Actividad 3 Objetivo: Aplicar la técnica de Divide y Vencerás en una lista de clientes con id, nombre y scoring, buscando los dos clientes con los scoring máximos. Tareas: Resolver mediante pseudocódigo Realizar el análisis de recurrencia mediante método inductivo (sin utilizar fórmulas matemáticas) para indicar la complejidad algorítmica. Implementar en java

Función encontrarDosMaximosScoring(clientes, inicio, fin):

Si inicio == fin:

Devolver (clientes[inicio], null) // Caso base: un solo cliente

Si fin - inicio == 1:

Si clientes[inicio].scoring > clientes[fin].scoring:

Devolver (clientes[inicio], clientes[fin])

Sino:

Devolver (clientes[fin], clientes[inicio])

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

// Resolver recursivamente para cada mitad

(maxIzquierda, segundoMaxIzquierda) = encontrarDosMaximosScoring(clientes, inicio, medio)

(maxDerecha, segundoMaxDerecha) = encontrarDosMaximosScoring(clientes, medio + 1, fin)

// Combinar: encontrar los dos clientes con los scoring máximos entre las dos mitades

Si maxIzquierda.scoring > maxDerecha.scoring:

max1 = maxIzquierda

max2 = max(segundoMaxIzquierda, maxDerecha)

Sino:

max1 = maxDerecha

max2 = max(segundoMaxDerecha, maxIzquierda)

Devolver (max1, max2)

Análisis de recurrencia mediante método inductivo

Caso base:

Si la lista tiene un solo cliente, se devuelve ese cliente y null como segundo máximo.

Si la lista tiene dos clientes, se comparan y se devuelven ordenados por scoring.

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 dos máximos en cada mitad.

Luego, combina los resultados comparando los máximos de ambas mitades para determinar los dos máximos globales.

Complejidad:

Cada llamada recursiva divide la lista en dos mitades, lo que resulta en

O(logn) niveles de recursión.

En cada nivel, se realizan operaciones constantes para combinar los resultados.

O(n), ya que cada elemento se procesa una vez en algún nivel de recursión.

Implementacion Java

class Cliente {

int id;

String nombre;

int scoring;

public Cliente(int id, String nombre, int scoring) {

this.id = id;

this.nombre = nombre;

this.scoring = scoring;

}

@Override

public String toString() {

return "Cliente{" +

"id=" + id +

", nombre='" + nombre + '\'' +

", scoring=" + scoring +

'}';

}

}

public class DosMaximosScoringDivideYConquista {

public static void main(String[] args) {

// Lista de clientes

Cliente[] clientes = {

new Cliente(1, "Juan", 85),

new Cliente(2, "Ana", 90),

new Cliente(3, "Luis", 78),

new Cliente(4, "Maria", 95),

new Cliente(5, "Pedro", 88)

};

// Encontrar los dos clientes con los scoring máximos

Cliente[] resultado = encontrarDosMaximosScoring(clientes, 0, clientes.length - 1);

// Mostrar el resultado

System.out.println("El cliente con el scoring máximo es: " + resultado[0]);

System.out.println("El segundo cliente con el scoring máximo es: " + resultado[1]);

}

// Método para encontrar los dos clientes con los scoring máximos usando Divide y Conquista

public static Cliente[] encontrarDosMaximosScoring(Cliente[] clientes, int inicio, int fin) {

// Caso base: si solo hay un cliente

if (inicio == fin) {

return new Cliente[]{clientes[inicio], null};

}

// Caso base: si hay dos clientes

if (fin - inicio == 1) {

if (clientes[inicio].scoring > clientes[fin].scoring) {

return new Cliente[]{clientes[inicio], clientes[fin]};

} else {

return new Cliente[]{clientes[fin], clientes[inicio]};

}

}

// Dividir la lista en dos mitades

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

// Resolver recursivamente para cada mitad

Cliente[] maxIzquierda = encontrarDosMaximosScoring(clientes, inicio, medio);

Cliente[] maxDerecha = encontrarDosMaximosScoring(clientes, medio + 1, fin);

// Combinar: encontrar los dos clientes con los scoring máximos entre las dos mitades

Cliente max1, max2;

if (maxIzquierda[0].scoring > maxDerecha[0].scoring) {

max1 = maxIzquierda[0];

max2 = (maxIzquierda[1] != null && maxIzquierda[1].scoring > maxDerecha[0].scoring) ? maxIzquierda[1] : maxDerecha[0];

} else {

max1 = maxDerecha[0];

max2 = (maxDerecha[1] != null && maxDerecha[1].scoring > maxIzquierda[0].scoring) ? maxDerecha[1] : maxIzquierda[0];

}

return new Cliente[]{max1, max2};

}

}