	UNIVERSIDAD AUTÓNOMA DE OCCIDENTE			
	FACULTAD DE INGENIERÍA *NUCLEO MIDIA*	ESTRUCTURAS DE DATOS Y ALGORITMOS 1	*GRUPO 1 HL*	
	CÓDIGO:	NOMBRE:		*Valoración*
	EXAMEN PARCIAL - CORTE 2		FECHA: Abril 15 de 2024	

PRIMERA PARTE	Tiempo de ejecución y complejidades **[3 Puntos]**	PUNTAJE	

a) [1.0 pts] A partir del siguiente fragmento en el que se han definido los bloques que permiten realizar su análisis, *n* que es igual al tamaño del arreglo *datos*, indicar: 1) cuantas veces se ejecuta la condición **n1** 2) indicar cuantas veces se ejecuta la instrucción **n2** en el ***peor de los casos.***

```
                 n1
for ( let i=7 ; i < n+1 ; i++ ) {
      if ( datos[i-3] < datos[0] ){
  n2 ⇨    aux = datos[i-4];
      }else{
          veces += ajustarSuma(datos);
      }
}
```

R://

1) n-6

2) n-6

b) [0.8 pts] A partir de las siguientes funciones $T_A(n)$ y $T_B(n)$, correspondientes, respectivamente, al número total de instrucciones, de los algoritmo A y B. a) indicar sus órdenes de complejidad b) indicar cuál de los algoritmos es más eficiente, con base las ordenes de ejecución obtenidos:

$$T_A(n) = 5*n^2+1000*n+50000, \quad T_B(n) = 0.1*n^2+0.0001*n^3-20$$

R://

$$T_A(n) = O(\ O(n^2)\ ), \quad T_B(n) = O(\ O(n^3)\ )$$

El algoritmo más eficiente es el A (indicar la letra)

c) [1.2 pts] A partir del siguiente fragmento, en el cual *n* es el tamaño del arreglo *datos* (obtenido con la instrucción *datos.length*), la complejidad del método ***calcularMetodo1*** es ***O*(lg n)** y la del método ***calcularMetodo2*** es ***O*(n)**, indicar las complejidades *O* de: a) todas las ejecuciones de las líneas solicitadas b) de todo el fragmento.

```
27  let valA = datos.length;
28  let valB = Math.trunc(Math.sqrt(valA));
29  for ( let i = 0 ; i < valA ; i++ ) {
30      for ( let j=1 ; j<valB ; j++ ){
31          res = calcularMetodo1(datos,j);
32          suma += calcularMetodo2(datos);
33      }
34  }
```

R://

Línea	Respuesta a)	Respuesta b)
27 y 28	$T(n) = O(1)$	27: T(n) = O(1) 28: T(n) = O(1) 29: T(n) = O(1) 30: T(n) = O(√n) 31: T(n) = O(√n *log n) 32: T(n) = O(n^3/2)
29	T(n)=O(1)	
30	T(n) = O(√n)	
31 y 32	O(√n *log n) - O(√n * n) = O(n^3/2)	

SEGUNDA PARTE	Recursividad **[2.0 Puntos]**	PUNTAJE	

A partir de un algoritmo A, del cual se muestra su seudocódigo y relación de recurrencia $T_A(n)$ que define su tiempo de ejecución:

```
PROCEDIMIENTO solucionG1 (A,n)
   SI  n es igual a 1
      retornar la posicion obtenida al buscar en A el valor 500
   SINO
      crear variable res y asignar el valor obtenido de buscar el numero mayor de A
      EJECUTAR cuatro veces en un ciclo
         acumular en res el valor obtenido de invocar a solucionG1 con un cuarto del
         valor de n
      retornar res
```

$$T_A(n) = \begin{cases} O(n) & n = 1 \\ O(n) + 4T(\frac{n}{4}) & n > 1 \end{cases}$$

.

a) [2.0 pts] Calcular su complejidad (realizando sustituciones o generando el árbol) y comparar su eficiencia contra la de un algoritmo B. Se requiere:
I) expandir $T_A(n)$ k veces e indicarla en términos de k
II) despejar k (cuantas veces se realizan las invocaciones recursivas)
III) despejar $T_A(n)$ al reemplazar el valor de k

IV) indicar la complejidad *O* del algoritmo A

V) si la complejidad de un algoritmo B que realiza la misma labor es $O(n^3)$, indicar cuál de los dos algoritmos (A o B) es más eficiente.

R://

I) [0.8 pts]) expandir $T_A(n)$ *k* veces e indicarla en términos de *k*

$T_A(n) =$ Primera expansión: TA(n) = O(n)+4TA(n/4)

Segunda expansión: se sustituye TA(n/4):

- TA(n)= O(n)+4(O(n/4)+4TA(n/16))

Simplicado
TA(n) = O(n)+O(n)+4^2TA(n/16) = 20(n) + 16TA(n/16)

Tercera expansión:
TA(n) = 20(n) + 16(O(n/16) + 4TA(n/64))

Simplicado:
TA(n) = 20(n) + 160(n/16) + 64TA(n/64) = 20(n) +O(n) +64TA(n/64)
TA(n) = 30(n) + 64TA(n/64)

K expansiones:

Sk = 4k -1 / 4-1 = 4^k-1/3

TA(n) = O(n) * 4^k -1/3 + 4^kTA(n/4^k)

II) [0.4 pts] despejar *k* (cuantas veces se realizan las invocaciones recursivas)

k = log2(n) / 2

III) [0.3 pts] despejar $T_A(n)$ al reemplazar el valor de *k*

$T_A(n) =$

TA(n) = O(n) * 4^k -1/3 + 4^kTA(n/4^k)
TA(n) = O(n) * 4log4^(n) -1 + 4log4^(n)TA (n/4log4 ^(n))
TA(n) = O(n) * n-1/3 +nTA(1)
TA(n) = O(n) * n-1/3 +n * c
TA(n) = O(n^2) + O(n) = O(n^2)

IV) [0.3 pts] indicar la complejidad *O* del algoritmo A

$T_A(n) =$ O(n^2)

V) [0.2 pts] indicar cual algoritmo es más eficiente

El algoritmo más eficiente es el __A__.