1. Implementar una función en C que acepte como argumentos un *array* de enteros, su longitud y un número $s \in \mathbb{Z}$. Devolver el índice de $s$ si es que se encuentra en el *array* o -1 de lo contrario.

   Analizar la cantidad de instrucciones que se ejecutan en el peor caso posible en función de $n$ (la dimensión del *array*). ¿Cuál es la cantidad de instrucciones en el mejor caso?

2. Escribir una función que reciba como entrada un *array* de enteros y cuente la cantidad de pares en el *array* que suman a cero. Usar dos *loops for* anidados. ¿Cuál es el orden de crecimiento de dicha función?

3. Escribir una función que reciba como entrada un *array* de enteros y cuente la cantidad de ternas en el *array* que suman a cero. Usar tres *loops for* anidados. ¿Cuál es el orden de crecimiento de dicha función?

4. Escribir un programa que acepte un número $n$ como argumento de línea de comandos. El usuario debe pensar un número de 0 a $n$ y el programa debe adivinar el número que elige el usuario. Para eso el programa tiene un número limitado de intentos y cada vez que el programa elige un número el usuario debe responder si el número que pensó es igual o mayor al intento del programa.

5. Implementar una función que realice una búsqueda binaria en un *array* de enteros. Devolver el índice si el valor se encuentra o -1 de lo contrario.

6. Implementar la función del ejercicio anterior pero de forma iterativa.

7. Usar el concepto de la búsqueda binaria para buscar raíces de funciones reales. En este contexto se denomina método de bisección y consiste de lo siguiente. Sea $f(x)$ una función con dominio en $\mathbb{R}$ y $f(a)$ y $f(b)$ tienen signos opuestos. Debe existir un valor de $x$, $x_0$ en el intervalo $[a, b]$ tal que $f(x_0) = 0$.

   Aplicar la misma idea que en la búsqueda binaria para encontrar el valor de $x_0$ con un margen de error $\varepsilon$.

8. Implementar búsqueda binaria para buscar en un *array* de *strings*. Escribir un programa que implemente un filtro de lista blanca (*whitelist filter*). Es decir, recibir una lista de *strings* ordenada (la *whitelist*) y quedarse esperando entrada del usuario. Si el usuario ingresa un *string* que está en la lista se imprime "autorizado" y si el *string* no está se imprime "no autorizado".

9. Escribir un programa que filtre según una *blacklist*, la idea contraria a una *whitelist*. Es decir, se permite el acceso a cualquiera que no esté en la lista.

10. Escribir una función para ordenar un *array* de enteros usando el algoritmo conocido como *insertion sort* (ordenamiento por inserción). ¿En qué orden está el tiempo de ejecución de este algoritmo?

11. Escribir una función para ordenar un *array* de enteros usando el algoritmo conocido como *selection sort* (ordenamiento por selección). Este algoritmo ordena un *array* de enteros recorriendo el *array* desde el primer elemento y buscando el mínimo elemento del *subarray* a la derecha del elemento seleccionado para intercambiarlos de ser necesario.

12. Escribir una función que implemente el algoritmo de ordenamiento conocido como *bubble sort* (ordenamiento de burbuja).

13. Implementar la función *mergesort* para ordenar un *array* de *strings*.

14. Escribir un programa que reciba por entrada estándar una tira de números y copie a salida estándar los números recibidos quitando los duplicados. El programa recibe primero un número $n$ con la cantidad de números que se van a leer y luego los $n$ números que se deben imprimir sin duplicados.

    Escribir el mismo programa pero usando como entrada un número arbitrario de *strings* en vez de enteros.

15. Escribir dos funciones que operen sobre un *array* de enteros. Una para encontrar la moda y otra para encontrar la mediana.

16. Implementar un algoritmo de ordenamiento que lea de entrada estándar una cantidad arbitraria de enteros en el intervalo $[0, 99]$ y los copie a la salida de manera ordenada.

17. Rehacer el ejercicio dos escribiendo una función con complejidad $O(n)$.

18. Rehacer el ejercicio tres escribiendo una función con complejidad $O(n^2)$.

19. Escribir un programa en Processing que represente un *array* de enteros como un histograma. Realizar una animación de *insertion sort*, volviendo a dibujar el histograma cada vez que se intercambien dos valores en el *array*.

20. Realizar animaciones como las del ejercicio anterior para *selection sort* y *bubble sort*.

21. Dado un *array* de números reales, diseñar un algoritmo de complejidad $O(n \log n)$ para encontrar el par de números que estén más cerca el uno del otro.

22. Dado un *array* de números reales, diseñar un algoritmo de complejidad $O(n)$ para encontrar el par de números que estén más lejos el uno del otro.