TRABAJO PRÁCTICO 1

PROBLEMA 1

Tenemos un array A de tamaño n con enteros diferentes (positivos, negativos o cero), ordenado de menor a mayor. Diseñar e implementar un algoritmo de División y Conquista que determine el índice *i* tal que *A[i]=i* o que indique si no existe tal *i*.

IMPORTANTE: el array ya viene ordenado. No incluir el ordenamiento en el análisis, en la determinación de la complejidad temporal ni en la medición de los tiempos de ejecución.

Se pide:

1. Análisis:
    a. Identificar supuestos, condiciones, limitaciones y/o premisas bajo los cuales funcionará el algoritmo desarrollado
    b. Describir los casos base, en cuántas partes se divide el problema y cómo se combinan los resultados parciales
2. Diseño:
    a. Incluir un Pseudocódigo
    b. Detallar las estructuras de datos utilizadas. Justificar su elección.
2. Seguimiento: Mostrar un ejemplo de seguimiento con un set de datos reducido
3. Complejidad: Realizar un análisis de la complejidad temporal a partir del pseudocódigo. Justificar aplicando el Teorema Maestro.
4. Sets de datos: diseñar sets de datos apropiados.
    a. Se pueden generar utilizando una función random con una semilla fija, para permitir la reproducibilidad de los resultados, o ser generados externamente e incluidos como archivos que lee el programa.
    b. Cada set de datos debe ser incluido en la entrega, junto con el resultado obtenido en cada caso.
    c. El programa entregado debe generar los sets de datos en tiempo de ejecución o leerlos desde archivos incluidos en la entrega.
5. Tiempos de Ejecución: medir los tiempos de ejecución de cada set de datos y presentarlos en un gráfico.
6. Informe de Resultados:
    a. Redactar un informe de resultados comparando los tiempos de ejecución con la complejidad temporal.
    b. El gráfico comparativo de tiempos debe incluir tanto la curva con los valores medidos como la curva correspondiente a complejidad temporal determinada.

PROBLEMA 2

Tenemos otro array A de tamaño n, como el del Problema 1, pero que no está ordenado. Llamaremos *intervalo positivo* al subarray *A[i..j] (i≠j)* tal que la suma de sus valores es mayor (estricto) que cero. Diseñar e implementar un algoritmo Greedy que calcule la mínima cantidad de intervalos positivos en todo el array A. Por ejemplo, si el array fuera el siguiente, el algoritmo debería devolver 3 (los *intervalos positivos* se identifican en gris):

+3	-5	+7	-4	+1	-8	+3	-7	+5	-9	+5	-2	+4

Se pide:

1. Análisis:
    a. Identificar supuestos, condiciones, limitaciones y/o premisas bajo los cuales funcionará el algoritmo desarrollado
    b. Describir la regla Greedy que sigue el algoritmo
    c. Analizar (y justificar) si el algoritmo propuesto encuentra siempre una solución óptima
2. Diseño:
    a. Incluir un Pseudocódigo
    b. Detallar las estructuras de datos utilizadas. Justificar su elección.
3. Seguimiento: Mostrar un ejemplo de seguimiento con un set de datos reducido
4. Complejidad: Realizar un análisis de la complejidad temporal a partir del pseudocódigo
5. Sets de datos: diseñar sets de datos apropiados.
    a. Se pueden generar utilizando una función random con una semilla fija, para permitir la reproducibilidad de los resultados, o ser generados externamente e incluidos como archivos que lee el programa.
    b. Cada set de datos debe ser incluido en la entrega, junto con el resultado obtenido en cada caso.
    c. El programa entregado debe generar los sets de datos en tiempo de ejecución o leerlos desde archivos incluidos en la entrega.
6. Tiempos de Ejecución: medir los tiempos de ejecución de cada set de datos y presentarlos en un gráfico.
7. Informe de Resultados:
    a. Redactar un informe de resultados comparando los tiempos de ejecución con la complejidad temporal.
    b. El gráfico comparativo de tiempos debe incluir tanto la curva con los valores medidos como la curva correspondiente a complejidad temporal determinada.

PROBLEMA 3

Tenemos un array A de números enteros (positivos, negativos o cero) como el del Problema 2, no ordenado. Queremos encontrar la secuencia de uno o más elementos contiguos que sumen el máximo valor posible.
Desarrollar un algoritmo de Backtracking para encontrar dicha secuencia.

Se pide:

1. Análisis:
    a. Identificar supuestos, condiciones, limitaciones y/o premisas bajo los cuales funcionará el algoritmo desarrollado
    b. Describir las podas que realiza el algoritmo
    c. Analizar el tamaño del espacio de soluciones factibles del problema
2. Diseño:
    a. Incluir un Pseudocódigo
    b. Detallar las estructuras de datos utilizadas. Justificar su elección.
3. Seguimiento: Mostrar un ejemplo de seguimiento con un set de datos reducido
4. Complejidad: Realizar un análisis de la complejidad temporal a partir del pseudocódigo. Comparar con el tamaño del espacio de soluciones factibles.
5. Sets de datos: diseñar sets de datos apropiados.
    a. Se pueden generar utilizando una función random con una semilla fija, para permitir la reproducibilidad de los resultados, o ser generados externamente e incluidos como archivos que lee el programa.
    b. Cada set de datos debe ser incluido en la entrega, junto con el resultado obtenido en cada caso.
    c. El programa entregado debe generar los sets de datos en tiempo de ejecución o leerlos desde archivos incluidos en la entrega.
6. Tiempos de Ejecución: medir los tiempos de ejecución de cada set de datos y presentarlos en un gráfico.
7. Informe de Resultados:
    a. Redactar un informe de resultados comparando los tiempos de ejecución con la complejidad temporal.
    b. El gráfico comparativo de tiempos debe incluir tanto la curva con los valores medidos como la curva correspondiente a complejidad temporal determinada.

PROBLEMA 4

Para el Problema 3, desarrollar un algoritmo de Programación Dinámica que encuentre la secuencia que sume el máximo valor posible.

Se pide:

1. Análisis:
    a. Identificar supuestos, condiciones, limitaciones y/o premisas bajo los cuales funcionará el algoritmo desarrollado
    b. Definir la Ecuación de Recurrencia
    c. Justificar el cumplimiento de los requisitos de Subestructura Óptima y Subproblemas Superpuestos
    d. Indicar cómo usa Memoization
2. Diseño:
    c. Incluir un Pseudocódigo
    d. Detallar las estructuras de datos utilizadas. Justificar su elección.
7. Seguimiento: Ejemplo de seguimiento con un set de datos reducido
8. Complejidad: Análisis de la complejidad temporal a partir del pseudocódigo
9. Sets de datos: Utilizar los sets de datos generados para el Problema 3
10. Tiempos de Ejecución: medir los tiempos de ejecución de cada set de datos y presentarlos en un gráfico.
11. Informe de Resultados:
    a. Redactar un informe de resultados comparando los tiempos de ejecución con la complejidad temporal.
    b. El gráfico comparativo de tiempos debe incluir tanto la curva con los valores medidos como la curva correspondiente a complejidad temporal determinada.
    c. Los algoritmos de los Problemas 3 y 4 resuelven la misma situación. Comparar las complejidades computacionales de ambos: ¿Cuál debería ser más eficiente? Comparar los tiempos de ejecución de ambos: ¿Cuál es más eficiente? ¿Por qué puede ser esto?

Condiciones Generales de Entrega

- El trabajo debe ser entregado en un archivo zip conteniendo:
    - Documento con carátula, índice y numeración de páginas. La carátula debe incluir nombre y padrón de los integrantes del grupo. Debe presentarse en formato PDF.
    - Archivos con los fuentes de los algoritmos desarrollados.
    - Archivo README indicando el lenguaje de programación utilizado, versión mínima y bibliotecas requeridas, e instrucciones para ejecutar.
    - Archivos con los sets de datos utilizados
    - Archivos con resultados obtenidos para cada set de datos
- IMPORTANTE: el tamaño de los sets de datos debe ser tal que permita obtener suficiente información como para graficar los tiempos de ejecución y verificar la complejidad temporal.
- Aquí no toleramos el plagio: Por ello, se pueden incluir citas en el texto del informe siguiendo el modelo propuesto por Rivas (2022) y luego incorporar el listado completo en un anexo al final, usando normas APA 7ma edición.

Referencias

- Rivas, A. (2022). Cómo hacer una lista de referencias con Normas APA. Guía Normas APA. https://normasapa.in/como-hacer-la-lista-de-referencias/