Solemne 1

Estructuras de Datos y Algoritmos (CIT-2006) **1ER** Semestre 2023

1)	
2)	
3)	
Nota	

Nombre: _____ Rut: _____

- Cada respuesta debe ser justificada con claridad.
- Durante el desarrollo de la prueba no se responde ningún tipo de pregunta.
- Ningún alumno puede salir de la sala durante el desarrollo de la evaluación.

DE ACUERDO AL ARTÍCULO 50 DEL REGLAMENTO DE ESTUDIANTES DE PREGRADO

"Cualquier conducta de un o una estudiante que vicie o tienda a viciar una actividad o evaluación académica, ya sea que se ejecute antes, durante o luego de su realización, dará origen a una o más de las siguientes sanciones, según la gravedad de la falta cometida: nota mínima (1,0) en la respectiva evaluación; reprobación del curso respectivo; suspensión por un semestre o año académico, o expulsión de la Universidad."

Pregunta 1: Algoritmo Misterioso

Usted recibe una carta con un algoritmo misterioso en ella. Dado que usted es un experto en la materia, responderá las siguientes preguntas acerca de este misterioso algoritmo:

```
1     static String AlgoritmoMisterioso(String s) {
2        Stack<Character> st = new Stack<Character>();
3        st.push(s.charAt(0));
4        for(int i = 1; i < s.length(); i++) {
5            if(st.size()!= 0 && st.peek() == s.charAt(i)) {
6                st.pop();
7                continue;
8            }
9            st.push(s.charAt(i));
10       }
11
12       String newS = "";
13       while(st.size()!= 0) {
14           newS = st.pop() + newS;
15       }
16       return newS;
17    }
```

Analicemos el código para entender su comportamiento:

1. Se crea una pila vacía `st`.
2. Se empuja el primer carácter de la cadena `s` a la pila.
3. Se itera a través de cada carácter en la cadena `s`, empezando por el segundo carácter:
   a) Si la pila no está vacía y el carácter en la cima de la pila es igual al carácter actual, se elimina el carácter en la cima de la pila y se continúa con la siguiente iteración.
   b) Si no, se empuja el carácter actual a la pila.
4. Finalmente, se construye una nueva cadena `newS` sacando caracteres de la pila hasta que esté vacía.

1. Para cada uno de los siguientes casos de prueba, indique el valor retornado por el método `AlgoritmoMisterioso` al recibir el caso como argumento: **(12 puntos)**

   - yyz:
     - Empuja 'y' a la pila.
     - El siguiente carácter es 'y', que es igual al carácter en la cima de la pila, así que se quita 'y' de la pila.

- Empuja 'z' a la pila.
- La cadena resultante es "z".

- abccba
  - Empuja ' a' a la pila.
  - Empuja ' b' a la pila.
  - Empuja ' c' a la pila.
  - El siguiente carácter es 'c', que es igual al carácter en la cima de la pila, así que se quita 'c' de la pila.
  - El siguiente carácter es 'b', que es igual al carácter en la cima de la pila, así que se quita 'b' de la pila.
  - El siguiente carácter es ' a', que es igual al carácter en la cima de la pila, así que se quita ' a' de la pila.
  - La cadena resultante es una cadena vacía.
- casas
  - Empuja ' c' a la pila.
  - Empuja ' a' a la pila.
  - Empuja ' s' a la pila.
  - Empuja ' a' a la pila.
  - Empuja ' s' a la pila.
  - No hay caracteres adyacentes duplicados en la cadena, por lo que todos los caracteres se empujan a la pila uno tras otro.
  - La cadena resultante se construye sacando los caracteres de la pila y es casas.
- felicitaciones
  - Empuja cada carácter a la pila ya que no hay caracteres adyacentes duplicados.
  - La cadena resultante es "felicitaciones".

2. En base a sus observaciones del punto anterior, explique en no más de **3 lineas**, ¿De qué forma el algoritmo modifica los datos que recibe? **(3 puntos)**

   a) El algoritmo procesa la cadena eliminando pares consecutivos de caracteres idénticos; si la eliminación crea nuevos pares consecutivos, estos también se eliminan hasta que no queden repeticiones adyacentes.

   b) Al recorrer la cadena, el algoritmo identifica y descarta caracteres que se repiten de manera consecutiva. Si esta eliminación genera nuevos duplicados consecutivos, el proceso se repite hasta que la cadena esté libre de tales repeticiones.

   c) El algoritmo elimina caracteres consecutivos duplicados y, si esta acción resulta en nuevos pares de caracteres idénticos, continúa eliminándolos hasta que la cadena no tenga caracteres repetidos adyacentes.

3. Analice el algoritmo en cuestión y describa su tiempo de ejecución en términos de $\mathcal{O}(f(n))$, dónde n es el largo del String recibido como argumento y f es una función matemática propuesta por usted. Fundamente su respuesta. **(15 puntos)**
   Hint: Para efectos prácticos en su análisis considere que la concatenación de valores de tipo String usando el operador $+$, tiene complejidad $\mathcal{O}(1)$.

   La notación $O(n)$ denota la complejidad temporal de un algoritmo en términos de cómo crece su tiempo de ejecución con respecto al tamaño de la entrada, donde n es el tamaño de la entrada.
   Para el algoritmo `AlgoritmoMisterioso`:

   a) **Recorrido de la cadena:** El algoritmo recorre la cadena s una vez, lo que toma $O(n)$ tiempo.

   b) **Operaciones de pila:** Las operaciones `push` y `pop` en una pila tienen una complejidad temporal constante, $O(1)$, para cada carácter.

   c) **Construcción de la cadena resultante:** Al final, el algoritmo construye la cadena resultante sacando todos los caracteres de la pila, lo cual también es $O(n)$ en el peor de los casos.

   Al combinar estos pasos, todas las operaciones dentro del algoritmo son proporcionales a n. Por lo tanto, la complejidad total del algoritmo es $O(n)$. Esto significa que el algoritmo realiza un trabajo proporcional al tamaño de la entrada y no tiene bucles anidados o recursividad que multiplique el número de operaciones.

Pregunta 2: Dos en uno

En tu trabajo como desarrollador de software, te han encargado la tarea de analizar y correguir un código que une dos listas ordenadas en una sola.

```
1     class Node {
2         int value;
3         Node next;
4
5         public Node(int value) {
6             this.value = value;
7         }
8     }
9
10    public static Node merge(Node l1, Node l2) {
11        if (l1 == null) return l2;
12        if (l2 == null) return l1;
13
14        Node head = null, tmp = null;
15
16        if (l1.value <= l2.value) {
17            head = l1;
18        } else {
19            head = l2;
20        }
21        tmp = head;
22
23        while (l1 != null && l2 != null) {
24            if (l1.value >= l2.value) {
25                tmp.next = l1;
26                l1 = l1.next;
27            } else {
28                tmp.next = l2;
29                l2 = l2.next;
30            }
31            tmp = tmp.next;
32        }
33
34        if (l1 == null) {
35            tmp.next = l2;
36        } else {
37            tmp.next = l1;
38        }
39
40        return head;
41    }
```

El método `merge` recibe dos cabezas de lista, donde ambas listas están ordenadas, y retorna la cabeza de una nueva lista que contiene ordenados todos los nodos de ambas listas. La nueva lista utiliza los nodos originales de las listas de tal forma que no se crean nodos nuevos. El problema es que la funcion tiene ciertos errores que no permiten su correcto funcionamiento y es su deber encontrar y corregir dichos errores. Considerando lo anterior, se solicita que responda lo siguiente:

1. Explique qué problemas presenta el código presentado y por qué está fallando. **(10 puntos)**

   El código presentado intenta combinar dos listas enlazadas ordenadas en una sola lista ordenada. Sin embargo, hay errores en la implementación. A continuación, se describen los problemas:

   a) **Asignación de la cabeza (`head`):** Al intentar decidir qué nodo debe ser la cabeza de la nueva lista combinada, el código simplemente compara `l1.value` y `l2.value` y asigna `head` según cuál sea menor. Sin embargo, esto no actualiza los punteros `l1` y `l2` correctamente. Si `l1.value` es menor, entonces `l1` debería avanzar a `l1.next` (y viceversa para `l2`). Si no se actualiza `l1` o `l2` en este paso inicial, terminaremos comparando y fusionando el mismo nodo dos veces, lo que puede dar lugar a resultados incorrectos o incluso a un bucle infinito.

   b) **Condición de comparación en el bucle `while`:** Dentro del bucle `while`, el código compara `l1.value` $\geq$ `l2.value` para decidir a cuál nodo asignar a `tmp.next`. El problema es que esto es exactamente opuesto a

la condición que queremos. Debería ser `l1.value` $\leq$ `l2.value`. Como está, si `l1.value` es menor o igual a `l2.value`, se asignará `l2` a `tmp.next` y viceversa, lo que dará lugar a una lista enlazada desordenada.

c) **Conexión de la cola de la lista:** Después del bucle `while`, el código intenta conectar la cola de la lista. Sin embargo, debido a cómo se ha escrito el bucle `while`, al menos uno de `l1` o `l2` siempre será `null` en este punto. La lógica actual es correcta, pero es redundante, ya que el bucle ya garantiza que uno de ellos será `null`. Así que esto podría simplificarse.

Para corregir estos problemas:

a) Después de decidir el nodo cabeza, debemos avanzar `l1` o `l2` respectivamente.

b) Cambiar la condición en el bucle `while` de $\geq$ a $\leq$.

c) (Opcional) Simplificar la conexión de la cola de la lista al final.

2. Indique las líneas o bloques de código que deben ser cambiados y cuáles serían los cambios para corregir los errores detectados. Si lo desea, puede reescribir la función completa. **(10 puntos)**

a) **Asignación de la cabeza (head) y actualización de *l*1 o *l*2**:

Líneas originales:

```
1     if (l1.value <= l2.value) {
2         head = l1;
3     } else {
4         head = l2;
5     }
6     tmp = head;
7
```

Cambios:

- Si ***l*1.**value es menor o igual que ***l*2.**value, se debe avanzar ***l*1** al siguiente nodo.
- Si ***l*1.**value es mayor que ***l*2.**value, se debe avanzar ***l*2** al siguiente nodo.

Líneas corregidas:

```
1     if (l1.value <= l2.value) {
2         head = l1;
3         l1 = l1.next;
4     } else {
5         head = l2;
6         l2 = l2.next;
7     }
8     tmp = head;
9
```

b) **Condición de comparación en el bucle while**:

Línea original:

```
1     if (l1.value >= l2.value) {
2
```

Cambio:

- Cambiar la condición $\geq$ por $\leq$.

Línea corregida:

```
1     if (l1.value <= l2.value) {
2
```

c) **Conexión de la cola de la lista**:

Líneas originales:

```
    if (l1 == null) {
        tmp.next = l2;
    } else {
        tmp.next = l1;
    }

```

Cambio:

- Dado que al menos uno de $l1$ o $l2$ siempre será null al final del bucle, podemos simplificar esta lógica. Si $l1$ no es null, conectamos $l1$. Si $l1$ es null, conectamos $l2$.

Líneas corregidas:

```
    tmp.next = (l1 != null) ? l1 : l2;

```

Con estas correcciones, la función `merge` debería funcionar correctamente.

3. Analice el algoritmo en cuestión y describa su tiempo de ejecución en términos de $\mathcal{O}(f(n,m))$, dónde n y m son la cantidad de datos en la primera y segunda lista y f es una función matemática propuesta por usted. Fundamente su respuesta. **(10 puntos)**

   Analisis del tiempo de ejecución del algoritmo paso a paso:

   a) **Asignación de la cabeza (head):**
   La asignación de la cabeza de la lista resultante y la actualización de los punteros $l1$ o $l2$ es una operación constante, por lo que tiene un tiempo de ejecución de $O(1)$.

   b) **Bucle while:**
   El bucle `while` se ejecuta mientras ambas listas tengan elementos. En el peor de los casos, recorrerá todos los nodos de ambas listas. Dado que la primera lista tiene n nodos y la segunda lista tiene m nodos, el bucle se ejecutará $n + m$ veces en el peor de los casos.
   Dentro del bucle, todas las operaciones (comparaciones, asignaciones) son constantes y tienen un tiempo de ejecución de $O(1)$. Por lo tanto, el bucle en su conjunto tiene un tiempo de ejecución de $O(n + m)$.

   c) **Conexión de la cola de la lista:**
   La conexión de la cola de la lista, después del bucle `while`, es una operación constante, por lo que tiene un tiempo de ejecución de $O(1)$.

   Combinando todos estos pasos, el tiempo de ejecución total del algoritmo es:

$$O(1) + O(n + m) + O(1) = O(n + m)$$

   Por lo tanto, la función $f(n,m)$ en este caso es $f(n,m) = n + m$, y el tiempo de ejecución del algoritmo es $O(n + m)$.

   Fundamentación: La base de este análisis es que estamos combinando dos listas enlazadas ordenadas. La naturaleza de este problema implica que debemos recorrer cada elemento de ambas listas una vez para combinarlas correctamente. Por lo tanto, el tiempo de ejecución está dominado por la longitud total de ambas listas, lo que se refleja en la complejidad $O(n + m)$. Todas las demás operaciones en el algoritmo (como comparaciones y asignaciones) son constantes en relación con el tamaño de las listas y, por lo tanto, no afectan a la complejidad general del algoritmo.

En las explicaciones y propuestas de cambio se recomienda hacer referencia a los números de línea que aparecen en el listing del código.

Para revisar el funcionamiento de la funcion le recomendamos hacer la simulacion de los siguientes casos:

```
Caso 1:
    Input:
        l1: 1 -> 3 -> 5 -> 7 -> null
        l2: 2 -> 4 -> 6 -> 8 -> null
    Output:
        1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> null

Caso 2:
   Input:
        l1: 1 -> 3 -> 5 -> 7 -> null
        l2: null
    Output:
        1 -> 3 -> 5 -> 7 -> null

Caso 3:
    Input:
        l1: null
        l2: 2 -> 4 -> 6 -> 8 -> null
    Output:
        2 -> 4 -> 6 -> 8 -> null
```

Pregunta 3: La mejor alternativa

Para cada consulta debe indicar la mejor respuesta.

1. ¿Cuáles son las operaciones básicas de una pila (cola LIFO)?

   a) Agregar un elemento a la cima de la pila y sacar el elemento del fondo de la pila.

   b) Agregar un elemento a la cima de la pila y editar el elemento en la cima de la pila.

   c) Agregar un elemento a la cima de la pila y sacar el elemento de la cima de la pila.

   d) Agregar un elemento al fondo de la pila y sacar el elemento de la cima de la pila.

   Una pila (LIFO: Last In, First Out) tiene dos operaciones principales:

   a) **Push**: Agrega un elemento a la cima de la pila.

   b) **Pop**: Saca el elemento de la cima de la pila.

   Respuesta: (c) Agregar un elemento a la cima de la pila y sacar el elemento de la cima de la pila.

2. ¿Cuál es la complejidad de Insertion-Sort al ejecutarse sólo con datos de entrada ordenados de forma creciente? (Suponiendo que los datos de salida deben ser ordenados de forma creciente.)

   a) $\mathcal{O}(\log(n))$

   b) $\mathcal{O}(n)$

   c) $\mathcal{O}(n \log(n))$

   d) $\mathcal{O}(n^2)$

   Si los datos ya están ordenados de forma creciente, el Insertion-Sort tendrá un comportamiento lineal, ya que no necesitará realizar intercambios.

   Respuesta: *b*) $O(n)$

3. ¿Cuáles son los algoritmos de ordenamiento estables (que no cambian el orden relativo de los elementos iguales con respecto a la relación de orden considerada)?

   a) Insertion-Sort y Merge-Sort.

   b) Selection-Sort.

   c) Selection-Sort y Merge-Sort.

   d) Merge-Sort.

   Un algoritmo de ordenamiento es estable si dos registros con claves iguales aparecen en el mismo orden en el archivo ordenado que en el archivo de entrada. Insertion-Sort y Merge-Sort son algoritmos de ordenamiento estables. Por otro lado, Selection-Sort no es estable en su implementación básica.

   Respuesta: *a*) Insertion-Sort y Merge-Sort.

4. ¿Cuál es la complejidad de Merge-Sort en el peor caso?

   a) $\mathcal{O}(\log(n))$

   b) $\mathcal{O}(n)$

   c) $\mathcal{O}(n \log(n))$

   d) $\mathcal{O}(n^2)$

   La complejidad de Merge-Sort en el peor caso (y en todos los casos, de hecho) es $O(n \log(n))$.

   Respuesta: *c*) $O(n \log(n))$

5. ¿Cuánto puede crecer, como máximo, el tiempo de ejecución de un algoritmo de complejdad cúbica ($\mathcal{O}(n^3)$) al duplicar la cantidad de datos de entrada?

a) **2**

b) **3**

c) **6**

d) **8**

Si un algoritmo tiene complejidad $\boldsymbol{O(n^3)}$, al duplicar $\boldsymbol{n}$ (haciéndolo $\boldsymbol{2n}$), el tiempo de ejecución crecería por el factor $\frac{(2n)^3}{n^3} = \frac{8n^3}{n^3} = \mathbf{8}$.

Respuesta: *d*) 8

Esta pregunta sirve de bonus. Cada respuesta correcta vale **1 punto**. El puntaje final de la prueba será el mínimo entre el puntaje total obtenido entre las tres preguntas y **60**.