Introducción a la Programación

Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Medidas de análisis de programas

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Solución: Contar la cantidad de operaciones que realiza un programa.

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Solución: Contar la cantidad de operaciones que realiza un programa.

Vamos a contar operaciones elementales (**OEs**).

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Solución: Contar la cantidad de operaciones que realiza un programa.

Vamos a contar operaciones elementales (**OEs**).

El tiempo de una **OE** es, por definición, **1**.

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Solución: Contar la cantidad de operaciones que realiza un programa.

Vamos a contar operaciones elementales (**OEs**).

El tiempo de una **OE** es, por definición, **1**.

Algunos ejemplos de OEs:

- Asignación de un valor a una variable.
- Comparación entre dos números.
- Operaciones aritméticas (+, -, *, %).
- Devolver el valor de una variable (return).

Contar operaciones

Problema: Queremos medir la eficiencia de nuestros programas.

Solución: Contar la cantidad de operaciones que realiza un programa.

Vamos a contar operaciones elementales (**OEs**).

El tiempo de una **OE** es, por definición, **1**.

Algunos ejemplos de OEs:

- Asignación de un valor a una variable.
- Comparación entre dos números.
- Operaciones aritméticas (+, -, *, %).
- Devolver el valor de una variable (return).

Para un programa dado, vamos a definir una función **T(n)** que nos diga la cantidad de OEs que realiza el programa, siendo n un parámetro de entrada del mismo.

Guía 11 - Ejercicio 4 (ítem 2)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_1(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= n:
        res = res * i
        i = i + 1
    return res
```

Guía 11 - Ejercicio 4 (ítem 2)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_1(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= n:
        res = res * i
        i = i + 1
    return res
```

¿Cómo es el crecimiento de T(n) con respecto a n?

Guía 11 - Ejercicio 4 (ítem 4)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_3(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= n:
        j: int = 1
        while j <= n:
            res = res * i * j
            j = j + 1
        i = i + 1
    return res
```

Guía 11 - Ejercicio 4 (ítem 4)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_3(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= n:
        j: int = 1
        while j <= n:
            res = res * i * j
            j = j + 1
        i = i + 1
    return res
```

¿Cómo es el crecimiento de T(n) con respecto a n?

Guía 11 - Ejercicio 4 (ítem 7)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_6(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= 2**n:
        producto: int = 1
        j: int = 1
        while j <= n:
            if (i // (2 ** (j-1))) %2 == 1:
                producto = producto * j
            else:
                producto = producto * 1
            j = j + 1
        i = i + 1
        res = res * producto
    return res
```

Guía 11 - Ejercicio 4 (ítem 7)

Calcular la cantidad de operaciones T(n) que realiza la siguiente función, siendo n el parámetro de entrada.

```
def producto_6(n: int) -> int:
    res: int = 1
    i: int = 1
    while i <= 2**n:
        producto: int = 1
        j: int = 1
        while j <= n:
            if (i // (2 ** (j-1))) %2 == 1:
                producto = producto * j
            else:
                producto = producto * 1
            j = j + 1
        i = i + 1
        res = res * producto
    return res
```

¿Cómo es el crecimiento de T(n) con respecto a n?

Guía 11 - Ejercicio 5.5

Calcular T(n) para ambas implementaciones del siguiente problema, siendo n la cantidad de filas de la matriz m.

`problema diag_principal` (in m:$seq\langle seq\langle \mathbb{Z} \rangle \rangle$) : $seq\langle \mathbb{Z} \rangle$ {
  `requiere:` { esMatriz(m) }
  `requiere:` { $|m| = |m[0]|$ }
  `asegura:` { $res =$ a los elementos de la diagonal principal de m }
}

Guía 11 - Ejercicio 5.5

Calcular T(n) para ambas implementaciones del siguiente problema, siendo n la cantidad de filas de la matriz m.

```
def diag_principal_v1(m: list[list[int]]) -> list[int]:
    res: list[int] = []
    n: int = len(m)
    i: int = 0
    while i < n:
        j: int = 0
        while j < n:
            if i == j:
                res.append(m[i][j])
            j = j + 1
        i = i + 1
    return res
```

Guía 11 - Ejercicio 5.5

Calcular T(n) para ambas implementaciones del siguiente problema, siendo n la cantidad de filas de la matriz m.

```
def diag_principal_v2(m: list[list[int]]) -> list[int]:
    res: list[int] = []
    n: int = len(m)
    i: int = 0
    while i < n:
        res.append(m[i][i])
        i = i + 1
    return res
```

Guía 11 - Ejercicio 5.5

Calcular T(n) para ambas implementaciones del siguiente problema, siendo n la cantidad de filas de la matriz m.

```
def diag_principal_v2(m: list[list[int]]) -> list[int]:
    res: list[int] = []
    n: int = len(m)
    i: int = 0
    while i < n:
        res.append(m[i][i])
        i = i + 1
    return res
```

Comparar entre sí los valores de T(n) obtenidos para cada implementación.

Guía 11 - Ejercicio 6.1

Buscar el mejor y el peor caso del parámetro de entrada. Calcular $T_{mejor}(n)$ y $T_{peor}(n)$ y comparar ambos valores entre sí, siendo n el tamaño de la lista s.

`problema contar_pares` (in s:$seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ $\{$
  `requiere:` $\{ True \}$
  `asegura:` $\{ res =$ a la suma de los elementos pares de s $\}$
$\}$

Guía 11 - Ejercicio 6.1

Buscar el mejor y el peor caso del parámetro de entrada. Calcular $T_{mejor}(n)$ y $T_{peor}(n)$ y comparar ambos valores entre sí, siendo n el tamaño de la lista s.

```
def contar_pares(s: list[int]) -> int:
    res: int = 0
    i: int = 0
    while i < len(s):
        if s[i] %2 == 0:
            res = res + s[i]
        i = i + 1
    return res
```

Guía 11 - Ejercicio 6.2

Buscar el mejor y el peor caso del parámetro de entrada. Calcular $T_{mejor}(n)$ y $T_{peor}(n)$ y comparar ambos valores entre sí, siendo n el tamaño de la lista s.

`problema suma_hasta_umbral` (in s:$seq\langle\mathbb{Z}\rangle$, in umbral: $\mathbb{Z}$) : $\mathbb{Z}$ $\{$
  `requiere:` $\{ True \}$
  `asegura:` $\{ res =$ a la suma de los elementos de s menores a umbral que aparecen en forma consecutiva al inicio de s $\}$
$\}$

Guía 11 - Ejercicio 6.2

Buscar el mejor y el peor caso del parámetro de entrada. Calcular $T_{mejor}(n)$ y $T_{peor}(n)$ y comparar ambos valores entre sí, siendo n el tamaño de la lista s.

```
def suma_hasta_umbral(s: list[int], umbral: int) -> int:
    res: int = 0
    i: int = 0
    while i < len(s) and s[i] < umbral:
        res = res + s[i]
        i = i + 1
    return res
```