Actividad 4 Objetivo: Encontrar los "n" elementos más grandes en una lista utilizando la técnica de Divide y Vencerás 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 encontrarNMaximos(lista, inicio, fin, n):

Si n <= 0:

Devolver lista vacía // Caso base: no se requieren elementos

Si inicio == fin:

Devolver [lista[inicio]] // Caso base: un solo elemento

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

// Resolver recursivamente para cada mitad

maximosIzquierda = encontrarNMaximos(lista, inicio, medio, n)

maximosDerecha = encontrarNMaximos(lista, medio + 1, fin, n)

// Combinar: encontrar los "n" máximos entre las dos mitades

maximosCombinados = combinarMaximos(maximosIzquierda, maximosDerecha, n)

Devolver maximosCombinados

Función combinarMaximos(maximosIzquierda, maximosDerecha, n):

maximosCombinados = []

i = 0

j = 0

// Combinar los máximos de ambas mitades hasta obtener "n" elementos

Mientras i < longitud(maximosIzquierda) Y j < longitud(maximosDerecha) Y longitud(maximosCombinados) < n:

Si maximosIzquierda[i] > maximosDerecha[j]:

Agregar maximosIzquierda[i] a maximosCombinados

i++

Sino:

Agregar maximosDerecha[j] a maximosCombinados

j++

// Agregar los elementos restantes de la mitad izquierda

Mientras i < longitud(maximosIzquierda) Y longitud(maximosCombinados) < n:

Agregar maximosIzquierda[i] a maximosCombinados

i++

// Agregar los elementos restantes de la mitad derecha

Mientras j < longitud(maximosDerecha) Y longitud(maximosCombinados) < n:

Agregar maximosDerecha[j] a maximosCombinados

j++

Devolver maximosCombinados

**Análisis de recurrencia mediante método inductivo**

1. **Caso base**:
   * Si n≤0, no se requieren elementos, por lo que se devuelve una lista vacía.
   * Si la lista tiene un solo elemento, se devuelve ese elemento.
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 n*n* máximos en cada mitad.
   * Luego, combina los resultados comparando los máximos de ambas mitades para determinar los n máximos globales.
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.

**Implementacion en java**

import java.util.ArrayList;

import java.util.List;

public class NMaximosDivideYConquista {

public static void main(String[] args) {

int[] lista = {10, 5, 20, 15, 30, 25}; // Lista de números

int n = 3; // Número de elementos máximos a encontrar

List<Integer> maximos = encontrarNMaximos(lista, 0, lista.length - 1, n);

System.out.println("Los " + n + " elementos más grandes son: " + maximos);

}

// Método para encontrar los "n" elementos más grandes usando Divide y Conquista

public static List<Integer> encontrarNMaximos(int[] lista, int inicio, int fin, int n) {

List<Integer> maximos = new ArrayList<>();

// Caso base: no se requieren elementos

if (n <= 0) {

return maximos;

}

// Caso base: un solo elemento

if (inicio == fin) {

maximos.add(lista[inicio]);

return maximos;

}

// Dividir la lista en dos mitades

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

// Resolver recursivamente para cada mitad

List<Integer> maximosIzquierda = encontrarNMaximos(lista, inicio, medio, n);

List<Integer> maximosDerecha = encontrarNMaximos(lista, medio + 1, fin, n);

// Combinar: encontrar los "n" máximos entre las dos mitades

return combinarMaximos(maximosIzquierda, maximosDerecha, n);

}

// Método para combinar los máximos de ambas mitades

private static List<Integer> combinarMaximos(List<Integer> maximosIzquierda, List<Integer> maximosDerecha, int n) {

List<Integer> maximosCombinados = new ArrayList<>();

int i = 0, j = 0;

// Combinar los máximos de ambas mitades hasta obtener "n" elementos

while (i < maximosIzquierda.size() && j < maximosDerecha.size() && maximosCombinados.size() < n) {

if (maximosIzquierda.get(i) > maximosDerecha.get(j)) {

maximosCombinados.add(maximosIzquierda.get(i));

i++;

} else {

maximosCombinados.add(maximosDerecha.get(j));

j++;

}

}

// Agregar los elementos restantes de la mitad izquierda

while (i < maximosIzquierda.size() && maximosCombinados.size() < n) {

maximosCombinados.add(maximosIzquierda.get(i));

i++;

}

// Agregar los elementos restantes de la mitad derecha

while (j < maximosDerecha.size() && maximosCombinados.size() < n) {

maximosCombinados.add(maximosDerecha.get(j));

j++;

}

return maximosCombinados;

}

}