**Unordered add(e) and remove(i) Methods for Scoreboard**

The goal is to implement add(e) and remove(i) without maintaining the game entries in sorted order. Additionally, these operations must run in **constant time (O(1))** without loops.

**Approach**

1. **add(e):**
   * If the scoreboard is not full, simply add the new entry at the **next available index**.
   * If the scoreboard is full, **replace a random existing entry** with e.
2. **remove(i):**
   * Instead of shifting elements (which takes O(n) time), we **replace the removed element with the last entry** in the array.
   * This maintains O(1) complexity and avoids unnecessary shifting.

**Java Code Implementation**

public class Scoreboard {

private GameEntry[] board;

private int size; // Keeps track of the number of entries

public Scoreboard(int capacity) {

board = new GameEntry[capacity];

size = 0;

}

// Add a new entry without keeping order

public void add(GameEntry e) {

if (size < board.length) {

board[size] = e; // Place at the next available index

size++;

} else {

int randomIndex = (int) (Math.random() \* board.length);

board[randomIndex] = e; // Replace a random existing entry

}

}

// Remove an entry at index i without shifting elements

public void remove(int i) {

if (i < 0 || i >= size) {

throw new IndexOutOfBoundsException("Invalid index");

}

board[i] = board[size - 1]; // Swap with the last element

board[size - 1] = null; // Remove reference

size--; // Reduce the size

}

// Print scoreboard for debugging

public void printScoreboard() {

for (int i = 0; i < size; i++) {

System.out.println(board[i]);

}

}

public static void main(String[] args) {

Scoreboard sb = new Scoreboard(5);

sb.add(new GameEntry("Alice", 50));

sb.add(new GameEntry("Bob", 75));

sb.add(new GameEntry("Charlie", 90));

sb.add(new GameEntry("David", 60));

sb.add(new GameEntry("Eve", 80));

System.out.println("Initial Scoreboard:");

sb.printScoreboard();

System.out.println("\nRemoving index 2...");

sb.remove(2); // Removes Charlie and replaces with the last entry

System.out.println("\nUpdated Scoreboard:");

sb.printScoreboard();

System.out.println("\nAdding a new score...");

sb.add(new GameEntry("Frank", 85)); // Replaces a random entry

System.out.println("\nFinal Scoreboard:");

sb.printScoreboard();

}

}

// Class for game entries

class GameEntry {

private String name;

private int score;

public GameEntry(String name, int score) {

this.name = name;

this.score = score;

}

@Override

public String toString() {

return name + ": " + score;

}

}

**Key Features of This Approach**

✅ **Constant Time Complexity O(1):**

* add(e): Directly inserts without shifting.
* remove(i): Replaces i with the last element instead of shifting.

✅ **No Loops Used:**

* Both add() and remove() execute in a **fixed number of steps**, regardless of n.

✅ **Efficient Memory Usage:**

* No extra arrays or shifting operations.

**Example Output**

Initial Scoreboard:

Alice: 50

Bob: 75

Charlie: 90

David: 60

Eve: 80

Removing index 2...

Updated Scoreboard:

Alice: 50

Bob: 75

Eve: 80

David: 60

Adding a new score...

Final Scoreboard:

Alice: 50

Bob: 75

Frank: 85

David: 60

**Why This Works Well?**

* **No need to maintain order** (which would require shifting elements).
* **Fast execution** due to direct indexing.
* **Easy to implement** with minimal logic.