**Data Structures Fundamentals – Exam**

# **Bitcoin Wallet Manager**

**Correctness - 100 pts**

You are tasked with implementing the data management of a **Bitcoin wallet management system**. The software should work with Bitcoin **transactions**, **wallets**, and **users** as entities. All entities are identified by a unique Id. The entities are defined as follows:

The **Transaction** entity contains the following properties:

* **Id** – string
* **SenderWalletId** – string (unique Id of the sender's wallet)
* **ReceiverWalletId** – string (unique Id of the receiver's wallet)
* **Amount** – long (amount of Sats involved in the transaction)
* **Timestamp** – DateTime

The **Wallet** entity contains the following properties:

* **Id** – string
* **UserId** – string (unique Id of the user owning the wallet)
* **Balance** – long(total balance of Sats in the wallet)
* **Transactions** – List of Bitcoin Transaction Ids (Ids of transactions associated with this wallet)

The **User** entity contains the following properties:

* **Id** – string
* **Name** – string
* **Email** – string

Implement the following functionalities to make the Bitcoin wallet management system fully operative:

* **void CreateUser(User user)** – adds a new user to the system.
* **void CreateWallet(Wallet wallet)** – creates a new wallet for a user and adds it to the system.
* **bool ContainsUser(User user)** – returns whether the user is registered in the system.
* **bool ContainsWallet(Wallet wallet)** – returns whether the wallet is present in the system.
* **IEnumerable<Wallet> GetWalletsByUser(string userId)** – returns all wallets owned by the given user.
* **void PerformTransaction(Transaction transaction)** – performs a Bitcoin transaction between two wallets, updating their balances accordingly. If neither the sender's nor receiver's wallet is not present in the system or if the sender's wallet balance is insufficient, throw ArgumentException().
* **IEnumerable<Transaction> GetTransactionsByUser(string userId)** – returns all Bitcoin transactions associated with the given user (either as a sender or receiver).
* **IEnumerable<Wallet> GetWalletsSortedByBalanceDescending()** – returns all wallets ordered by their balances in descending order.
* **IEnumerable<User> GetUsersSortedByBalanceDescending()** – returns all users ordered by their balances in descending order.
* **IEnumerable<User> GetUsersByTransactionCount()** – returns all users ordered by the total count of transactions they've been involved in (as a sender or receiver) in descending order. If there aren’t any users – return an empty collection.

Please implement the above functionalities in a class named "**BitcoinWalletManager**" that implements the "**IBitcoinWalletManager**" interface. The "**IBitcoinWalletManager**" interface should contain the method signatures for the above functionalities.

**NOTE: If all sorting criteria fail, you should order by order of input. This is for all methods with ordered output.**

**Performance – 50 pts**

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 a 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: **add**, **size**, **remove**, **get,** etc… Also, make sure you are using the correct data structures. ☺

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, and **edge** **cases** will mostly be ignored such as throwing exceptions, etc…