# Práctica 3

Matú Hernández Diana

Rojo Mata Daniel

---

El algoritmo que realiza la eliminación de los elementos repetidos en la lista es el siguiente:

---

- Algoritmo: eliminarRpetidos
- Entrada: Un arreglo de números enteros, llamado arregloConRepetidos.
- Salida: Un arreglo de números enteros en donde no están repetidos los elementos, llamado, arregoSinRepetidos.

```
1  public static int[] eliminarRepetidos(int[] arregloConRepetidos) {
2
3      int tamañoSinRepetidos = 1;
4
5      for (int i = 1; i < arregloConRepetidos.length; i++) {
6        if (arregloConRepetidos[i] != arregloConRepetidos[i - 1]) {
7          tamañoSinRepetidos++;
8        }
9      }
10
11     int[] arregloSinRepetidos = new int[tamañoSinRepetidos];
12     arregloSinRepetidos[0] = arregloConRepetidos[0];
13
14     int j = 1;
15
```

```
16      for (int i = 1; i < arregloConRepetidos.length; i++) {
17        if (arregloConRepetidos[i] != arregloConRepetidos[i - 1]) {
18          arregloSinRepetidos[j] = arregloConRepetidos[i];
19          j++;
20        }
21      }
22      return arregloSinRepetidos;
23    }
```

### Corrección del algoritmo

Para diseñar el algoritmo se hace uso de dos ciclos *for*, a saber, el ciclo 5-9 catalogado como *ciclo 1*, y el ciclo 16-21, llamado *ciclo 2*.

El *ciclo 1* es usado para contabilizar cuantos elementos no se repiten dentro del arreglo de entrada (arregloConRepetidos), mientras que el *ciclo 2* usa un nuevo arreglo en donde se almacenan los valores que no se han repetido del arreglo inicial, es así que el ciclo 1 no es usado (directamente) para resolver el problema.

Es por ello que se propone un invariante para el *ciclo 2* quien es el encargado de construir el arreglo de salida.

Se propone el siguiente invariante para el ciclo 2, en donde, por notación, $A[0:j]$ denota todos los elementos del arreglo $A$ desde el índice 0 hasta el índice $j$.

> *LEMA: Después de la iteración $i$ el subarreglo, $arregloSinRepetios[0:j]$ contiene elementos únicos de $arregloConRepetidos[0:i]$.*

Dicho de otro modo:

> *LEMA: Después de la iteración $i$, el elemento $j$-ésimo de arregloSinRepetidos no se ha repetido.*

Al inicio del *ciclo 2*, $j$ es igual a 1 y $arregloSinRepetidos[0]$ contiene el primer elemento del arreglo original $arregloConRepetidos[0]$, y hasta este momento, *arregloSinRepetidos* contiene elementos únicos del arreglo inicial, así, el invariante es cierto.

A medida que el *ciclo 2* avanza, compara cada elemento de *arregloConRepetidos* con el elemento

anterior ($arregloConRepetidos[i] != arregloConRepetidos[i-1]$). Si el elemento actual es diferente al anterior, lo agrega al arreglo *arregloSinRepetidos* y aumenta $j$ en 1. De esta manera, $arregloSinRepetidos[0:j]$ siempre contiene elementos únicos hasta el momento, cumpliéndose el invariante.

Así, el invariante asegura que al final del *ciclo 2*, *arregloSinRepetidos* contendrá todos los elementos únicos del arreglo original *arregloConRepetidos*, garantizando además, la terminación del algoritmo.

### Discusión de resultados

A continuación se muestran los resultados para el tiempo de ejecución obtenidos en el laboratorio. Éstos fueron obtenidos con ayuda del algoritmo visto en clase.

```
1  public class Tiempo {
2    public static void main(String[] args) {
3      long t0 = System.nanoTime();
4      Clase_seis.eliminarRepetidos(Arreglos.Arr);
5      long t1 = System.nanoTime();
6      Arreglos.printArray(Arreglos.Arr);
7      System.out.println("\nEl tiempo en nanosegundos es:");
8      System.out.println(t1 - t0);
9    }
10 }
```

En la línea 5 se hace uso de una clase externa llamada *Arreglos* y el objeto *Arr* pertenece al conjunto de arreglos definidos como $\alpha = \{arr5, arr10, arr15, arr20, arr25\}$

| Arreglo | Tiempo (ns) | Tiempo ($\mu$s) |
|---|---|---|
| arr5 | 273189 | 273.189 |
| arr10 | 291801 | 291.801 |
| arr15 | 278689 | 278.689 |
| arr20 | 277079 | 277.079 |
| arr25 | 288542 | 288.542 |

La figura 2 muestra lo obtenido experimentalmente y lo teórico para el tiempo de ejecución $T(n)$. La línea de color azul es la función obtenida al analizar el algoritmo *elimarRepetidos*, ésta

corresponde a la siguiente función:

$$T(n) = 15n + 10 \tag{1}$$

Por otra parte, los puntos de color rojo representan las parejas ordenas de la forma $(5, T(5))$, $(5, T(10))$, $(5, T(15))$, $(5, T(20))$ y $(5, T(25))$.

Los puntos de color verde hacen referencia a las parejas ordenadas de la forma $(5, 273.189)$, $(10, 291.801)$, $(15, 278.689)$, $(20, 277.079)$ y $(25, 288.542)$, es decir, lo obtenido con la computadora (valores presentes en la tabla).

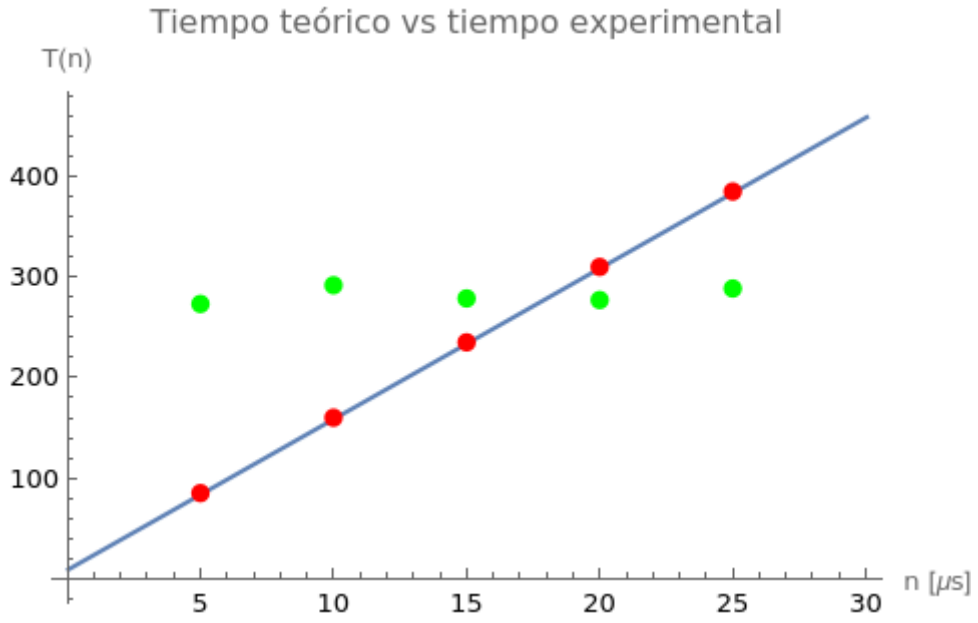


Figura 1: Datos obtenidos para el tiempo de ejecución

La principal característica que se puede apreciar en la figura 1 es el hecho de que los valores experimentales están muy alejados de los datos teóricos, pues estos (de color verde) no parecen tener un comportamiento lineal así como lo dicta la teoría, estos parecerían presentar un comportamiento 'constante' .[1] La discrepancia de los valores puede ocurrir debido a distintos factores que alteran el tiempo de ejecución del algoritmo, tales como tratamientos que ocurren dentro del procesador de la máquina que no se están examinando con profundidad o bien otros factores extra.

[1] Valdría la pena ejecutar el programa varias veces y hacer una estadística para comprobar que lo anterior en efecto es verdadero, de momento solo se menciona lo que se observa

# Anexos

Cálculos para obtener la ecuación 1.

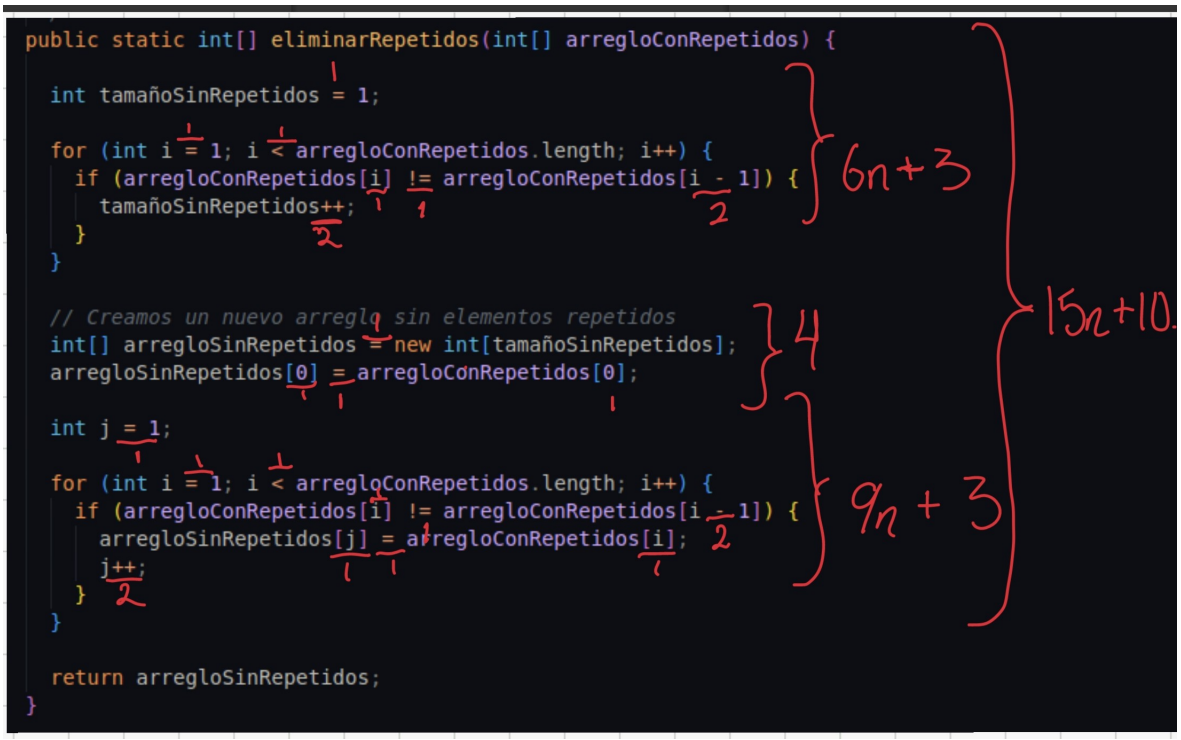


Figura 2: Cálculo para la función $T(n)$.