UNIDAD TEMÁTICA 2: DISEÑO Y ANÁLISIS DE ALGORITMOS

TRABAJO DE APLICACIÓN 1

Ejercicio #1

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
public static int enRango (int[] a, int bajo, int alto) {
      int contador = 0;
      for (int i=0; i<a.length; i++) {
            if (a[i] >= bajo && a[i] < alto)
                  contador++;
      }
      return contador;
}
```

SUMATORIA:
O(1)
O(n)
O(1)
O(1+n+1) ≅ O(n)

O(1)
O(1)
O(n.(1+1)) ≅ O(2n) ≅ O(n).
¿por el for cant. de veces que lo hace?

por orden ya que n es de mayor orden que los de Orden 1

ANOTACIONES
- Orden 1 = constante.

asigna ←

Ejercicio #2

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
unaFunción ( N de tipo entero)
  i ← 1
  j ← N
  mientras i < N hacer
    j ← J
    i ← i * 2
  fin mientras
  [illegible] (j)
fin
```

O(1) O(1) O(N) O(1) $\log_2(N)$ $\log_2 N$ O(1)

$O(1) + O(1) + \log_2 N + O(1) \cong O(\log_2 N)$

N° iteración

# it	i	j	while - cond	N = 64
0	1	64	–	
1	2	63	✓	
2	4	62	✓	
3	8	61	✓	
4	16	60	✓	
5	32	59	✓	
6	64	58	✓	
7			×	

$2^6 = 64$

$\Rightarrow N = 2^{\#it} \Rightarrow \log_2(N) = \#it$

Ejercicio #3

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
int[] cuentas = new int [100];
for (int i = 0; i<100; i++) {
  cuentas[i] = enRango (notas, i, i+1);
}
```

O(1) O(1) O(n) — O(1+1+n) ≅ O(1)

O(1) O(1) O(n) } O(100·n) ≅ O(n)

Ejercicio #4

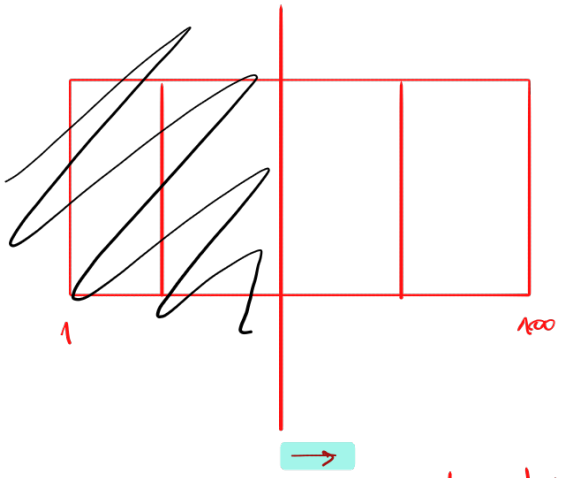
Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
unValor (A, N de tipos enteros)
i ← 0
Si N < 3 entonces
devolver (A)
fin si
mientras i < 3 hacer
      si arreglo[i] = A entonces
            devolver ((arreglo[0] + arreglo[N-1]) div 2)
      fin si
      i ← i + 1
fin mientras
devolver (A div N)
Fin
```

Ejercicio #5

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
otraFunción (claveAbuscar)
  inicio ← 0  O(1)
  fin ← N-1  O(1)
  mientras inicio ≤ fin hacer
    medio ← (inicio + fin) div 2  O(1)
    si (arreglo[medio] < claveAbuscar) entonces
      inicio ← medio + 1  O(1)
    sino
      si (arreglo[medio] > claveAbuscar) entonces
        fin ← medio - 1  O(1)
      sino
        devolver medio  O(1)
      fin si
    fin si
  fin mientras
  devolver -1  O(1)
fin
```



busca en la mitad en la que pueda estar mi "claveAbuscar" ej.

58 → tomo la mitad de la derecha y lo vuelvo a div en 2.

Por lo tanto podemos deducir que es de $O(\log_2 N)$

Ejercicio #6

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
function particion( i, j: integer; pivote: TipoClave): integer;
{divide V[i], .., V[j] para que las claves menores que pivote estén a la izquierda y las mayores o iguales a la derecha. Devuelve el lugar donde se inicia el grupo de la derecha.}
COMIENZO
  L ← i;
  R ← j;
  Repetir
    intercambia(V[L],V[R]);
    mientras V[L].clave < pivote hacer   L := L + 1; fin mientras
    mientras V[R].clave >= pivote hacer R := R - 1; fin mientras
  Hasta que L > R
  Devolver L;
FIN; {particion}
```

Ejercicio #7

Analiza el orden del tiempo de ejecución del siguiente algoritmo, y responde las preguntas presentadas en pantalla:

```
miFunción
  Desde i = 1 hasta N-1 hacer
    Desde j = N hasta i+1 hacer
      Si arreglo[j].clave < arreglo[j-1].clave entonces
        Intercambia(arreglo[j], arreglo[j-1])
      Fin si
    Fin desde
  Fin desde
Fin
```