NIVEL BASICO

Ejercicio 1. Dado un array de enteros, devolver el producto de todos sus elementos. Si el array está vacío se devuelve 1 por conveniencia.

Función no Final (pseudocódigo)

```
F(v[ ], n) ( empezando en v[0])
inicio
si n=0 entonces -> devolver 1;

sino -> devolver v[ ] * F(v[ ] + 1, n-1)

fin.
```

Función Final (pseudocódigo)

```
//FuncionFinalAux

f(v[ ], n, acumulador) (empezando en v[0]
inicio
si n=0 entonces devolver acumulador
sino -> devolver f(v +1, n-1, acumulador * v[])
fin


//Función Final

F(v[0 ], n)
inicio
devolver f(v[0],n,1);
fin.
```

Función Iterativa (pseudocódigo)

```
inicio
declaramos entero aux
bucle for( declaramos i=0, mientras i<n, i+1 en cada iteracion)
aux *= v[i] en cada iteracion
fin bucle
devolver aux
fin
```

// Calculo T(n) y orden de complejidad EJ1

#Función no final → orden de complejidad: O(n)
Caso base: Comprobación igualdad n=0 (1) + return 1 (1) = 2
Recursión: Comprobación igualdad n!=0(1) + acceso *v (1) +multiplicacion * (1) + return recursivo (1) = 4

Caso base(2) + 4 por cada recursión

$$T(n) = 4n + 2$$

#Función final → orden de complejidad: O(n)
Constante inicial: return función final auxiliar (1)=1
Caso base: Comprobacion igualdad n=0 (1) + return acum (1)= 2 + cte inicial
Recursion: Comprobacion igualdad n!=0 (1) + return (1) + * (1) + Acceso *v = 4

Caso base(3) + 4 por recursión

$$T(n) = 4n + 3$$

#Función Iterativa → orden de complejidad: O(n)
Caso base:Inicializacion int i (1) + Inicializacion variable aux (1) + Validación negativa i<n (1) + return aux (1) = 4
Bucle: Validacion i<n(1) + Suma i++ (1) + Multiplicacion(1) + Asignacion(1) + LLamada v (1) = 5

$$T(n) = 5n + 4$$

Ejercicio 2. Dado un array de float, suma el cuadrado de la raíz cúbica de sus elementos.

Función no Final (pseudocódigo)

```
F(v[ ], n) ( empezando en v[0])
inicio
si n=0 entonces -> devolver 0;

sino -> devolver pow(cbrt(v[ ]), 2) + F(v[ ] + 1, n-1)

fin.
```

Función Final (pseudocódigo)

```
//FuncionFinalAux

f(v[ ], n, acumulador) (empezando en v[0]
inicio
si n=0 entonces devolver acumulador
sino -> devolver f(v +1, n-1, acumulador + pow(cbrt(v),2))
fin

//Función Final

F(v[0 ], n)
inicio
devolver f(v[0],n,0);
fin.
```

Función Iterativa (pseudocódigo)

```
inicio
declaramos entero aux
bucle for( declaramos i=0, mientras i<n, i+1 en cada iteracion)
aux += pow(cbrt(v[i]),2) en cada iteracion
fin bucle
devolver aux
fin
```

// Calculo T(n) y orden de complejidad EJ2

#Función no final → orden de complejidad: O(n)
Caso base: Comprobación igualdad n=0 (1) + return 0 (1) = 2
Recursión: Comprobación igualdad n!=0(1) + return recursivo (1) + uso pow (1) + uso cbrt (1) + acceso *v (1) + suma (1) = 6

Caso base(2) + 6 por cada recursión

$$T(n) = 6n + 2$$

#Función final → orden de complejidad: O(n)
Constante inicial: return función final auxiliar (1)=1
Caso base: Comprobacion igualdad n=0 (1) + return acum (1)= 2 + cte inicial
Recursion: Comprobacion igualdad n!=0 (1) + return (1) + (uso de +) (1) + uso pow (1) + uso cbrt (1) + Acceso *v (1) = 6

Caso base(3) + 6 por recursión

$$T(n) = 6n + 3$$

#Función Iterativa → orden de complejidad: O(n)
Caso base: Inicializacion int i (1) + Inicializacion variable acum(1) + Validacion i<n (1) + return acum(1) = 4
Bucle: Validacion i<n(1) + Suma i++ (1) + Suma(1) + Asignacion(1) + pow (1) + cbrt (1) + LLamada v (1) = 7

$$T(n) = 7n + 4$$

Ejercicio 3. Dada una cadena de caracteres, contar cuántas vocales contiene.

Función auxiliar (pseudocódigo)

```
función esVocal (caracter)
   si caracter = 'a' o ‘A’ o 'e' o ‘E’ o 'i' o ‘I’ o 'o' o ’O’ o 'u' o ‘U’ entonces
      devolver verdadero
   sino
      devolver falso
   fin si
fin
```

Función no Final (pseudocódigo)

```
funcionNFaux(cadena, i)
   si i = longitud(cadena) entonces
      devolver 0
   fin si
   si esVocal(cadena[i]) entonces
      devolver 1 + funcionNF(cadena, i + 1)
   sino
      devolver funcionNF(cadena, i + 1)
   fin si
fin
funcionNF(cadena)
   devolver funcionNFaux(cadena, 0)
fin
```

Función Final (pseudocódigo)

```
funcion FAux(cadena, variablei, variable acum)
        si (i == cad.length()) devuelvo acum;
        devolver funcion FAux(cad, i + 1, acum + (esVocal(cad[i])));
fin FAux

funcion F(cadena)
        devolver funcionFAux(cad, 0, 0);
fin F
```

Función Iterativa (pseudocódigo)

```
funcionIterativa(cadena)
        variable acum = 0;
        bucle (variable i = 0; i < cad.length(); i++ en cada iteracion)
                si (esVocal(cad[i])) acum++
                fin si
        fin bucle
        devolver acum;
fin
```

// Calculo T(n) y orden de complejidad EJ3

#Funcion Auxiliar esVocal → return (1) + comparacion (10) + operadores OR (9) = 20

#Función no final → orden de complejidad: O(n)
Caso base: return (1) = 1
Recursion: Comprobación igualdad i==cad.length (1) + return 0 (1) + declaracion suma (1) + uso esVocal (20) + return (1) + suma del return (2) = 26

Caso base(1) + 26 por cada recursión

$$T(n) = 26n + 1$$

#Función final → orden de complejidad: O(n)
Constante inicial: return función final auxiliar (1)=1
Caso base: Comprobacion igualdad n=0 (1) + return acum (1)= 2 + cte inicial
Recursion: Comprobacion igualdad n!=0 (1) + return (1) + (uso de +) (1) + uso pow (1) + uso cbrt (1) + Acceso *v (1) = 6

Caso base(3) + 6 por recursión

$$T(n) = cn + c0$$

#Función Iterativa → orden de complejidad: O(n)
Caso base: Inicializacion int i (1) + Inicializacion variable acum(1) + Validacion i<n (1) + return acum(1) = 4
Bucle: Validacion i<n(1) + Suma i++ (1) + Suma(1) + Asignacion(1) + pow (1) + cbrt (1) + LLamada v (1) = 7

$$T(n) = cn + c0$$

Nivel Medio

Ejercicio 8. Dado un array de cadenas, encontrar la cadena con mayor longitud (la primera del vector en caso de empate).

```
funcion funcionNoFinal(v, n)
    si n == 0 entonces
        devolver ""
    si n == 1 entonces
        devolver v[0]

    resto ← funcionNoFinal(v+1, n-1)

    si tamaño(v[0]) >= tamaño(resto) entonces
        devolver v[0]
    sino
        devolver resto
fin funcion


funcion funcionFinalAux(v, n, mayor)
    si n == 0 entonces
        devolver mayor

    si tamaño(v[0]) > tamaño(mayor) entonces
        mayor ← v[0]

    devolver funcionFinalAux(v+1, n-1, mayor)
fin funcion


funcion funcionFinal(v, n)
    si n == 0 entonces
        devolver ""
    devolver funcionFinalAux(v+1, n-1, v[0])
fin funcion


funcion funcionI(v, n)
    si n == 0 entonces
        devolver ""

    mayor ← v[0]

    para i desde 1 hasta n-1 hacer
```

```
        si tamaño(v[i]) > tamaño(mayor) entonces
            mayor ← v[i]
        fin si
    fin para

    devolver mayor
fin funcion
```

Ejercicio 9. Dado un array, decidir si todos los elementos son numeros perfectos (se ´ puede hacer uso de una funcion auxiliar que compruebe si un elemento es ´ un cuadrado perfecto o no, y cuya complejidad sea lineal).

```
funcion funcionNoFinal(v, n)
    si n == 0 entonces devolver ""
    si n == 1 entonces devolver v[0]
    resto ← funcionNoFinal(v+1, n-1)
    si tamaño(v[0]) >= tamaño(resto) entonces
        devolver v[0]
    sino
        devolver resto
fin funcion


funcion funcionFinalAux(v, n, mayor)
    si n == 0 entonces devolver mayor
    si tamaño(v[0]) > tamaño(mayor) entonces
        mayor ← v[0]
    devolver funcionFinalAux(v+1, n-1, mayor)
fin funcion

funcion funcionFinal(v, n)
    si n == 0 entonces devolver ""
    devolver funcionFinalAux(v+1, n-1, v[0])
fin funcion


funcion funcionI(v, n)
    si n == 0 entonces devolver ""
    mayor ← v[0]
    para i desde 1 hasta n-1 hacer
        si tamaño(v[i]) > tamaño(mayor) entonces
            mayor ← v[i]
```

```
    fin para
    devolver mayor
fin funcion
```

Ejercicio 10. Dada una cadena de caracteres, decidir si es un pal´ındromo. Una cadena es un palíndromo si es igual a su inversa. Por ejemplo: ana, arenera, aviva, radar, reconocer, salas, …

```
funcion esPalindromo(cadena, i, j)
    si i >= j entonces
        devolver verdadero
    si cadena[i] != cadena[j] entonces
        devolver falso
    devolver esPalindromo(cadena, i+1, j-1)
fin funcion


funcion funcionNoFinal(cadena)
    devolver esPalindromo(cadena, 0, longitud(cadena) - 1)
fin funcion



funcion funcionFinalAux(cadena, i, j, acum)
    si i >= j entonces
        devolver acum
    acum ← acum Y (cadena[i] == cadena[j])
    devolver funcionFinalAux(cadena, i+1, j-1, acum)
fin funcion

funcion funcionFinal(cadena)
    devolver funcionFinalAux(cadena, 0, longitud(cadena)-1, verdadero)
fin funcion

funcion funcionIterativa(cadena)
    i ← 0
    j ← longitud(cadena) - 1

    mientras i < j hacer
        si cadena[i] != cadena[j] entonces
            devolver falso
```

```
        i ← i + 1
        j ← j - 1
    fin mientras

    devolver verdadero
fin funcion
```

EJERCICIO 15:

```
función invertir(cadena, i, mitad)
    si i < mitad entonces
        devolver ""
    devolver cadena[i] + invertir(cadena, i - 1, mitad)
fin función
```

T(n) = T(n-2) + 1
O(Tn) = n

```
función funcionNoFinal(cadena)
    mitad ← longitud(cadena) / 2 + longitud(cadena) % 2
    invertida ← invertir(cadena, longitud(cadena) - 1, mitad)
    devolver invertida + subcadena(cadena, mitad)
fin función
```

T(n) = 1
O(1)

```
función funcionFinalAux(cadena, i, mitad, acumulador)
    si i < mitad entonces
        devolver acumulador + subcadena(cadena, mitad)
    devolver funcionFinalAux(cadena, i - 1, mitad, acumulador + cadena[i])
fin función
```

T(n) = T(n-2) + 1
O(n)

```
función funcionFinal(cadena)
    mitad ← longitud(cadena) / 2 + longitud(cadena) % 2
    devolver funcionFinalAux(cadena, longitud(cadena) - 1, mitad, "")
fin función
```

T(n) = 1
O(1)

```
función funcionIterativa(cadena)
    mitad ← longitud(cadena) / 2 + longitud(cadena) % 2
    aux ← ""
    para i desde longitud(cadena) - 1 hasta mitad hacer
        aux ← aux + cadena[i]
    devolver aux + subcadena(cadena, mitad)
fin función
```

T(n) = n/2 + 1
O(n)

EJERCICIO 16

```
función esPrimo(n)
    si n == 1 devolver verdadero
    contar ← 0
    para i desde 1 hasta n hacer
        si n % i == 0 entonces contar++
    devolver contar == 2
fin función
```

T(n) = √n
O(√n)

```
función funcionNoFinal(n, i)
    si i > n devolver lista vacía
    lista ← funcionNoFinal(n, i + 1)
    si n % i == 0 y esPrimo(i) entonces
        insertar i al inicio de lista
    devolver lista
fin función
```

T(n) = T(n-1) + √n
O(n√n)

```
función funcionNoFinal(n)
    devolver funcionNoFinal(n, 1)
```

```
fin función

T(n) = n√n
O(n√n)

función funcionFinalAux(n, i, lista)
    si i > n devolver lista
    si n % i == 0 y esPrimo(i) entonces
        añadir i a lista
    devolver funcionFinalAux(n, i + 1, lista)
fin función

T(n) = T(n-1) + √n
O(n√n)

función funcionFinal(n)
    devolver funcionFinalAux(n, 1, lista vacía)
fin función

T(n) = n√n
O(n√n)

función funcionIterativa(n)
    lista ← vacía
    para i desde 1 hasta n hacer
        si n % i == 0 y esPrimo(i) entonces
            añadir i a lista
    devolver lista
fin función

T(n) = n√n O(n√n)
```

EJERCICIO 17

```
función funcionNoFinal(n)
    si n == 0 devolver {1, 0, 0}
    r ← funcionNoFinal(n / 10)
    digito ← n % 10
    si digito par entonces r.pares++
    r.producto ← r.producto * digito
    r.suma ← r.suma + digito
    devolver r
```

```
fin función
```

T(n) = T(n/10) + 1
O(log n)

```
función funcionFinalAux(n, multi, pares, suma)
    si n == 0 devolver {multi, pares, suma}
    digito ← n % 10
    si digito par entonces pares++
    multi ← multi * digito
    suma ← suma + digito
    devolver funcionFinalAux(n / 10, multi, pares, suma)
fin función
```

T(n) = T(n/10) + 1
O(log n)

```
función funcionFinal(n)
    si n == 0 devolver {0, 1, 0}
    devolver funcionFinalAux(n, 1, 0, 0)
fin función
```

T(n) = 1
O(1)

```
función funcionIterativa(n)
    multi ← 1
    pares ← 0
    suma ← 0
    mientras n > 0 hacer
        digito ← n % 10
        multi ← multi * digito
        suma ← suma + digito
        si digito par entonces pares++
        n ← n / 10
    devolver {multi, pares, suma}
fin función
```

T(n) = log n
O(log n)

EJERCICIO 18

```
función funcionNoFinal(n)
    si n == 0 devolver {0, 0, 9}
    r ← funcionNoFinal(n / 10)
    digito ← n % 10
    si digito par entonces r.suma ← r.suma + digito²
    sino r.impares++
    si digito < r.minimo entonces r.minimo ← digito
    devolver r
fin función
```

T(n) = T(n/10) + 1
O(log n)

```
función funcionFinalAux(n, suma, impares, minimo)
    si n == 0 devolver {suma, impares, minimo}
    digito ← n % 10
    si digito par entonces suma ← suma + digito²
    sino impares++
    si digito < minimo entonces minimo ← digito
    devolver funcionFinalAux(n / 10, suma, impares, minimo)
fin función
```

T(n) = T(n/10) + 1
O(log n)

```
función funcionFinal(n)
    devolver funcionFinalAux(n, 0, 0, 9)
fin función
```

T(n) = 1
O(1)

```
función funcionIterativa(n)
    suma ← 0
    impares ← 0
    minimo ← 9
    mientras n > 0 hacer
        digito ← n % 10
        si digito par entonces suma ← suma + digito²
        sino impares++
```

```
        si digito < minimo entonces minimo ← digito
        n ← n / 10
    devolver {suma, impares, minimo}
fin función
T(n) = log n
O(log n)
```

EJERCICIO 19

```
función divisionNF(A, B)
    si A < B devolver {0, A}
    r ← divisionNF(A - B, B)
    r.S ← r.S + 1
    devolver r
fin función

T(n) = T(n-1) + 1
O(n)

función divisionFAux(A, B, S)
    si A < B devolver {S, A}
    devolver divisionFAux(A - B, B, S + 1)
fin función

T(n) = T(n-1) + 1
O(n)

función divisionF(A, B)
    devolver divisionFAux(A, B, 0)
fin función

T(n) = 1
O(1)

función divisionI(A, B)
    S ← 0
    mientras A >= B hacer
        A ← A - B
        S ← S + 1
    devolver {S, A}
fin función
```

T(n) = n
O(n)

EJERCICIO 20

```
función gRecursiva(n)
   si n < 3 devolver n²
   devolver 2*gRecursiva(n-1) - gRecursiva(n-2) + gRecursiva(n-3)
fin función
```

T(n) = T(n-1) + T(n-2) + T(n-3) + 1
O(3^n)

```
función gIterativa(n)
   si n < 3 devolver n²
   g0 ← 0
   g1 ← 1
   g2 ← 4
   para i desde 3 hasta n hacer
      g3 ← 2*g2 - g1 + g0
      g0 ← g1
      g1 ← g2
      g2 ← g3
   fin para
   devolver g3
fin función
```

T(n) = n
O(n)