Universidad Autónoma de Occidente

FACULTAD DE INGENIERÍA Y CIENCIAS BÁSICAS
PROGRAMA DE INGENIERÍA INFORMÁTICA
ASIGNATURA ESTRUCTIRAS DE DATOS Y ALGORITMOS 1

TAREA FINAL CONCURSO DE TRIPLES

Sebastián Leiton Goyes – 2235779

David Torres – 2235622

María Camila Serna Izquierdo – 2235504

Asignatura
Estructuras de datos y algoritmos 1

Profesor
Orlando Arboleda Molina

Universidad Autónoma de occidente UAO
Facultad de Ingeniería y Ciencias Básicas

IS06 – Ingeniería Informática

Resumen del enunciado

El proyecto consiste en desarrollar un aplicativo web para el Concurso de Triples de la NBA. Este concurso tiene entre sus participantes al menos 3 jugadores (n≥3), al menos 5 estaciones de tiro (m≥5), y cada estación cuenta con al menos 5 balones (p≥5). Se requiere recoger y procesar los intentos de cada jugador con un formato específico, evaluando el puntaje obtenido bajo diferentes reglas, teniendo en cuenta balones estándar (1 punto) y un "money ball" (2 puntos). Los resultados deben ser procesados y ordenados según tres criterios: puntos totales descendentes, número de "money balls" descendentes y, en caso de empate, por nombre del jugador ascendente. La implementación debe incluir una función en JavaScript llamada `resolverConcursoTriples(concurso)` que genere la salida en formato específico, evaluando su correctitud y rendimiento.

Código/pseudocódigo de la función resolverConcursoTriples()

```
FUNCION RESOLVERCONCURSOTRIPLES(CONCURSO)

    SI (CONCURSO ES NULO) O (TIPODE(CONCURSO) ≠ CADENA) ENTONCES
        RETORNAR "INGRESA UNA CADENA VÁLIDA CON EL FORMATO DEL                                  CONCURSO."
    FINSI

    INTENTAR
        // 1. PARSEAR LOS DATOS DEL CONCURSO
        (JUGADORES, RONDAS) ← UTIL.PARSEARCONCURSO(CONCURSO)

        // 2. VALIDAR QUE EL CONCURSO TENGA SENTIDO
        UTIL.VALIDARCONCURSO(JUGADORES, RONDAS)

        // 3. CALCULAR ESTADÍSTICAS (PUNTOS TOTALES Y MONEY BALLS)
        ESTADISTICAS ← UTIL.CALCULARESTADISTICAS(RONDAS)

        // 4. CONVERTIR ESTADÍSTICAS A UNA LISTA ORDENABLE
        RESULTADOS ← LISTAVACIA()
        PARA CADA (JUGADOR, DATOS) EN ESTADISTICAS HACER
            RESULTADO ← {
                PLAYER: JUGADOR,
                POINTS: DATOS.POINTS,
                MONEYBALLS: DATOS.MONEYBALLS
            }
            AGREGAR RESULTADO A RESULTADOS
        FINPARA

        // 5. DEFINIR COMPARADOR DE ACUERDO A LAS REGLAS
        DEFINIR FUNCION COMPARAR(A, B)
            SI (B.POINTS ≠ A.POINTS) ENTONCES
                RETORNAR B.POINTS - A.POINTS
            FINSI
            SI (B.MONEYBALLS ≠ A.MONEYBALLS) ENTONCES
                RETORNAR B.MONEYBALLS - A.MONEYBALLS
            FINSI
            RETORNAR COMPARARCADENAS(A.PLAYER, B.PLAYER)
```

```
        FINFUNCION

        // 6. ORDENAR LA LISTA USANDO MERGE SORT GENÉRICO
        ORDENADOS ← ALGORITMOS.MERGESORTOBJETOS(RESULTADOS, COMPARAR)

        // 7. FORMATEAR LA SALIDA FINAL
        RETORNAR UTIL.FORMATEARRESULTADOS(ORDENADOS)

    CAPTURAR ERR
        RETORNAR "ERROR: " + ERR.MENSAJE
    FININTENTAR

FINFUNCION
```

Complejidades individuales y totales

El sistema procesa concursos de triples con n jugadores, m rondas por jugador y p balones por ronda. La función parsearConcurso() tiene complejidad O(n×m×p) al dividir y convertir cada lanzamiento de la cadena de entrada. La función validarConcurso() opera en O(n×m) verificando restricciones agrupando rondas por jugador e iterando sobre cada una. La función calcularEstadisticas() recorre todas las rondas procesando cada balón, resultando en O(n×m×p) al sumar puntos normales y money balls. La función formatearResultados() es O(n) simplemente formateando el texto final del ranking. El ordenamiento con MergeSort aporta O(n log n) comparando jugadores por puntos, money balls y nombre. La función principal resolverConcursoTriples() integra todas las anteriores sumando O(n×m×p) + O(n×m) + O(n×m×p) + O(n log n) + O(n). Simplificando términos redundantes, la complejidad temporal total del sistema es O(n×m×p + n log n), donde el parseo y cálculo de estadísticas dominan. La complejidad espacial es O(n×m) almacenando todas las rondas y agrupaciones temporales en memoria. Para el escenario real de la NBA con 8 jugadores, 5 rondas y 5 balones, el sistema opera en tiempo constante O(224) siendo extremadamente eficiente. En conclusión, el cuello de botella está en procesar cada lanzamiento individualmente, mientras que el ordenamiento tiene impacto menor.

MergeSort (algoritmos.js)

- **mergeSortObjetos:**
  - (1) Verificar caso base: O(1).
  - (2) Dividir array: O(n).

  - (3) Llamadas recursivas: 2·T(n/2).
  - (4) Mezclar: O(n).
  - **Total:** O(n log n) complejidad total.

- **mergeObjetos:**
  - (1) Inicializar: O(1).
  - (2) Comparar y mezclar: O(n).
  - (3) Agregar restantes: O(n).
  - **Total:** O(n) complejidad total.

Util.js - Funciones del Sistema

- **resolverConcursoTriples:**
  - (1) Validar entrada: O(1).
  - (2) Parsear: O(n×m×p).
  - (3) Validar restricciones: O(n×m).
  - (4) Calcular estadísticas: O(n×m×p).
  - (5) Convertir a array: O(n).
  - (6) Ordenar: O(n log n).
  - (7) Formatear: O(n).
  - **Total:** O(n×m×p + n log n), complejidad total (n×m×p)

- **parsearConcurso:**
  - (1) Dividir cadena completa: O(n×m×p).
  - (2) Procesar jugadores: O(n).
  - (3) Dividir por "*": O(n×m).
  - (4) Para cada ronda separar nombre: O(1) por ronda.
  - (5) Dividir balones por espacios: O(p) por ronda.
  - (6) Convertir a número y validar: O(1) por balón, O(p) por ronda.
  - (7) Procesar n×m rondas con p balones: O(n×m×p).
  - **Total:** O(n×m×p) complejidad total.

- **validarConcurso:**
  - (1) Verificar n≥3: O(1).
  - (2) Agrupar rondas por jugador: O(n×m).
  - (3) Recorrer jugadores: O(n).
  - (4) Verificar m≥5 por jugador: O(n).
  - (5) Recorrer rondas de cada jugador: O(n×m).
  - (6) Verificar p≥5 por ronda: O(n×m).

  - (7) Crear Set de jugadores: O(n).
  - (8) Verificar existencia en todas las rondas: O(n×m).
  - **Total:** O(n×m) complejidad total.

- **calcularEstadisticas:**
  - (1) Crear Map: O(1).
  - (2) Recorrer rondas: O(n×m).
  - (3) Verificar/inicializar jugador en Map: O(1).
  - (4) Recorrer balones normales: O(p) por ronda.
  - (5) Sumar punto si acierta: O(1) por balón.
  - (6) Procesar money ball: O(1).
  - (7) Sumar 2 puntos si acierta: O(1).
  - **Total:** O(n×m×p) complejidad total.

- **formatearResultados:**
  - (1) Recorrer jugadores: O(n).
  - (2) Construir string por jugador: O(1).
  - (3) Join con saltos de línea: O(n).
  - **Total:** O(n) complejidad total.

Bibliografía

1. https://campus.uaovirtual.edu.co/pluginfile.php/326515/mod_folder/content/0/4_EDyA1_Recursividad.pdf?forcedownload=1
2. https://campus.uaovirtual.edu.co/pluginfile.php/326515/mod_folder/content/0/5_EDyA1_Ordenamiento.pdf?forcedownload=1
3. https://campus.uaovirtual.edu.co/pluginfile.php/326515/mod_folder/content/0/6_EDyA1_Busqueda.pdf?forcedownload=1