**Análisis de complejidad temporal**

i) add(E element) [Class CustomLinkedList]

1 public void add(E element) {

2 Node<E> newNode = new Node<>(element);

3 newNode.next = head;

4 head = newNode;

5 size++;

6 }

La línea Node<E> newNode = new Node<>(element) tiene una complejidad temporal O(1) (constante), porque sólo asigna e inicializa un objeto.

La línea newNode.next = head tiene una complejidad temporal O(1) (constante), porque sólo asigna un nuevo valor a un objeto.

La línea head = newNode tiene una complejidad temporal O(1) (constante), porque sólo asigna un nuevo valor a un objeto.

La línea size++ tiene una complejidad temporal O(1) (constante), porque sólo asigna un nuevo valor a un objeto.

Por ende, la complejidad temporal del algoritmo es O(1) (constante).

ii) get(int index) [Class CustomLinkedList]

1 public E get(int index) {

2 if (index < 0 || index >= size) {

3 throw new IndexOutOfBoundsException("Index out of bounds");

4 }

5 Node<E> current = head;

6 for (int i = 0; i < index; i++) {

7 current = current.next;

8 }

9 return current.data;

10 }

Las líneas 2 - 4 sólo hacen una comparación, por lo que su complejidad es O(1).

La línea 5 sólo asigna un valor a un objeto, por lo que su complejidad es O(1).

Las líneas 6 y 7 ocurren dentro de un for, por lo que su complejidad es, en el peor de los casos, O(N).

La línea 10 es un return, por lo que su complejidad es O(1).

Por ende, la complejidad temporal del algoritmo es O(N) (lineal).

**Análisis de complejidad espacial**

i) add(E element) [Class CustomLinkedList]

(mismo código)

El método sólo asigna un nuevo espacio de memoria para el nuevo nodo, por lo que su complejidad es O(1) (no aumenta a medida que se recorre la lista).

Por ende, la complejidad temporal del algoritmo es O(1) (constante).

ii) ii) resize() [Class CustomHashMap]

private void resize() {

Entry<K, V>[] oldTable = table;

table = new Entry[table.length \* 2];

size = 0;

for (Entry<K, V> entry : oldTable) {

while (entry != null) {

put(entry.key, entry.value);

entry = entry.next;

}

}

}

La línea 1 asigna un espacio de memoria a una nueva variable, oldTable, por lo que su complejidad es O(1).

Las líneas 2 y 3 del método sólo modifican valores de variables ya designadas previamente, por lo que su complejidad también es O(1).

El for tiene complejidad O(N) porque recorre todos los elementos en oldTable.

El while