1. Búsqueda en Listas (Buscar un elemento)

El ejercicio se trató de comparar las 2 maneras de encontrar un elemento en una lista, demostrando la eficiencia de un algoritmo que depende del método que usamos.

- **Búsqueda Lineal (O(n)):** Es el método más simple, pero el menos eficiente. La función *búsqueda lineal* revisa cada elemento de la lista 1 por 1 hasta encontrar el objetivo.

  Como buscar un libro en una biblioteca desorganizada. Por esto, el tiempo de ejecución es directamente proporcional al tamaño de la lista (n).

- **Búsqueda Binaria (O(logn)):** Este método es mucho más rápido, aunque funciona mejor en listas ya ordenadas y divide la lista por la mitad repetidamente. Si el elemento del medio no es el que buscas, la función descarta la mitad de la lista donde el objetivo no puede estar.

  Como buscar una palabra en un diccionario: no revisas cada página, sino que abres el libro por la mitad, y en cada paso, reduces la búsqueda a la mitad restante.

2. Ordenamiento de Listas (Organizar elementos)

Aquí se comparan dos algoritmos que se usan para ordenar listas, destacando cómo la forma de organizar los datos impacta enormemente el rendimiento.

- **Bubble Sort (O(n2)):** Es un algoritmo de ordenamiento ineficiente. *bubble sort* recorre la lista varias veces, comparando pares de elementos adyacentes y cambiándolos de lugar si están en el orden incorrecto.

- **Merge Sort (O(nlogn)):** Es un algoritmo muy eficiente. La función *merge sort* divide la lista por la mitad una y otra vez hasta que solo quedan elementos individuales. Luego, las va uniendo de nuevo en el orden correcto. Este método es tan eficaz que su tiempo de ejecución se mantiene rápido y constante, sin importar si la lista original estaba casi ordenada, completamente desordenada o al revés.

3. Cálculo de Fibonacci (Recursivo vs. Iterativo)

Esta parte del ejercicio demuestra una diferencia fundamental en la programación: cómo la **recursión** puede ser menos eficiente que un enfoque **iterativo** para el mismo problema.

- **Versión Recursiva (O(2n)):** La función *fib recursivo* se llama a sí misma para calcular los números anteriores. Aunque el código es corto, esta técnica es extremadamente ineficiente porque calcula los mismos valores una y otra vez.

- **Versión Iterativa (O(n)):** La función *fib dinamico* utiliza un bucle simple para calcular los números de Fibonacci de manera secuencial, almacenando solo los dos números anteriores. Este método es mucho más eficiente porque cada número se calcula una sola vez. Es la manera recomendada para resolver este tipo de problemas cuando se necesita rapidez.