# Algoritmos y Estructuras de Datos II

TALLER - 29 de marzo 2022

## Laboratorio 2: Ordenación

Revisión 2018: Sergio CanchiRevisión 2021: Marco RocchiettiRevisión 2022: Marco Rocchietti

## Objetivos

- 1. Implementar el algoritmo de ordenación por inserción (*insertion-sort*)
- 2. Implementar el algoritmo de ordenación rápida (quick-sort)
- 3. Comparar desempeño de los algoritmos selection-sort, insertion-sort y quick-sort en distintos ejemplos
- 4. Lectura y comprensión del código entregado por la cátedra
- 5. Trabajar con implementaciones opacas de funciones leyendo su documentación
- 6. Abstraer la noción de orden
- 7. Usar procedimientos en C
- 8. Uso de funciones locales en módulos en C

# Ejercicio 1: Insertion Sort

Dentro de la carpeta ej1 se encuentran los siguientes archivos

Archivo	Descripción
array_helpers.h	Prototipos y descripciones de la funciones auxiliares para manipular arreglos.
array_helpers.c	Implementaciones de las funciones de la librería array_helpers
sort_helpers.h	Prototipos y descripciones de las funciones goes_before(), swap() y array_is_sorted()
sort_helpers.o	Archivo binario con las Implementaciones las funciones declaradas en sort_helpers.h (código compilado para la arquitectura x86-64)
sort.h	Prototipo de la función insertion_sort() y su descripción
sort.c	Contiene una implementación incompleta de insertion_sort(), falta implementar insert()
main.c	Programa principal que carga un <i>array</i> de números, luego lo ordena con la función insertion_sort() y finalmente comprueba que el arreglo sea permutación ordenada del que se cargó inicialmente.



Si se trabaja en una computadora con arquitectura distinta a x86-64, entonces seleccionar y renombrar uno de los siguientes archivos, sort\_helpers.o\_32 o sort\_helpers.o\_macos según la arquitectura de su máquina.

#### Parte A: Ordenación por Inserción

Se debe realizar una implementación del algoritmo de ordenación por inserción (alias *insertion-sort*). Para ello es necesario completar la implementación del "procedimiento" <code>insert()</code> en el módulo <code>sort.c</code>. Como guía se puede examinar el resto del archivo <code>sort.c</code> y la definición del algoritmo de ordenación por inserción vista en el teórico. El algoritmo debe ordenar con respecto a la relación <code>goes\_before()</code> declarada en <code>sort\_helpers.h</code> cuya implementación está oculta puesto que viene ya compilada en <code>sort\_helpers.o</code>.

#### Parte B: Chequeo de Invariante

Se debe modificar el "procedimiento" insertion\_sort() agregando la verificación de cumplimiento de la invariante del ciclo for que se vio en el teórico. Por simplicidad sólo se debe verificar la siguiente parte de la Invariante:

• el segmento inicial a[0,i) del arreglo está ordenado.

Para ello usar las funciones assert () y array is sorted().

#### Compilación

Una vez implementados los incisos a) y b), se puede compilar ejecutando:

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c main.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -no-pie array_helpers.o sort.o sort_helpers.o main.o -o sorter
```

la opción -no-pie tiene que ver con que se están "linkeando" los objetos array\_helpers.o, sort.o y main.o compilados en nuestra computadora con el objeto precompilado sort\_helpers.o, cuya compilación fue realizada en una computadora distinta. En consecuencia esta opción puede ser necesaria para lograr compatibilidad entre los archivos binarios durante el "linkeo" y así poder generar el ejecutable. El programa puede ejecutarse de la siguiente manera:

```
$ ./sorter ../input/example-unsorted.in
```

Si el programa funciona bien en ese ejemplo (es decir, si no reporta error), probar con otros archivos de la carpeta ../input, sin olvidar realizar una prueba con el archivo ../input/empty.in

Analizar los resultados del programa y responder: ¿Qué relación implementa la función goes before()?

## Ejercicio 2: Quick Sort I

En este ejercicio se realizará una implementación *top-down* del algoritmo de ordenación rápida vista en el teórico. En la carpeta **ej2** se encuentran los siguientes archivos:

Archivo	Descripción
array_helpers.h	Es el mismo que en el ejercicio anterior.
array_helpers.c	Es el mismo que en el ejercicio anterior.
sort_helpers.h	Contiene además la declaración y descripción de partition()
sort_helpers.o	Contiene implementaciones ilegibles de esas funciones (código compilado para la arquitectura x86-64)
sort.h	Contiene descripción de la función quick_sort()
sort.c	Contiene una implementación muy incompleta de quick_sort(), además falta implementar quick_sort_rec()
main.c	Contiene el programa principal que carga un arreglo de números, luego lo ordena con la función quick_sort() y finalmente comprueba que el arreglo sea una permutación ordenada del que se cargó inicialmente.



Si se trabaja en una computadora con arquitectura distinta a x86-64, entonces seleccionar y renombrar uno de los siguientes archivos, sort\_helpers.o\_32 o sort\_helpers.o\_macos según la arquitectura de su máquina.

## Parte A: Implmentación de quick sort rec()

Implementar el "procedimiento" quick\_sort\_rec() en el archivo sort.c. Tener en cuenta que no es necesario implementar la función partition() puesto que la misma ya está implementada (aunque no puede leerse su código por estar compilada en sort\_helpers.o). Para saber cómo utilizarla, examinar su descripción en sort\_helpers.h.

A modo de guía se puede revisar la presentación del algoritmo de ordenación rápida realizada en la <u>clase del teórico</u>.

#### Parte B: Función main()

Se debe abrir el archivo main.c y completar la función main() con una llamada al "procedimiento" quick sort(). Para entender cómo utilizar este "procedimiento", examinar el archivo sort.h.

## Compilación

Una vez completadas las partes A y B, compilar el código con gcc siguiendo el mismo método del ejercicio 1.

# Ejercicio 3: Quick Sort II

En la carpeta ej3 se encuentran los siguientes archivos

Archivo	Descripción
sort_helpers.h	Contiene descripciones de las funciones goes_before(), swap() y array_is_sorted()
sort_helpers.o	Contiene implementaciones ilegibles de todo lo descripto en sort_helpers.h (código compilado para la arquitectura x86-64). Notar que la función partition() no está más aquí.
sort.h	Contiene descripción de la función quick_sort()
sort.c	<pre>contiene una implementación incompleta de quick_sort(), falta implementar quick_sort_rec() y partition().</pre>



Si se trabaja en una computadora con arquitectura distinta a x86-64, entonces seleccionar y renombrar uno de los siguientes archivos, sort\_helpers.o\_32 o sort\_helpers.o\_macos según la arquitectura de su máquina.

Copiar los archivos array\_helpers.h, array\_helpers.c y main.c del ejercicio 2. Luego copiar el "procedimiento" quick\_sort\_rec() (también del ejercicio 2) en el archivo sort.c y definir allí la función partition() usando como guía la presentación que se dio del algoritmo de ordenación rápida en la clase del teórico.

## Compilación

Una vez completada la definición de partition(), compilar el código con gcc siguiendo el mismo método del ejercicio 1.

# Ejercicio 4: Versus

Realizar una comparación de todos los algoritmos de ordenación implementados en este laboratorio. En la carpeta **ej4** se encuentran los siguientes archivos:

Archivo	Descripción
sort_helpers.h	Se agregan nuevas declaraciones de funciones para manejo de contadores
sort_helpers.o	Contiene implementaciones ilegibles de todo lo descripto en sort_helpers.h (código compilado para la arquitectura x86-64)
sort.h	Contiene las declaraciones y descripciones de las implementaciones de los métodos de ordenación selection-sort, insertion-sort y quick-sort
sort.c	Contiene las definiciones incompletas de las funciones declaradas en sort.h. Deben completarse con el código de los ejercicios anteriores.
main.c	Contiene el programa principal que carga un arreglo de números, luego lo ordena usando alguno de los algoritmos de ordenación implementados y muestra:  • Tiempo de ejecución  • Número de comparaciones  • Intercambios realizados.



Si se trabaja en una computadora con arquitectura distinta a x86-64, entonces seleccionar y renombrar uno de los siguientes archivos, sort\_helpers.o\_32 o sort\_helpers.o\_macos según la arquitectura de su máquina.

Copiar los archivos array\_helpers.h y array\_helpers.c del ejercicio anterior y luego:

- 1. Abrir el archivo **sort.c** y copiar el código de cada uno de los algoritmos de ordenación resueltos en los ejercicios anteriores.
- 2. Abrir el archivo main.c y completar la función main() siguiendo los pasos indicados en los comentarios.

## Compilación y Ejecución

Una vez completados los ítems 1 y 2, compilar el código con gcc siguiendo el mismo método del ejercicio 1.

Analizar los resultados de la ejecución del programa para distintos ejemplos y sacar conclusiones sobre el desempeño de cada algoritmo de ordenación.