1. Cree dos nuevas clases de puertas, una llamada **PuertaNOR** y otra llamada **PuertaNAND**. Las puertas **NAND** funcionan como puertas **AND** que tienen una **NOT** conectada a la salida. Las puertas **NOR** funcionan como puertas **OR** que tienen una **NOT** conectada a la salida. Cree la clase **XOR**
2. Cree una serie de puertas que demuestren que la siguiente ecuación **NOT((A and B) or (C and D))** es equivalente a **NOT(A and B) and NOT (C and D)**. Asegúrese de usar algunas de sus nuevas puertas en la simulación.
3. Modifique el constructor para la clase **Fracción** de modo que compruebe que el numerador y el denominador sean ambos enteros. Si alguno no es un entero, el constructor debe generar una excepción. Implemente los métodos simples **getNum** y **getDen** eso devolverá el numerador y el denominador de una fracción. Implementar los operadores relacionales restantes (>, >=, <, <=, y !=)
4. En la definición de fracciones asumimos que las fracciones negativas tienen un numerador negativo y un denominador positivo. El uso de un denominador negativo haría que algunos de los operadores relacionales dieran resultados incorrectos. En general, ésta es una restricción innecesaria. Modifique el constructor para permitir que el usuario pase un denominador negativo y que todos los operadores continúen funcionando correctamente.
5. Reescribir la clase común **Clase Fecha**, de manera que sea robusta y completarla con métodos necesarios
6. Implementar la clase de uso común **Clase Hora**, de manera que sea robusta y completarla con métodos necesarios
7. Implementar con todos sus componentes **la clase Polinomio**, de manera que se puedan ejecutar todas las operaciones sobre **polinomios**
8. Implementar con todos sus componentes **la clase Complejo**, de manera que se puedan ejecutar todas las operaciones sobre **complejos**
9. En un juego de dados, todos los participantes tienen el mismo número de dados (con n caras, de la 1 a la n). En cada turno, cada jugador tira todos sus dados, sumando cada dado para obtener una puntuación. Gana el jugador con mejor puntuación tras x turnos.
10. Completar la clase Fracción en C++ para que sea como la desarrollada en Python

11. Escribe dos funciones para encontrar el número mínimo en una lista. La primera función debe comparar cada número de una lista con todos los demás de la lista. $\boldsymbol{O(n^2)}$. La segunda función debe ser lineal $\boldsymbol{O(n)}$.
12. Diseña un experimento para verificar que el operador **indexación** para listas es $\boldsymbol{O(1)}$
13. Diseña un experimento para verificar que las operaciones de obtención y asignación de ítems para diccionarios son $\boldsymbol{O(1)}$**.**
14. Diseña un experimento que compare el desempeño del operador **del** en listas y en diccionarios.
15. Dada una lista de números en orden aleatorio, escribe un algoritmo que funcione en tiempo $\boldsymbol{O(nlog(n))}$ para encontrar el k-ésimo número más pequeño de la lista.
16. ¿Puedes mejorar el algoritmo del problema anterior para que sea lineal? **Explica**.
17. Escribir en lenguaje Python una función que calcule la suma de los elementos de un vector. Calcular la complejidad temporal de dicha función y expresarla en notación asintótica. Existe algún caso en el que dicha complejidad pueda ser $\boldsymbol{O(1)}$? **Explícalo**
18. Escribir una función que devuelva el número de valores que aparecen dos o más veces en un vector. Calcular la complejidad temporal de dicha función y expresarla en notación asintótica. **¿Lista o diccionario?**

Exámenes años anteriores

Implementar en **C++ la clase Complejo** con todos sus componentes, de manera que se puedan ejecutar todas las operaciones sobre **complejos: suma, resta, multiplicación, división, conjugado**

Usando la **clase Estudiantes**, implementar los métodos necesarios para hacer un programa para la evaluación de la presente asignatura y hacer un programa que, usando un fichero .csv, nos de las notas de los alumnos.