Práctica 4

Matú Hernández Diana

Rojo Mata Daniel

- Algoritmo: arreglo-2n
- Entrada: Dos arreglos ordenados de manera creciente de números enteros y de longitud n cada uno.
- Salida: Un arreglo de números enteros que conteng los elementos de ambos arreglos ordenados de manera creciente.

```
public static int[] arreglo_2n(int[] arreglo1, int[] arreglo2) {
    int longitud = arreglo1.length;

    int[] arreglo_auxiliar = new int[2 * longitud];

    int i = 0;
    int j = 0;

    while (i < longitud && j < longitud) {
        if (arreglo1[i] <= arreglo2[j]) {
            arreglo_auxiliar[i + j] = arreglo1[i];
            i++;
        } else {
            arreglo_auxiliar[i + j] = arreglo2[j];
            j++;
        }
    }
```

```java

        while (i < longitud) {
            arreglo_auxiliar[i + j] = arreglo1[i];
            i++;
        }

        while (j < longitud) {
            arreglo_auxiliar[i + j] = arreglo2[j];
            j++;
        }

        return arreglo_auxiliar;
    }
```

Descripción del algoritmo

Se denomina como $arreglo1$ y $arreglo2$ a los arreglos de entrada para el algoritmo. Se crea una variable llamada *longitud*, la cual almacena la longitud de uno de los dos arreglos de entrada (como ambos arreglos tienen el mismo número de elementos es indistinto tomar $arreglo1.length$ o $arreglo2.length$.

Se crea un arreglo vacío de tamaño $2n$ llamado *arreglo-auxiliar* en donde se almacenarán los elementos de ambos arreglos en orden ascendente, además, se contemplan dos contadores; a saber j e i, los cuales son inicializados con el valor de 0.

Se realiza un ciclo *while* en donde la condición booleana de inicio es que los contadores j e i sean, ambos, menores a la variable *longitud*. La primer condicional es verdadera cuando el elemento i del arreglo1 es menor o igual al elemento j del arreglo2, es decir, $arreglo1[i] <= arreglo2[j]$. Si esto ocurre, entonces al elemento $i + j$ del arreglo auxiliar se le asocia el valor i del arreglo1, esto es $arreglo - auxiliar[i + j] = arreglo1[i]$, más aún, se incrementa en una unidad el valor de i. Si lo anterior no ocurre es porque se satisface lo siguiente; $arreglo1[i] >= arreglo2[j]$, en esta circunstancia se sigue que $arreglo - auxiliar[i + j] = arreglo2[j]$ y se incrementa en uno el valor de j.

En esencia, la variable i se utiliza para rastrear la posición actual en arreglo1, mientras que j se utiliza para rastrear la posición actual en arreglo2. La comparación $arreglo1[i] <= arreglo2[j]$ o $arreglo1[i] >= arreglo2[j]$ determina el elemento que se encontrará en $arreglo - auxiliar[i+j]$.

Las lineas 19 a 27 se utilizan para agregar los elementos restantes de los arreglos 1 y 2.

Resultados

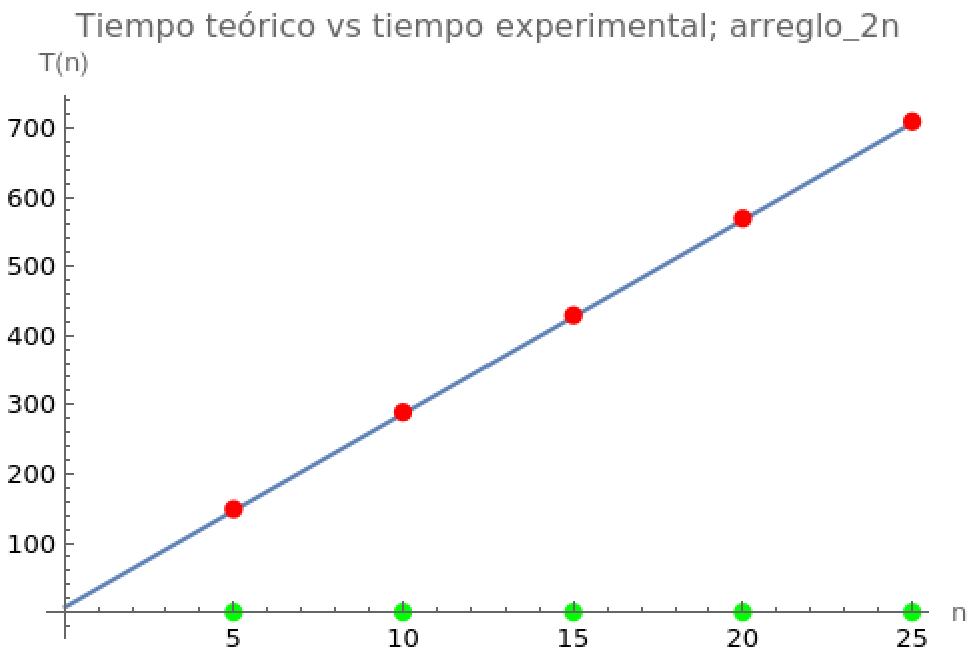

Figura 1: Gráfica de $T(n)$

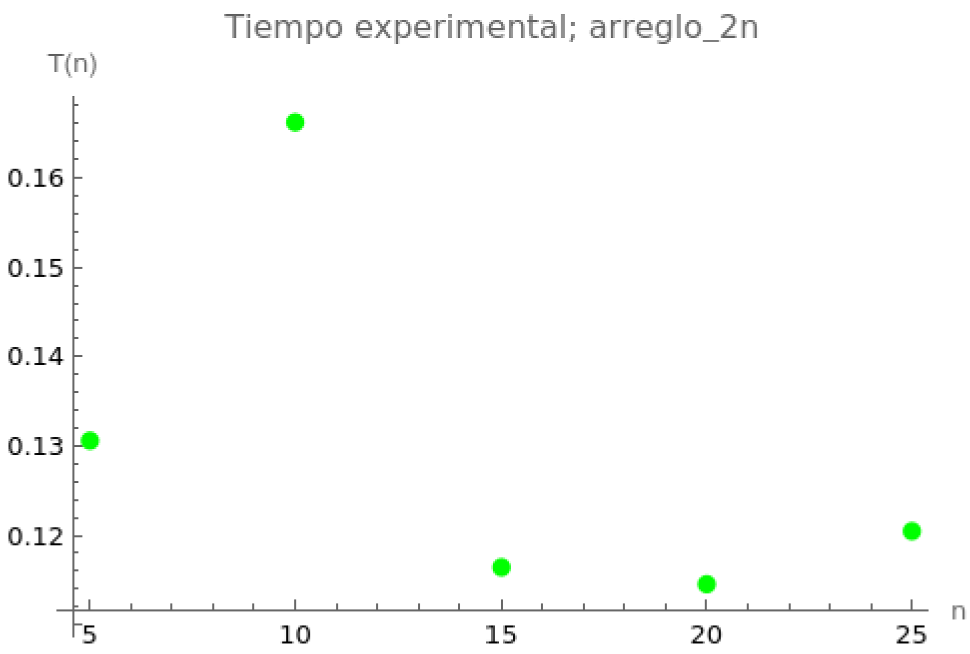

Figura 2: Gráfica de los valores experimentales

Al analizar el algoritmo se obtuvieron los siguiente valores para $T(n)$ y para $M(n)$.

$$T(n) = 28n + 9$$

$$M(n) = 2n$$

La gráfica 1 muestra lo obtenido para la función $T(n)$, mientras que la gráfica 2 muestra lo que se obtuvo para la parte experimental. El cálculo de $T(n)$ se muestra en el anexo.

El valor obtenido para $M(n)$ se debe a que en la línea 4 se está 'creando' un arreglo de longitud $2n$, esto es, se está utilizando $2n$ elementos como memoria extra para el diseño del algoritmo.

Demostración de la complejidad

Afirmación: El algoritmo tiene complejidad en tiempo O(n).

Demostración:

Por demostrar: $T(n) = 28n + 9 \in O(n)$, es decir, veamos que existen $c, n \in Z^+$ tales que $28n + 9 \leq c \cdot n$ para toda $n_0 \leq n$.

Se propone $c = 37$ y $n_0 = 1$.

Con base a lo anterior:

Por demostar: $28n + 9 \leq 37n$ para toda $n \geq 1$.

Sea $n \geq 1$, entonces es cierto que $28n \geq 28n$, además, $9n \geq 9$.

Sumando las dos expresiones anteriores se tiene:

$$28n + 9n \geq 28n + 9 \Longrightarrow 37n \geq 28n + 9$$

Esto es $28n + 9 \leq 37n$, para toda $n \geq 1$.

Por lo tanto existen $c, n_0 \in Z^+$ tales que $0T(n) \leq c \cdot n$ para toda $n \geq n_0$.

Por lo tanto el algoritmo tiene complejidad en tiempo $O(n)$. ✓✓

Afirmación: El algoritmo tiene complejidad en memoria O(n).

Demostración:

Por demostrar: $M(n) = 2n \in O(n)$, es decir, veamos que existen $c, n \in Z^+$ tales que $2n \leq c \cdot n$ para toda $n_0 \leq n$.

Se propone $c = 3$ y $n_0 = 1$.

Con base a lo anterior:

Por demostar: $2n \leq 3n$ para toda $n \geq 1$.

Se sabe que $3 \geq 2$, y como $n \geq 1$, entonces es cierto que $3n \geq 2n$.

Esto es $2n \leq 3n$, para toda $n \geq 1$.

Por lo tanto existen $c, n_0 \in Z^+$ tales que $M(n) \leq c \cdot n$ para toda $n \geq n_0$.

Con base a lo anterior el algoritmo tiene complejidad en memoria $O(n)$. ✓✓

¿El comportamiento de tu algoritmo es lineal?

Sí, puesto que el término dominante para las expresiones $T(n)$ y $M(n)$ es el término n, lo que indica que es cierto que ambos son lineales, más aún, la complejidad en tiempo del algoritmo pertenece a $O(n)$ pues se ha encontrado una cota de la forma $T(n) \leq c \cdot n$ y $M(n) \leq c \cdot n$ para las funciones mencionadas.

Anexos

Tiempos obtenidos para la graficación de 2.

Arreglo	Tiempo (ns)	Tiempo (s)
a5, b5	130577	0.130577
a10, b10	166152	0.166152
a15, b15	116416	0.116416
a20, b20	114532	0.114532
a25, b25	120463	0.120463

Figura 3: Cálculo de $T(n)$ y $M(n)$