2.2 Poker Game Simulator

Poker is a family of comparing card games in which players bet on the set of cards they received. Player who has got best collection of cards wins the game and gets all the bets and other players lose their bets. Your task is to use `Heap` data structure, you need to solve the design a Poker Game Simulator.
In a city named `'COL106'`, citizens decides to earn extra income by using their knowledge of poker. The population of the city is `city_size` and citizens have their unique id numbered as `0,1,...,city_size-1`. There is a `Poker arena` where citizens come and play poker. Each citizen has `$100,000` initially.
When a citizen enter the `Poker arena`, they come up with their maximum loss that they can bear and maximum profit they wish to get. When a citizen has suffered loss more than it's maximum loss or has gained more profit than it's maximum profit, he/she leaves the `Poker arena`.
In each game, a set of m players play and each player bids his/her amount (m can vary for each game and $1 < m \leq n$, where n is the number of citizens present in `Poker arena` currently). The players playing a poker game will be present in the `Poker arena`. Winner of a poker game is decided using a random generator and the winner gets all the money raised by other players and other players lose their bids.
Number of citizens in `Poker arena` will change when an existing citizen leave the `Poker arena` as he/she has reached his/her maximum profit/maximum loss or when a citizen enters the `Poker arena`. Any citizen could enter the `Poker arena` irrespective of whether he/she has entered the `Poker arena` earlier or not.

You need to implement methods for the class `Poker`. The only global data structure you are allowed to use is `Heap` data structure computed in 2.1. Furthermore, the number of instances allowed of `Heap` data structure is at max **four**. You may use `ArrayList` locally inside member functions.

The class `Poker` will contain the following components.

- **Instance variables:**
    1. `private int city_size`: Population of the city `COL106`.
    2. `public int[] money`: An array of size `city_size` denoting the current money each citizen has. Initially, this is `100,000` for all citizens.
- **Constructor:**
    1. `public Poker(int city_size, int[] players, int[] max_loss, int[] max_profit)`: The variable `city_size` denotes the population of the city `COL106`. The array `players` contains the id of the citizens who are first to come to the `Poker arena` to play Poker, and the size of the array is bounded by `city_size`. The value `max_loss[i]` denotes the maximum loss player `players[i]` can bear. Similarly, `max_profit[i]` denotes the maximum profit player `players[i]` wishes to get,

before he/she decides to exit the arena. The array `players` will contains distinct values. You can initialize the `Heap` data structure (if needed).

The time complexity of this function must be $O(n)$ where n denotes the size of array `players`.

- **Member functions:**

1. `public void initMoney()`: Initialize the `money` array. Do not change anything in this function.
2. `public ArrayList<Integer> Play(int[] players, int[] bids, int winnerIdx)`: The array `players` stores the id of the citizens who will play the poker game. `winnerIdx` is index of player who will win the game, so player `players[winnerIdx]` will win this game. It will be ensured that the players who have come to play the poker game will be present in the `Poker arena`. Size of `players` and `bids` will be equal and `players` will contain distinct values. The value `bids[i]` denotes the bid made by player `players[i]` in this game. You must update the money of the players who has played in this game in array `money`, and return `ArrayList` storing all players who will leave the poker arena after this game. (In case no player leaves, return an empty `ArrayList`).

   The time complexity of this function must be $O(m \log n)$ where m denotes the size of `players` and n denotes the number of citizens present in the `Poker arena` currently.
3. `public void Enter(int player, int max_loss, int max_profit)`: A citizen with id `player` enters the `Poker arena`. Here `max_loss` denotes the (new) maximum loss the `player` can bear, and `max_profit` is (new) maximum profit `player` wants to get. It will be ensured that `player` is not in the `Poker arena`. In particular, a citizen enters only if he/she had either left the `Poker arena` earlier, or was not added to the `Poker arena` during the initialization phase. The Money that this citizen has will be same as the money that he/she had before leaving the `Poker arena`. If the citizen is entering for the first time, then the money he/she has will be 100,000.

   The time complexity of this function must be $O(\log n)$ where n is the number of citizens in the `Poker arena` currently.
4. `public ArrayList<Integer> nextPlayersToGetOut()`: Returns `ArrayList` storing the id of citizens who are likely to get out of `Poker arena` in the next game. Citizens in `Poker arena` are ranked based on score and you have to return the citizens who have minimum score. The score for citizen with id `idx` who entered the `Poker arena` with money `M[idx]`, maximum profit `max_profit[idx]` and maximum loss `max_loss[idx]` is calculated as :

   `score[idx]` = `minimum(max_profit[idx]+M[idx]-money[idx], money[idx]+max_loss[idx]-M[idx])`.

The time complexity of this function should be $O(\ell)$, where ℓ denotes the size of the output.

5. `public ArrayList<Integer> playersInArena()`: Returns `ArrayList` storing id of citizens present in the `Poker arena`.

   The time complexity of this function must be $O(n)$ where n is the number of citizens in the `Poker arena` currently.

6. `public ArrayList<Integer> maximumProfitablePlayers()`: Returns `ArrayList` storing id of citizens who has gained maximum profit. You need to include all citizens present in the city and not only those currently present in the `Poker arena`. Here, profit for citizen with id `idx` is calculated as `money[idx]-100000`.

   The time complexity of this function should be $O(\ell)$, where ℓ denotes the size of the output.

7. `public ArrayList<Integer> maximumLossPlayers()`: Returns `ArrayList` which stores id of citizens who has suffered maximum loss. You need to include all citizens present in the city and not only those currently present in the `Poker arena`. Here, loss for citizen with id `idx` is calculated as `100000-money[idx]`.

   The time complexity of this function should be $O(\ell)$, where ℓ denotes the size of the output.

Note:

1. Whenever return type is ArrayList, ordering of elements in ArrayList doesn't matter.
2. `max_loss` for a citizen with id `idx` will be in range [0, `money[idx]`] always.
3. `bids` for a citizen with id `idx` will be in range [0, `money[idx]`] always.
4. `key` will be in range [0, `max_size` $-$ 1] for heap in all the methods.