Mastery II — Data Struct. & OOP (T. III/19–20)

Directions:

- You have 11 hours and 50 minutes to complete the following *four* problems. Each problem comes with performance expectations. You'll need to meet them to obtain full credit. Partial credit will be given to correct solutions that don't fully meet the bar.
- **WHAT IS PERMITTED:**
  - Reading the official Java documentation
  - Accessing Canvas for submission.
  - Browsing (online) tutorials and stack overflow (and similar websites).
- **WHAT IS *NOT* PERMITTED:**
  - Asking or attempting to get help from a person or AI either online or in person.
  - Communicating with other person or using any other aid.
- We're providing a starter package. The fact that you're reading this PDF means you have successfully downloaded the starter pack.
- At the top level, the starter package contains 4 folders, one for each problem. We're supplying `build.gradle` suitable for each problem. For this reason, it is recommended that you use one IntelliJ project for each problem. To save you some typing, we're supplying some simple tests (in the class's main method). Passing these tests doesn't mean you will pass our further testing. Failing them, however, means you will not pass the real one.
- To submit your work, run `gradle clean` for each problem, and zip the folders for all problems as one zip file called `mastery2.zip` and upload it to Canvas.

Problem 1: Lost Items (10 points)

Dr. Piti keeps a rare collection of numbers. In this collection, some numbers are repeated multiple times. He treasures this collection so much that he has a backup. Thus, there are two identical copies, a and b.

One day, he took collection a to a Deadly Math lecture and some kids took home a few numbers after watching card tricks. He is mad and wants to 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.

Expectations: Promises, Constraints, and Grading

- $1 \leq$ `a.length` $\leq$ `b.length` $\leq 100,000$.
- We promise that every number in a appears in b.
- Each `b[i]` satisfies $0 \leq$ `b[i]` $\leq 16,384$. (*Hint:* it is possible to keep an **int** array of length 16,384 or more. For example, `int[] counts = new int[16385];`)
- Your solution must finish within 2 seconds per test. Let n be the length of b. The desired solution must run in $O(n \log n)$ time or faster. (*Hint:* Both $O(n \log n)$ and $O(n)$ solutions are within reach.)

- 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 2: Game of *k* Deques (10 points)

G and P are besties. In return for P's special present, G is inviting P to a specially-designed game.

G has neatly arranged k deques $D_1, D_2, \ldots, D_k$, where each D_i is a deque whose values are sorted from large (front of the deque) to small (back of the deque). Every integer is guaranteed to be positive (i.e., > 0).

The pair is playing the following game: At the beginning, P is given a number x.

- In each move, P can remove one integer from the back of one of the deques and hand the integer to G.
- G keeps a running sum of the integers P's have handed to her.
- The game ends as soon as this running sum becomes greater (>) than x given at the beginning.
- P's *final score* is the count of integers handed to G before the game is over. This count does *not* include the final number that causes the game to be over. P's goal, of course, is to maximize the final score.

If P manages to remove all the numbers without exceeding x, the score is the total number of elements.

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

```
public static int maximizeScore(List<Deque<Integer>> theDeques, int x)
```

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

Sample Input: Suppose $x = 9$ and the 3 input deques are:

```
Deque #1: 6, 3, 1 (back)
Deque #2: 9, 5, 2, 1 (back)
Deque #3: 4, 1 (back)
```

The best possible score for this input is 5. One way (among several others) to achieve this is to remove numbers as shown in the following diagram:

No.	Action	Running Sum
1	Remove back from #2	1
2	Remove back from #3	2
3	Remove back from #1	3
4	Remove back from #2	5
5	Remove back from #1	8
6	Remove back from #3	12

The score is 5 because in the 6th move the sum is above $x = 9$.

Expectations: Promises, Constraints, and Grading

- There will be at least 1 deque and $x \geq 1$. We guarantee that the sum of all the numbers in every deque 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 deques and k is the number of deques. All test cases have $N \leq 5,000,000$ and $k \leq 1,000$.
- If your solution runs slower than that, you will receive some partial credit.

Problem 3: The Count of Ones (10 points)

We will count the total of number of ones in an array of packed sorted bit-arrays (PaSBA). A sorted bit-array satisfies the following interface:

```
public interface SortedBitArray {
    // returns the value at index k.
    int get(int k);

    // returns the length of the array.
    int length();
}
```

When a bit-array is sorted, all the 0s come before all the 1s. Storing a bit-array of length n generally requires $O(n)$ space. However, when a bit-array is sorted, it is possible to store it using much less memory, for example, by storing the index at which the run of 0s ends.

A *packed* sorted bit-array (PaSBA), given as a `PackedSortedBitArray` class, satisfies the `SortedBitArray` interface and further guarantees that both its `.get` and `.length` run in worst-case $O(1)$ time.

Your Task: Without modifying the given `PackedSortedBitArray` class or `SortedBitArray` interface, implement the following static method inside a class `PaSBACount`:

```
public static long count(SortedBitArray[] packedArrays)
```

which counts the total number of 1s in the whole array of PaSBAs.

To test your code, you will likely want to use the `PackedSortedBitArray` in the starter pack. The constructor `PackedSortedBitArray(int n, int k)` makes an instance of length `n` where the first index of a 1 is `k`. As an example, the array of PaSBAs on the left compactly encodes the arrays on the right (which are never constructed for real).

```
packedArrays = {
    new PackedSortedBitArray(5, 2),
    new PackedSortedBitArray(3, 1),
    new PackedSortedBitArray(4, 2),
    new PackedSortedBitArray(7, 3)
};
```

```
{
    {0, 0, 1, 1, 1},
    {0, 1, 1},
    {0, 0, 1, 1},
    {0, 0, 0, 1, 1, 1, 1}
}
```

In this example, `packedArrays[1].get(1)` will return 1 and `packedArrays[3].get(0)` will return 0. Furthermore, if we run `count` on this `packedArrays`, the return value should be 3 + 2 + 2 + 4 = 11, which is the number of 1s in the above.

Expectations: Promises, Constraints, and Grading

- $0 \leq$ `packedArrays.length` $\leq 100,000$. The length of each PaSBA is between 0 and $1,000,000,000$.
- For each `SortedBitArray` instance given as part of the input, its `.get` and `.length` run in worst-case $O(1)$ time. It will *not* be any slower than the `PackedSortedBitArray` supplied in your starter pack.
- For full credit, your code must run in $O(n \log k)$ or faster, where n is `packedArrays.length` and k is the length of the longest PaSBA. Your code must finish within 2 seconds on each of these. (*Hint:* This suggests we could use some variant of binary search.)
- Although the number of 1s in each PaSBA will fit in a Java **int**, the grand total may be significantly larger than what could be stored in an **int**. Notice that the return type of the function is a **long**.

Problem 4: The Perfect Hiding Spot (10 points)

The CS student committee for recreation and well-being is renting a huge castle for a week-long party this summer. The entrance opens into Chamber 1. In this chamber, they're planning to set up several large TV monitors so they can play console games without seeing daylight.

Not everyone is a console gamer, though. Your task is to help find the perfect hiding spot for people who seek solitude from the gaming crowd. You will be presented with a map of the castle. The chambers (vertices) are numbered 1 through n. You'll also be given a list of one-directional passages from one chamber to another (directed edges). Every passage is labeled with its distance. Importantly, each chamber has exactly one passage that opens into it. More precisely, the input is a directed tree where every vertex, except for vertex 1, has precisely one incoming edge. Note: vertex 1 has no incoming edge.

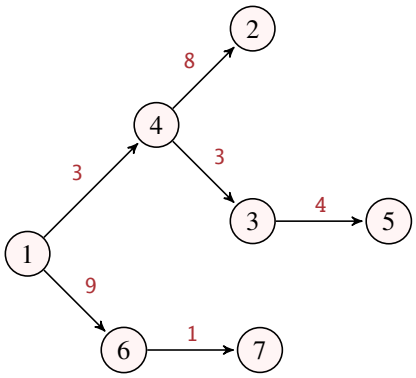
Inside the class `PerfectHiding`, you are to implement a static method

```
public static int bestSpotDistance(int n, List<WeightedEdge> passages)
```

where n is the number of chambers and `passages` is a list of directed edges. If `e` is a `WeightedEdge` in this list, then there is a passage from Chamber `e.first` to Chamber `e.second` of length `e.cost` meters. Notice that because this structure is a tree, the length of the list `passages` is always $n - 1$.

Your static method will compute the distance to the chamber (vertex) farthest away from chamber number 1. If it turns out that you cannot reach any chamber other than 1, the answer will be 0.

Example:



This graph has $n = 7$ nodes and the edges are:

```
(1, 4, 3)   (4, 2, 8)   (4, 3, 3)
(3, 5, 4)   (1, 6, 9)   (6, 7, 1)
```

Here, the notation (u, v, c) denotes a directed edge from u to v with a weight of c meters. Calling `bestSpotDistance` on this input will result in the number 11 because Chamber 2 is the farthest from 1 via $1 \to 4 \to 2$, totaling $3 + 8$ meters.

Expectations: Promises, Constraints, and Grading We'll test your program with n up to $100,000$. On this size of input, your program must finish within 3 seconds to receive full credit. The cost of an edge will be a number between 1 and $1,000$ (inclusive).