Leader Board

Design a Leaderboard class, which has 3 functions:

1. `addScore(playerId, score)`: Update the leaderboard by adding `score` to the given player's score. If there is no player with such id in the leaderboard, add him to the leaderboard with the given `score`.
2. `top(K)`: Return the score sum of the top `K` players.
3. `reset(playerId)`: Reset the score of the player with the given id to 0 (in other words erase it from the leaderboard). It is guaranteed that the player was added to the leaderboard before calling this function.

Initially, the leaderboard is empty.

1. 方法1, 用Heap

```
class Leaderboard:

    def __init__(self):
        self.scores = {}

    def addScore(self, playerId: int, score: int) -> None:
        # O(1)
        if playerId not in self.scores:
            self.scores[playerId] = 0
        self.scores[playerId] += score

    def top(self, K: int) -> int:
        # N log (K)
        # This is a min-heap by default in Python.
        heap = []
        for x in self.scores.values():
            heapq.heappush(heap, x)
            if len(heap) > K:
                heapq.heappop(heap)
        res = 0
        while heap:
            res += heapq.heappop(heap)
        return res

    def reset(self, playerId: int) -> None:
        # O(1)
        self.scores[playerId] = 0
```

2. 方法2, 用sortedDict()

```
from sortedcontainers import SortedDict

class Leaderboard:

    def __init__(self):
        self.scores = {}
        self.sortedScores = SortedDict()

    def addScore(self, playerId: int, score: int) -> None:

        # The scores dictionary simply contains the mapping from the
        # playerId to their score. The sortedScores contain a BST with
        # key as the score and value as the number of players that have
        # that score.
        if playerId not in self.scores:
            self.scores[playerId] = score
            self.sortedScores[-score] = self.sortedScores.get(-score, 0) + 1
        else:
            preScore = self.scores[playerId]
            val = self.sortedScores.get(-preScore)
            if val == 1:
                del self.sortedScores[-preScore]
            else:
                self.sortedScores[-preScore] = val - 1

            newScore = preScore + score;
            self.scores[playerId] = newScore
            self.sortedScores[-newScore] = self.sortedScores.get(-newScore, 0) + 1

    def top(self, K: int) -> int:
        count, total = 0, 0;

        for key, value in self.sortedScores.items():
            times = self.sortedScores.get(key)
            for _ in range(times):
                total += -key;
                count += 1;

                # Found top-K scores, break.
                if count == K:
                    break;

            # Found top-K scores, break.
            if count == K:
                break;
```

```
        return total;

    def reset(self, playerId: int) -> None:
        preScore = self.scores[playerId]
        if self.sortedScores[-preScore] == 1:
            del self.sortedScores[-preScore]
        else:
            self.sortedScores[-preScore] -= 1
        del self.scores[playerId];
```