Algoritmos y Estructuras de Datos II 2020.

Laboratorio 1: Algoritmos de ordenamiento

Ejercicio 1: Ordenamiento por inserción

Dentro de la carperta ej1 vas a encontrar los siguientes archivos:

- **array_helpers.h** : contiene descripciones de funciones auxiliares para manipular arreglos.
- **array_helpers.c** : contiene implementaciones de dichas funciones.
- **sort_helpers.h** : contiene descripciones de goes_before, array_is_sorted y swap
- **sort_helpers.o** : contiene implementaciones ilegibles de esas funciones (código compilado para la arquitectura x86-64)

  Nota: Si usted esta trabajando en una computadora con arquitectura distinta a la mencionada, entonces seleccione y renombre uno de los siguientes archivos, sort_helpers.o_32 o sort_helpers.o_macos según la arquitectura de su máquina.
- **sort.h** : contiene descripción de la función insertion_sort
- **sort.c** : contiene una implementación incompleta de insertion_sort, falta implementar insert
- **main.c** : contiene el programa principal que carga un arreglo de números, luego lo ordena con la función insertion_sort y finalmente comprueba que el arreglo sea permutación ordenada del que cargó inicialmente.

Parte a: Implementación del algoritmo de ordenación por inserción

Para esta parte es necesario que abras el archivo sort.c e implementes el procedimiento insert. Para guiarte, no dudes en examinar el resto del archivo sort.c y la definición del algoritmo de ordenación por inserción que hemos visto en clase. El algoritmo debe ordenar con respecto a la relación goes_before, provista por sort_helpers.h

Parte b) Verificar el cumplimiento del invariante del "ciclo for del procedimiento insertion_sort"

Para esta parte es necesario que modifiques el procedimiento insertion_sort agregando

la verificación de cumplimiento del invariante del ciclo for del procedimiento insertion_sort visto en el teórico.

Por simplicidad solo verifique esta parte del Invariante:

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

Para esto debes hacer uso de las funciones assert y array_is_sorted.

Una vez implementado los incisos a) y b), compilá ejecutando:

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando:

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

Si anda bien (o sea, si no reporta error) probá con otros archivos de la carpeta ../input

no te olvides de probar con el archivo ../input/empty.in

¿Te das cuenta qué relación implementa la función goes_before?

Ejercicio 2: Implementación top-down del algoritmo de ordenación rápida, primera parte

En la carpeta ej2 se encuentran los siguientes archivos:

- **array_helpers.h** : es el mismo que en el ejercicio anterior.
- **array_helpers.c** : también.
- **sort_helpers.h** : contiene además una descripción de partition
- **sort_helpers.o** : contiene implementaciones ilegibles de todo lo descripto en sort_helpers.h (código compilado para la arquitectura x86-64)

  Nota: Si usted esta trabajando en una computadora con arquitectura distinta a la mencionada, entonces seleccione y renombre uno de los siguientes archivos, sort_helpers.o_32 o sort_helpers.o_macos segun la arquitectura de su maquina.
- **sort.h** : contiene descripción de la función quick_sort

- **sort.c** : contiene una implementación muy incompleta de quick_sort, 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 permutación ordenada del que cargó inicialmente.

a) Implemente el procedimiento quick_sort_rec en el archivo sort.c

Para esta parte es necesario que abras el archivo sort.c e implementes el procedimiento quick_sort_rec. Ojo: no es necesario que implementes la función partition puesto que la misma ya está implementada (salvo que no podés leer el código porque solo disponés de la versión compilada). Para saber cómo utilizarla, examiná su descripción en sort_helpers.h. A modo de guía para implementar quick_sort_rec, no dudes en revisar la presentación que hicimos del algoritmo de ordenación rápida en clase.

b) Completar la función main en el archivo main.c

Para esta parte es necesario que abras el archivo main.c y completes la función main con una llamada al procedimiento quick_sort. Para saber como utilizar el procedimiento quick_sort examiná el archivo sort.h

Una vez completados los incisos a) y b), compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

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

Ejercicio 3: Implementación top-down del algoritmo de ordenación rápida, segunda parte

En la carpeta ej3 se encuentran los siguientes archivos

- **sort_helpers.h** : contiene descripciones de goes_before, array_is_sorted y swap

- **sort_helpers.o** : contiene implementaciones ilegibles de todo lo descripto en sort_helpers.h (código compilado para la arquitectura x86-64)

  Nota: Si usted esta trabajando en una computadora con arquitectura distinta a la mencionada, entonces seleccione y renombre uno de los siguientes archivos, sort_helpers.o_32 o sort_helpers.o_macos segun la arquitectura de su maquina.
- **sort.h** : contiene descripción de la función quick_sort
- **sort.c** : contiene una implementación incompleta de quick_sort, falta implementar quick_sort_rec y partition

Ahora tenés que copiar los archivos array_helpers.h, array_helpers.c y main.c del ejercicio 2.

Luego copiar el procedimiento quick_sort_rec del ejercicio 2 en el archivo sort.c y definir la función partition.

Para finalizar la implementación de quick_sort tenés que abrir el archivo sort.c, copiar tu implementación de quick_sort_rec del ejercicio anterior y, ahora sí, implementar la función partition usando como guía la presentación que hicimos del algoritmo de ordenación rápida en clase.

Una vez implementada la función partition, compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

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

Ejercicio 4: Comparación de todos los algoritmos de ordenación implementados

En la carpeta ej4 estarán los siguientes archivos:

- **sort_helpers.h** : contiene además descripciones de manejadores de contadores
- **sort_helpers.o** : contiene implementaciones ilegibles de todo lo descripto en sort_helpers.h (código compilado para la arquitectura x86-64)

  Nota: Si usted esta trabajando en una computadora con arquitectura distinta a la mencionada, entonces seleccione y renombre uno de los siguientes archivos, sort_helpers.o_32 o sort_helpers.o_macos segun la arquitectura de su maquina.
- **sort.h** : contiene descripción de las funciones de ordenación implementadas
- **sort.c** : contiene una implementaciones incompletas de los algoritmos de ordenación trabajados
- **main.c** : contiene el programa principal que carga un arreglo de números, luego lo ordena uno de los algoritmos de ordenación implementados y muestra el tiempo de ejecución, número de comparaciones e intercambios realizados.

Ahora tenés que copiar los archivos array_helpers.h y array_helpers.c de un ejercicio

anterior.

a) Abrí el archivo sort.c y copiar el código de cada uno de los algoritmos de ordenación resueltos en los ejercios anteriores.

b) Abrí el archivo main.c y completar la función main siguiendo los pasos indicados en los comentarios.

Una vez completados los incisos a) y b), compilá ejecutando

```
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c sort.c
gcc -Wall -Werror -Wextra -pedantic -std=c99 -o sorter *.o main.c
```

y ya podés correr el programa, ejecutando

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