Ejercicio 1:Dado un array de enteros, devolver el producto de todos sus elementos.

Si el array está vacío se devuelve 1 por conveniencia.

funcionNoFinal

Pseudocódigo:

```
funcionNoFinal(v, n):
    si n == 0:
        devolver 1
    sino:
        devolver v[0] * funcionNoFinal(v+1, n-1)
```

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

funcionFinalaux

Pseudocódigo:

```
funcionFinalaux(v, n, acc):
    si n == 0:
        devolver acc
    sino:
        devolver funcionFinalaux(v+1, n-1, v[0] * acc)
```

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

funcionFinal

Pseudocódigo:

```
funcionFinal(v, n):
    devolver funcionFinalaux(v, n, 1)
```

T(n) = 1
O(1)

funcionIterativa

Pseudocódigo:

```
funcionIterativa(v, n):
    aux ← 1
    para i desde 0 hasta n-1:
        aux ← aux * v[i]
    devolver aux
```

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

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

funcionNoFinal
Pseudocódigo:

```
funcionNoFinal(v, n):
    si n == 0:
        devolver 0
    sino:
        devolver (cbrt(v[0])^2) + funcionNoFinal(v+1, n-1)
```

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

funcionFinalaux
Pseudocódigo:

```
funcionFinalaux(v, n, acc):
    si n == 0:
        devolver acc
    sino:
        devolver funcionFinalaux(v+1, n-1, acc + (cbrt(v[0])^2))
```

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

funcionFinal
Pseudocódigo:

```
funcionFinal(v, n):
    devolver funcionFinalaux(v, n, 0)
```

T(n) = 1
O(1)

funcionIterativa
Pseudocódigo:

```
funcionIterativa(v, n):
    aux ← 0
    para i desde 0 hasta n-1:
        aux ← aux + (cbrt(v[i])^2)
    devolver aux
```

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

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

esVocal
Pseudocódigo:

```
esVocal(c):
   devolver (c es 'a','e','i','o','u' mayúscula o minúscula)
```

T(n) = 1
O(1)

funcionNFaux
Pseudocódigo:

```
funcionNFaux(cad, i):
   si i == longitud(cad):
      devolver 0
   suma ← esVocal(cad[i])
   devolver suma + funcionNFaux(cad, i+1)
```

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

funcionNF
Pseudocódigo:

```
funcionNF(cad):
   devolver funcionNFaux(cad, 0)
```

T(n) = 1
O(1)

funcionFaux
Pseudocódigo:

```
funcionFaux(cad, i, acc):
   si i == longitud(cad):
      devolver acc
   devolver funcionFaux(cad, i+1, acc + esVocal(cad[i]))
```

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

funcionF
Pseudocódigo:

```
funcionF(cad):
   devolver funcionFaux(cad, 0, 0)
```

T(n) = 1
O(1)

funcionIterativa

Pseudocódigo:

```
funcionIterativa(cad):
    acumulador ← 0
    para i desde 0 hasta longitud(cad)-1:
        acumulador ← acumulador + esVocal(cad[i])
    devolver acumulador
```

$T(n) = n + 1$

$O(n)$

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

funcionNoFinal
Pseudocódigo:

```
funcionNoFinal(v, n):
    si n == 0:
        devolver ""
    si n == 1:
        devolver v[0]
    resto ← funcionNoFinal(v+1, n-1)
    si longitud(v[0]) >= longitud(resto):
        devolver v[0]
    sino:
        devolver resto
```

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

funcionFinalAux
Pseudocódigo:

```
funcionFinalAux(v, n, mayor):
    si n == 0:
        devolver mayor
    si longitud(v[0]) > longitud(mayor):
        mayor ← v[0]
    devolver funcionFinalAux(v+1, n-1, mayor)
```

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

funcionFinal
Pseudocódigo:

```
funcionFinal(v, n):
    si n == 0:
        devolver ""
    devolver funcionFinalAux(v+1, n-1, v[0])
```

T(n) = 1
O(1)

funcionI
Pseudocódigo:

```
funcionI(v, n):
    si n == 0:
        devolver ""
    mayor ← v[0]
    para i desde 1 hasta n-1:
        si longitud(v[i]) > longitud(mayor):
            mayor ← v[i]
    devolver mayor
```

T(n) = n
O(n)

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

esPerfecto

Pseudocódigo:

```
esPerfecto(n):
    si n < 0:
        devolver falso
    divisores ← {1}
    para i desde 2 hasta n/2:
        si n mod i == 0:
            añadir i a divisores
    aux ← 0
    para cada elemento t en divisores:
        aux ← aux + t
    devolver (aux == n)
```

T(n) = n + n/2

O(n)

funcionNoFinal

Pseudocódigo:

```
funcionNoFinal(v, n):
    si n == 0:
        devolver verdadero
    devolver esPerfecto(v[0]) AND funcionNoFinal(v+1, n-1)
```

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

$O(n^2)$

funcionFinalaux

Pseudocódigo:

```
funcionFinalaux(v, n, a):
    si n == 0:
        devolver a
    devolver funcionFinalaux(v+1, n-1, a AND esPerfecto(v[0]))
```

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

$O(n^2)$

funcionFinal

Pseudocódigo:

```
funcionFinal(v, n):
    devolver funcionFinalaux(v, n, verdadero)
```

T(n) = 1

O(1)

funcionIterativa
Pseudocódigo:

```
funcionIterativa(v, n):
    si n == 0:
        devolver verdadero
    para i desde 0 hasta n-1:
        si esPerfecto(v[i]) == falso:
            devolver falso
    devolver verdadero
```

$T(n) = n * n$
$O(n^2)$

Ejercicio 15. Encontrar el espejo de la parte derecha de una cadena de caracteres. Por ejemplo, dada la palabra "antonio", el resultado sería "oinonio".

funcionNoFinal (versión auxiliar)
Pseudocódigo:

```
funcionNoFinal(entrada, i, n):
    si i ≥ n:
        devolver
    funcionNoFinal(entrada, i+1, n-1)
    entrada[i] ← entrada[n]
```

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

funcionNoFinal (versión pública)
Pseudocódigo:

```
funcionNoFinal(entrada):
    si tamaño(entrada) ≤ 1:
        devolver entrada
    funcionNoFinal(entrada, 0, tamaño(entrada)-1)
    devolver entrada
```

T(n) = 1
O(1)

funcionFinal (versión auxiliar)
Pseudocódigo:

```
funcionFinal(entrada, i, n):
    si i ≥ n:
        devolver
    entrada[i] ← entrada[n]
    funcionFinal(entrada, i+1, n-1)
```

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

funcionFinal (versión pública)
Pseudocódigo:

```
funcionFinal(entrada):
    si tamaño(entrada) ≤ 1:
        devolver entrada
    funcionFinal(entrada, 0, tamaño(entrada)-1)
    devolver entrada
```

T(n) = 1
O(1)

funcionIterativa
Pseudocódigo:

```
funcionIterativa(entrada):
    n ← tamaño(entrada)-1
    i ← 0
    mientras i < n:
        entrada[i] ← entrada[n]
        i ← i + 1
        n ← n - 1
    devolver entrada
```

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

$O(n)$

Ejercicio 16. Dado un entero, obtener una lista de sus divisores primos (se puede implementar una función que determine si un número es primo o no, y cuya complejidad sea lineal). Consideraremos el 1 como número primo.

esPrimo
Pseudocódigo:

```
esPrimo(n):
    si n == 1 o n == 0:
        devolver falso
    si n == 2:
        devolver verdadero
    si n mod 2 == 0:
        devolver falso
    para i desde 3 hasta √n, paso 2:
        si n mod i == 0:
            devolver falso
    devolver verdadero
```

$T(n) = \sqrt{n}$
$O(\sqrt{n})$

---

funcionNoFinal (auxiliar)
Pseudocódigo:

```
funcionNoFinal(n, i, div):
    si i ≥ n:
        devolver
    si n mod i == 0 y esPrimo(i):
        añadir i a div
    funcionNoFinal(n, i+1, div)
```

$T(n) = T(n-1) + \sqrt{n}$
$O(n\sqrt{n})$

funcionNoFinal (principal)
Pseudocódigo:

```
funcionNoFinal(n):
    div ← lista vacía
    añadir 1 a div
    si esPrimo(n):
        añadir n a div
        devolver div
    funcionNoFinal(n, 2, div)
    devolver div
```

$T(n) = n\sqrt{n}$
$O(n\sqrt{n})$

---

funcionFinal (auxiliar)
Pseudocódigo:

```
funcionFinal(n, i, div):
    si i ≥ n:
        devolver
    si n mod i == 0 y esPrimo(i):
        añadir i a div
    funcionFinal(n, i+1, div)
```

$T(n) = T(n-1) + \sqrt{n}$
$O(n\sqrt{n})$

funcionFinal (principal)
Pseudocódigo:

```
funcionFinal(n):
    div ← lista vacía
    añadir 1 a div
    si esPrimo(n):
        añadir n a div
        devolver div
    funcionFinal(n, 2, div)
    devolver div
```

$T(n) = n\sqrt{n}$
$O(n\sqrt{n})$

---

funcionIterativa
Pseudocódigo:

```
funcionIterativa(n):
    si n ≤ 0:
        devolver lista vacía
    div ← lista vacía
    añadir 1 a div
    si esPrimo(n):
        añadir n a div
        devolver div
    para i desde 2 hasta n-1:
        si n mod i == 0 y esPrimo(i):
            añadir i a div
    devolver div
```

$T(n) = n\sqrt{n}$
$O(n\sqrt{n})$

Ejercicio 17. Dado un número, determinar el producto, el número de cifras pares y la suma de sus cifras. Por ejemplo: dado 5394, el resultado debe ser {540, 1, 21}.

esPar
Pseudocódigo:

```
esPar(n):

    devolver (n mod 2 == 0)
```

T(n) = 1
O(1)

---

funcionNoFinal
Pseudocódigo:

```
funcionNoFinal(n):

    si n == 0:

        devolver {0, 1, 0}

    si n < 10:

        digito ← n

        devolver {digito, esPar(digito), digito}

    r ← funcionNoFinal(n / 10)

    digito ← n mod 10

    r.producto ← r.producto * digito

    si esPar(digito):

        r.nPares ← r.nPares + 1

    r.suma ← r.suma + digito

    devolver r
```

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

funcionFinal (auxiliar)
Pseudocódigo:

```
funcionFinal(n, r):

    si n == 0:

        devolver r

    digito ← n mod 10

    r.producto ← r.producto * digito

    si esPar(digito):

        r.nPares ← r.nPares + 1

    r.suma ← r.suma + digito

    devolver funcionFinal(n / 10, r)
```

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

funcionFinal (principal)
Pseudocódigo:

```
funcionFinal(n):

    si n == 0:

        devolver {0, 1, 0}

    devolver funcionFinal(n, {1, 0, 0})
```

$T(n) = 1$
$O(1)$

funcionIterativa
Pseudocódigo:

```
funcionIterativa(n):

    si n == 0:

        devolver {0, 1, 0}

    producto ← 1

    pares ← 0

    suma ← 0

    mientras n > 0:

        digito ← n mod 10

        producto ← producto * digito

        si esPar(digito):

            pares ← pares + 1

        suma ← suma + digito

        n ← n / 10

    devolver {producto, pares, suma}
```

$T(n) = \log n$
$O(\log n)$

Ejercicio 18. Dado un número entero, obtener la suma de los cuadrados de las cifras pares, la cantidad de cifras impares, y el valor mínimo de sus cifras. Por ejemplo: dado 5863, el resultado sería {100, 2, 3}.

funcionNoFinal
Pseudocódigo:

```
funcionNoFinal(n):

    si n == 0:

        devolver {0, 0, 0}

    si n < 10:

        digito ← n

        si digito es par:

            devolver {digito*digito, 0, digito}

        sino:

            devolver {0, 1, digito}

    r ← funcionNoFinal(n / 10)

    digito ← n mod 10

    si digito es par:

        r.suma ← r.suma + digito*digito

    si digito es impar:

        r.nImpares ← r.nImpares + 1

    si digito < r.minimo:

        r.minimo ← digito

    devolver r
```

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

funcionFinal (auxiliar)
Pseudocódigo:

```
funcionFinal(n, r):

    si n == 0:

        devolver r

    digito ← n mod 10

    si digito es par:

        r.suma ← r.suma + digito*digito

    si digito es impar:

        r.nImpares ← r.nImpares + 1

    si digito < r.minimo:

        r.minimo ← digito

    devolver funcionFinal(n / 10, r)
```

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

funcionFinal (principal)
Pseudocódigo:

```
funcionFinal(n):

    si n == 0:

        devolver {0, 0, 0}

    devolver funcionFinal(n, {0, 0, 9})
```

T(n) = 1
O(1)

funcionIterativa

Pseudocódigo:

```
funcionIterativa(n):

    si n == 0:

        devolver {0, 0, 0}

    r ← {0, 0, 9}

    mientras n > 0:

        digito ← n mod 10

        si digito es par:

            r.suma ← r.suma + digito*digito

        si digito es impar:

            r.nImpares ← r.nImpares + 1

        si digito < r.minimo:

            r.minimo ← digito

        n ← n / 10

    devolver r
```

T(n) = log n

O(log n)

Ejercicio 19. Realice la División por restas sucesivas. Si se desea realizar A/B y obtener el resultado S y el resto R, el proceso consistirá en restar a A la cantidad B hasta que el resultado de la resta sea menor que el propio B. S será el número de operaciones de resta realizadas y R el resultado de la última resta.

divisionNF

Pseudocódigo:

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

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

O(n)

---

divisionF (auxiliar)

Pseudocódigo:

```
divisionF(A, B, r):
    si A < B:
        r.R ← A
        devolver r
    r.S ← r.S + 1
    devolver divisionF(A - B, B, r)
```

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

O(n)

divisionF (principal)

Pseudocódigo:

```
divisionF(A, B):
    r ← {0, 0}
    devolver divisionF(A, B, r)
```

T(n) = 1

O(1)

---

divisionI
Pseudocódigo:

```
divisionI(A, B):
    r ← {0, 0}
    mientras A ≥ B:
        A ← A - B
        r.S ← r.S + 1
    r.R ← A
    devolver r
```

T(n) = n
O(n)

Ejercicio 20. Realizar la transformación a iterativo de la siguiente función recursiva definida para n ≥ 0:

funcionRecursiva
Pseudocódigo:

```
funcionRecursiva(n):
    si n < 3:
        devolver n*n
    devolver 2*funcionRecursiva(n-1)
          - funcionRecursiva(n-2)
          + funcionRecursiva(n-3)
```

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

---

funcionIterativa
Pseudocódigo:

```
funcionIterativa(n):
    si n < 3:
        devolver n*n
    g0 ← 0
    g1 ← 1
    g2 ← 4
```

```
    para i desde 3 hasta n:
        g ← 2*g2 - g1 + g0
        g0 ← g1
        g1 ← g2
        g2 ← g
    devolver g2
```

$T(n) = n$

$O(n)$