

**CONCEPTO DE CORRECCION  
CONCEPTO DE EFICIENCIA  
EJEMPLOS**

**Teoría– Cecilia Sanz**

# Temas que presentaremos

- ✓ ANÁLISIS DE LOS FACTORES DE CALIDAD DE LOS PROGRAMAS.
- ✓ IMPORTANCIA DE LA DOCUMENTACIÓN DE UN PROGRAMA.
- ✓ CORRECCIÓN DE ALGORITMOS. VERIFICACIÓN.
- ✓ EFICIENCIA DE UN ALGORITMO.
- ✓ ANÁLISIS DE EJEMPLOS DE PROGRAMAS SIMPLES.

# Repaso



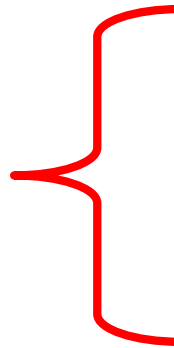
Programa



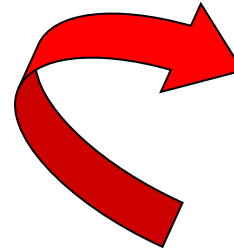
¿Cómo elijo entre distintos programas que resuelven el mismo programa?

**Criterios de Calidad**

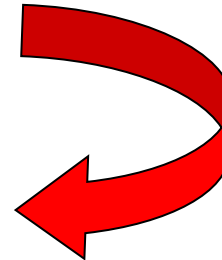
Por ejemplo



Instrucciones  
+  
Datos



Estructuras de control



Pertenece a un

TIPO de Datos

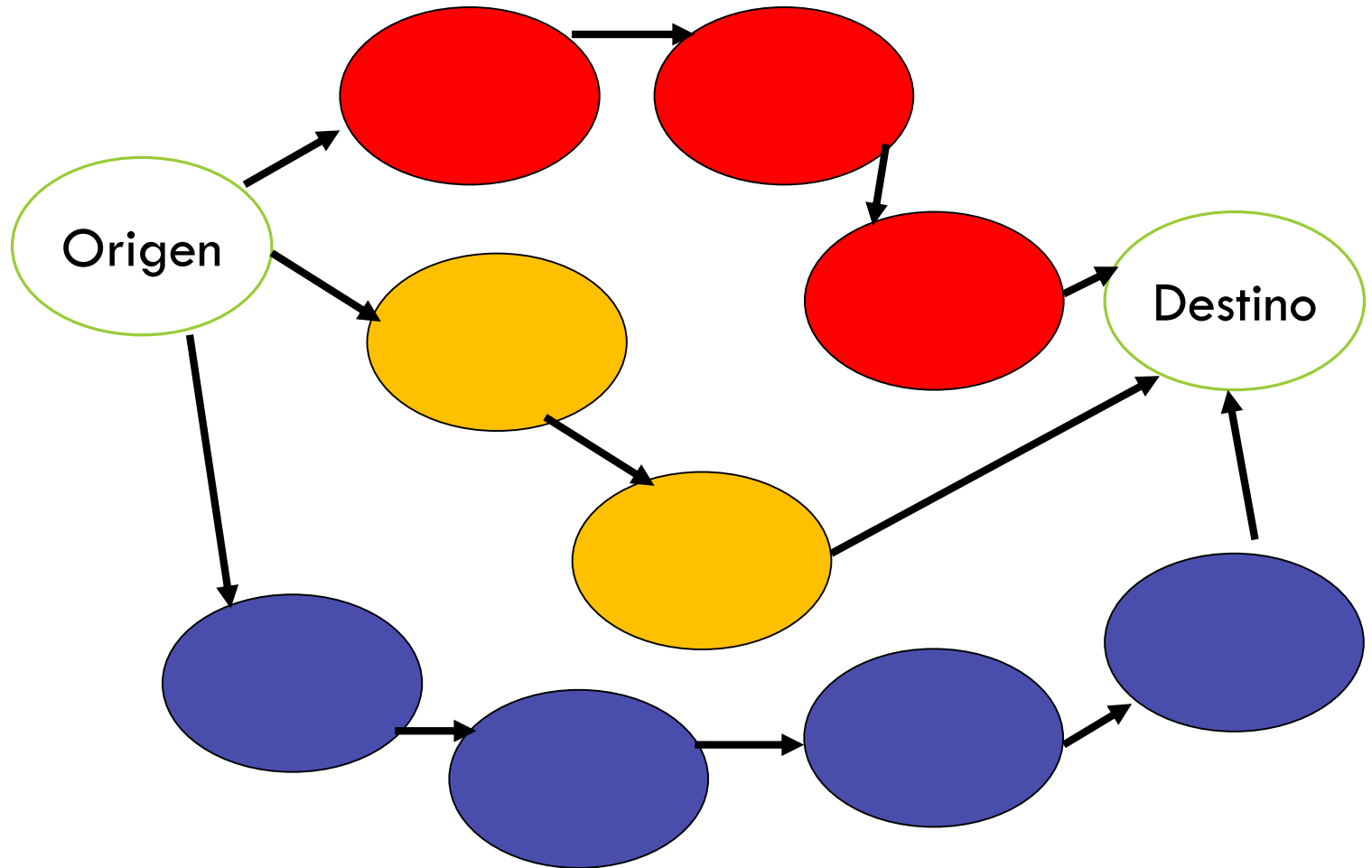


Hoy veremos...

MOTIVACIÓN

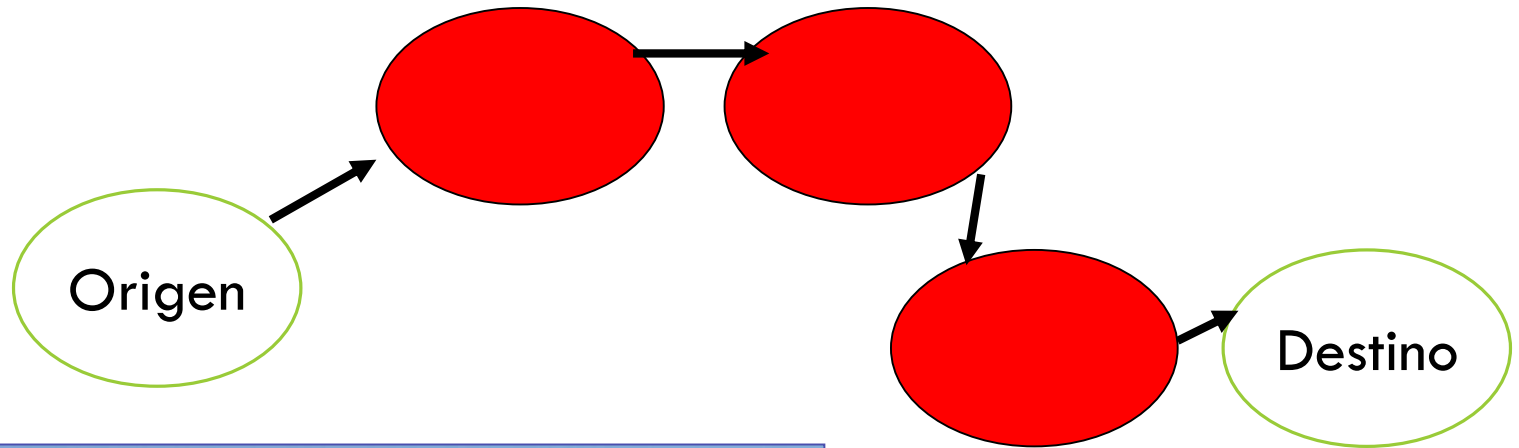


# ¡Vamos de viaje!



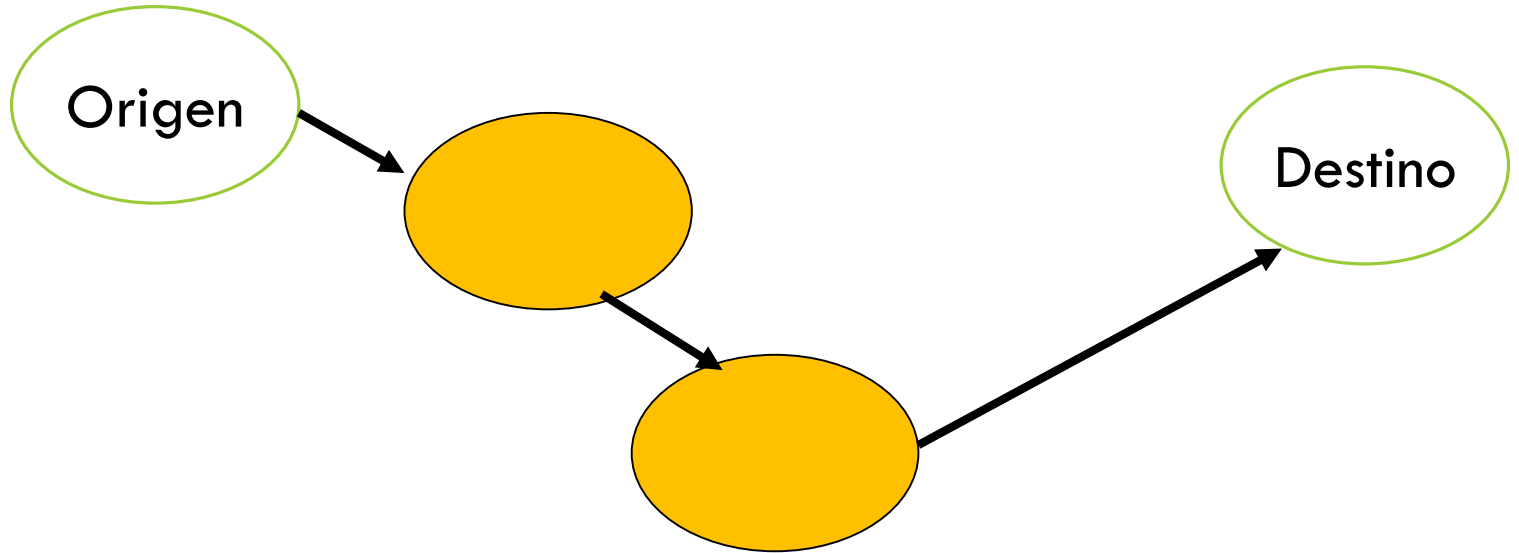
¿Cómo elegimos el camino?

# Vamos de viaje



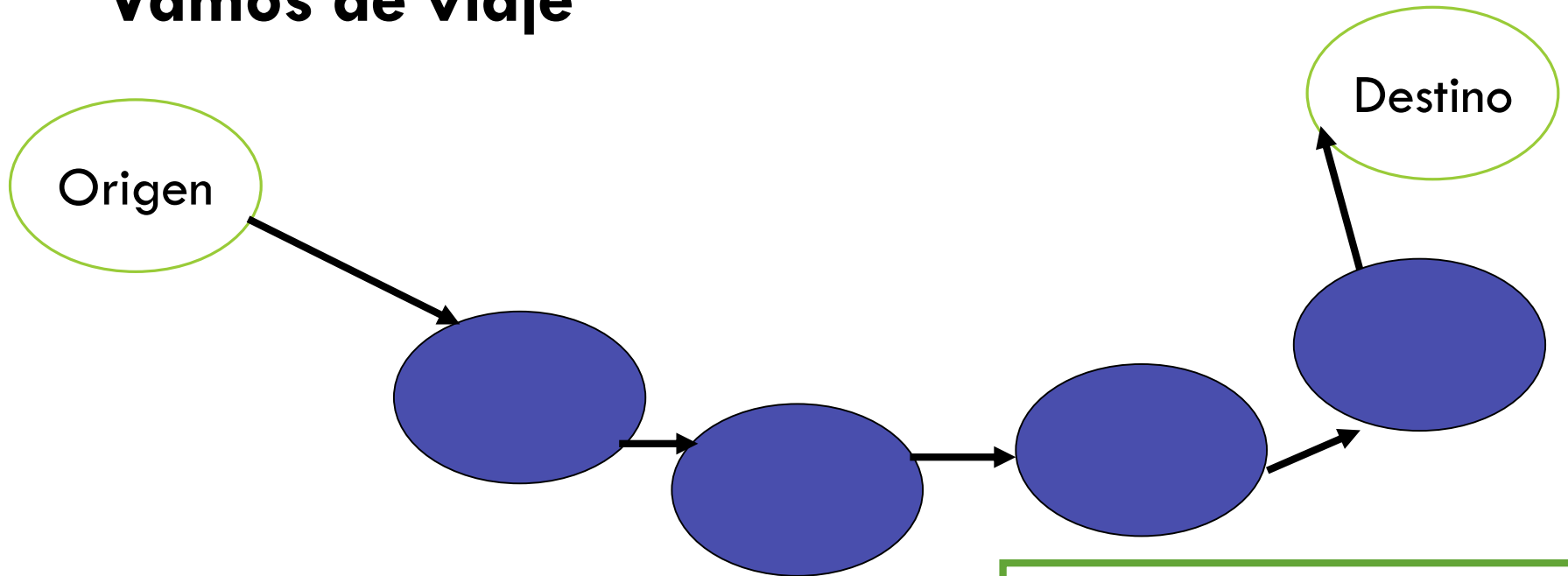
El camino rojo tiene muy buenas vistas, pero es más largo que el naranja.

# Vamos de viaje



El camino naranja es corto. Pero la vista no es muy agradable.

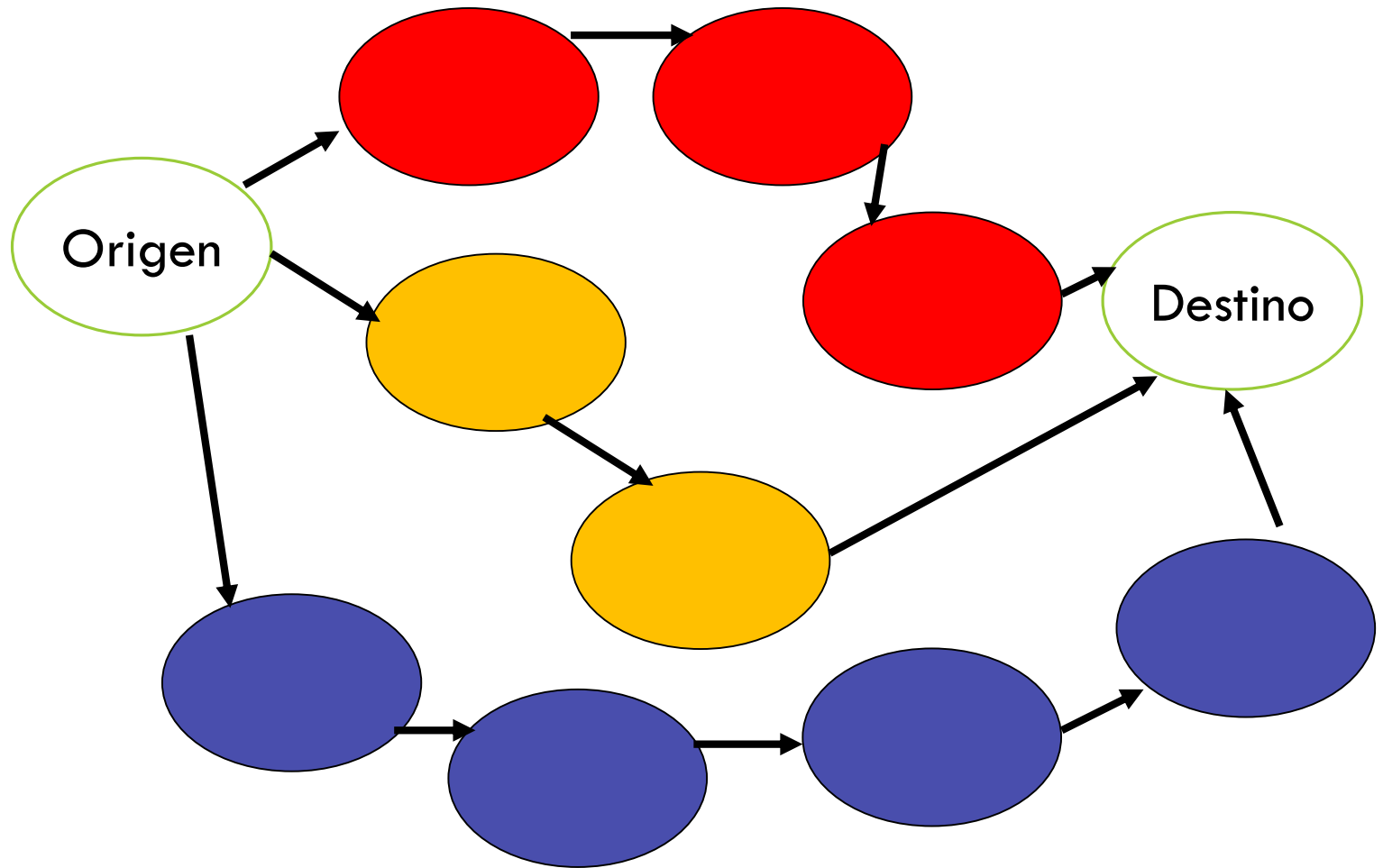
# Vamos de viaje



El camino azul es largo.  
Pero está muy bien y  
tiene muchos servicios.



# Vamos de viaje



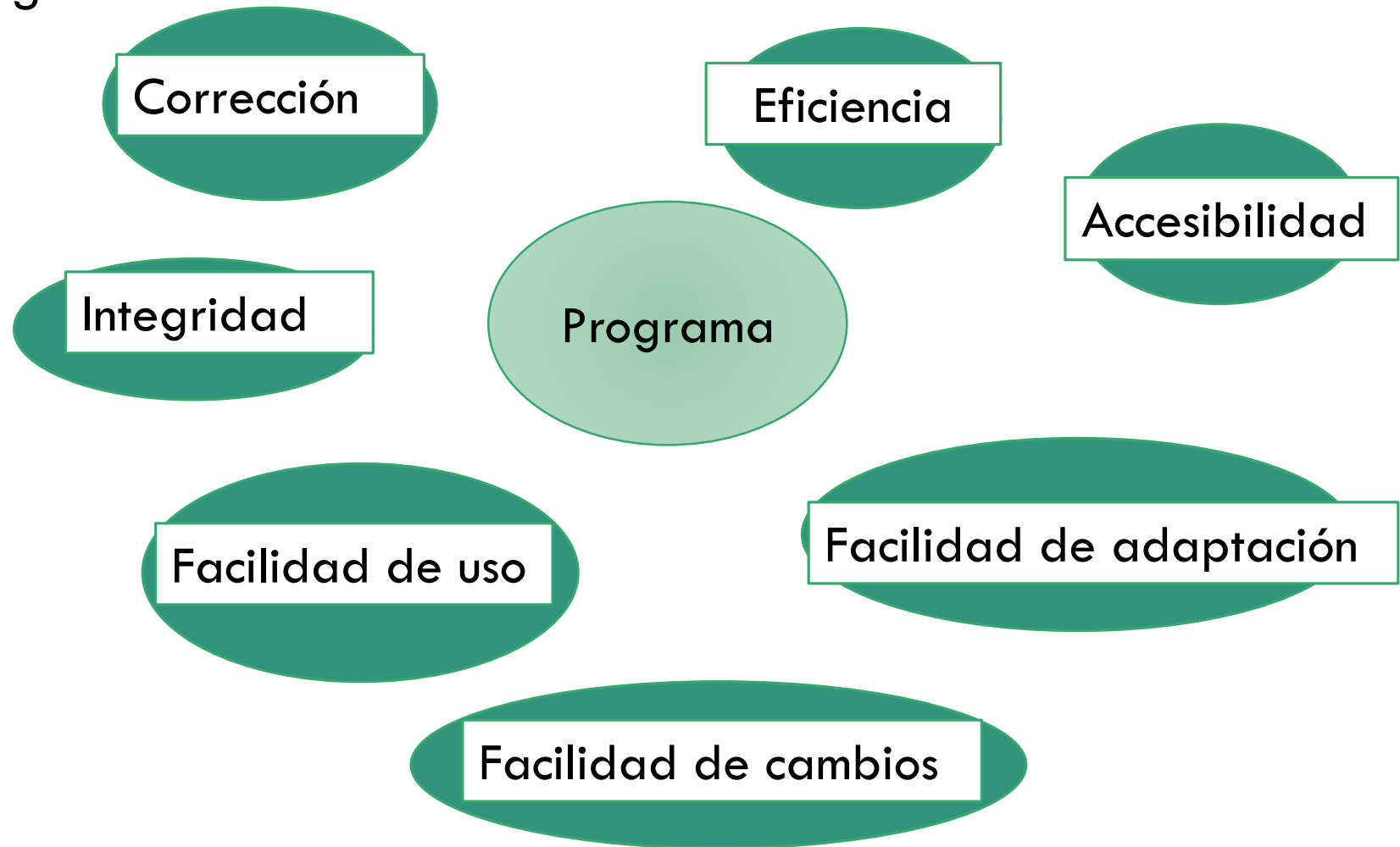
¿Qué camino elijo?

# Vamos de viaje

- Cuando debemos tomar decisiones, existe siempre una serie de factores que se deben analizar y buscar optimizar.
- En algunos casos, debemos buscar optimizar algunos criterios en pos de sacrificar otros, dependiendo del contexto y las necesidades específicas de la situación.

# Diferentes criterios de Calidad de un Programa

Los programas deben atender a diferentes criterios que hacen a su calidad. Algunos criterios de calidad son los que se muestran en la figura.



# Factores que afectan a la Calidad de un Programa

## Algunos factores que determinan la calidad del software

Se clasifican en tres grupos:

- **Operaciones del producto: CARACTERÍSTICAS OPERATIVAS**

- **Corrección** (¿Hace lo que se le pide?)

El grado en que una aplicación satisface sus especificaciones y consigue los objetivos encomendados por el cliente

- **Eficiencia** (¿Cómo se usan los recursos del sistema?)

Atiende a cómo el programa usa la memoria y el tiempo que consume su ejecución

# Factores que afectan a la Calidad de un Programa

Algunos factores que determinan la calidad del software

- Operaciones del producto: **CARACTERISTICAS OPERATIVAS**

- *Integridad de los datos que maneja* (¿Puedo controlar su uso?)

El grado con que puede controlarse el acceso al software o a los datos a personal no autorizado

- *Facilidad de uso* (¿Es fácil y cómodo de manejar?)

El esfuerzo requerido para aprender el manejo de una aplicación, trabajar con ella, introducir datos y conseguir resultados

# Factores que afectan a la Calidad de un Programa

## Factores que determinan la calidad del software

- **Revisión del producto: CAPACIDAD PARA SOPORTAR CAMBIOS**

- *Facilidad de mantenimiento* (¿Puedo localizar los fallos?)

El esfuerzo requerido para localizar y reparar errores → Se va a vincular con la modularización y con cuestiones de legibilidad y documentación.

- *Flexibilidad* (¿Puedo añadir nuevas opciones?)

El esfuerzo requerido para modificar una aplicación en funcionamiento

- *Facilidad de prueba* (¿Puedo probar todas las opciones?)

El esfuerzo requerido para probar una aplicación de forma que cumpla con lo especificado en los requisitos

# Calidad de un Programa

## Factores que determinan la calidad del software

- **Transición del producto: ADAPTABILIDAD A NUEVOS ENTORNOS**

- *Portabilidad* (¿Podré usarlo en otra máquina?)

El esfuerzo requerido para transferir la aplicación a otro hardware o sistema operativo

- *Reusabilidad* (¿Podré utilizar alguna parte del software en otra aplicación?)

Grado en que partes de una aplicación pueden utilizarse en otras aplicaciones

- *Interoperabilidad* (¿Podrá comunicarse con otras aplicaciones o sistemas informáticos?)

El esfuerzo necesario para comunicar la aplicación con otras aplicaciones o sistemas informáticos

# Otras Características deseables de un Programa que hacen a su calidad

**Legibilidad :** el código fuente de un programa debe ser fácil de leer y entender. Esto obliga a acompañar a las instrucciones con comentarios adecuados. **Relacionado con la presentación de documentación.**

**Documentados:** todo el proceso de análisis y diseño del problema y su solución debe estar documentado mediante texto y/o gráficos para favorecer la comprensión, la modificación y la adaptación a nuevas funciones.



# Veremos

En este curso vamos a trabajar con dos conceptos fundamentales a la hora de desarrollar programas y atender a su calidad:

- **Corrección**
- **Eficiencia**

# CORRECCIÓN



# ¿Cuándo se considera correcto un programa?

Un programa es correcto cuando cumple con la función especificada; esto significa que cumple con los requerimientos propuestos.

Para determinar cuáles son esos requerimientos se debe tener una especificación **completa**, **precisa** y **no ambigua** del problema a resolver antes de escribir el programa.

# Corrección de algoritmos.

Suponga que se debe calcular un promedio de números enteros que se ingresan hasta ingresar el -1 (no se procesa).

**Program** promedio;

*Var num, total, suma: integer;*

**Begin**

total:=0; suma:=0;

readln(num);

while (num<>-1) do

begin

total:=total+1;

suma:=suma+num;

readln(num);

End;

writeln('El promedio es: ', suma/total);

**End.**

¿Este programa  
es correcto?

Revisemos esto.  
¿Funciona siempre?

---

# **PROCESO DE VERIFICACIÓN Y VALIDACIÓN**

# Corrección de Programas

## PROCESOS DE VERIFICACIÓN Y VALIDACIÓN

---

Los procesos de verificación y validación se llevan a cabo para analizar si un programa es correcto. La verificación y la validación no son la misma cosa , aunque es muy fácil confundirlas:

**Verificación: ¿Estamos construyendo el producto correctamente?**

El papel de la verificación comprende comprobar que el software está de acuerdo con su especificación. Se comprueba que el sistema cumple los requerimientos funcionales y no funcionales que se le han especificado.

**Validación: ¿Estamos construyendo el producto concreto?**

La validación es un proceso más general. **Se debe asegurar que el software cumple las expectativas del cliente.** Va mas allá de comprobar si el sistema está acorde con su especificación, para probar **que el software hace lo que el usuario espera**, a diferencia de lo que se ha especificado-

# TÉCNICAS DE CORRECCIÓN



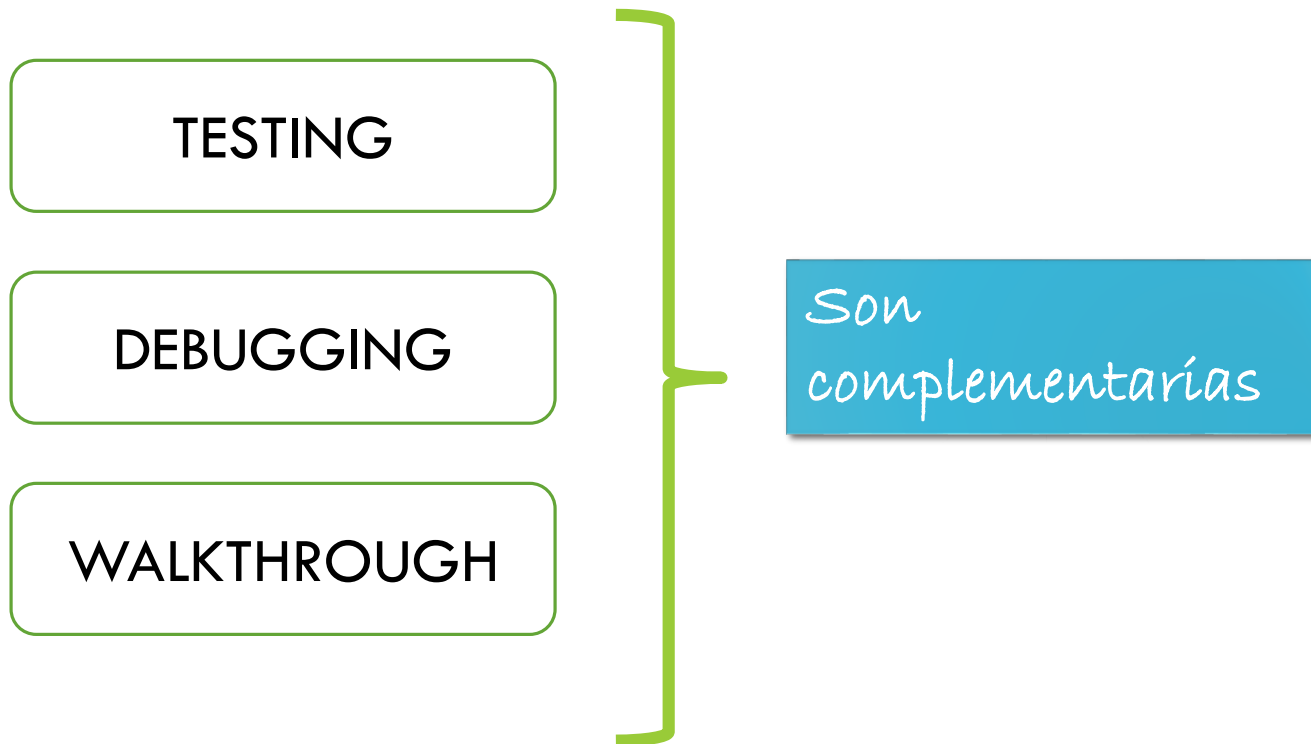
---

# **TÉCNICAS PARA MEDIR CORRECCIÓN**



# TÉCNICAS PARA MEDIR/ANALIZAR CORRECCIÓN

---



---

# **TESTING**

# **Pruebas**

# Corrección de Programas: Testing

La técnica de **Testing** es el proceso mediante el cual se proveen evidencias convincentes respecto a que el programa hace el trabajo esperado.

¿Como se proveen evidencias?

|   |  |
|---|--|
| ✓Diseñar un plan de pruebas.  | ✓Poner atención en los casos límite  |
| ✓Decidir cuales aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos. | ✓Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa.   |
| ✓Determinar el resultado que se espera que el programa produzca para cada caso de prueba                              | ✓Mejor aún, diseñar casos de prueba antes de que comience la escritura del programa. (Esto asegura que las pruebas no están pensadas a favor del que escribió el programa) |

Cuando se tiene el plan de pruebas y el programa, el plan debe aplicarse sistemáticamente.

# Corrección de Programas: Testing

Durante este proceso es importante analizar las postcondiciones en función de las precondiciones establecidas.

- Las precondiciones, junto con las postcondiciones, permiten describir la función que realiza un programa, sin especificar un algoritmo determinado.
- Las precondiciones describen los aspectos que se consideran verdaderos antes que el programa comience a ejecutarse, por ejemplo: entradas de datos disponibles.
- Las postcondiciones describen los aspectos que deben cumplirse cuando el programa terminó.

---

# **DEBUGGING**

## **Depuración**

# Corrección de Programas: Debugging

La técnica de **Debugging** es el proceso de localización del error

## Puede involucrar:

- el diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.
- el agregado de sentencias adicionales en el programa para poder monitorear su comportamiento más cercana

## Los errores pueden provenir de varios caminos, por ejemplo:

- El diseño del programa puede ser defectuoso.
- El programa puede usar un algoritmo defectuoso.

*A veces el error es tan evidente que se reconoce rápidamente en qué lugar está la falla.*

*Otras veces se puede necesitar agregar sentencias de salida adicionales que sirven como punto de control o para señalar cambios en ciertas variables claves.*

---

# **WALKTHROUGH**

## Corrección de Programas: Walkthroughs

---

### **Consiste en recorrer el programa ante una audiencia**

- La lectura de un programa a alguna otra persona provee un buen medio para detectar errores.
- Esta persona no comparte preconcepciones y está predispuesta a descubrir errores u omisiones.



**OTRO CRITERIO DE  
CALIDAD ES LA  
EFICIENCIA**

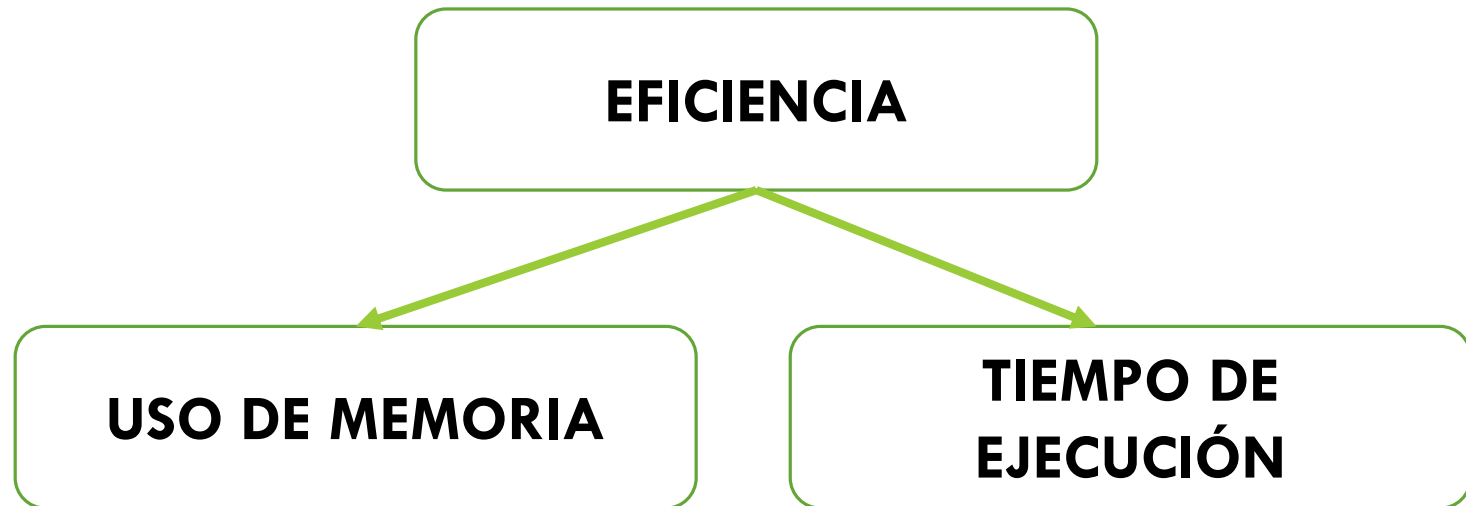
# Eficiencia de un Algoritmo

- ✓ Para cada problema se pueden tener muchas soluciones algorítmicas (por ejemplo para encontrar el menor valor de una lista de números o para que el robot vaya de un punto a otro de la ciudad).
- ✓ Sin embargo el uso de recursos (tiempo, memoria) de cada una de esas soluciones puede ser muy diferente.

Definiremos **eficiencia** como una métrica de calidad de los algoritmos, asociada con una utilización óptima de los recursos del sistema de cómputo donde se ejecutará el programa. Principalmente la memoria utilizada y el tiempo de ejecución empleado

# Eficiencia

---



# Eficiencia – Cálculo de la memoria utilizada

## USO DE MEMORIA

### Medición de la Memoria utilizada en un programa

Se puede calcular antes de la ejecución únicamente la cantidad de memoria estática que utiliza el programa.

Se deben analizar las variables declaradas y el tipo correspondiente.

# Eficiencia – Cálculo de la memoria utilizada

## ➡ Medición de la Memoria utilizada en un programa

- Se puede calcular únicamente la cantidad de memoria estática que utiliza el programa.
- Se analizan las variables declaradas y el tipo correspondiente.

¿Cuánta memoria se utiliza?

```
{declaración de tipos}
Type
cadena10 = string [10];
PtrString = ^cadena10;
Datos = record
    Nombre: cadena10;
    Apellido: cadena10;
    Edad: integer;
    Altura: real
End;
Personas = array [1..100] of Datos;
PtrDatos = ^Datos;

var
    frase : PtrString;
    s : cadena10;
    puntero : PtrDatos;
    p : personas;
```

# Eficiencia – Cálculo del tiempo

## TIEMPO DE EJECUCIÓN

## Medición del Tiempo de ejecución de un programa

Depende de distintos factores:

- ✓ **Los datos de entrada al programa**
  - Tamaño
  - Contenido y disposición
- ✓ **La calidad del código generado por el compilador utilizado**
- ✓ **La naturaleza y rapidez de las instrucciones de máquina empleadas en la ejecución del programa**
- ✓ **El tiempo del algoritmo base.**

## Eficiencia

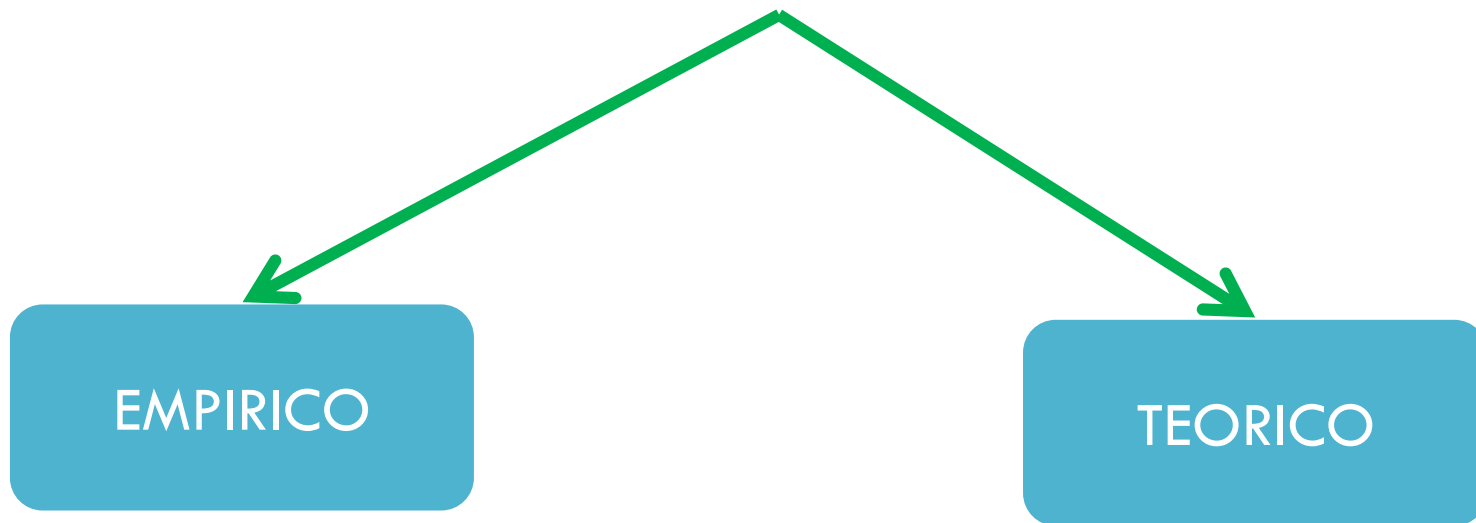
**El tiempo de ejecución de un programa debe definirse como una función de la entrada.**

Para muchos programas, el tiempo de ejecución depende de la cantidad de **datos de entrada** o **su tamaño**

Para otros programas, el tiempo de ejecución es una función de la entrada “específica”, en estos casos se habla del tiempo de ejecución del “peor” caso. En estos casos, se obtiene una cota superior del tiempo de ejecución para cualquier entrada.

# Eficiencia – Técnicas para el análisis del tiempo de ejecución

El tiempo de ejecución de un programa puede calcularse de dos maneras:





# Eficiencia

## EMPIRICO

- ✓ Para realizar un análisis empírico, es necesario realizar el programa y medir los recursos consumidos.
- **Inconveniente:** este análisis tiene varias limitaciones: puede dar una información pobre de los recursos consumidos en caso de que el programa sea aplicado a pruebas de escritorio. Por ejemplo:
  - Obtiene valores exactos para una máquina y unos datos determinados
  - Completamente dependiente de la máquina donde se ejecuta
  - Requiere implementar el algoritmo y ejecutarlo repetidas veces.

## TEORICO

- ✓ Para realizar un **análisis teórico**, es necesario establecer una medida intrínseca de la cantidad de trabajo realizado por el algoritmo; esta medida nos permite comparar algoritmos y seleccionar la mejor implementación.
- ✓ VENTAJAS
  - Obtiene valores aproximados
  - Es aplicable en la etapa de diseño de los algoritmos, uno de los aspectos fundamentales a tener en cuenta
  - Independiente de la máquina donde se ejecute
  - Permite analizar el comportamiento
  - Se puede aplicar sin necesidad de implementar el algoritmo.
  - La predicción del costo puede evitar una implementación posiblemente compleja.

# CÁLCULO TEÓRICO DEL TIEMPO DE EJECUCIÓN

# Eficiencia de Programas – Tiempo de Ejecución

## Reglas Generales

**Regla 1:** For

**Regla 2:** For anidados

**Regla 3:** sentencias consecutivas

**Regla 4:** If / else

Los comentarios y  
declaraciones no se  
consideran para el  
cálculo

Puede darse el caso que el algoritmo mas adecuado en cada momento dependa del tamaño de la entrada:

Algoritmo1:  $f(n) = 4n - 1$

Algoritmo2:  $h(n) = 2n + 10^6$

En estas situaciones hay que considerar cuál es el algoritmo a utilizar teniendo en cuenta el tamaño de la entrada.

# Eficiencia – Consideraciones generales

---

- Se tendrá en cuenta como una medida de eficiencia en cuanto al tiempo de ejecución **el número de operaciones elementales que emplea el algoritmo.**
- Una operación elemental **utiliza un tiempo constante para su ejecución**, independientemente del tipo de dato con el que trabaje.
- Se considera que cada operación elemental **se ejecuta en una unidad de tiempo.**
- Una operación elemental es
  - una asignación,
  - una comparación
  - una operación aritmética simple

# Eficiencia de Programas – Tiempo de Ejecución

Sentencias consecutivas

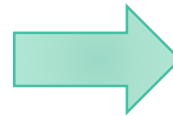
instruccion 1  
instruccion 2  
instrucción 3

For

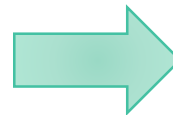
for i:= 1 to N do  
begin  
instruccion 1  
instruccion 2  
end;

If

If (condicion) then  
instruccion 1  
Else  
instruccion 2



$N * \text{MAX} (\text{tiempo de cada instruccion})$



$\text{MAX} (\text{tiempo if} , \text{tiempo else})$

# Eficiencia de Programas – Tiempo de Ejecución



Recordemos un ejemplo ya visto: Se necesita conocer la cantidad de veces que aparece la temperatura con valor 10 en un vector de temperaturas.

```
Type temperaturas = array [1..30] of real;  
  
Function contar ( tem:temperaturas): integer;  
Var i: 1..30; can10 : integer;  
begin  
    can10 := 0;  
    {recorrido total del vector}  
    For i := 1 to 30 do  
        If (tem [i] = 10 ) then can10 := can10 + 1;  
    contar := can10;  
end.
```

¿Memoria  
utilizada por el  
módulo?

¿Tiempo  
empleado por el  
módulo?

## ■ Cálculo Teórico del tiempo de ejecución:

```

Type temperaturas = array [1..30] of real;

Function contar ( tem:temperaturas): integer;
Var i: 1..30; can10 : integer;
begin
    can10 := 0;                                {1}
    {recorrido total del vector}
    For i := 1 to 30 do                        {2}
        If (tem[i] = 10) then                  {3}
            can10 := can10 + 1;                {4}
        contar := can10;                       {5}
    end.

```

- Las líneas {1} y {5} cuentan una unidad cada una, entonces tenemos 2
- La línea {3} evalúa una condición, cuenta 1 unidad y la línea {4} cuenta 2 unidades. Por la tanto, cada vez que se ejecutan utilizan 3 unidades ->  $3 * 30$
- La línea {2} tiene una inicialización, testeo de  $i \leq 30$  e incremento de  $i$ , entonces 1 de la asignación, más 31 para todos los test y  $30 * 2$  para el incremento, entonces  $1 + 31 + 60 = 92$
- Total =  $2 + 90 + 92$  (como máximo!!!) ¿Por qué?



# Eficiencia de Programas – Tiempo de Ejecución



Supongamos que disponemos de un módulo que calcula la suma de los  $n$  primeros números naturales.

¿Memoria?  
¿Cantidad de  
operaciones?

```
Function sumar ( n : integer): integer;  
  Var total: integer; i: integer;  
  begin  
    total := 0; {1}  
    For i := 1 to n do {2}  
      total := total + i; {3}  
    sumar := total; {4}  
  end;
```

- {1} → 1 asignación = 1 unidad de tiempo
- {4} → 1 asignación = 1 unidad de tiempo
- {3} → (1 asignación + 1 suma) \*  $n$  =  $2 * n$
- {2} → una asignación ( $i:=1$ ) + testeos de  $i \leq n$  + incremento de  $i$  ( $i:= i+1$ )  
→  $1 + (n + 1) + 2*n = 3*n + 2$
- **$T(N) ==> \text{total} = 1 + 1 + 2*n + 3*n + 2 = 5*n + 4 \rightarrow O(n)$**

# Eficiencia de un Algoritmo

RETOMEMOS EL EJEMPLO DE OBTENER EL MÍNIMO M DE TRES NÚMEROS A, B Y C QUE SE LEEN.

HAY VARIOS CAMINOS PARA OBTENERLO:

## Solución 1

```
m := a;  
if b < m then m := b;  
if c < m then m := c;
```

## Solución 2

```
if a <= b then  
    if a <= c then m := a  
    else m := c  
else  
    if b <= c then m := b  
    else m := c
```

Calculemos la cantidad de operaciones que se realizan en ambas soluciones

**Solución 1:** en el peor de los casos, sumamos 5 operaciones

**Solución 2:** siempre 3 operaciones

# Eficiencia

## Programas Eficientes:

---

Se puede mejorar la eficiencia de los programas, observando algunas guías simples relacionadas con la eficiencia del código.

**Un punto importante a tener en cuenta es no repetir cálculos innecesarios.**

Es decir, podemos escribir:

$$t := x * x * x;$$
$$y := 1/(t-1) + 1/(t-2) + 1/(t-3) + 1/(t-4)$$

en lugar de

$$y := 1/(x*x*x-1) + 1/(x*x*x-2) + 1/(x*x*x-3) + 1/(x*x*x-4)$$

## REFLEXIONES FINALES

---

Los siguientes factores inciden en el tiempo de ejecución:

- Cantidad de datos de entrada
- Orden de los datos de entrada
- Cantidad de iteraciones

**No incide de forma directa en la eficiencia:**

- La modularización
- La cantidad de líneas del programa