Data Structures Advanced with Java - Regular Exam

Do not modify the interface or the package, or anything from the given resources. In Judge, you only upload the archive of the corresponding package.

1. Solar Service – 100pts

In our solar service, we are working with Inverters and PVModules attached to them. Since the PVModules are not directly attached to an Inverter we should maintain information about the arrays in which the PVModules are grouped before being connected to the Inverter. Each Inverter and Array is identified by an ID of type String.

- `void addInverter(Inverter inverter)` - adds an Inverter to our system. If the Inverter already exists throw **IllegalArgumentException**
- `void addArray(Inverter inverter, String arrayId)` - reserves new arrayId to our inverter. We'll use this so that we can attach PVModules later on. If the inverter is missing or this arrayId is already in use or the inverter doesn't have the capacity for more arrays- throw **IllegalArgumentException.** Each array will have a unique name for the whole application.
- `void addPanel(Inverter inverter, String arrayId, PVModule pvModule)` - Adds a panel to an existing array for a given inverter. If the Inverter is missing - throw **IllegalArgumentException.** If this array is not associated with this Inverter or the PVModule is already in use - throw **IllegalArgumentException**
- `boolean containsInverter(String id)` - return if the inverter with this id is present in the system
- `boolean isPanelConnected(PVModule pvModule)` - return if this PVModule is present in the system
- `Inverter getInverterByPanel(PVModule pvModule)` - get the inverter to which this PVModule is connected. If it is not connected to any Inverter - return **null**
- `void replaceModule(PVModule oldModule, PVModule newModule)` - replaces an old module with a new one. If the oldModule is not in use or the newModule is already used somewhere - throw **IllegalArgumentException**
- `Collection<Inverter> getByProductionCapacity()` - return all Inverters sorted by total production capacity - **lowest to highest.** The capacity is calculated based on the sum of **maxWattProduction** of each PVModule connected to this Inverter.
- `Collection<Inverter> getByNumberOfPVModulesConnected()` - return all Inverters sorted by number of modules connected - **lowest to highest**. If 2 inverters have the same number of modules connected - sort them by the number of arrays present for each inverter.
- `Collection<PVModule> getByWattProduction()` - return all PVModules sorted by **maxWattProduction.** For panels with the same production return them by order of addition

2. Solar Service – Performance – 50pts

For this task, you will only be required to submit the **code from the previous problem**. If you are having a problem with this task you should **perform detailed algorithmic complexity analysis**, and try to **figure out weak** spots inside your implementation.

For this problem, it is important that other operations are **implemented correctly** according to the specific problems: **addInverter**, **addArray,** etc…

You can submit code to this problem **without full coverage** from the previous problem, not all test cases will be considered, only the **general behavior** will be important, **edge cases** will mostly be ignored such as throwing exceptions, etc…