# Algoritmos y Estructuras de datos
# Ejercicios previos al segundo examen parcial

- Escribe una función que tome una secuencia de enteros y la ordene. No es necesario implementar un algoritmo eficiente, basta con que funcione. Puede usar cualquier utilidad de la biblioteca de C++ excepto `std::sort` y `std::stable_sort`.

```
void ordena(int* ini, int* fin) {
   //...
}
```

- Escribe una función que tome dos enteros y pueda ser usada por `std::sort` como predicado para ordenar enteros de la siguiente forma: los enteros negativos deben quedar primero y los no negativos después, desempatando enteros del mismo grupo por valor absoluto de menor a mayor. Por ejemplo, usar esta función con `std::sort` sobre la secuencia { 8, -7, 1, -2, 6, -4, 0 } debe resultar en { -2, -4, -7, 0, 1, 6, 8 }. Puedes usar cualquier utilidad de la biblioteca de C++.

```
bool signo_magnitud(int a, int b) {
   //...
}
```

- Escribe una función que tome dos alumnos y pueda ser usada por `std::sort` como predicado para ordenar alumnos por calificación de mayor a menor; si hay alumnos que tienen la misma calificación, éstos deben ordenarse por número de lista de menor a mayor.

```
struct alumno {
   int lista, calif;
};

bool predicado(alumno a, alumno b) {
   //...
}
```

- Escribe una función que tome un entero positivo *n* y que devuelva un arreglo que contenga los enteros de 0 a *n-1*. El arreglo devuelto debe seguir siendo válido aún después de que la función termine. No se permite usar variables globales ni estáticas.

```
int* genera(int n) {
   //...
}
```

- Escribe una función que tome una doble cola de enteros y devuelva otra con una copia de los elementos de la doble cola original, pero que aparezcan el orden contrario. No se permite modificar doble cola original ni usar las utilidades de `algorithm` de la biblioteca de C++, pero sí las de `deque`.

```
std::deque<int> invierte(const std::deque<int>& d) {
   //...
}
```

- Escribe una función que tome una doble cola y un entero *n* y rote los elementos de la doble cola *n* veces hacia la izquierda. Por ejemplo, si la doble cola guarda la secuencia { 1, 2, 3, 4, 5 } y *n*=2 entonces la doble cola debe quedar { 3, 4, 5, 1, 2 }. No se permite usar las utilidades de `algorithm` de la biblioteca de C++, pero sí las de `deque`.

```
void rota_izquierda(std::deque<int>& d, int n) {
   //...
}
```

- Escribe la implementación de una función que toma dos dobles colas e intente modificar la primera mediante rotaciones para que coincida con la segunda. Si es posible hacerlas coincidir, la función debe realizar la modificación y debe devolver verdadero; en caso contrario, la doble cola no debe presentar modificaciones al término de la función y se debe devolver falso. Por ejemplo, si las dobles colas guardan las secuencias { 1, 2, 3 } y { 2, 3, 1 } respectivamente, entonces la primera doble cola sí puede modificarse para que quede igual que la segunda. No se permite usar las utilidades de `algorithm` de la biblioteca de C++, pero sí las de `deque`.

```
bool intenta_rotacion(deque<int>& s1, const deque<int>& s2) {
   // Pista: puede usar la expresión s1 == s2 para preguntar si las dobles colas son iguales
   // Dos dobles colas son iguales si guardan los mismos valores en el mismo orden
}
```