Ayudantía 3: Ordenación

Problema 1. Dinámica de introducción

Pedimos voluntarios y los marcamos con un numerito. Intentamos que los alumnos nos guíen en recordar cómo ordena cada uno de los algoritmos que se vieron (excepto *heap sort*, que se trabajará después en detalle).

Problema 2. Discriminando algoritmos

Supongamos que tenemos 5 cajas negras capaces de ordenar y se sabe que cada una corresponde a un algoritmo de ordenación (*selection sort, insertion sort, merge sort, quick-sort* y *heap sort*), sin saber cuál es cuál.

Proponga una estrategia basada en inputs de prueba para decidir qué caja corresponde a cada algoritmo.

Problema 3. Una implementación de Heaps

Desarrolle una implementación el algoritmo de *heap sort*. Para ello, se pide

1. Proponer un pseudocódigo *abstracto* del algoritmo
2. Proponer un pseudocódigo C-*like* del algoritmo
3. Programar una implementación en C, con la base que se entrega

Comentarios: en la carpeta `Ayudantias/A3` del repositorio del curso se encuentra la carpeta programa. Esta contiene una makefile para compilar el programa que vimos en la ayudantía. En el archivo `main.c` hay dos main, de forma que puedan ver que ambas estrategias de construcción del *max-heap* funcionan.

¡Ojo! Como comentamos, esta implementación del *max-heap* funciona aceptando un arreglo de enteros entregados al ejecutar el programa. Por ejemplo, luego de hacer `make` en el directorio programa, podrán ejecutar

```
./methods 3 4 3 10 9
```

y se mostrará el proceso de construcción del *heap* a partir del arreglo `{3, 4, 3, 10, 9}`, seguido de su ordenamiento en orden creciente. En cada etapa del ordenamiento se muestra el arreglo completo: primero la parte del arreglo que forma el *max-heap*, y segundo la parte ya ordenada. Ambas partes se muestran separadas por `||`. Para el ejemplo dado, uno de los pasos del ordenamiento muestra

```
| 4 | 3 | 3 || 9 | 10 |
```

donde la secuencia 9, 10 ya está ordenada.

Cabe destacar que esta implementación de *heap sort* no ocupa memoria adicional. Al investigar el código como hicimos en la ayudantía, podrán verificar que en todo momento se utiliza el arreglo dado al comienzo.

Problema 4. *Quick sort*

Quick Sort no se comporta bien cuando existen muchos datos repetidos en el arreglo a ordenar. Proponga una modificación al algoritmo visto en clases que permita mejorar esta situación.