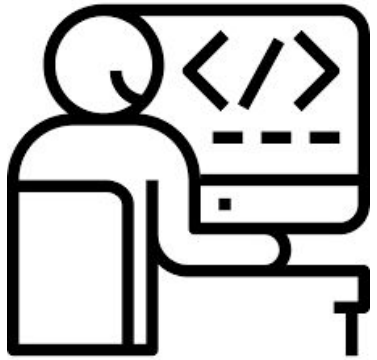


Algoritmos y Programación I

AyPI – **Temas de las clases pasadas**

- Tipo de datos lista
- Operaciones

AyPI – **Temas de la clase de hoy**

- Corrección
- Eficiencia

AyPI – **CORRECCIÓN y EFICIENCIA**

Cuando se desarrollan los algoritmos hay dos aspectos importantes que se deben tener en cuenta:

CORRECCIÓN

EFICIENCIA

CORRECCIÓN

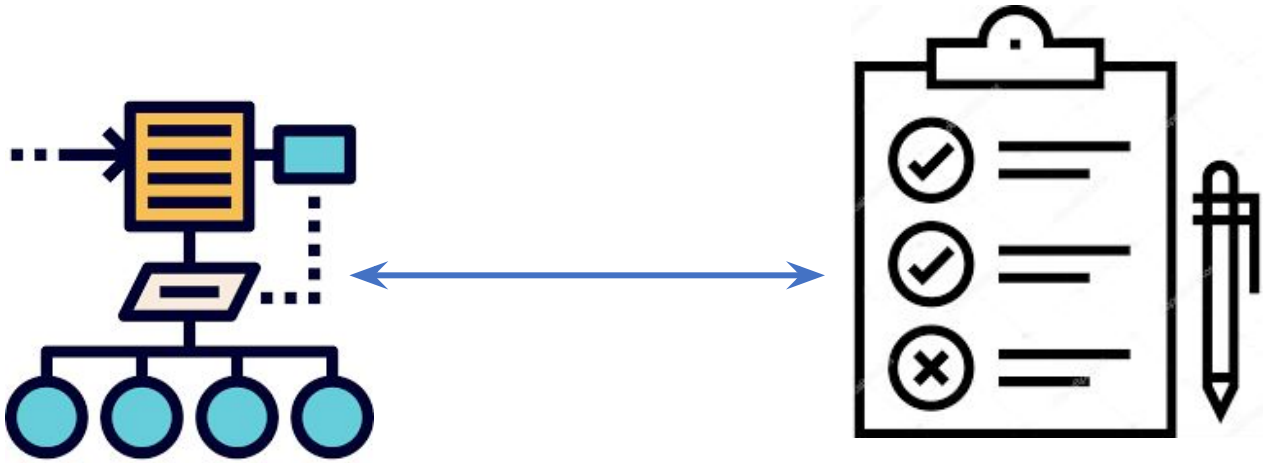
Un programa es correcto si se realiza de acuerdo a sus especificaciones.

Técnicas para corrección de programas

- Testing
- Debugging
- Walkthroughs
- Verificación

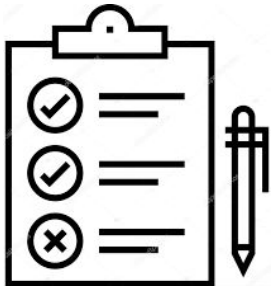
TESTING

El propósito del Testing es proveer evidencias convincentes que el programa hace el trabajo esperado.



Diseñar un plan de pruebas

AyPI – **CORRECCIÓN y EFICIENCIA**

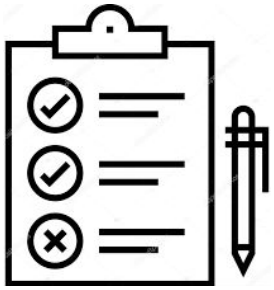


PLAN de PRUEBAS

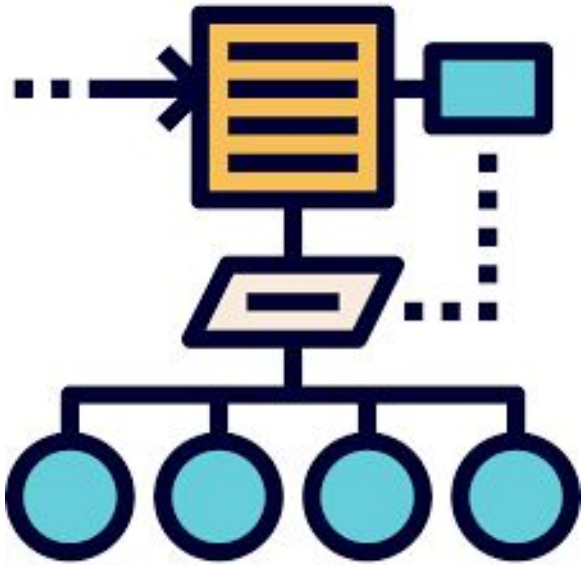
- Decidir cuáles aspectos del programa deben ser testeados y encontrar datos de prueba para cada uno de esos aspectos.
- Determinar el resultado que se espera que el programa produzca para cada caso de prueba.
- Poner atención en los casos límite.
- Diseñar casos de prueba sobre la base de lo que hace el programa y no de lo que se escribió del programa. Lo mejor es hacerlo antes de escribir el programa.

AyPI – **CORRECCIÓN y EFICIENCIA**

CORRECCIÓN



PLAN de PRUEBAS: una vez que el programa ha sido implementado y se tiene el plan de pruebas:



Se analizan los casos de prueba.

Si hay errores se corrigen.

Estos dos pasos se repiten hasta que no haya errores.

DEBUGGING

Es el proceso de descubrir y reparar la causa del error.

El diseño y aplicación de pruebas adicionales para ubicar y conocer la naturaleza del error.

- Es posible agregar sentencias adicionales en el programa para poder monitorear su comportamiento más cercanamente.
- También se puede utilizar los debuggers: herramientas que proveen algunos entornos de programación.

AyPI – **CORRECCIÓN y EFICIENCIA**

CORRECCIÓN

Los errores pueden provenir de dos fuentes:

El diseño del programa no es el adecuado.

El programa no está escrito correctamente.

Errores Sintácticos	Errores Lógicos	Errores externos
• Se detectan en la compilación	• Generalmente se detectan en la ejecución	• No se pueden solucionar por nosotros

WALKTHROUGH

Es recorrer el código fuente de un programa frente a una audiencia.

La lectura de un programa por otra persona provee un buen medio para detectar errores.

Esta persona no comparte preconceptos y está predispuesta a descubrir errores u omisiones.

A menudo, cuando no se puede detectar un error, el programador trata de probar que no existe, pero mientras lo hace, puede detectar el error, o bien puede que el otro lo encuentre.

AyPI – **CORRECCIÓN y EFICIENCIA**

CORRECCIÓN

VERIFICACIÓN

Verificar un programa significa controlar que se cumplan las pre y post condiciones del mismo.

AyPI – **CORRECCIÓN y EFICIENCIA**

CORRECCIÓN

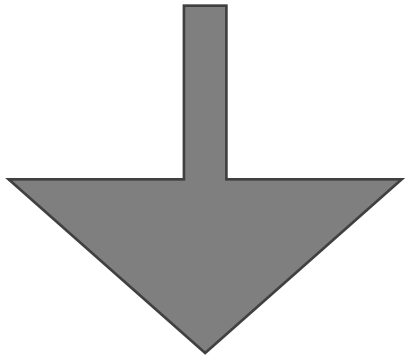
PARA DETERMINAR LA CORRECCIÓN DE UN PROGRAMA PUEDO UTILIZAR UNO, DOS, TRES O LAS CUATRO TÉCNICAS DE CORRECCIÓN

- Testing
- Debugging
- Walkthroughs
- Verificación

¿Cuál/es técnicas creen que se utilizan en la materia?

AyPI – **CORRECCIÓN y EFICIENCIA**

Una vez que se obtiene un algoritmo y se decide que es correcto, es importante determinar la **eficiencia** del mismo.

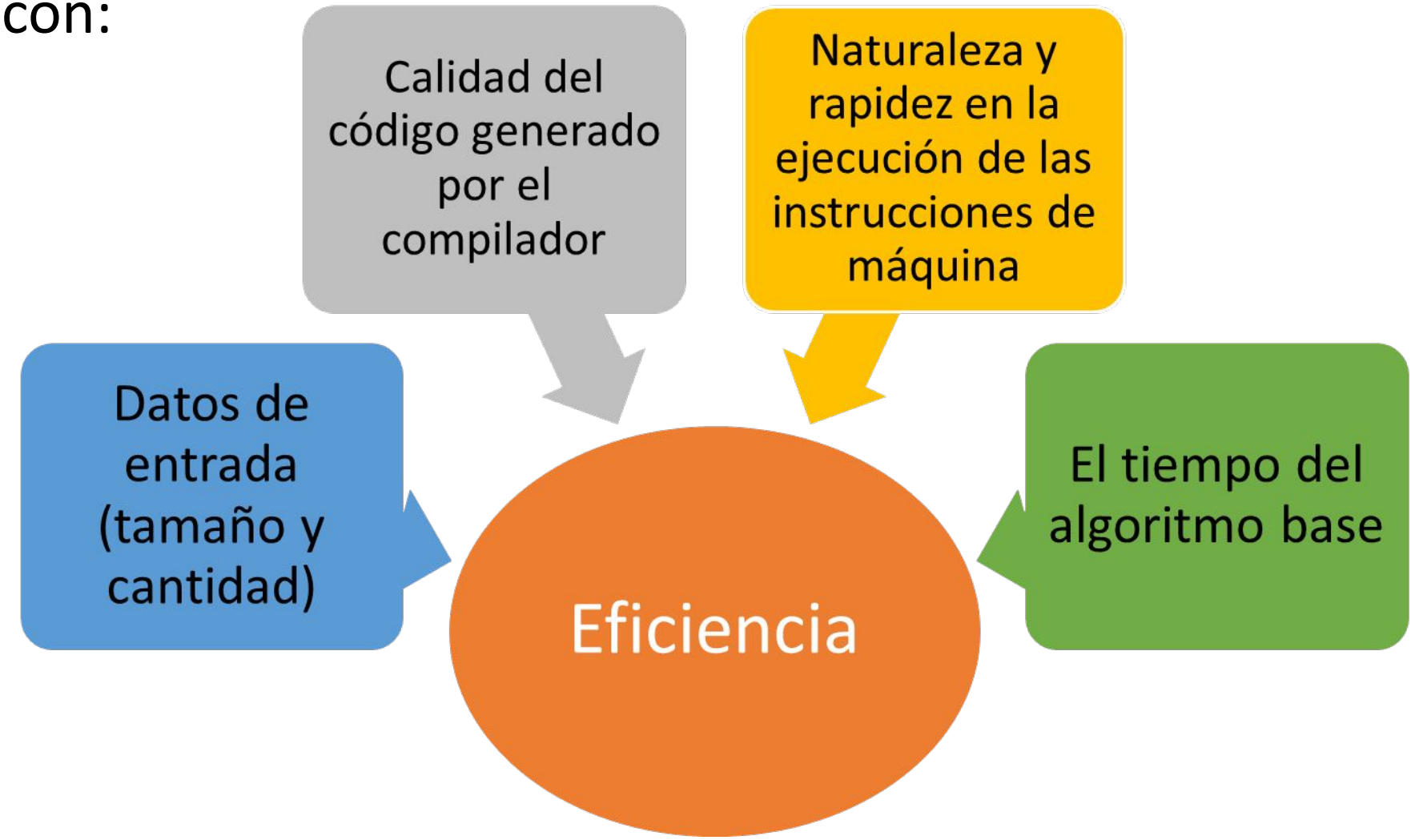


EFICIENCIA

El análisis de la eficiencia de un algoritmo estudia el **tiempo** que tarda un algoritmo en ejecutarse y la **memoria** que requiere.

EFICIENCIA

Se relaciona con:

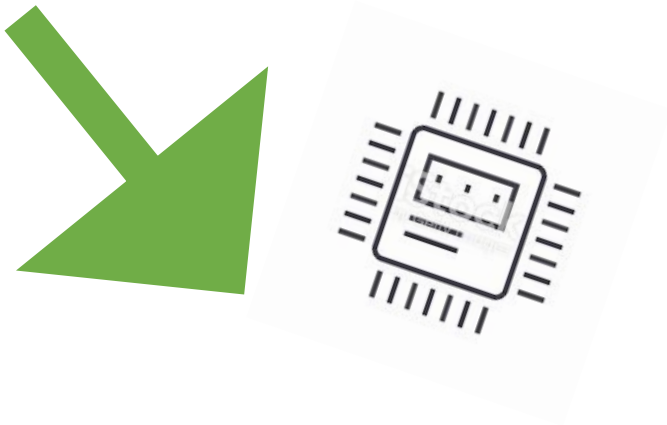


AyPI – **CORRECCIÓN y EFICIENCIA**

EFICIENCIA

EFICIENCIA

TIEMPO



MEMORIA

¿Cómo se miden?

TIEMPO DE EJECUCIÓN

El tiempo de un algoritmo puede definirse como una función de entrada :

- Existen algoritmos que el tiempo de ejecución no depende de las características de los datos de entrada sino de la cantidad de datos de entrada o su tamaño

- Existen otros algoritmos que 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

AyPI – **CORRECCIÓN y EFICIENCIA**

EFICIENCIA

TIEMPO DE EJECUCIÓN

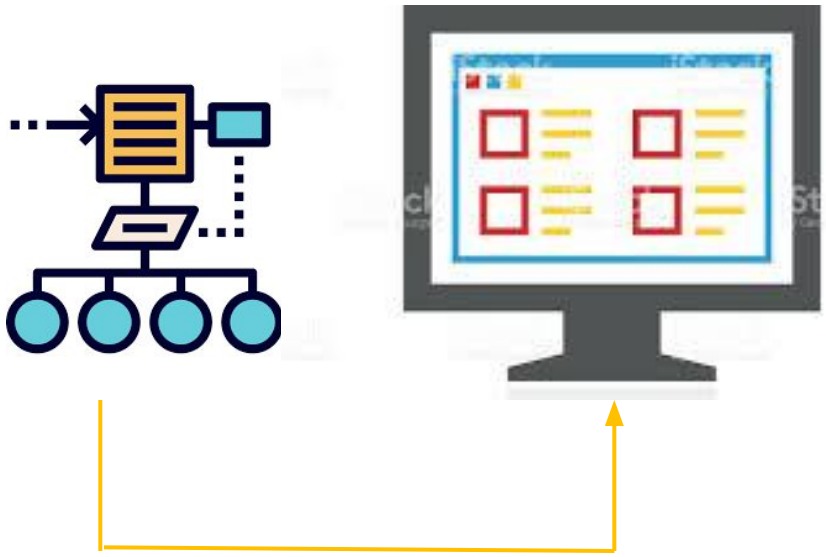
Puede calcularse haciendo dos tipos de análisis:

ANÁLISIS EMPÍRICO

ANÁLISIS TEÓRICO

ANÁLISIS EMPÍRICO

Para realizar un análisis empírico, es necesario realizar el programa y medir el tiempo consumido.



Fácil de realizar.

Obtiene valores exactos para una máquina y unos datos determinados

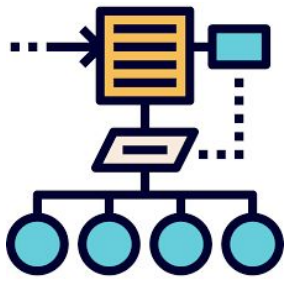
Completamente dependiente de la máquina donde se ejecuta.

Requiere implementar el algoritmo y ejecutarlo repetidas veces.

TIEMPO DE EJECUCIÓN

ANÁLISIS TEÓRICO

Implica encontrar una cota máxima para expresar el tiempo de nuestro algoritmo, sin necesidad de ejecutarlo.



O (log)

O (n)

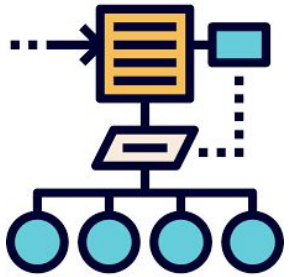
$O(n^2)$

Se obtiene el tiempo teórico del algoritmo y luego se obtiene el orden del mismo.

¿Cómo obtengo el tiempo teórico?

ANÁLISIS TEÓRICO

El cálculo del tiempo de ejecución de un algoritmo se realiza analizando el tiempo de ejecución de cada una de sus instrucciones: asignación, lectura, operaciones aritmético/lógicas, modificación de listas, etc.



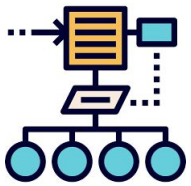
TIEMPO DE EJECUCIÓN

ANÁLISIS TEÓRICO

Sentencia	**Unidades de tiempo**
Asignacion	1
Comparación	1
Operación lógica	1
Operación aritmética	1
Lectura	1
Creación de lista (Create)	1
Agregar/eliminar elemento (add, addFirst, InsertCurrent, RemoveCurrent)	4
Obtener y modificar el valor actual (current, setCurrent)	1
Final de la lista (eol)	1
Ir al inicio de la lista (reset)	1
Avanzar al siguiente nodo de la lista (next)	1

ANÁLISIS TEÓRICO

T (alg) = T(1) + T(2) + T(3) + T(4)



```
Program uno;

var
 aux,temp,x: integer;

Begin
  aux:= 58;
  aux:= aux * 5;
  temp:= aux;
  read (x);
End.
```

T(1)= asignación = 1UT

T(2)= multiplicación + asignación = 2UT

T(3)= asignación = 1UT

T(4)= asignación = 1UT

Ejemplos

T (alg) = 5UT

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

```
Program uno;

var
 aux,temp,x: integer;

Begin
  aux:= 58;
  aux:= aux * 5;
  if (aux > 45) then
   begin
    temp:= aux - 5;
    x:= temp + aux + 2;
   end;
  x:= x * 10;
end;
```

T (alg) = T(1) + T(2) + T(3) + T(4)

T(1)= asignación = 1UT

T(2)= multiplicación + asignación = 2UT

T(3)= evaluar la condición + resta + asignación
suma + suma + asignación
= 1UT + 2UT + 3UT = 6UT

T(4)= multiplicación + asignación = 2UT

T (alg) = 11 UT

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

```
Program uno;
var
 aux,temp,x: integer;

Begin
  aux:= 58;
  aux:= aux * 5;
  if (aux > 45) then
   begin
    temp:= aux – 5;
    x:= temp + aux + 2;
   end
  else
   x:= x * 10;
end;
```

T (alg) = T(1) + T(2) + T(3)

T(1)= asignación = 1UT

T(2)= multiplicación +
asignación = 2UT

T(3)= evaluar la condición +
MAX (if,else)
if = resta + asignación
suma + suma + asignación
= 5UT
else = multiplicación +
asignación = 2 UT
1 + MAX (5,2) = 6 UT

T (alg) = 9 UT

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

```
Program uno;
var
 aux,temp,x,i: integer;

Begin
  aux:= 58;
  aux:= aux * 5;
  for i:= 1 to 10 do
    begin
    x:= x + 1;
    temp:= aux * 2;
    end;
end;
```

T (alg) = T(1) + T(2) + T(3)

T(1)= asignación = 1UT

T(2)= multiplicación + asignación = 2UT

T(3)= tiempo del for = 3(N)+2 + N(cuerpo)

N= 10

3(N) + 2 = 32 UT

N (cuerpo) =

10 (suma + asignación = 2UT) +
(multip + asignación = 2UT)

10 (4) = 40 UT

32 + 40 = 72 UT

T (alg) = 75 UT

A veces no se conoce N y quedará expresado en función de N

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

```
Program uno;
var
 aux,temp,x,i: integer;

Begin
   read (aux);
   for i:= 1 to aux do
     begin
     x:= x + 1;
     temp:= aux * 2;
     end;
end;
```

T (alg) = T(1) + T(2)

T(1)= lectura = 1UT

T(2)= tiempo del for = 3(N)+2 + N(cuerpo)

N= no sabemos el valor, ya que depende de aux, entonces se escribe de manera genérica

3(N) + 2 = se deja así

N (cuerpo) =

N (suma + asignación = 2UT) +

(multip + asignación = 2UT)

N*(4) UT

T (alg) = 1 + 3N + 2 + 4N = 7N + 3

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

```
Program uno;
var
 aux,temp,x,i: integer;

Begin
  aux:= 5;
  while (aux > 0) do
    begin
    x:= x + 1;
    aux:= aux - 1;
    end;
end;
```

T (alg) = T(1) + T(2)

T(1)= asignación = 1UT

T(2)= tiempo del while =
(N + 1) (evaluar cond) + N (cuerpo)

N= 5

6 (cond 1 UT) = 6 UT

5 ((suma + asignación = 2UT) +
(suma + asignación = 2UT))
5*(4)= 20UT

T (alg) = 6UT + 20UT = 26 UT

AyPI – **EFICIENCIA**

ORDEN DE LOS ALGORITMOS

```
Program uno;
var

Begin
 ....
 ....
end;
```

T (alg) = 36 UT

O (alg) = constante

```
Program dos;
var

Begin
 ....
 ....
end;
```

T (alg) = 2N + 4 UT

O (alg) = N

```
Program tres;
var

Begin
 ....
 ....
end;
```

T (alg) = $2N^2 + 4N$ UT

O (alg) = N^2

Mirando el orden puedo comparar algoritmos y elegir el más eficiente en cuanto a tiempo de ejecución

AyPI – **EJERCICIOS**

TIEMPO DE EJECUCIÓN

```
Program uno;
var
 aux,temp,i,x: integer;

Begin
  aux:= 58;
  for i:= 1 to 10 do
    begin
    x:= x + 1;
    temp:= temp * 2;
    end;

  for i:= 2 to 4 do
   temp:= temp + 1;
end;
```

```
Program dos;
var
 aux,temp,i,x: integer;

Begin
  read (aux);
  if (aux >5) and (aux >10) then
     for i:= 1 to 6 do
        x:= x + 1;
  else
     for i:= 2 to 4 do
        temp:= temp + 1;
end;
```

AyPI – **EFICIENCIA**

TIEMPO DE EJECUCIÓN

Dados dos programas que resuelven el mismo problema. ¿Si me piden que elija el más eficiente, con cuál me quedo?

Dados dos programas que resuelven el mismo problema. ¿La solución con menos instrucciones de código es la más eficiente en cuanto a tiempo de ejecución?

AyPI – **Ejercicio**

¿Cuál es el tiempo de ejecución del módulo aumentarPrecio_2?
(*El módulo realizado en clases anteriores, que recibe una lista de inmuebles y modifica el precio de los inmuebles dentro de la misma lista*).

```
Program uno;
uses GenericLinkedList;

Type
      inmueble = record
                    ...
                 end;
   ListaI = specialize LinkedList<inmuebles>;

Var
   listaInmuebles: ListaI;
```

AyPI – **Ejercicio**

```
procedure aumentarPrecio_2(l:listaI; porcentaje:real);
var  inmu: inmueble;
Begin
l.reset();
while(not l.eol()) do
   begin
   inmu:= l.current();
   inmu.precio:= inmu.precio * (1 + porcentaje / 100);
   l.setCurrent( inmu );
   l.next();
   end;
end;
```