Ejercicio: Campeonato de Atletismo - Encontrar los Mejores Tiempos por Categoría Descripción: Se organiza un campeonato de atletismo con varias categorías. Cada corredor tiene un nombre, una categoría y un tiempo en segundos registrado en su carrera. Se desea encontrar el mejor tiempo (mínimo) y el nombre del corredor correspondiente para cada categoría, utilizando la técnica de Divide y Vencerás.

Función encontrarMejoresTiempos(corredores, inicio, fin):

Si inicio == fin:

Devolver {categoría: corredores[inicio].categoría, mejorTiempo: corredores[inicio].tiempo, nombre: corredores[inicio].nombre}

medio = (inicio + fin) / 2 // Dividir la lista en dos mitades

// Resolver recursivamente para cada mitad

mejoresIzquierda = encontrarMejoresTiempos(corredores, inicio, medio)

mejoresDerecha = encontrarMejoresTiempos(corredores, medio + 1, fin)

// Combinar: encontrar el mejor tiempo por categoría

mejoresCombinados = combinarMejoresTiempos(mejoresIzquierda, mejoresDerecha)

Devolver mejoresCombinados

Función combinarMejoresTiempos(mejoresIzquierda, mejoresDerecha):

mejoresCombinados = {}

// Combinar los mejores tiempos de ambas mitades

Para cada categoría en mejoresIzquierda:

Si la categoría no está en mejoresCombinados O mejoresIzquierda[categoría].mejorTiempo < mejoresCombinados[categoría].mejorTiempo:

mejoresCombinados[categoría] = mejoresIzquierda[categoría]

Para cada categoría en mejoresDerecha:

Si la categoría no está en mejoresCombinados O mejoresDerecha[categoría].mejorTiempo < mejoresCombinados[categoría].mejorTiempo:

mejoresCombinados[categoría] = mejoresDerecha[categoría]

Devolver mejoresCombinados

**Análisis de recurrencia mediante método inductivo**

1. **Caso base**:
   * Si la lista tiene un solo corredor, se devuelve su categoría, tiempo y nombre como el mejor tiempo para esa categoría.
2. **Paso inductivo**:
   * Supongamos que el algoritmo funciona correctamente para listas de tamaño n/2.
   * Al dividir la lista en dos mitades de tamaño n/2, el algoritmo encuentra los mejores tiempos para cada categoría en cada mitad.
   * Luego, combina los resultados comparando los mejores tiempos de ambas mitades para determinar los mejores tiempos globales por categoría.
3. **Complejidad**:
   * Cada llamada recursiva divide la lista en dos mitades, lo que resulta en O(log⁡n) niveles de recursión.
   * En cada nivel, se realizan operaciones lineales para combinar los resultados.
   * Por lo tanto, la complejidad total es O(nlog⁡n), ya que cada elemento se procesa en cada nivel de recursión.

import java.util.\*;

class Corredor {

String nombre;

String categoria;

int tiempo;

public Corredor(String nombre, String categoria, int tiempo) {

this.nombre = nombre;

this.categoria = categoria;

this.tiempo = tiempo;

}

}

public class CampeonatoAtletismo {

public static void main(String[] args) {

// Lista de corredores

List<Corredor> corredores = Arrays.asList(

new Corredor("Juan", "A", 300),

new Corredor("Ana", "B", 280),

new Corredor("Luis", "A", 290),

new Corredor("Maria", "B", 270),

new Corredor("Pedro", "C", 320)

);

// Encontrar los mejores tiempos por categoría

Map<String, Corredor> mejoresTiempos = encontrarMejoresTiempos(corredores, 0, corredores.size() - 1);

// Mostrar los resultados

for (Map.Entry<String, Corredor> entry : mejoresTiempos.entrySet()) {

System.out.println("Categoría: " + entry.getKey() +

", Mejor Tiempo: " + entry.getValue().tiempo +

", Corredor: " + entry.getValue().nombre);

}

}

// Método para encontrar los mejores tiempos por categoría usando Divide y Conquista

public static Map<String, Corredor> encontrarMejoresTiempos(List<Corredor> corredores, int inicio, int fin) {

Map<String, Corredor> mejoresTiempos = new HashMap<>();

// Caso base: un solo corredor

if (inicio == fin) {

Corredor corredor = corredores.get(inicio);

mejoresTiempos.put(corredor.categoria, corredor);

return mejoresTiempos;

}

// Dividir la lista en dos mitades

int medio = inicio + (fin - inicio) / 2;

// Resolver recursivamente para cada mitad

Map<String, Corredor> mejoresIzquierda = encontrarMejoresTiempos(corredores, inicio, medio);

Map<String, Corredor> mejoresDerecha = encontrarMejoresTiempos(corredores, medio + 1, fin);

// Combinar: encontrar el mejor tiempo por categoría

return combinarMejoresTiempos(mejoresIzquierda, mejoresDerecha);

}

// Método para combinar los mejores tiempos de ambas mitades

private static Map<String, Corredor> combinarMejoresTiempos(Map<String, Corredor> mejoresIzquierda, Map<String, Corredor> mejoresDerecha) {

Map<String, Corredor> mejoresCombinados = new HashMap<>();

// Combinar los mejores tiempos de la mitad izquierda

for (Map.Entry<String, Corredor> entry : mejoresIzquierda.entrySet()) {

String categoria = entry.getKey();

Corredor corredor = entry.getValue();

if (!mejoresCombinados.containsKey(categoria) || corredor.tiempo < mejoresCombinados.get(categoria).tiempo) {

mejoresCombinados.put(categoria, corredor);

}

}

// Combinar los mejores tiempos de la mitad derecha

for (Map.Entry<String, Corredor> entry : mejoresDerecha.entrySet()) {

String categoria = entry.getKey();

Corredor corredor = entry.getValue();

if (!mejoresCombinados.containsKey(categoria) || corredor.tiempo < mejoresCombinados.get(categoria).tiempo) {

mejoresCombinados.put(categoria, corredor);

}

}

return mejoresCombinados;

}

}