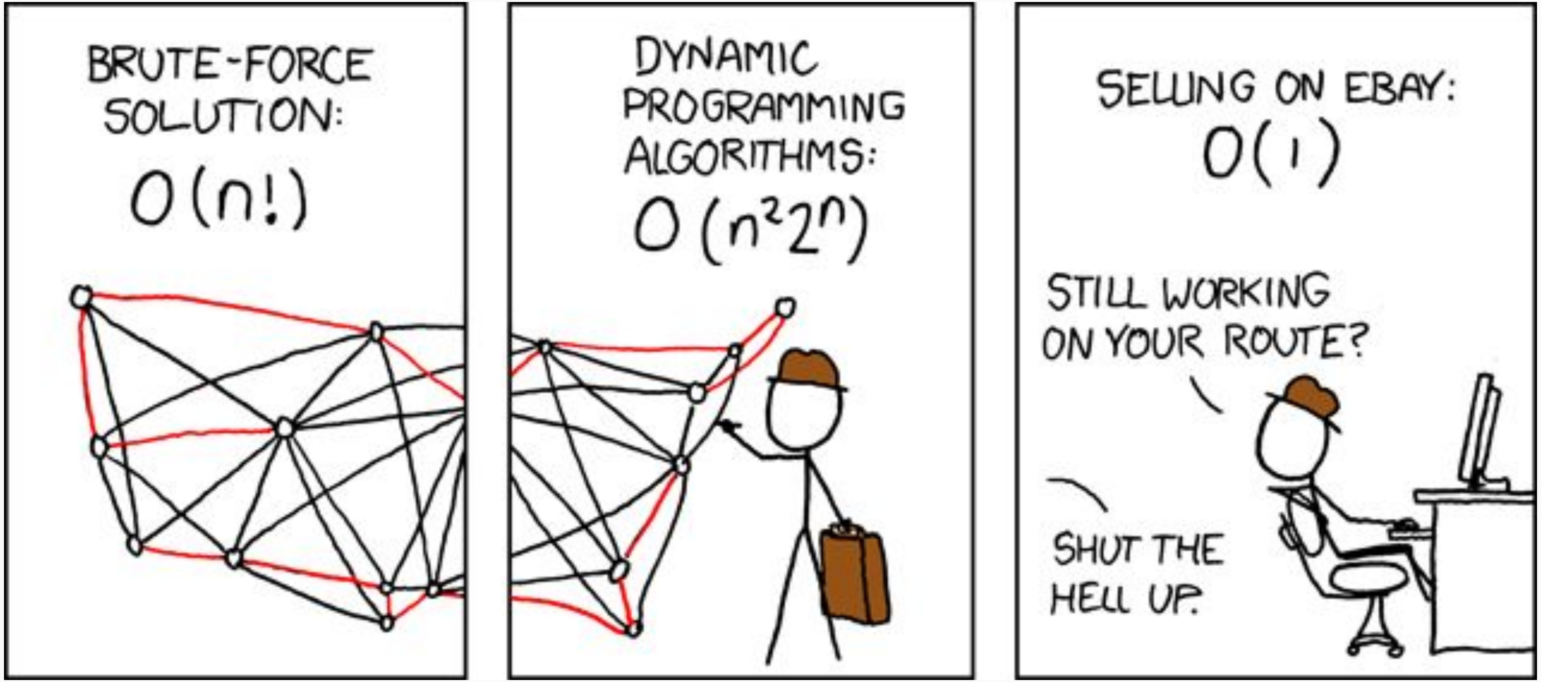
Licenciatura en Ciencias de la Computación.
Facultad de Ingeniería - Universidad Nacional de Cuyo

Algoritmos y Estructuras de Datos I

1. tema = ”análisis de complejidad”

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



El objetivo último de esta clase es que entiendan el chiste de xkcd

Enfoque teórico

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 $T1(n) \leq cT2(n)$.

La realidad es que...

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

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 **únicamente** 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 $T(F)$ 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

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

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

$$T(n)=8+6n = c_0+c_1n$$

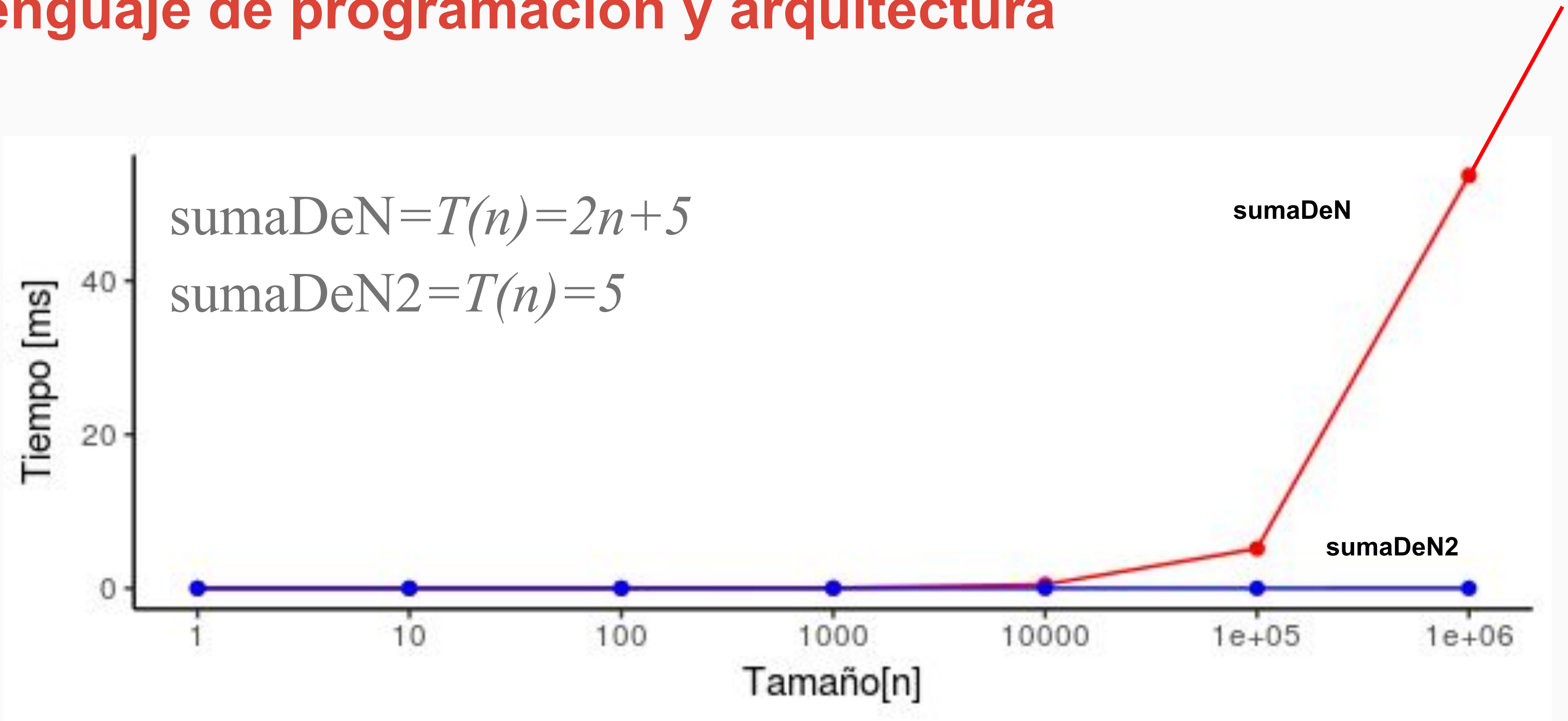
```
1: def sumDeN2(n):
2:    return (n*(n+1))/2
```

1: 2 OE
2: 5 OE

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

$$T(n)=7=c_0$$

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



Ejercicios...
de practica

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

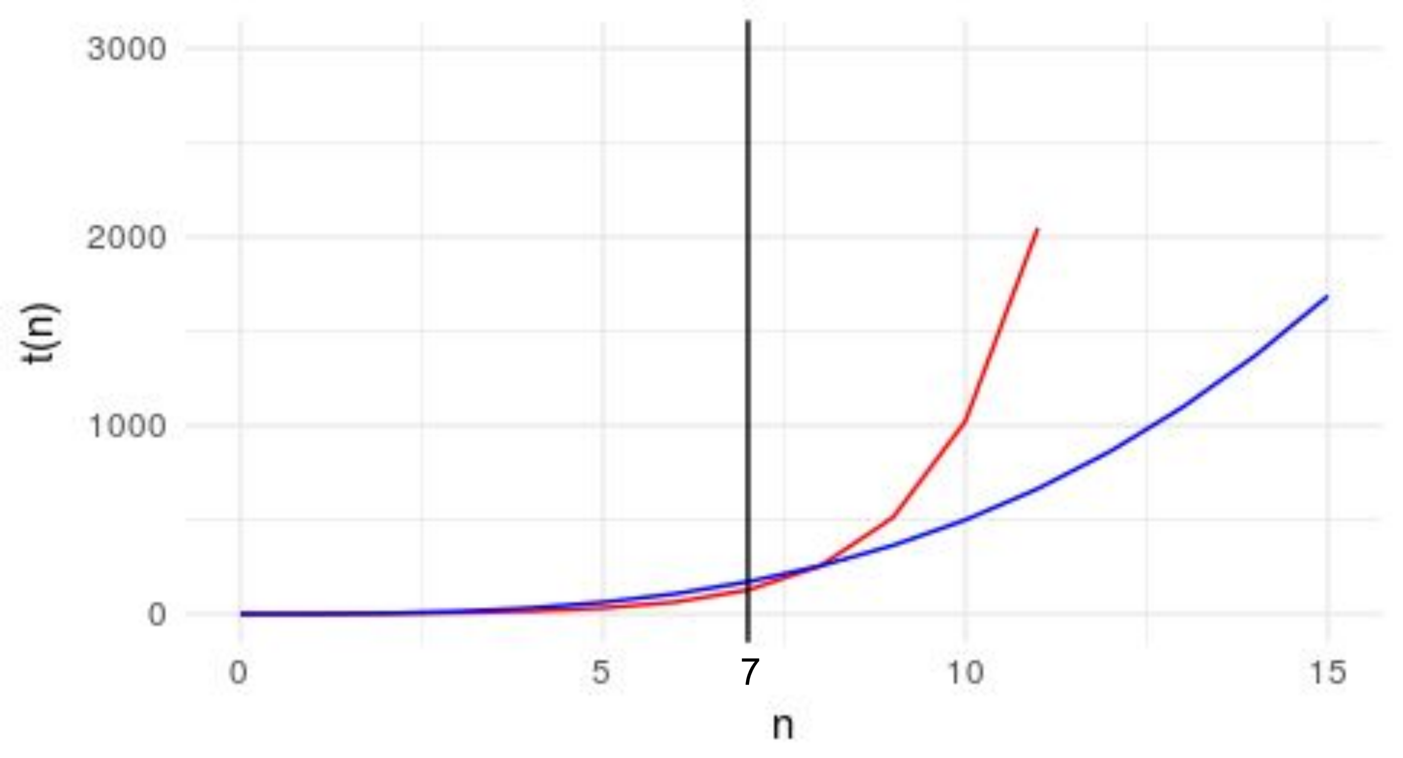
Algoritmos y Estructuras de Datos I

1. tema = "análisis de complejidad"

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

Cómo comparamos la complejidad temporal de 2 algoritmos?

Tasa de crecimiento



Análisis Asintótico: Notación $O(f)$

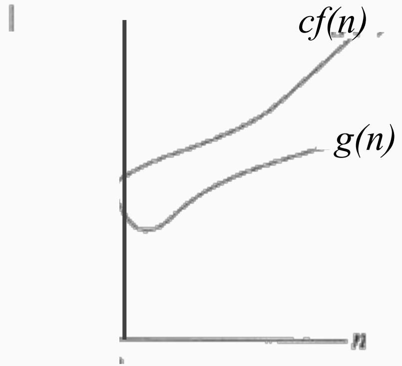
Tambien llamada Big Oh

La denominación $O(f)$ hace referencia a la clase de equivalencia compuesta por las funciones g que **van a crecer a lo sumo tan deprisa como** f.

Notación Big-oh: Cota superior

- Dada una función f, nos interesan aquellas funciones g que a lo **sumo crecen tan deprisa como f.**

- Para el conjunto de tales funciones g, la función f constituye una **cota superior** .



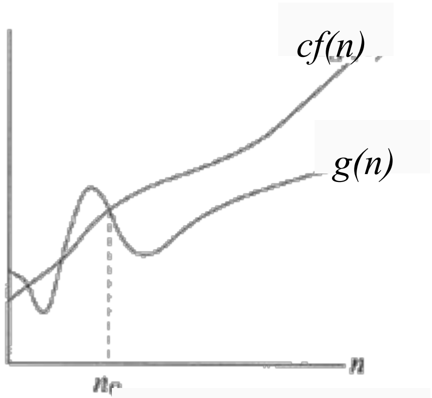
$g(n)=O(f(n))$

Notación Big-Oh Definición:

Sea f: $\mathbf{N} \rightarrow [0,\infty)$. Se define el conjunto de funciones de orden O (Omicron) de f como:

$$O(f) = \{g: \mathbf{N} \rightarrow [0,\infty) \mid \exists c \in \mathbf{R},\ c>0,\ \exists n_0 \in \mathbf{N} \bullet g(n) \leq cf(n)\ \ \forall n \geq n_0\}.$$

Diremos que una función t: $\mathbf{N} \rightarrow [0,\infty)$ es de orden O de f si $t \in O(f)$.



$g(n)=O(f(n))$

Normalmente estaremos interesados en la menor función f tal que t pertenezca a $O(f)$.

3 casos

En el análisis asintótico aplicado a la complejidad de los algoritmos es posible considerar 3 casos:

1. El comportamiento del algoritmo en el **peor caso**
2. El comportamiento del algoritmo en el **mejor caso**
3. El comportamiento del algoritmo en el **caso promedio**

Mejor y peor caso.

Buscar el elemento $\boldsymbol{E}$ en un arreglo de N elementos.

0	1	2	3	4	5	6	7	..	N
$\boldsymbol{E}$									
									$\boldsymbol{E}$

Normalmente la notación Big-Oh hace referencia al peor caso.

Al considerar la complejidad temporal de un algoritmo en el **peor caso**, la notación Big Oh nos permite descartar los términos de menor grado.

Notación Big-Oh: Ejemplos

$T(n)=231 \;\to O(1)$

$T(n)=4n+2 \;\to O(n)$

$T(n)=32n^2+3n+2 \;\to O(n^2)$

$T(n)=5n^3+5n^2+12n+12 \;\to O(n^3)$

Algunas de las clases más relevantes en el análisis de algoritmos.

Función Constante

$$T(n)=f(n)=c$$

- Sin importar el valor de n, el resultado de T siempre será igual a una constante c.
- Normalmente asociada a operaciones básicas como asignaciones, comparaciones, sumas, etc.

Funcion Lineal

$$T(n)=f(n)=n$$

- Para cualquier el valor de n, el resultado de T será igual n.

- Normalmente asociada a operaciones básicas como iteraciones sobre n elementos.

Función Cuadrática

$$T(n)=f(n)=n^2$$

- Para cualquier valor de n, el resultado de T siempre será igual al producto de n por si misma.
- Normalmente asociada a iteraciones anidadas.

Funcion Cubica

$$T(n)=f(n)=n^3$$

- Sin importar el valor de n, el resultado de T siempre será igual al producto de n por si misma 3 veces.
- Se observa en iteraciones anidadas.

Function Logaritmica

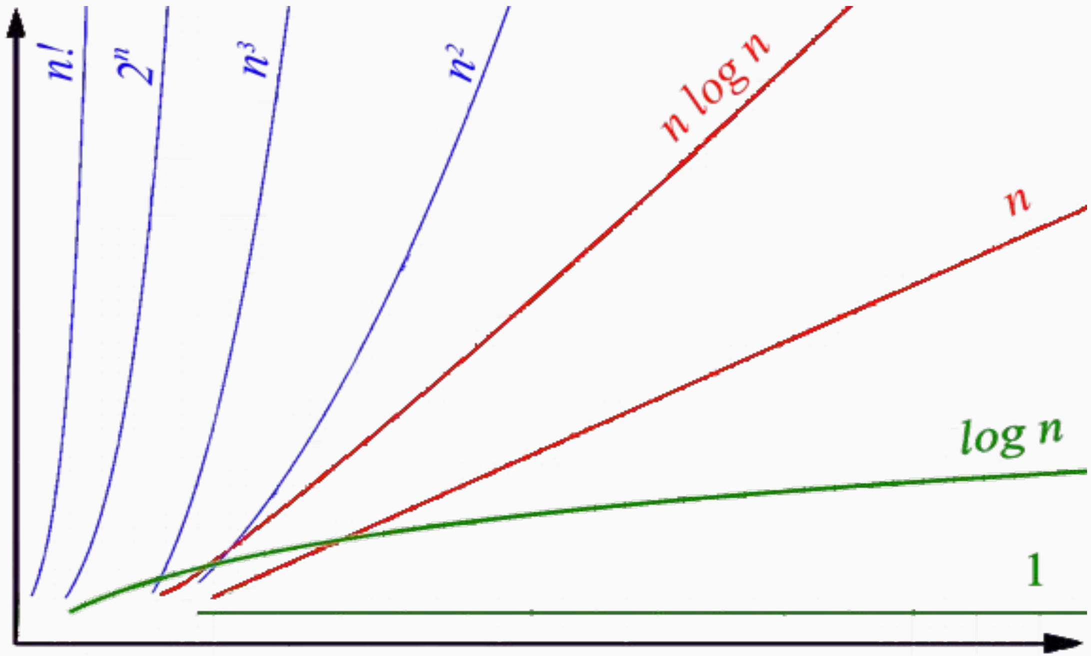
$$T(n)=f(n)=log(n)$$

- Para cualquier valor de n, el resultado de T será igual *a*l exponente al cual hay que elevar la base (normalmente 2)
- Normalmente asociada a operaciones recursivas.

Función Exponencial

$$T(n)=f(n)=2^n$$

- Dado un valor de n, el resultado de T será igual al producto de una constante c n veces.
- Se observa en iteraciones donde en cada paso se duplica el número de operaciones.



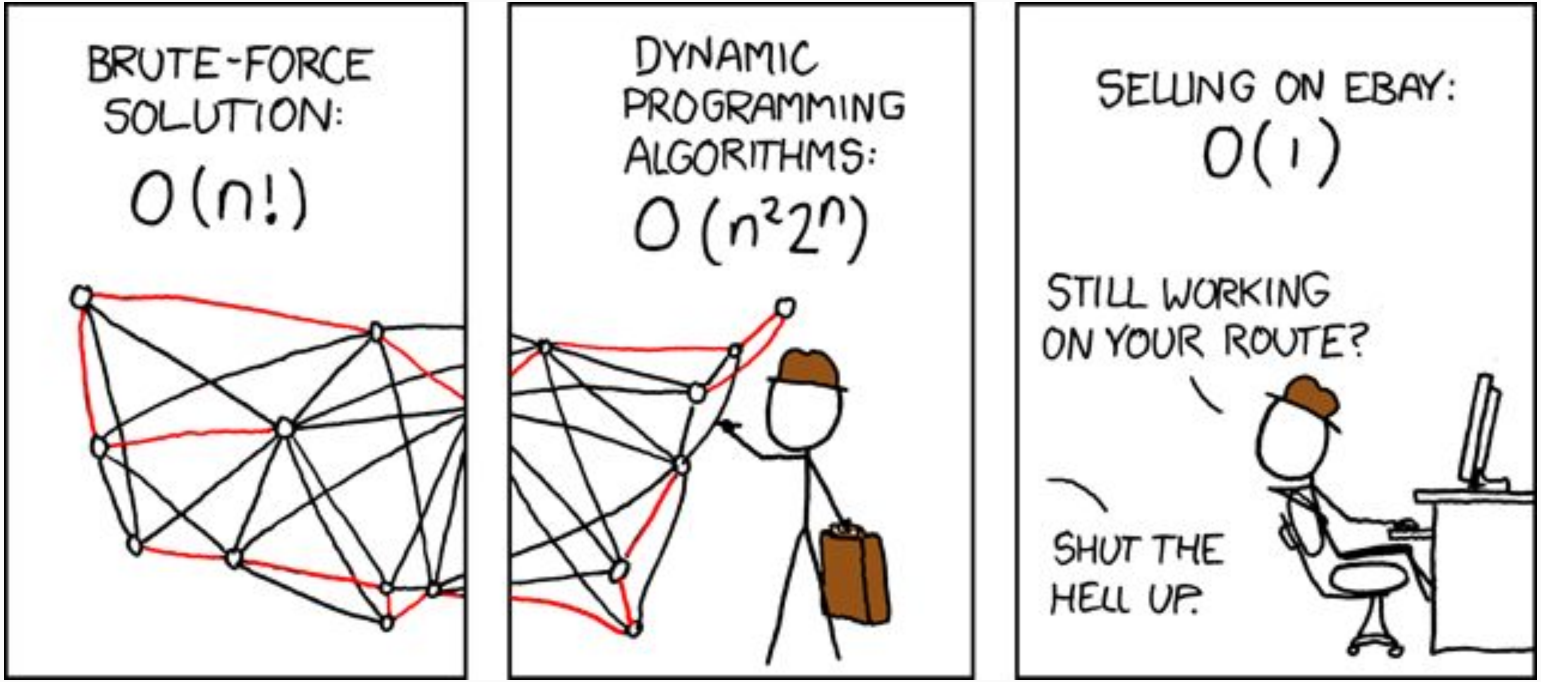
Comparación entre las distintas funciones consideradas

Conclusiones

La utilización del enfoque teórico nos va a permitir calcular la complejidad temporal de manera analitica.

La notación Big-Oh junto a las distintas clases de funciones f, nos van a permitir simplificar la estimación de la complejidad temporal.

Estas herramientas las vamos a usar para diferenciar y comparar cada una de las estructuras y algoritmos que vamos a ver durante el cursado de la materia.



Que dicen... Objetivo cumplido?

Algunos ejemplos:

```
if a>b:
  c=a+b
else:
  for d in range(1,10):
    c=a+b*d
```

Algunos ejemplos:

```
a=1
while a<n:
    a=a+1
```

Algunos ejemplos:

```
for i in range(1,n):
    j=0
     while j<i:
       a=a*(1+j)
       j=j+1
```

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!