Actividad #1

Algoritmos y Estructuras de Datos (08GIAR)

- **Fecha límite de entrega en 1ra Convocatoria: 16 de mayo de 2025 hasta las 23:59**
- **Fecha límite de entrega en 2da Convocatoria: 20 de junio de 2025 hasta las 23:59**

Objetivo y Descripción general:

El **objetivo** de esta actividad es que el alumnado desarrolle competencias resolviendo ejercicios prácticos relacionados con los siguientes contenidos:

- Eficiencia y Complejidad Algorítmica.
- Estructuras de Datos Lineares.

¡MUY IMPORTANTE! Leer cuidadosamente las siguientes instrucciones.

- **Solo debes resolver UNO de los problemas propuestos**.
- Escoge el problema que prefieras resolver y entrégalo como solución única. **Si entregas más de un problema, solo se evaluará/calificará el primero en orden del documento**.
- **Para cada problema está indicada su calificación máxima.**
- Asegúrate de incluir tus comentarios en el **código Python** y detallar en células de **MarkDown** del **Jupyter-Lab** cualquier decisión tomada en la solución del problema.

Instrucciones de entrega:

- El trabajo que debe realizar en esta actividad es **individual**.
- El informe de esta actividad se entregará a través de plataforma del aula en un único fichero **.ipynb** (**Notebook Jupyter Lab**) con el problema resuelto.
- **No se aceptarán entregas de la actividad fuera de los plazos establecidos**.
- Si durante la calificación de la actividad se detecta cualquier vestigio de fraude, copia o plagio, la actividad será penalizada con una calificación de **cero (0)** sin posibilidad de recuperación.

Problema 1: (máximo 8 puntos)

Enunciado:

(a) Dados dos arreglos (*arrays*) de enteros positivos del mismo tamaño, arr[] e index[], implemente un programa en **Python** de **complejidad temporal lineal**, **O(n)**, que reorganice los elementos de arr[] de acuerdo con el arreglo index[] dado.

Nota: Justifique debidamente por qué su solución es de complejidad temporal O(n).

Ejemplo:

Entrada:

arr[] = [10, 11, 12]

index[] = [1, 0, 2]

Salida:

arr[] = [11, 10, 12]

index[] = [0, 1, 2]

(b) Dado un arreglo (*array*) de números enteros, arr[], implemente un programa en **Python** de **complejidad temporal lineal**, **O(n)**, que mueva todos los ceros (0) de arr[] al final del arreglo mientras se mantiene el orden relativo de todos los elementos distintos de cero.

Nota: Justifique debidamente por qué su solución es de complejidad temporal O(n).

Ejemplos:

Entrada:

arr[] = [3, 1, 0, 7, 4, 0, 5, 0]

Salida:

arr[] = [3, 1, 7, 4, 5, 0, 0, 0]

Entrada:

arr[] = [3, 1, 7, 4, 5]

Salida:

arr[] = [3, 1, 7, 4, 5]

Problema 2: (máximo 9 puntos)

Enunciado:

(a) Dado un arreglo (*array*) de elementos de enteros, arr[], de longitud n, con valores en el rango de 0 a n−1 en el cual, además, puede aparecer -1 si un dado valor está ausente. Implementa un programa en **Python** de **complejidad temporal lineal**, **O(n)**, que reorganiza (en orden creciente) el arreglo dado de manera que arr[i] = i y, si el valor i no está presente, entonces escribe -1 en esa posición.

Nota: Justifique debidamente por qué su solución es de complejidad temporal O(n).

Ejemplo:

Entrada: arr[] = [-1, -1, 6, 1, 9, 3, 2, -1, 4, -1]

Salida: arr[] = [-1, 1, 2, 3, 4, -1, 6, -1, -1, 9]

Explicación: Este array tiene longitud 10. En el rango de 0 a 9 todos los valores excepto el 0, 5, 7 y 8 no están presentes. Por lo tanto, escribimos -1 en lugar de ellos.

(b) Implemente un programa en **Python** que, dada una cadena de símbolos, imprime la longitud de la subcadena más larga que no tenga caracteres repetidos.
Ejemplo:

Entrada: "bbbaaabcccc"

Salida: 3 (Subcadena "abc")

(c) ¿Cuál es la **complejidad temporal** (en términos de O, *big O*) del programa implementado en el inciso anterior ((b))? *Justifique debidamente su respuesta.*

Problema 3: (máximo 10 puntos)

Enunciado:

(a) Dada una estructura de datos **Cola** (**Queue ADT**) que soporta las operaciones estándar como **enqueue()** y **dequeue()**, la tarea es implementar en **Python** una estructura de datos **Pila** (**Stack ADT**) utilizando una **Cola**.
*Nota: Solo podrá utilizar el **recurso externo** de **Python*** `from collections import deque`.

(b) ¿Cuál es la **complejidad temporal** (en términos de O, *big O*) de las operaciones **push()** y **pop()** en el programa implementado en el inciso (a)? *Justifique debidamente su respuesta.*

(c) Implemente un programa en **Python** que evalúe si una expresión matemática con paréntesis está balanceada. Por ejemplo, la expresión `((2+3)*(4-1))` está balanceada, pero `((2+3)*4-1))` no lo está.

Notas: Deberá usar una estructura de datos **Pila** (**Stack ADT**). *El programa debe retornar* True *si la expresión está balanceada en sus paréntesis y* False *en caso contrario.*

BUEN TRABAJO