# Algoritmos y Estructuras de Datos I


Universidad Católica del Uruguay

Orden del tiempo de ejecución

# TIEMPO DE EJECUCIÓN DE UN PROGRAMA

Universidad Católica del Uruguay

- Al resolver un problema, debemos elegir el algoritmo a utilizar, siguiendo los siguientes objetivos:
  - algoritmo fácil de entender, codificar y depurar.
  - uso eficiente de los recursos y ejecución veloz.
- En la mayoría de los casos, estos objetivos son contrapuestos.
- Depende, en gran forma, de la cantidad de veces que el programa se va a ejecutar.

# Medición del tiempo de ejecución de un programa.

- El tiempo de ejecución de un programa depende de factores como:
  - Los datos de entrada al programa.
  - La calidad del código generado por el compilador.
  - La naturaleza y rapidez de las instrucciones usadas en el programa.
  - La complejidad de tiempo del algoritmo base del programa.
- Muchas veces el tiempo depende del tamaño de la entrada más que de la entrada en sí misma.
- La longitud de la entrada es una medida apropiada de tamaño.
- Se denomina ***T(n)*** el tiempo de ejecución de un programa con entrada de tamaño ***n***.

# Medición del tiempo de ejecución de un programa

- Cuando el tiempo de ejecución es función de una entrada específica y no sólo del tamaño, entonces se define ***T(n)*** como el tiempo de ejecución del *peor caso.*
- También se podría calcular el $T_{prom}(n)$, o sea, el tiempo promedio de ejecución, pero suele ser bastante más difícil.
- El hecho de que el tiempo de ejecución dependa del compilador y de la máquina impide que se pueda expresar en unidades de tiempo normales.
- Se dirá entonces que, por ejemplo, “el algoritmo es proporcional a $n^2$”.

# Notación asintótica - Orden del tiempo de ejecución.

- Se dice que el tiempo de ejecución de un programa es de ***orden n,*** ( ***T(n)*** es ***O(n)*** ), si existen dos constantes enteras positivas ***c*** y $n_0$ tales que,
  - para ***todo n >= $n_0$ , T(n) <= c * n***
- Se dice que ***T(n)*** es ***O(f(n))*** si existen dos constantes enteras positivas ***c*** y $n_0$ tales que,
  - para ***todo n >= $n_0$ , T(n) <= c * f(n)***
  - en este caso, se dice que el programa tiene *velocidad de crecimiento* ***f(n)***.
- Si ***T(n)*** es ***O(f(n)) , f(n)*** es una cota superior para la velocidad de crecimiento de ***T(n)***.

# Notación asintótica - Orden del tiempo de ejecución.

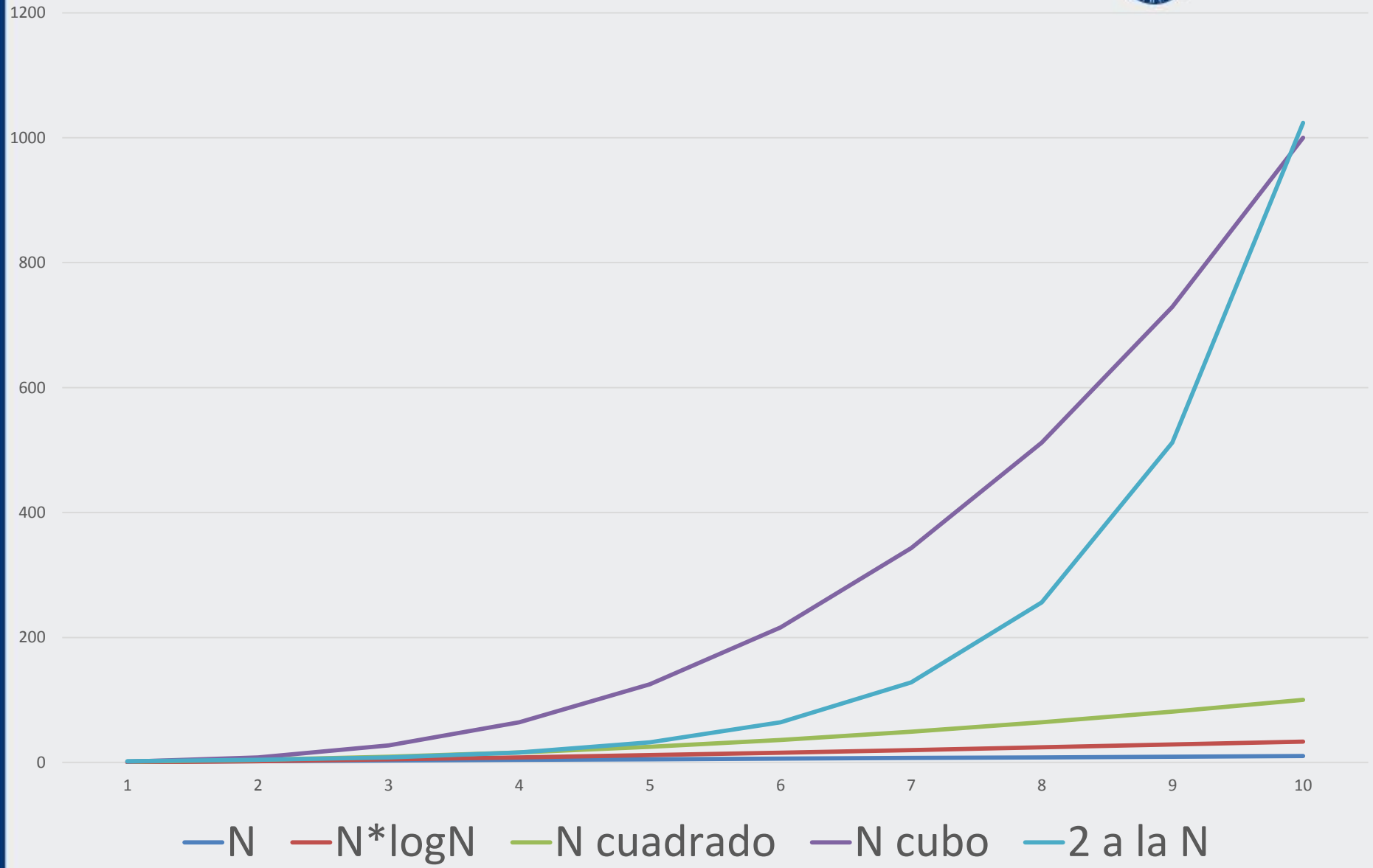
- La cota inferior se especifica notando ***T(n)*** es ***Ω(g(n)).*** Esto significa que existe una constante ***c*** tal que

  ***T(n) >= c*g(n)*** para un número infinito de valores de *n*.

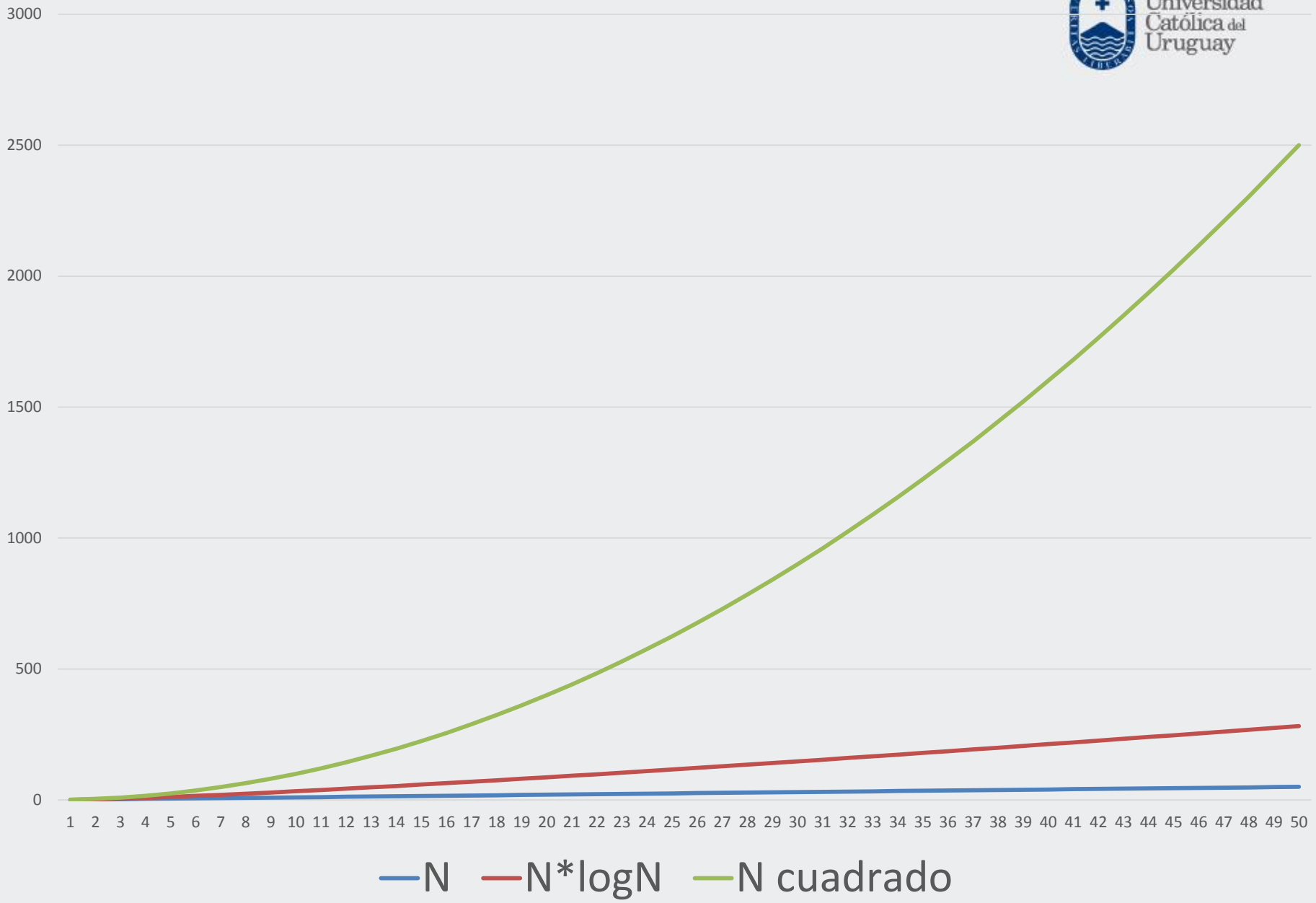
- Suponemos que podemos evaluar programas comparando sus funciones de tiempo de ejecución, sin considerar las constantes de proporcionalidad. De acuerdo a esto, un programa con tiempo de ejecución $O(n^2)$ es mejor que uno con tiempo de ejecución $O(n^3)$.
- Además de los factores del compilador y la máquina, existe un factor debido al programa mismo.

- Ej: 100 * $n^2$ milisegundos vs. 5 * $n^3$ milisegundos.

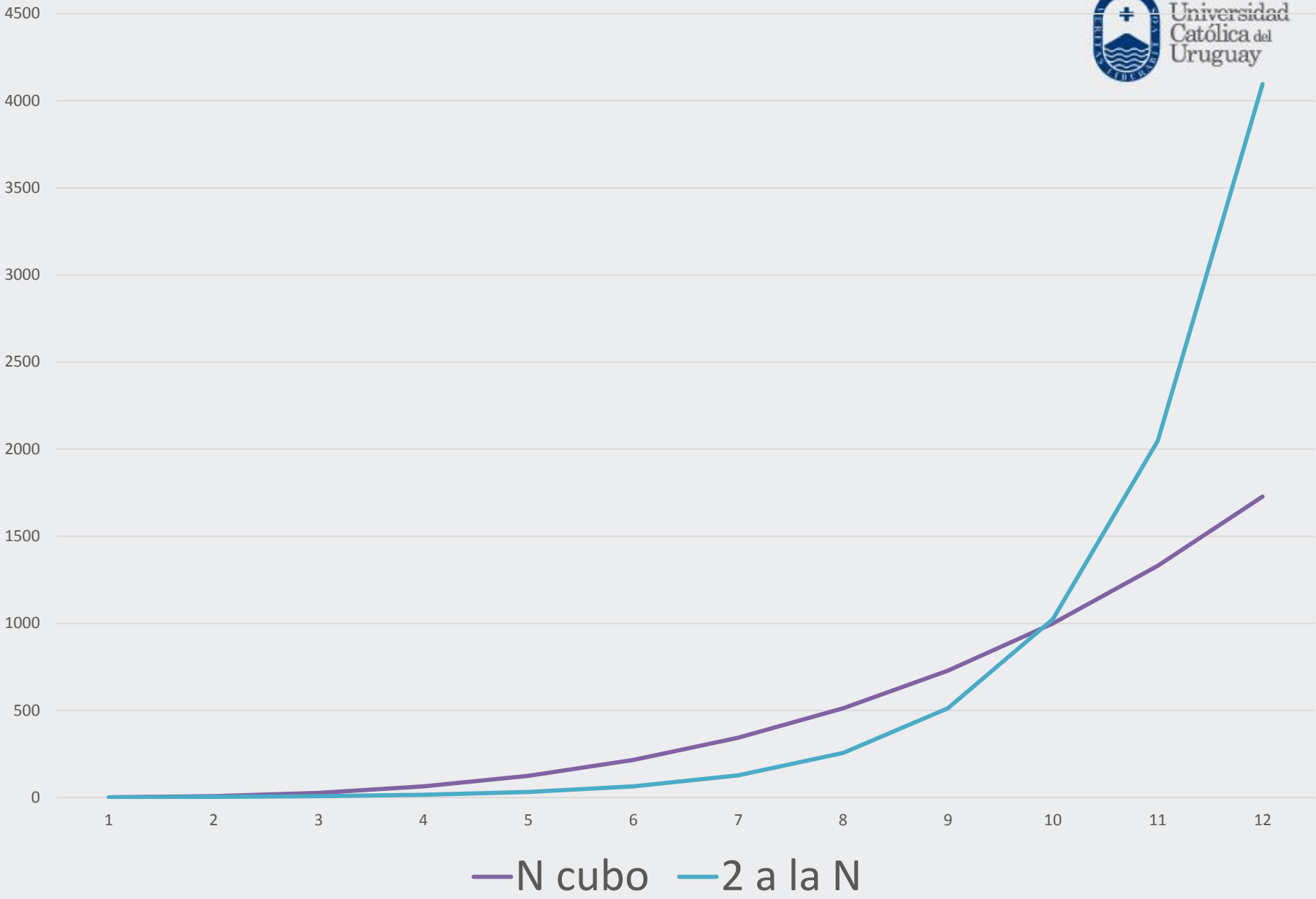
# Comparación de órdenes (todos)



# Comparación de órdenes



# Comparación de órdenes



# Efecto de multiplicar por diez la velocidad del procesador

| T(n) | Tamaño para $t=10^3$ segs | Tamaño para $t=10^4$ segs | Incremento |
|---|---|---|---|
| 50n | | | |
| $3n^2$ | | | |
| n3/3 | | | |
| 2n | | | |

# Efecto de multiplicar por diez la velocidad del procesador

Universidad Católica del Uruguay

| T(n) | Tamaño para $t=10^3$ segs | Tamaño para $t=10^4$ segs | Incremento |
|---|---|---|---|
| 50n | 20 | 200 | 10 |
| $3n^2$ | 18 | 58 | 3.2 |
| n3/3 | 7 | 14 | 2 |
| $2^n$ | 10 | 13 | 1.3 |

# Tiempo de ejecución

- Si un programa se va a ejecutar poco, el costo de su desarrollo es el dominante.
- Si el programa se ejecuta sólo con entradas pequeñas, la velocidad de crecimiento puede ser menos importante que el factor constante.
- Un algoritmo eficiente pero complicado aún puede no ser adecuado, por razones de mantenimiento.
- Algunos algoritmos eficientes requieren demasiado espacio, debiéndose utilizar almacenamiento secundario, con lo cual pierden su eficiencia.
- En los algoritmos numéricos, la precisión y la estabilidad son tan importantes como la eficiencia.

# Cálculo del tiempo de ejecución de un programa

- Reglas de la suma y producto en notación asintótica.
- El tiempo de ejecución de cada proposición de asignación puede tomarse como O(1).
- El tiempo de ejecución de una secuencia de proposiciones se determina por la regla de la suma.
- El tiempo de ejecución de una proposición condicional if es el costo de las proposiciones que se ejecutan condicionalmente, más el tiempo para evaluar la condición (se toma O(1)).
- El tiempo para ejecutar un ciclo es la suma sobre todas las iteraciones del ciclo, del tiempo de ejecución del cuerpo y del empleado para evaluar la condición de terminación. Normalmente este tiempo es el producto del número de iteraciones del ciclo por el mayor tiempo posible para una ejecución del cuerpo.

# Cuál es el orden de las siguientes expresiones:

- 22
- n(n-1) / 3
- máx(n3,10n2)
- polinomio de grado K donde el coeficiente del término de mayor grado es positivo.

# El siguiente algoritmo encuentra el lugar LUG y el valor MAX del elemento mayor del array DATOS, no vacío, con N valores numéricos.

Universidad Católica del Uruguay

**1. Inicializar K := 1, LUG := 1 y MAX := DATOS[1]**
**2. Repetir pasos 3 y 4 mientras K <= N:**
**3. Si MAX < DATOS[K] entonces**
**LUG .= K y MAX := DATOS[K]**
**Fin Si**
**4. K := K + 1**

**Fin ciclo paso 2**

**5. Escribir LUG, MAX**
**6. Salir**

Hallar el orden de ejecución para el mejor caso, peor caso y caso medio.

# Considere el siguiente algoritmo, en el cual:

- **R, x, m** son variables enteras,
- **tabla** es un vector ordenado de enteros,
- **N** es el tamaño máximo de la entrada

```
L ← 0
R ← N
Mientras L < R hacer
      m ← (L + R) div 2
      Si tabla[m] < x
        entonces L ← m + 1
         sino    R ← m
      FinSi
FinMientras
```

- Explicar cómo funciona y qué es lo que produce.
- Calcular el orden de ejecución.

# ¿Cuál es el tiempo de ejecución del siguiente algoritmo para calcular $X^N$?

```
POTENCIA(x: real, n: entero): real
COM
    Resultado = 1.0
PARA i entre 0 y n-1, hacer
    Resultado = Resultado * x
FIN PARA
Devolver Resultado
FIN
```

# ¿Qué valor calcula f?

- Considere el siguiente fragmento de código de un programa en Java. ¿Qué valor calcula **f**? (Exprese su respuesta como una función de **n**). Calcule el orden de ejecución del método

```
class Example
{
   static int f (int n)
   {
     int sum = 0;
     for (int i = 1; i <= n; ++i)
        sum = sum + i;
     return sum;
   }
   // ...
}
```

# ¿Qué valor calcula g?

- Considere el siguiente fragmento de código de un programa en Java. (El método **f** se da en el ejercicio anterior). ¿Qué valor calcula **g**? (Exprese su respuesta como una función de n). Calcule el orden de ejecución del método **g**

```
class Example
{
   // ...
   static int g (int n)
   {
     int sum = 0;
     for (int i = 1; i <= n; ++i)
        sum = sum + i + f (i);
     return sum;
   }
}
```

# ¿Qué valor calcula h?

- Considere el siguiente fragmento de código de un programa en Java. (Los métodos **f** y **g** se dan en los ejercicios anteriores). ¿Qué valor calcula **h**? (Exprese su respuesta como una función de n). Calcule el orden de ejecución del método **h**.

```
class Example
{
   // ...
   int h (int n)
    { return f (n) + g (n); }
}
```

# Ejercicios para revisar (Weiss)

- Subsecuencia de suma máxima (5.3)
  - Algoritmo O($N^3$) obvio (5.3.1)
  - Algoritmo mejorado O($N^2$) (5.3.2)
  - Algoritmo lineal (5.3.3)

- ¿qué limitaciones tiene el análisis de órdenes O?
- Resuelva las cuestiones breves, problemas teóricos, problemas prácticos y prácticas de programación que se encuentran al final del Capítulo 5.