URL VIDEOS

Explicación enunciado del problema: https://youtu.be/4f0udH_KLyY?feature=shared
Explicación código: https://youtu.be/O45bE-dBp6Q?feature=shared
Análisis de complejidad: https://youtu.be/8CrAJxGKWZw?feature=shared

ENUNCIADO DEL PROBLEMA

Dado un arreglo ordenado arr con posibles duplicados, la tarea consiste en encontrar la primera y última aparición de un elemento x en la matriz dada.

Nota: Si el número x no se encuentra en la matriz, devuelve ambos índices como -1.

Ejemplo:

Entrada: arr[] = [1, 3, 5, 5, 5, 5, 67, 123, 125], x = 5

Salida: [2, 5]

Explicación: La primera aparición del 5 está en el índice 2 y la última aparición del 5 está en el índice 5

1 ≤ arr.size() ≤ 10^6 1 ≤ arr[i], x ≤ 10^9
URL DE GITLAB O GITHUB PROYECTO EN FORMA PRIVADA, USUARIO madarme@ufps.edu.co, con rol maintener
https://gitlab.com/ejercicios-proyecto-ada/ejercicio-11-first-and-last-ocurrences

MÉTODO 1

```java
public int[] metodoCandido(int[] arr, int x) {
    int primera = -1, ultima = -1;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == x) {
            if (primera == -1) {
                primera = i;
            }
            ultima = i;
        }
    }

    return new int[]{primera, ultima};
}
```

Eficacia	El método es eficaz para todas las entradas posibles
	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado de la invocación: [2,5]
	Instancia 2:

	X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado de la invocación: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado de la invocación: [-1,-1]
Eficiencia	El método es eficiente solo para las entradas donde el arreglo es pequeño.
	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado de la invocación: [2,5] **Tiempo del proceso: 127 microsegundos**
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321,

	54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado de la invocación: [5,11] **Tiempo del proceso: 5 microsegundos**
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado de la invocación: [-1,-1] **Tiempo del proceso: 4 microsegundos**
Correctitud	El método devuelve salidas esperadas para todas las entradas posibles.
	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado esperado: [2,5] Resultado método candido: [2,5]
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321,

	54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado esperado: [5,11] Resultado método candido: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado esperado: [-1,-1] Resultado método cándido: [-1,-1]
Completitud	El método es completo, ya que la lógica usada para encontrar los índices es correcta, aunque en algunos casos es lento asegura siempre un resultado correcto, si es que este existe
	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado del método cándido: [2,5]
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321,

	54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado del método cándido: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado método cándido: [-1,-1]

MÉTODO 2

```java
public int[] metodoOptimo(int[] arr, int x) {
    int[] res = new int[2];
    res[0] = metodoOptimoIzq(arr, 0, arr.length - 1, x);
    res[1] = metodoOptimoDer(arr, 0, arr.length - 1, x);
    return res;
}

private int metodoOptimoIzq(int[] arr, int low, int high, int x) {
    if (low > high) {
        return -1;
    }
    int mid = (low + high) / 2;
    if ((mid == 0 || arr[mid - 1] < x) && arr[mid] == x) {
        return mid;
    } else if (x <= arr[mid]) {
        return metodoOptimoIzq(arr, low, mid - 1, x);
    } else {
        return metodoOptimoIzq(arr, mid + 1, high, x);
    }
}
```

```java
private int metodoOptimoDer(int[] arr, int low, int high, int x) {
    if (low > high) {
        return -1;
    }
    int mid = (low + high) / 2;
    if ((mid == arr.length - 1 || arr[mid + 1] > x) && arr[mid] == x) {
        return mid;
    } else if (x >= arr[mid]) {
        return metodoOptimoDer(arr, mid + 1, high, x);
    } else {
        return metodoOptimoDer(arr, low, mid - 1, x);
    }
}
```

Eficacia	El método es eficaz para todas las entradas posibles.

	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado de la invocación: [2,5]
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321, 54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado de la invocación: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado de la invocación: [-1,-1]
Eficiencia	El método es eficiente ya que al utilizar llamados recursivos se agiliza la búsqueda de los índices, pero esto es más notable en arreglos de mayor tamaño.
	Instancia 1: X = 5

	arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado de la invocación: [2,5] **Tiempo del proceso: 17 microsegundos**
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321, 54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado de la invocación: [5,11] **Tiempo del proceso: 5 microsegundos**
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado de la invocación: [-1,-1] **Tiempo del proceso: 2 microsegundos**
Correctitud	El método devuelve salidas esperadas para todas las entradas posibles.
	Instancia 1: X = 5

	arr = {1, 3, 5, 5, 5, 5, 67, 123, 125} Resultado esperado: [2,5] Resultado método candido: [2,5]
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado esperado: [5,11] Resultado método candido: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado esperado: [-1,-1] Resultado método cándido: [-1,-1]
Completitud	El método es completo ya que con la lógica usada se asegura una respuesta correcta si existe.
	Instancia 1: X = 5 arr = {1, 3, 5, 5, 5, 5, 67, 123, 125}

	Resultado del método óptimo: [2,5]
	Instancia 2: X=54321 arr = {10000, 20000, 30000, 40000, 50000, 54321, 54321, 54321, 54321, 54321, 54321, 54321, 80000, 90000, 90000, 90000, 90000, 90000, 100000, 110000, 120000, 130000, 140000, 150000}; Resultado del método óptimo: [5,11]
	Instancia 3: X = 4 arr = {1, 2, 3, 3} Resultado del método óptimo: [-1,-1]