UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
ESCUELA DE INGENIERÍA DE SOFTWARE

PRACTICA CALIFICADA
ANÁLISIS Y DISEÑO DE ALGORITMOS
Ciclo 2024-2

1. Calcular el tiempo del método y la notación asintótica del siguiente método

```
Metodo Clista.Calculo()
        Para I desde 1 hasta N²
                K ← N
                Mientras (K < N²) hacer
                        Para L desde N hasta 1 hacer
                                L ← L div 3                A        B        C
                        fPara
                        K ← k * 4
                fMientras
                I ← I / 5        Esto se corrige por I ← I * 5
        fPara
fMetodo
```

Sección A:

Convertir el Para en un Mientras

L←N	t = 1		
Mientras (L >= 1) hacer	t = 1	T = 4 * # ciclos	T = 4 * # ciclos
L ← L div 3	t = 2		
fMientras	t = 1		

Calculo del # de ciclos

i. Función implicada en el control del bucle
$f = L - 1 + 1 = N - 1 + 1 = N$

ii. Para que condición llega al termino el mientras
$L < 1$

iii. Como las variables implicadas tienden al término del ciclo del mientras
$L \leftarrow N$, N div 3, N div 3^2, N div $3^3 \ldots N$ div 3^K
Para la pasada N div 3^K no se ingresa al mientras
Entonces, el valor de k es el número de ciclos
Remplazando el L en la condición que llega al termino el mientras
N div $3^K < 1$
Convertimos la división entera en una división normal
$N / 3^K < 1 \Rightarrow N < 3^k$
Tomamos logaritmo en base 3 de ambos términos de la ecuación
$Log_3 N < Log_3 3^k \Rightarrow k > Log_3 N \Rightarrow k = Log_3 N + c$ es el número de ciclos del mientras
El tiempo de esta sección es $T = \#ciclos * T_{mientras} + 2$
$= (Log_3 N + c)*4 + 2 = 4Log_3 N + 4c + 2$
El orden asintótico de esto es $O(Log_3 N)$

Sección B:

Evaluar el Mientras

$K \leftarrow N$ — $t = 1$

Mientras ($K < N^2$) hacer — $t = 1$

  $K \leftarrow K * 4$ — $t = 2$

fMientras — $t = 1$

$T = 4 * \#\ ciclos$ — $T = 4 * \#\ ciclos$

Calculo del # de ciclos

i. Función implicada en el control del bucle
   $f = N^2 – K = N^2 – N$

ii. Para que condición llega al termino el mientras
   $K >= N^2$

iii. Como las variables implicadas tienden al término del ciclo del mientras
   $K \leftarrow N, 4N, 4^2N, 4^3N \ldots 4^xN$
   Para la pasada 4^xN no se ingresa al mientras
   Entonces, el valor de x es el número de ciclos
   Remplazando el K en la condición que llega al termino el mientras
   $4^xN >= N^2$
   Factorizamos un factor N a cada lado de la ecuación
   $4^x >= N$
   Tomamos logaritmo en base 4 de ambos términos de la ecuación
   $Log_4N =< Log_44^x => x =< Log_4N => x = Log_4N + c$ es el número de ciclos del mientras

El tiempo de esta sección es $T = \#ciclos * T_{mientras} + 2$

$= (Log_4N + c)*(4 + (Seccion A)) + 2$

$= (Log_4N + c)(4+ 4Log_3N + 4c + 2) + 2 = (Log_4N + c)(4Log_3N + 4c + 6) + 2$

$= 4Log_4NLog_3N + 4cLog_4N + cLog_4N + 4cLog_3N + 4c^2 + 6c$

$= 4Log_4NLog_3N + 5cLog_4N + 4cLog_3N + 4c^2 + 6c$

El orden asintótico de esto es $O(Log_4NLog_3N)$

Sección C:

Convertir el Para en un Mientras

$I \leftarrow 1$ — $t = 1$

Mientras ($I <= N^2$) hacer — $t = 1$

  $I \leftarrow I * 5$ — $t = 2$

fMientras — $t = 1$

$T = 4 * \#\ ciclos$ — $T = 4 * \#\ ciclos$

Calculo del # de ciclos

i. Función implicada en el control del bucle
   $f = N^2 – I + 1 = N^2 – 1 + 1 = N$

ii. Para que condición llega al termino el mientras
   $I > N^2$

iii. Como las variables implicadas tienden al término del ciclo del mientras
   $I \leftarrow 1, 5, 5^2, 5^3 \ldots 5^K$

Para la pasada 5^K no se ingresa al mientras
Entonces, el valor de k es el número de ciclos
Remplazando el L en la condición que llega al termino el mientras
$5^K > N^2$
Tomamos logaritmo en base 5 de ambos términos de la ecuación
$Log_5 5^k > Log_5 N^2 \Rightarrow k > Log_5 N^2 \Rightarrow k = Log_5 N^2 + c$ es el número de ciclos del mientras

El tiempo de esta sección es $T = \#ciclos * T_{mientras} + 2$

$= (Log_5 N^2 + c)*(4 + (Seccion\ B)) + 2$
$= (Log_5 N^2 + c)(4+ Log_4 N Log_3 N + 5cLog_4 N + 4cLog_3 N + 4c^2 + 6c) + 2$
$= (Log_5 N^2 + c)(Log_4 N Log_3 N + 5cLog_4 N + 4cLog_3 N + 4c^2 + 6c + 4) + 2$
$= Log_5 N^2 Log_4 N Log_3 N + 5cLog_5 N^2 Log_4 N + 4cLog_5 N^2 Log_3 N + 4c^2 Log_5 N^2$
$+ 6cLog_5 N^2 + 4Log_5 N^2 + cLog_4 N Log_3 N + 5c^2 Log_4 N + 4c^2 Log_3 N + 4c^3 + 6c^2 + 4c$
$= Log_5 N^2 Log_4 N Log_3 N + 5cLog_5 N^2 Log_4 N + 4cLog_5 N^2 Log_3 N + (4c^2 + 6c + 4)Log_5 N^2$
$+ cLog_4 N Log_3 N + 5c^2 Log_4 N + 4c^2 Log_3 N + 4c^3 + 6c^2 + 4c$

El orden asintótico de esto es $O(Log_5 N^2 Log_4 N Log_3 N)$

2. **Recursividad:** Un polinomio se puede representar a través de una lista enlazad simple (el primer atributo es el coeficiente, el segundo es el exponente de la variable y el último es el apuntador al siguiente nodo). Construya un algoritmo recursivo para multiplicar dos polinomios que pueden estar en cualquier orden. El resultado será representado en otra lista eficiente y ordenada. Elabore su función tiempo y su notación asintótica.

```
Clase CNodoP
    Atributos
        Entero coeficiente
        Entero exponente
        CNodoP siguiente
    Metodos
        contructor()
        // metodos accesores
fClase

Metodo CPolinomio.multiplicacion( CNodoP primero, CNodoP segundo) →
    Objeto result ejemplar-de CNodoP  → 1
    result ← nulo  → 1
    Entero base ← 0  → 1
    Entero expo ← 0  → 1
    CNodoP ptr ← primero  → 2
    Mientras (ptr <> nulo ) hacer  → 1
        CNodoP qtr ← segundo  → 2
        Mientras ( qtr <> nulo ) hacer  → 1
            base ← ptr.obtenerCoeficiente() * qtr.obtenerCoeficiente()  → 4
            expo ← ptr.otenerExponente() + qtr.obtenerExponente()  → 4      } 4   } (4+2)n   } (6n+2)n
            result ← insertarNodo(base, expo, result)  → 2                         6n          6n²+2n
            qtr ← qtr. obtenerSiguiente()  → 2
        fMientras  → 1
        ptr ← ptr.obtenerSiguiente()  → 2
    fMientras  → 1
    Mientras (ptr <> nulo ) hacer  → 1
        result ← insertarNodo(ptr.obtenerCoeficiente(), ptr.otenerExponente(), result)  → 4   } (4+2)n
    fMientras  → 1                                                                              6n
    Mientras (qtr <> nulo ) hacer  → 1
        result ← insertarNodo(qtr.obtenerCoeficiente(), qtr.otenerExponente(), result)  → 4   } (4+2)n
    fMientras  → 1                                                                              6n
    retornars result  → 1
fMetodo
```

Siendo el primer polinomio de n elementos y el segundo polinomio de m elementos
Además, se tiene que n > m entonces:

Tiempo = Tiempo sin llamadas recursivas + Tiempo primera llamada recursiva
Tiempo = $(6n^2+2n+6n+6n) + T(x)$
Tiempo = $(6n^2+14n) + T(x)$

Donde: x es llamado n*n veces en el peor caso
Entonces $T(x) = 4n^2$
Tiempo = $(6n^2+14n) + 4n^2 = 10n^2 + 14n$
El orden es $O(n^2)$

```
Metodo CPolinomio.insertarNodo( Entero base, Entero expo, CNodoP result) →
    Si (result = nulo) entonces  → 1
        Objeto nodo ejemplar-de CNodoP  → 1
        nodo.colocarCoeficiente( base )  → 1        1
        nodo.colocarExponente( expo )  → 1
        result ← nodo  → 1
    Sino
        CNodoP presente ← result  → 1    1
        CNodoP anterior ← result  → 1
        Mientras (presente <> nulo y expo > presente.obterneExponente()) hacer  → 2
            anterior ← presente  → 1                                            4n²
            presente ← presente.obtenerSiguiente()  → 1
        fMientras  → 1
        // Si el exponente ya existe en la lista de salida
        Si (presente <> nulo y presente.obtenerExponente() = expo) entonces  → 3
            presente.colocarBase( presente.obtenerBase() + base)  → 3
        Sino
            // Si el presente es nulo entonces el exponente es mayor que el último nodo    4n²
            Si (presente = nulo ) entonces  → 1
                Objeto nodo ejemplar-de CNodoP  → 1
                nodo.colocarCoeficiente( base )  → 1      1
                nodo.colocarExponente( expo )  → 1                           3
                anterior.colocarSiguiente( nodo)  → 1
            sino                                                       1
                // el presente existe y el exponente a insertar es menor
                Objeto nodo ejemplar-de CNodoP  → 1
                nodo.colocarCoeficiente( base )  → 1
                nodo.colocarExponente( expo )  → 1      1
                nodo.colocarSiguiente( presente )  → 1
                anterior.colocarSiguiente( nodo)  → 1
            fSi
        fSi
    fSi
    retornar result  → 1
fMetodo
```

El tiempo del método será de $4n^2$ por ser el más complejo cuando n tiende al infinito

3. Especifique y construya un algoritmo eficiente que lea una secuencia de enteros, halle la mayor cantidad de elementos consecutivos que constituyan la subsecuencia creciente continua más grande. Evalué el tiempo y la complejidad asintótica del algoritmo. Ejemplo de muestra:

```
Método CVector.subsecuenciaConsecutivaMaxima(CArreglo Vector)
    Entero maxima ← 0          // cantidad parcial                 → 1
    Entero parcial ← 0         // cantidad máxima                  → 1
    Entero pInicio ← 0         // índice menor de cantidad parcial → 1    1
    Entero pFinal ← 0          // índice mayor de cantidad parcial → 1
    Entero mInicio ← 0         // índice menor de cantidad máxima  → 1
    Entero mFinal ← 0          // índice mayor de cantidad máxima  → 1
    Para I desde 1 hasta n hacer → 2
        Si (I = 1) entonces → 1
            parcial ← I → 1
            pInicio ← I → 1    1
            pFinal ← I → 1
        Sino
            Si (Vector[I-1] < Vector[I] ) entonces → 1
                parcial ← parcial + 1 → 2
                pFinal ← I → 1                          2    2
                Si (pInicio = 0) entonces → 1
                    parcial ← parcial + 1 → 2    2    2    2    2
                    pInicio ← I - 1 → 2
                fSi
            sino
                parcial ← 0 → 1
                pinicio ← 0 → 1    1
                pFinal ← 0 → 1
            fSi
        fSi
        Si (parcial > maxima) entonces → 1
            maxima ← parcial → 1         1
            mInicio ← pInicio → 1    1
            mFinal ← pFinal → 1
        fSi
    fPara
    Escribir "Indice menor: " + mInicio → 1
    Escribir "Indice mayor: " + mFinal → 1    1
    Escribir "Cantidad: " + maxima → 1
fMetodo
```

Tiempo del algoritmo = tiempo del interior * numero ciclos = 2 * n

Orden del algoritmo es O(n)