

Ej 1

mi función

desde  $i = 1$  hasta  $N - 1$  hacer  $O(N)$ desde  $j = N$  hasta  $i + 1$  hacer  $O(N)$ 

si arreglo[j].clave < arreglo[j-1].clave entonces  $O(1)$   
 Intercambia (arreglo[j], arreglo[j-1])  $O(1)$

Fin si

Fin donde

Fin donde

$$O(N) + O(N) + O(1) + O(1)$$

Fin

$$\underline{O(N^2)} + O(1) + O(1)$$

$$O(N^2)$$

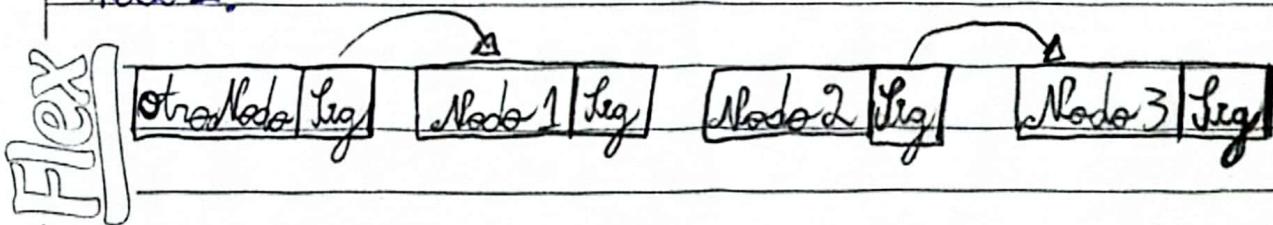
UT3 - PD1

ejercicio 1

Nuevo nodo otroNodo

otroNodo.siguiente  $\leftarrow$  nodo1nodo2.siguiente  $\leftarrow$  nodo3

1) inserto "otroNodo" en la lista, quedando como anterior a nodo1.



Papiror

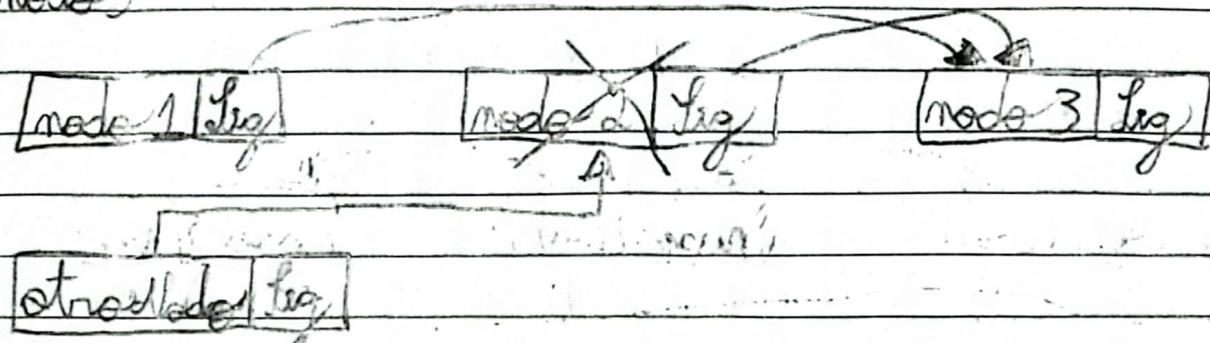
## Ejercicio #2

Nuevo nodo otrosNodo

otrosNodo ← nodo1.sigiente

nodo1.sigiente ← nodo3

- c) Elimina nodo2 de la lista. El siguiente de nodo1 es nodo3



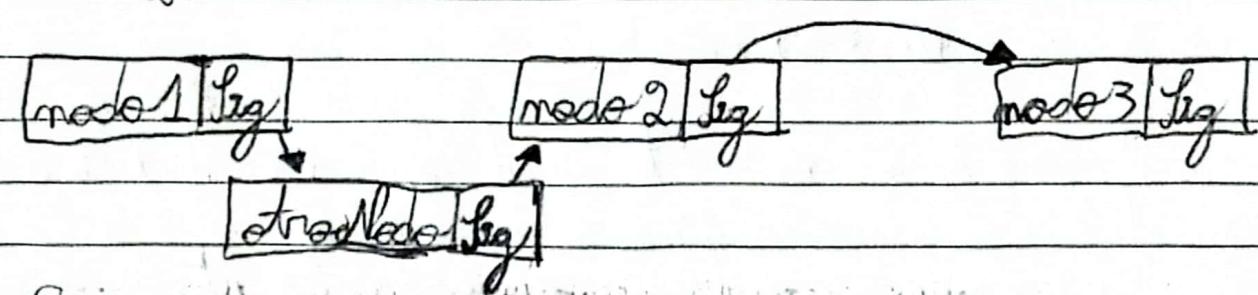
## Ejercicio #3

Nuevo nodo otrosNodo

otrosNodo.sigiente ← nodo1.sigiente

nodo1.sigiente ← otrosNodo

- b) Inserta "otrosNodo" en la lista, quedando entre nodo1 y nodo2.



Flex

Papirer

## Ejercicio #4

Nuevo nodo otrosNodo

Nodo actual < nodoActual

nodoActual < primeros

mientras nodoActual <> nulo hacer

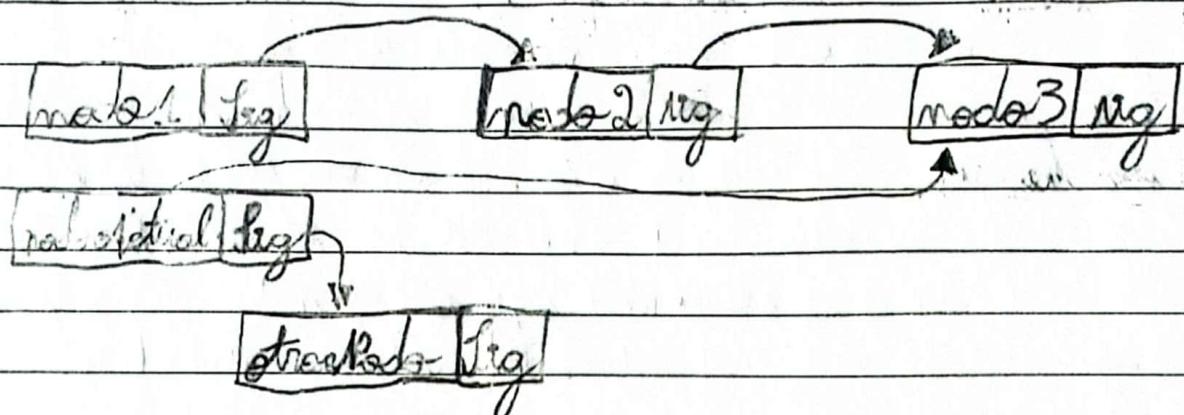
nodoActual <-> nodoActual.siguiente

fin mientras

nodoActual.siguiente <-> otrosNodo

a) Inserta correctamente

"otrosNodo" en la lista,  
quedando como último nodo.



## Ejercicio #5

Nodo actual otrosNodo

Nodo actual nodoActual

nodoActual < primeros

mientras nodoActual.siguiente <> nulo hacer

nodoActual <-> nodoActual.siguiente

fin mientras

nodoActual.siguiente <-> otrosNodo

2) Inserta correctamente "otrosNodo" en la lista, quedando como último nodo. Se crean dos nodos "otrosNodo" y "nodoActual", asigna a este último como el primero.

Luego, entra en un bucle hasta que el nodo siguiente sea nulo, que significa que no ha llegado al

final de la lista. Dentro del bucle, "nodoActual" se actualiza para apuntar al siguiente nodo en la lista. Cuando sale del bucle, "nodoActual.siguiente" se establece en "otroNodo", manteniendo así "otroNodo" como el último nodo de la lista.

## Ejercicio #6

a) Comparando entre la Linked List y la Array List, la Linked List tendrá más memoria enuso que la ArrayList debido al exceso de gastos generales para los punteros, next y previous para cada nodo en la lista enlazada. Sin embargo, la lista enlazada puede tener ventajas en ciertas situaciones, como cuando se necesita insertar o eliminar elementos con frecuencia, ya que estas operaciones son más eficientes en una Linked List que en un array.

b) En una Linked List, solo asigna memoria para cada alumno que se registra, mientras que en un array se debe asignar memoria para una cantidad fija de elementos, lo que implica un desperdicio de memoria, mientras que si se asigna un tamaño muy grande, puede haber desperdicio de memoria, mientras que si se asigna un tamaño muy pequeño, se puede superar el tamaño del array y perder datos o tener que readimensionar.

En una lista enlazada, no hay límite para la cantidad de alumnos que se pueden registrar, ya que se puede ir agregando nodos a medida que se registran nuevos alumnos.

Papiror

# UT2 - PD3



5.4

caso 1

$$a) 150 \cdot (10000) \cdot \log_2(10000) = 1500000 \cdot 13,29 \\ = 19935000$$

$$\text{caso 2: } 10000^2 = 100000000$$

→ es muy favorable ya que el caso  
la mitad de mi valor

caso 1

$$b) 150 \cdot (100) \cdot \log_2(100) = 15000 \cdot 6,64 = 99600$$

$$\text{caso 2: } (100)^2 = 10000 \rightarrow \text{es muy favorable ya que es un} \\ \text{valor menor}$$

$$c) \text{caso 1: } 150 \cdot (1000) \cdot \log_2(1000) = 150000 \cdot 9,97 = 1495500$$

$$\text{caso 2: } (1000)^2 = 1000000 \rightarrow \text{el caso muy} \\ \text{favorable}$$

d) No, va a llegar a un punto donde la entrada va a ser tan grande que va a convenir utilizar el primer código.

5.5

a) Línea 14: utilizando el test `inicio < fin`.

Resurre todos los elementos menos el último

b) Línea 16: haciendo la asignación `medio = inicio + fin / 2`  
Iría cambiando los valores de medio, que no sean iguales  
al principio pero a medida que avanza se acercarán a dar  
resultados erróneos, debido a que el diferente la división de  
una suma que la suma de una división con otro número.

c) Línea 19: ~~avanzando medio = medio~~.

Podría formarse un bucle infinito, porque el índice ~~medio~~ nunca se actualiza, y el bucle seguiría iterando en el mismo rango de búsqueda una y otra vez, lo que provocaría un bucle infinito. Esto sucede porque no hay progreso en la búsqueda, y la misma posición media se selecciona una y otra vez.

d) Idem c.

## 5.6

a) Sumar dos enteros de  $N$  dígitos

$\Theta$  de orden  $O(n)$  debido a que hay que recorrer todos los dígitos de los números.

b) Multiplicar dos enteros de  $N$  dígitos

$\Theta$  de orden  $O(n^2)$ , esto es debido a que hay que por cada cifra de uno se multiplicar la del otro.

c) Dividir dos enteros de  $N$  dígitos.

$\Theta$  de orden  $O(1)$  debido a que no hay que recorrer nada, solo depende de las cifras.



5.10

a) Si el algoritmo es lineal, entonces el tiempo de ejecución es proporcional al tamaño de la entrada, es decir, si la entrada se incrementa en un factor de  $k$ . En este caso, como la entrada se incrementa en un factor de 5, el tiempo de ejecución también se incrementará en un factor de 5. Por lo tanto, el tiempo de ejecución con una entrada de tamaño 500 será:

$$\text{Tiempo de ejecución} = 5 \cdot 0,5 \text{ ms} = 2.5 \text{ ms}$$

b) Si el algoritmo es  $O(N \log N)$ , entonces el tiempo de ejecución aumenta en un factor logarítmico cuando la entrada se incrementa en un factor lineal. En este caso, como la entrada se incrementa en un factor de 5, el tiempo de ejecución se incrementará en un factor de  $\log(5) \sim 0.7$ . Por lo tanto, el tiempo de ejecución con una entrada de tamaño 500 será:

$$\text{Tiempo de ejecución} = 0.7 \times 0.5 \text{ ms} = 0.35 \text{ ms}.$$

c) Si es cuadrático, entonces el tiempo de ejecución aumenta en un factor cuadrático cuando la entrada se incrementa en un factor lineal. En este caso, como la entrada se incrementa en un factor de 5, el tiempo de ejecución se incrementará en un factor de 5 al cuadrado = 25. Por lo tanto, el tiempo de ejecución con una entrada de tamaño 500 será:

$$\text{Tiempo de ejecución} = 25 \times 0.5 \text{ ms} = 12.5 \text{ ms}$$

d) Si es cúbico, entonces el tiempo de ejecución aumenta en un factor cúbico cuando la entrada se incrementa en un factor lineal. En este caso, como la entrada se incrementa en un factor de 5, el tiempo de ejecución se incrementará en un factor de 5 al cubo = 125. Por lo tanto, el tiempo de ejecución con una entrada de tamaño 500 será:

$$\text{T. de ejecución} = 125 \times 0,5 \text{ ms} = 62,5 \text{ ms.}$$

### 5.11

a) Si el algoritmo es lineal, entonces el tiempo de ejecución es proporcional al tamaño de la entrada, es decir, si la entrada se incrementa en un factor de  $k$ , el tiempo de ejecución también se incrementará en un factor de  $k$ . Por lo tanto, el tiempo de ejecución para una entrada de tamaño  $n$  será:

$$\text{T. de ejecución} = (n/100) \times 0,5 \text{ ms.}$$

Para saber qué tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de  $n$  tal que el tiempo de ejecución sea igual a 60 segundos (o 60000 ms). Despejando  $n$ :

$$(n/100) \times 0,5 \text{ ms} = 60000 \text{ ms}$$

$$n = (60000 \text{ ms} / 0,5 \text{ ms/1}) \times 100$$

$$n = 12000000$$

Por lo tanto, con un algoritmo lineal que tarda 0,5 ms en procesar una entrada de tamaño 100, puede procesar una entrada de tamaño 12000000 en un minuto.

- b) Si el algoritmo es  $O(N \log N)$ , entonces el tiempo de ejecución aumenta en un factor logarítmico cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño  $n$  será:

$$\text{T. de ejecución} = (n/100) \times 0,5 \text{ ms} \times \log(n/100)$$

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de  $n$  tal que el tiempo de ejecución sea igual a 60 segundos (o 60000 ms). Utilizando una calculadora en línea encontramos que  $n$  es aproximadamente igual a 259200.

- c) Si el algoritmo es cuadrático, entonces el tiempo de ejecución aumenta en un factor cuadrático cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño  $n$  será:

$$\text{T. de ejecución} = (n/100)^2 \times 0,5 \text{ ms}$$

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de  $n$  tal que el tiempo de ejecución sea igual a 60 segundos (o 60,000 ms). Despejando  $n$ :

$$(n/100)^2 \times 0,5 \text{ ms} = 60,000 \text{ ms}$$

Por lo tanto, con un algoritmo cuadrático que tarda 0,5 ms en  $n = \sqrt{(60,000 \text{ ms} / 0,5 \text{ ms}) \times 100}$  procesar una entrada de tamaño 100, puede procesar una entrada de tamaño 34641 en un minuto.

$$n = 34641$$

d) Si el cubo, entonces el tiempo de ejecución aumenta en un factor cúbico cuando la entrada se incrementa en un factor lineal. Por lo tanto, el tiempo de ejecución para una entrada de tamaño  $N$  será:

$$\text{t. de ejecución: } (m/100)^3 \times 0,5 \text{ ms}$$

Para saber que tamaño de entrada puede procesar en un minuto, necesitamos encontrar el valor de  $m$  tal que el tiempo de ejecución sea igual a 60 segundos ( $\approx 60000 \text{ ms}$ ). Despejando  $m$ :

$$(m/100)^3 \times 0,5 \text{ ms} = 60000 \text{ ms}$$

$$m = \sqrt{(60000 \text{ ms} / 0,5 \text{ ms})} \cdot 100$$

$$m = 4932$$

## 5.12

$N$	$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
10	0,000103	0,00045	0,00066	0,00034
100	0,47015	0,01112	0,00486	0,00063
1000	448,77	1,1233	0,05843	0,00333
10000	NA	111,13	0,68631	0,03042
100000	NA	NA	8,01130	0,2932
1000000	$1 \times 10^8 T(N)$	$1 \times 10^{12} T(N)$	$2 \times 10^7 T(N)$	$1 \times 10^6 T(N)$

Para entradas muy grandes siempre va a convenir un algoritmo que tenga el más pequeño orden debido a que cada vez va aumentando más y más el tiempo que tarda. Con valores de entrada pequeños esas diferencias no se nota, pero cuando crecen esto diferencial puede ser abusivo.



5.13

$$37 \rightarrow \frac{2}{N} \rightarrow \sqrt{N} \rightarrow N \log \log N \rightarrow N \rightarrow N \log N \rightarrow N \log N^2 \rightarrow N \log^2 N \\ \rightarrow N^{1.5} \rightarrow N^2 \log N \rightarrow N^2 \rightarrow N^3 \rightarrow 2^{\frac{N}{2}} \rightarrow 2N$$

Tanto  $2^{\frac{N}{2}}$  como  $2^N$  crecen a la misma velocidad, tambien  $N^2 \log N$  y  $N^2$ , y  $N \log N$  con  $N \log^2 N$ .

5.14

Fragmento 1

$$\text{for (int } i=0; i < n; i++) \text{ O}(N) \quad O(N) + O(1) = O(N) \\ \text{suma++; O(1)} \quad O(N)$$

Fragmento 2

$$\text{for (int } i=0; i < n; i++) \text{ O}(N) \quad \left. \right\} O(N^2) \quad O(N^2) + O(1) \\ \text{for (int } j=0; j < n; j++) \text{ O}(N) \quad \left. \right\} \\ \text{suma++; O(1)} \quad O(N^2)$$

Fragmento 3

$$\text{for (int } i=0; i < n; i++) \text{ O}(N) \quad O(N) + O(N) + O(1) + O(1) \\ \text{suma++; O(1)} \\ \text{for (int } j=0; j < n; j++) \text{ O}(N) \quad O(N) \\ \text{suma++; O(1)}$$

Fragmento 4

$$\text{for (int } i=0; i < n; i++) \text{ O}(N) \quad \left. \right\} O(N^2) \\ \text{for (int } j=0; j < n * n; j++) \text{ O}(N) \quad \left. \right\} \\ \text{suma++; O(1)} \quad O(N^2) + O(1)$$

### Fragmento #5

```

for (int i=0; i<n; i++) O(N) } O(N2) O(N2) + O(1)
for (int j=0; j<1; j++) O(N)
    suma++; O(1) O(N2)

```

### Fragmento #6

```

for (int i=0; i<n; i++) O(N)
for (int j=0; j<n * n; j++) O(N) } O(N3)
for (int k=0; k<j; k++) O(N)
    suma++; O(1) O(N3)

```

$$O(N^3) + O(1)$$

Veces	1	2	3	4	5	6
Suma con N=10	10	100	20	1000	45	49500
N=10	0ms	0ms	0ms	0ms	0ms	0ms
Suma con N=1000	1000	1000000	2000	10000000000	499500	882236
N=1000	0ms	2ms	0ms	15ms	1ms	1251ms

### 5.15

```

for (int i=1; i<n; i++) O(N)
for (int j=0; j<1 * i; j++) O(N) } O(N3)
if (j % i == 0) O(1)
for (int k=0; k<j; k++) O(N)
    suma++; O(1)

```

$$O(N^3) + O(1) + O(1)$$

Suma con N=1000: 296566661486 ms

Suma con N=10: 870/0 ms



5.16

Leyendo el gráfico de  $2^n$ , por lo que es de este orden. En el día N basta con sustituir  $n$  en  $2^9$