Práctica 5

1. Lea la implementación provista de `bsort`[1].

2. Implemente el algoritmo selección para arreglos (`ssort`) y listas (`ssortl`). En la versión de listas, para llevar la parte ordenada de la lista, evalúe crear una lista nueva.

3. Implemente el algoritmo selección para arreglos (`isort`) y listas (`isortl`). En la versión de listas, para llevar la parte ordenada de la lista, evalúe crear una lista nueva.

4. Escriba una versión recursiva del algoritmo de selección sobre arreglos.

5. En el algoritmo burbuja, cambie la estructura `do { ... } while (sorted != 1)` por un `for` que repita el bloque cierta cantidad de veces fija dependiente de la longitud del arreglo. ¿Cuál es una cantidad apropiada de veces?

6. Escriba una función `int are_permutable(int *, int *, int)` que toma dos arreglos, y la longitud de los mismos, y verifica si una lista es permutación de otra, i.e. si tienen los mismos elementos en la misma cantidad, sin importar el orden [2].

7. Proponga versiones modificadas (`bsortg`, `ssortg`, `isortg`) de las versiones de ordenamiento implementadas. Estas nuevas funciones deberían ser generales, y trabajar con una función:

```
typedef int (*CmpFunc)(void *, void *);
```

8. Re-escriba el ejemplo de `main.c` para que funcione con `bsortg` en lugar de `bsort`.

9. Implemente una función de tipo `CmpFunc` que compare dos enteros de manera inversa a la usual. Utilícela para ordenar el arreglo $[1, 4, -2, 3]$ de mayor a menor.

10. Implemente una función de tipo `CmpFunc` que compare dos cadenas de caracteres. Esta función ya viene implementada en C, y está declarada en la cabecera `string.h`. Busque una implementación de esta función en internet o algún libro, y compárela con la suya.

11. Imagine que ejecuta el algoritmo de selección sobre un arreglo de 10 elementos. ¿Cuántas comparaciones (llamadas a la función `CmpFunc`) realiza? Haga un cálculo a mano, y luego programe una función de comparación que lleve cuenta de la cantidad de veces que ha sido llamada. Para esto último puede ser útil la keyword `static`. Repita esto para el algoritmo de inserción.

12. Imagine que tiene una estructura `carta` como fue definida en la práctica 0. Implemente una función `CmpFunc` que dadas dos cartas, las ordene por su número, sin importarle el palo que tengan.

13. En el ejercicio anterior, puede resultar que hayan cartas distintas que tienen el mismo número, y por ende no hay a una preferencia sobre cuál va antes.
Por ej., para el arreglo de cartas

$$[4b, 3c, 4c, 1o, 6o, 2b]$$

no hay una preferencia para poner a $4b$ antes que $4c$ o viceversa al ordenarlo.
Un algoritmo de ordenamiento se dice *estable* si mantiene el orden original de los elementos en caso de que no hubiera preferencia según la función de ordenamiento. Determinar cuáles de los algoritmos implementados son estables.

[1] `http://dcc.fceia.unr.edu.ar/~erivas/estructuras/src/ordenamiento/`

[2] Formalmente, una permutación entre dos arreglos a, b de longitud n es una función biyectiva $f : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ tal que $a_{[i]} = b_{[f(i)]}$, y el ejercicio se reduce a determinar si existe tal función permutación.