Importante: Los ejercicios deben entregarse a través de web (**Domjudge y Blackboard**).

Para el primer ejercicio deberá entregarse el siguiente fichero:

quicksortLomuto.cpp (Utiliza como base el fichero proporcionado en Blackboard)

Para el segundo ejercicio deberá entregarse los siguientes ficheros:

Nodo.h y ListaCircular.h (Estos ficheros no deberán ser modificados)

ListaCircular.cpp (Utiliza como base el fichero proporcionado en Blackboard)

quicksortMedia.cpp

La fecha de entrega: consultar la página de la actividad en blackboard

Quicksort Lomuto(3 puntos): Implementa un algoritmo que permita ordenar una lista contigua de enteros utilizando el algoritmo de ordenamiento denominado quicksort que es una solución basada en una estrategia divide y vencerás. En concreto usa el esquema de partición diseñado por Lomuto que elige como pivote el último elemento y que puedes encontrar en la siguiente dirección: https://en.wikipedia.org/wiki/Quicksort
El **input** constará de dos partes. En primer lugar, con formato `"%d\n"` se indicará la cantidad N de enteros que se introducirán a continuación. La segunda parte de la entrada será una lista de enteros separados por espacios.
El **output** será una línea por cada iteración del algoritmo de ordenación en el que se mostrará una lista de números mostrando la ordenación parcial. Esta lista se imprimirá después de realizar la combinación de las dos listas. **OJO:** Observad que existe un espacio después de cada número incluso en el caso del último.

Input:	Output:
10 9 8 7 6 5 4 3 2 1 0	4 5 4 5 6 3 4 5 6 3 4 5 6 7 2 3 4 5 6 7 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

Quicksort Media(4 puntos): Implementa un algoritmo que permita ordenar una lista enlazada circular de enteros utilizando el algoritmo de ordenamiento denominado quicksort que es una solución basada en una estrategia divide y vencerás. En concreto usa el esquema de partición que elige como pivote la media entre el primer y último valor de la lista inicial. Para cada recursión, una vez ordenadas ambas sublistas se deberán concatenar con una operación O(1) así que será necesario incluir una operación de concatenación en la clase de ListaCircular que tenga dicha complejidad.

El **input** constará de dos partes. En primer lugar, con formato `"%d\n"` se indicará la cantidad N de enteros que se introducirán a continuación. La segunda parte de la entrada será una lista de enteros separados por espacios.

El **output** será una línea por cada iteración del algoritmo de ordenación en el que se mostrará una lista de números mostrando la ordenación parcial. Esta lista se imprimirá después de realizar la combinación de las dos listas.

OJO: Observad que existe un espacio después de cada número incluso en el caso del último.

Input:	Output:
10 9 8 7 6 5 4 3 2 1 0	0 1 0 1 2 0 1 2 3 4 3 4 0 1 2 3 4 5 6 5 6 7 5 6 7 8 9

	8 9 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9

NOTA: Para cada una de las funciones implementados se deberá incluir una pequeña descripción de su funcionamiento, sus precondiciones mediante assertdomjudge si las hubiera y el análisis de su complejidad temporal y espacial. Esta información deberá incluirse en la cabecera de cada función.