Licenciatura en Ciencias de la Computación.
Facultad de Ingeniería - Universidad Nacional de Cuyo

Algoritmos y Estructuras de Datos I

```
1. tema = "análisis de complejidad"
```

Dr. Carlos A. Catania
Ing. Lucia Cortes
Lic. Javier Rosenstein
Dr. Claudio Careglio

Algoritmo

sustantivo

Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

Programa

sustantivo

Algoritmo codificado en un lenguaje y ejecutado sobre una arquitectura en particular.

Para resolver un problema determinado, pueden existir múltiples algoritmos y múltiples programas.

...pero no todos seran igual de eficientes...

Cómo determinamos la eficiencia de un algoritmo?

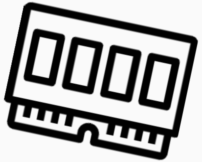
Primero hay que ponerse de acuerdo. Eficiencia respecto a que?

Algunas posibilidades

- La legibilidad del código?
- La cantidad de líneas de código?
- El tiempo de ejecución?
- La memoria que consume?

Se miden fundamentalmente dos características:

Cantidad de memoria (**Complejidad Espacial**)



El tiempo de ejecución (**Complejidad Temporal**)

Se miden fundamentalmente dos características:

El tiempo de ejecución (**Complejidad Temporal**)

Un ejemplo:
calcular la suma de n números enteros

Dado un valor ***n*** sumar todos los números enteros de 1 hasta n

Ejemplo:

```
n=10

1+2+3+4+5+6+7+8+9+10 = 55
```

Un ejemplo:
calcular la suma de n números enteros

```
def sumadeN(n):
  theSum = 0
   for i in range(1,n+1):
      theSum = theSum + i
  return theSum
print(sumOfN(10))
```

```
def sumadeN2(n):
  return (n*(n+1))/2
print(sumOfN2(10))
```

Cómo calcular la complejidad Temporal de un algoritmo?

Un enfoque Experimental

Que pasaria si...

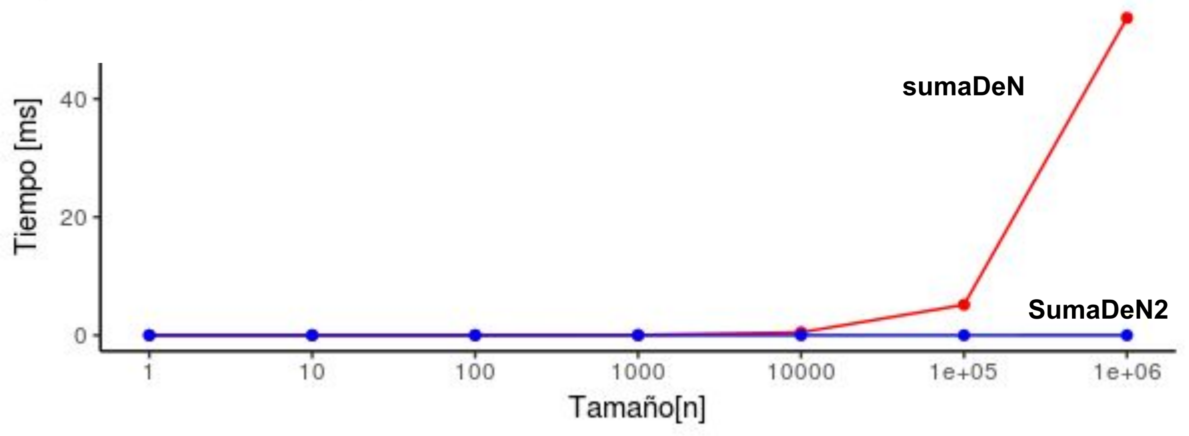
1. Probamos con distintos valores de ***n***
   Ej: n={10,100,1000,10000,100000...}

2. Luego calculamos el tiempo de ejecución ***T*** ?

Tabla 1: Tiempo de ejecución para distintos valores de n-10^z

	z	sumaDeN	sumaDeN2
1	0	1.9073486328125e-06	1.19209289550781e-06
2	1	2.38418579101562e-06	4.76837158203125e-07
3	2	6.4373016357421 9e-06	4.76837158203125e-07
4	3	4.79221343994141e-05	2.38418579101562e-07
5	4	0.000494718551635742	2.38418579101562e-07
6	5	0.0051567554473877	4.76837158203125e-07
7	6	0.0536763668060303	2.14576721191406e-06
8	7	0.552401781082153	2.62260437011719e-06

Cual de los 2 algoritmos tiene una menor complejidad Computacional?



AHORA BIEN...

1. Este resultado será válido si ejecutamos los mismos programas en otra **computadora**?

2. Que pasaria si reescribimos los programas en otro **lenguaje**?

La realidad es que...

- Si nos apartamos de detalles como **arquitectura** o **lenguaje**, las curvas van a presentar un comportamiento similar

Mas formalmente...

Principio de Invarianza:

Dado un algoritmo y dos implementaciones suyas $I1$ e $I2$, que tardan $T1(n)$ y $T2(n)$ existe una constante real $c > 0$ y un número natural **n0** tales que para todo $n \geq n0$ se verifica que $T1(n) \leq cT2(n)$.

Enfoque teórico

A pensar...en abstracciones

- Imaginemos una computadora ideal, la cual ejecuta una instrucción en tiempo **constante** predeterminado.

Entonces...

- El tiempo de ejecución puede expresarse como una función $T(n)$, donde T va a depender de los datos de entrada n.

Un enfoque teórico: Estimación del número de operaciones

- Estimar $T(n)$ en función del número de operaciones elementales (**OE**)
- **Se consideran como 1 OE:**
    - Operaciones aritméticas básicas
    - Asignaciones a variables de tipo predefinido por el compilador,
    - Saltos (llamadas a funciones),
    - Comparaciones lógicas
    - Acceso a estructuras indexadas básicas(vectores y matrices).
- **Tiempo de una OE es de orden 1**

Otras consideraciones sobre las OE

- El tiempo de ejecución de la sentencia :

```
if c:
    s1
else:
    s2
```

- es $T = T(c) + max\{T(s1), T(s2)\}$.

Otras consideraciones sobre las OE

- El tiempo de ejecución de una llamada a

```
F(P1,., Pn)
```

- Tiempo es 1 (por la llamada), más el tiempo de evaluación de los parámetros `P1, P2,..., Pn`, más el tiempo que tarda en ejecutarse F, esto es, $T = 1\ (salto) + T(P1) + T(P2) + \ldots + T(Pn) + T(F)$.

Otras consideraciones sobre las OE

- Donde $\mathrm{T(F)}$ será igual a:
  - $T(Op)$ Tiempo de las operaciones realizadas dentro de la función
  - $T(R)$ Tiempo de evaluar el retorno (salto) + su valor (2 OE)

Otras consideraciones sobre las OE

- El tiempo de ejecución de un bucle de sentencias

```
while c:
    s
```

- es $T = T(c) + (n^o\ iteraciones) * (T(s) + T(c))$.

Obsérvese que tanto $T(c)$ como $T(s)$ pueden variar en cada iteración, y por tanto habrá que tenerlo en cuenta para su cálculo

Otras consideraciones sobre las OE

- El tiempo de ejecución de un bucle de sentencias

```
for c in range(1,10):
    s
```

Se lo expresa como una sentencia **while** y se calcula de la misma manera

```
c=0
while c<10:
    s
    c=c+1
```

Sumar N números enteros: Algoritmo 1

```
1: def sumadeN(n):
2:    theSum = 0
3:     for i in range(1,n+1):
4:        theSum = theSum + i
5:    return theSum
```

1: 2 OE

2: 1 OE

3: 3+n OE

4: 2OE+4OE

5: 2 OE

$$T(n)=2+1+3+\sum^{n} 6+2$$

$$T(n)=2+1+3+6n+2$$

$$T(n)=8+6n$$

Sumar N números enteros: Algoritmo 2

```
1: def sumadeN(n):
2:    theSum = 0
3:     for i in range(1,n+1):
4:        theSum = theSum + i
5:    return theSum
```

1: 2 OE
2: 1 OE
3: 3+n OE
4: 2OE+4OE
5: 2 OE

$$T(n)=2+1+3+\sum^{n}6+2$$

$$T(n)=2+1+3+6n+2$$

$$T(n)=8+6n = \boldsymbol{c_0+c_1n}$$

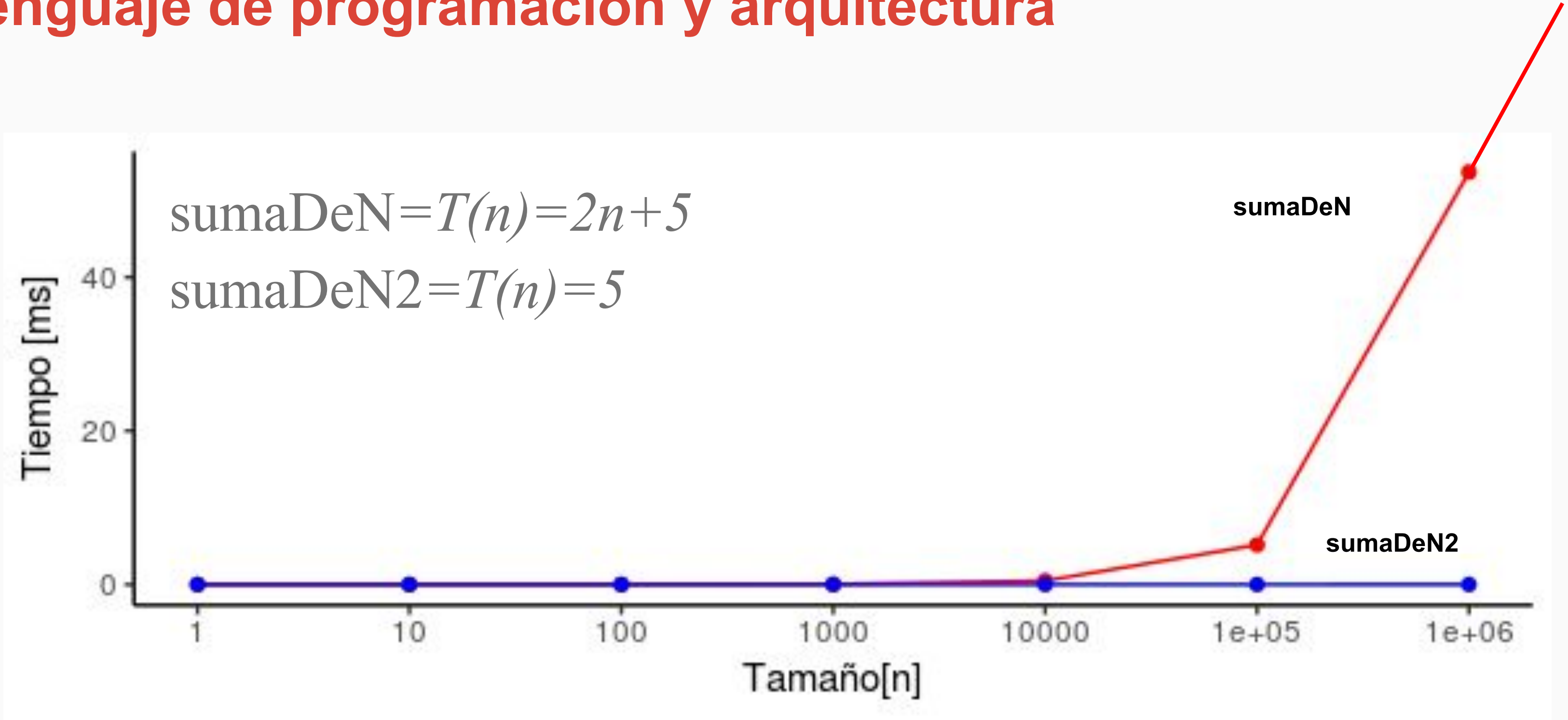
```
1: def sumDeN2(n):
2:    return (n*(n+1))/2
```

1: 2 OE
2: 5 OE

$$T(n)=2+5$$

$$T(n)=7=\boldsymbol{c_0}$$

Verificamos el resultado experimental válido para todo lenguaje de programación y arquitectura



Titular: Dr. C.A. Catania **<harpomaxx@gmail.com>** **@harpolabs**
Adjunto: Ing. L. Cortés **<luciacortes5519@gmail.com>**
JTP: Lic. J. Rosenstein **<rosensteinjavier@gmail.com>**

HAPPY HACKING!

Sumar N números enteros: Algoritmo 1

```
1: def sumadeN(n):
2:   theSum = 0
3:    for i in range(1,n+1):
4:        theSum = theSum + i
5:   return theSum
```

1: 1 OE
2: 1 OE
3: n OE
4: 2 OE
5: 1 OE

$$T(n)=1+1+\sum_{n} 2+1$$

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

$$T(n)=3+2n$$

Sumar N números enteros: Algoritmo 2

```
1: def sumaDeN(n):
2:   theSum = 0
3:    for i in range(1,n+1):
4:     theSum = theSum + i
5:   return theSum
```

1: 1 OE
2: 1 OE
3: n OE
4: 2 OE
5: 1 OE

$$T(n)=1+1+\sum_{n}2+1$$

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

$$T(n)=3+2n$$

```
1: def sumDeN2(n):
2:   return (n*(n+1))/2
```

1: 1 OE
2: 2 OE

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

$$T(n)=3$$