TAREA 3: ALGORITMOS DE ORDENAMIENTO

LA TAREA

Esta tarea es individual. Puedes elegir hacerla en Python o C, independientemente de la carrera que estudies.

El objetivo de esta tarea es implementar los 4 algoritmos de ordenamiento documentados en webcursos y evaluar empíricamente su complejidad computacional. Para esto, deberás completar el esqueleto de código disponible en webcursos (hay una versión para C y otra para Python) cumpliendo con los siguientes requerimientos:

1. No puedes cambiar los prototipos de las funciones. Es decir, el nombre de la función, su valor de retorno y los argumentos de entrada no pueden ser modificados.

2. Durante el desarrollo de tu tarea, puedes modificar el código principal para probar que tu código funciona correctamente (función main() en C y el código desde donde se invocan las funciones en Python), pero la entrega final debe tener el código principal tal cual se entrega en esta tarea.

3. También puedes agregar tus propias funciones si eso te facilita la programación

4. Las funciones a implementar son las siguientes:

FUNCIÓN 1

Nombre	BubbleSort
Argumentos de entrada	• A: arreglo con números (enteros) a ordenar • asc: variable booleana que indica el tipo de ordenamiento. Si es TRUE, el ordenamiento es ascendente (de menor a mayor). Si es FALSE, el ordenamiento es descendente (de mayor a menor)
Operación	La función debe ordenar los números que recibe en el arreglo A siguiendo el seudocódigo de BubbleSort entregado en webcursos. El orden debe ser de manera ascendente (si la variable `asc` es TRUE) o descendente (si la variable `asc` es FALSE)
Datos de salida	La función no retorna valor alguno. Como salida, debe agregar (no sobre escribir) en el archivo de salida Bubble.txt los siguientes datos: • El contenido del arreglo original y su tamaño • La elección de ordenamiento (ascendente o descendente) • El conjunto de números ordenado • El número de operaciones realizadas Recuerda, cada vez que se llama a la función, se debe agregar al final del archivo los nuevos datos (no sobre escribirlos).

Ejemplo de ejecución:

Supón que se ejecuta el siguiente seudo-código:

```
A[7]={4,2,7,4,8,2,9}
BubbleSort(A,1) //Bubble Sort debe ordenar de manera ascendente
BubbleSort(A,0) //BubbleSort debe ordenar de manera descendente
```

Entonces, en el archivo Bubble.txt debe aparecer la siguiente información:

```
- Arreglo Original: 4 2 7 4 8 2 9
- Tamaño: 7
- Asc: True
- Arreglo Ordenado: 2 2 4 4 7 8 9
- Operaciones realizadas: 78
- Arreglo Original: 4 2 7 4 8 2 9
- Tamaño: 7
- Asc: False
- Arreglo Ordenado: 9 8 7 4 4 2 2
- Operaciones realizadas: 83
```

Nota: El número de operaciones para este ejemplo ha sido inventado.

FUNCIÓN 2

Nombre	InsertionSort
Argumentos de entrada	• A: arreglo con números (enteros) a ordenar • asc: variable booleana que indica el tipo de ordenamiento. Si es TRUE, el ordenamiento es ascendente (de menor a mayor). Si es FALSE, el ordenamiento es descendente (de mayor a menor)
Operación	La función debe ordenar los números que recibe en el arreglo A siguiendo el seudocódigo de InsertionSort entregado en webcursos. El orden debe ser de manera ascendente (si la variable `asc` es TRUE) o descendente (si la variable `asc` es FALSE)
Datos de salida	La función no retorna valor alguno. Como salida, debe agregar en el archivo de salida Insertion.txt los siguientes datos: • El contenido del arreglo original y su tamaño • La elección de ordenamiento (ascendente o descendente) • El conjunto de números ordenado • El número de operaciones realizadas Recuerda, cada vez que se llama a la función, se debe agregar al final del archivo los nuevos datos (no sobre escribirlos).

FUNCIÓN 3

Nombre	SelectionSort
Argumentos de entrada	• A: arreglo con números (enteros) a ordenar • asc: variable booleana que indica el tipo de ordenamiento. Si es TRUE, el ordenamiento es ascendente (de menor a mayor). Si es FALSE, el ordenamiento es descendente (de mayor a menor)
Operación	La función debe ordenar los números que recibe en el arreglo A siguiendo el seudocódigo de SelectionSort entregado en webcursos. El orden debe ser de manera ascendente (si la variable `asc` es TRUE) o descendente (si la variable `asc` es FALSE)
Datos de salida	La función no retorna valor alguno. Como salida, debe agregar en el archivo de salida Selection.txt los siguientes datos: • El contenido del arreglo original y su tamaño • La elección de ordenamiento (ascendente o descendente) • El conjunto de números ordenado • El número de operaciones realizadas Recuerda, cada vez que se llama a la función, se debe agregar al final del archivo los nuevos datos (no sobre escribirlos).

FUNCIÓN 4

Nombre	CountingSort
Argumentos de entrada	• A: arreglo con números (enteros) a ordenar • k: Máximo valor en arreglo A • asc: variable booleana que indica el tipo de ordenamiento. Si es TRUE, el ordenamiento es ascendente (de menor a mayor). Si es FALSE, el ordenamiento es descendente (de mayor a menor)
Operación	La función debe ordenar los números que recibe en el arreglo A siguiendo el seudocódigo de InsertionSort entregado en webcursos. El orden debe ser de manera ascendente (si la variable `asc` es TRUE) o descendente (si la variable `asc` es FALSE)
Datos de salida	La función no retorna valor alguno. Como salida, debe agregar en el archivo de salida Counting.txt los siguientes datos: • El contenido del arreglo original y su tamaño • La elección de ordenamiento (ascendente o descendente)

	• El conjunto de números ordenado • El número de operaciones realizadas. Recuerda, cada vez que se llama a la función, se debe agregar al final del archivo los nuevos datos (no sobre escribirlos).

5. Para determinar el número de operaciones que ejecuta cada función debes crear una variable local (en cada función) llamada num_op que se incrementa en una unidad cada vez que:
   - Se ejecuta una asignación (con o sin operación aritmética incluida). Por ejemplo:
     - a=3
     - a=x+3 (en este caso en realidad se ejecutan dos operaciones, la asignación y la suma, pero como ya sabemos que las constantes no son importantes, solo contaremos una vez)
     - cuando se inicializa la variable de control de un ciclo for
     - cuando se actualiza la variable de control de un ciclo for
   - Se evalúa una condición. Por ejemplo: if (cambio==true), while(cambio), condición del ciclo for. No nos importará la complejidad de la condición. Por ejemplo, evaluar (x>y) toma menos operaciones que (x>y AND y<z), pero en ambos casos solo contaremos una vez.

   IMPORTANTE: No cuentes la operaciones relacionadas con la escritura al archivo de salida. Solo aquellas relacionadas con la ejecución del ordenamiento.

6. Un ejemplo del formato del archivo de salida de las funciones es el siguiente:

```
- Arreglo Original: 5 6 9 8 7 4
- Tamaño: 6
- Asc: True
- Arreglo Ordenado: 4 5 6 7 8 9
- Operaciones realizadas: 18
- Arreglo Original: 5 6 9 8 7 4 15 69 12565 1
- Tamaño: 10
- Asc: True
- Arreglo Ordenado: 1 4 5 6 7 8 9 15 69 12565
- Operaciones realizadas: 59
```

Revisa el archivo Ejemplo_Bubble.txt para confirmar que estás generando tu archivo en el formato correcto. Esto es MUY IMPORTANTE porque la revisión de la tarea se automatizará y por lo tanto, si no cumples EXACTAMENTE con los nombres que deben tener los archivos y su formato, tu tarea no podrá ser revisada.

7. Debes asegurarte de que tu programa funcionará correctamente para los siguientes casos “normales” y “extremos”:

- Arreglo vacío
- Arreglo ya ordenado
- Arreglo con todos los números iguales
- Arreglos con varios números iguales
- Arreglos de 10 a 10 millones de elementos

8. Tu archivo debe llamarse Tarea3.c o Tarea3.py. Si no usas ese nombre, no se podrá revisar (ya que se usarán tests automatizados).

LOS PLAZOS

Viernes 5 de Abril a las 10 AM: debes subir a webcursos dos documentos:

- documento con el código funcionando correctamente para el algoritmo BubbleSort.
- Documento de 1 página con un gráfico en el que se compare la complejidad computacional teórica de Bubble Sort vs. la complejidad computacional empírica (el número de operaciones que calcula tu programa), en función del tamaño del arreglo (desde 10 hasta 10 millones).

Viernes 12 de Abril a las 10 AM: debes subir a webcursos dos documentos:

- documento con el código funcionando correctamente para los 4 algoritmos de ordenamiento.
- Documento de 1 página con un gráfico en el que se compare la complejidad computacional teórica de los 4 algoritmos vs. la complejidad computacional empírica (el número de operaciones que calcula tu programa), en función del tamaño del arreglo (desde 10 hasta 10 millones).

LA EVALUACIÓN

Dado que el viernes 29 de Marzo se llevará a cabo una ayudantía donde se definirán los compiladores a usar, a partir de la tarea 2 no se revisarán las tareas que no puedan ser ejecutadas.

Si la tarea no puede ser ejecutada, tiene la nota mínima.
Si la tarea no cumple con los requerimientos, no podrá ser evaluada (y tiene la nota mínima).
Si la tarea puede ser ejecutada, se asignarán los siguientes puntajes:

Entrega del 5 de Abril:

- BubbleSort opera correctamente para los casos descritos, escribiendo los datos de salida correctamente en el archivo indicado (4 puntos)
- Documento de comparación completo y correcto (2 puntos)

Entrega del 12 de Abril:

- InsertionSort opera correctamente para los casos descritos (1.5 puntos)
- SelectionSort opera correctamente para los casos descritos (1.5 puntos)

- CountingSort opera correctamente para los casos descritos (1.5 puntos)
- Documento de comparación de los 4 algoritmos completo y correcto (1.5 puntos)

La ponderación de cada entrega es la siguiente:
Entrega del 5 de Abril: 25%
Entrega del 12 de Abril: 75%

La clase del día 5 de Abril se dedicará a atención de consultas. Pueden venir a programar en clases si así lo desean. Si prefieren quedarse en casa, programando mientras toman desayuno en cama, también pueden hacerlo.

El 5 de Abril también debe subir a webcursos la Actividad en Aula que empezaron el viernes 29 de Marzo (análisis asintótico de los algoritmos de ordenamiento).