## Solución al desafío técnico Robert Benítez ([robertbs9805@gmail.com](mailto:robertbs9805@gmail.com))

1. Escribe una función llamada mergeArrays que acepte dos arrays ordenados de enteros como parámetros y devuelva un solo array ordenado que contenga todos los elementos de ambos.
2. Escribe una función llamada findMedian que acepte un array de enteros como parámetro y devuelva la mediana del conjunto.
3. Escribe una clase llamada BinaryTree que implemente un árbol binario de búsqueda. La clase debe tener métodos para insertar, buscar, y debe ser capaz de imprimir el árbol en orden ascendente.
4. Escribe una función llamada isAnagram que acepte dos cadenas de texto como parámetros y determine si son anagramas (es decir, si tienen exactamente las mismas letras, pero en diferente orden).
5. Escribe una función llamada findCommonElements que acepte una lista de listas como parámetro y devuelva una lista con los elementos comunes a todas las sub-listas.
6. Escribe una implementación para el algoritmo de ordenamiento mergesort.
7. Escribe una función llamada removeDuplicates que acepte una lista como parámetro y devuelva una nueva lista sin elementos duplicados.
8. Dada una lista de números en orden ascendente y un número objetivo, escribe una función recursiva que encuentre si el número objetivo está en la lista utilizando una búsqueda binaria.

## Indicaciones:

1. Implemente su solución para los problemas 1, 2, 3, 7 y 8 en Java.
2. Implemente su solución para los problemas 4, 5, 6 utilizando Node con Javascript o TypeScript. [RobertBenitez9805/datamart-test: Technical test to validate Backend Engineer at Datamart (github.com)](https://github.com/RobertBenitez9805/datamart-test)
3. Analice la complejidad computacional de cada operación implementada en su solución al problema 3.

Para el caso de las operaciones **search()** e **insert()** puede ser **O(log n)** en el mejor caso (árbol balanceado) y **O(n)** en el peor caso (árbol no balanceado), *de esto depende el tamaño del árbol*.

La complejidad computacional de la operación **printInOrder()** es **O(n)**, ya que se recorre todo el árbol para imprimir sus elementos en orden.

1. Implemente un algoritmo que reciba en la entrada dos listas de números enteros y haga un map de cada elemento de la segunda lista con la función del problema 8 utilizando la primera lista como dominio de la búsqueda. Teniendo en cuenta que el tamaño de la primera lista podría ser muy grande explique qué beneficios tiene utilizar una búsqueda binaria en lugar de utilizar una búsqueda secuencial y cómo impacta la complejidad computacional de la solución. En circunstancias donde el segundo listado sea también grande es conveniente procesar las peticiones en paralelo; en este escenario sería conveniente poder configurar una máxima cantidad de workers para que procesen de forma paralela el segundo arreglo. Proponga una solución general a este problema utilizando Java.

La búsqueda binaria es más rápida que la búsqueda secuencial porque reduce el número de elementos que se deben comparar para encontrar el elemento deseado. La complejidad para la búsqueda binaria sería de **O(log n)** mientras que la complejidad computacional del a búsqueda secuencial es de **O(n)**, siendo la búsqueda binaria más efectiva en este caso.