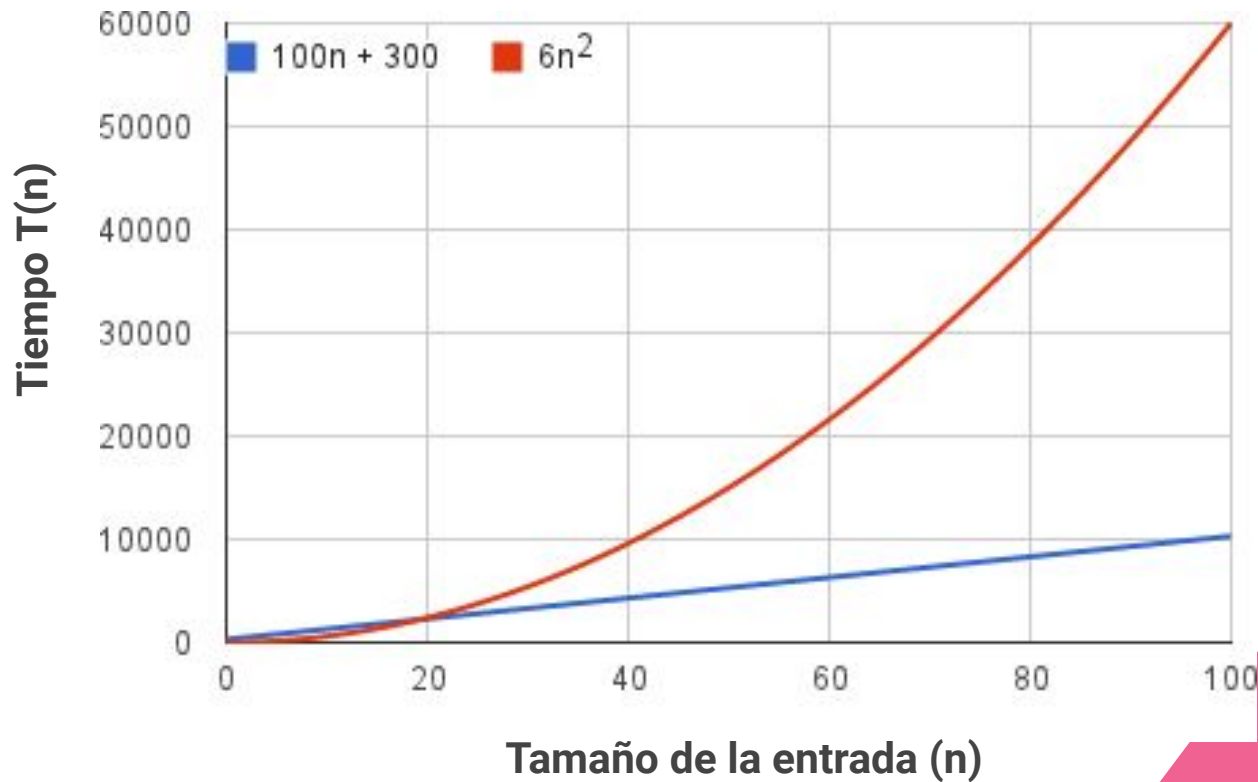


Tiempo de Ejecución (I)

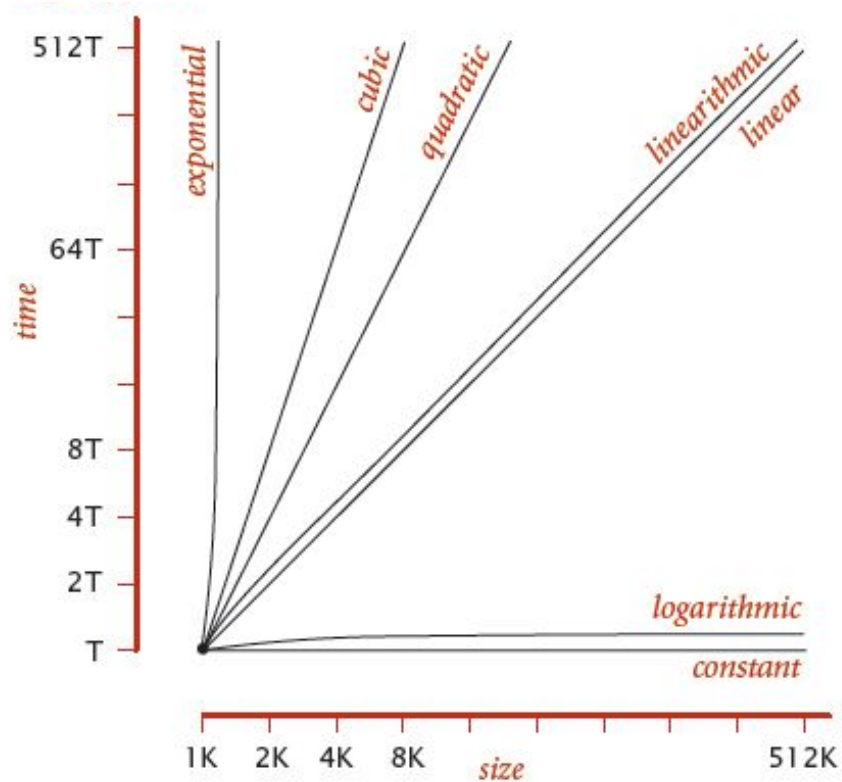
Tiempo de Ejecución de un Algoritmo

El Tiempo de Ejecución $T(n)$ muestra el desempeño del algoritmo en función de la cantidad de datos (n)



Orden de complejidad

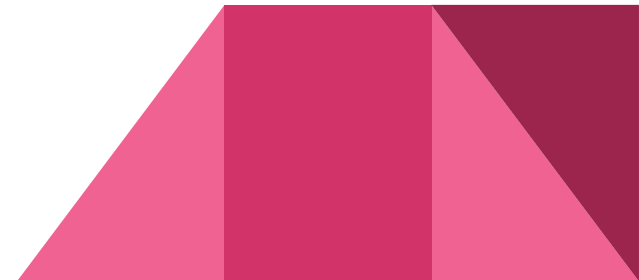
$O(N)$, permite enfocarnos en la tasa de crecimiento, por eso descartamos los términos menos significativos del $T(n)$



Definición de Big-Oh

$T(n)$ es $O(n)$ si existen constantes $c > 0$ y n_0 , tal que

$$T(n) \leq c \cdot O(n), \quad c > 0, \text{ para todo } n \geq n_0$$



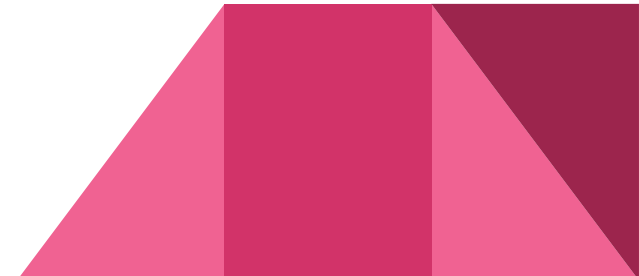
Ejercicios de demostración

¿ 2^{n+1} es $O(2^n)$?

(es decir, 2^{n+1} crece a una velocidad \leq que 2^n ?)

Para justificar la veracidad o falsedad de la afirmación:

- Usamos la definición de Big-Oh (encontrar c y n_0 que confirmen la veracidad) ó ...
- Demostramos la falsedad mediante un absurdo



¿ 2^{n+1} es $O(2^n)$?

(es decir, 2^{n+1} crece a una velocidad \leq que 2^n ?)

Para que 2^{n+1} sea $O(2^n)$, usando definición de Big-Oh, tiene que verificarse que $2^{n+1} \leq c2^n$, $c > 0$ y para todo $n \geq n_0$.

Ahora bien, $2^{n+1} = 2 \cdot 2^n$

En particular, podemos decir que $2^{n+1} \leq 2 \cdot 2^n$

Considerando $c=2$ y dado que vale para todo $n_0 \geq 0$ logramos acotar 2^{n+1} con $c2^n$ por lo cual,
 2^{n+1} es $O(2^n)$

¿ 2^{2n} es $O(2^n)$?

Usando definición de Big-Oh, tiene que verificarse que $2^{2n} \leq c2^n$ para todo $n \geq n_0$

Ahora bien, $2^{2n} = 2^n * 2^n$.


Por lo cual, podemos escribir $2^n * 2^n \leq c2^n$

Despejando c , $2^n * 2^n / 2^n \leq c$.

Simplificando, $2^n \leq c$.

Sin embargo, esto es absurdo, puesto que **nunca se puede acotar con una constante a una función creciente.**

Por lo cual 2^{2n} no es $O(2^n)$.

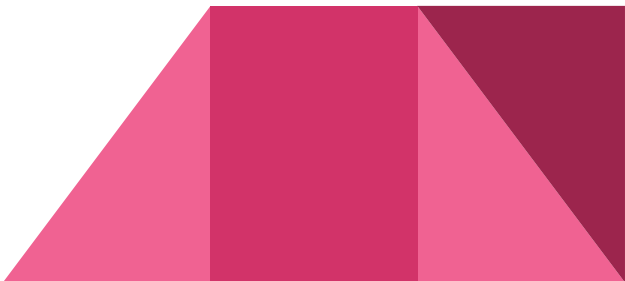


Ejercicio de $T(n)$

Caso de Algoritmo Iterativo

Encontrar el $T(n)$

```
public static void uno (int n)  {
    int  i, j, k ;
    int [] [] a, b, c;
    a = new int [n] [n];
    b = new int [n] [n];
    c = new int [n] [n];
    for ( i=1; i<=n-1; i++) {
        for ( j=i+1; j<=n; j++) {
            for ( k=1; k<=j; k++)  {
                c[i][j] = c[i][j]+ a[i][j]*b[i][j];
            }
        }
    }
}
```



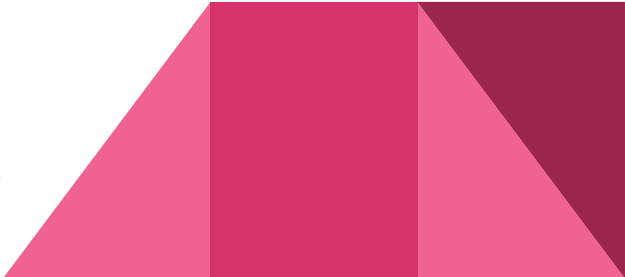
A tener en cuenta...

- las declaraciones, asignaciones y operaciones matemáticas tienen un tiempo constante
- los bucles se traducen como sumatorias, que indican la **cantidad de veces** que se ejecuta dicho bucle
- existe la posibilidad de relación entre el índice de una sumatoria y el contenido anidado dentro de la sumatoria
- la resolución de sumatorias anidadas se realiza desde las más internas hacia las externas

$$T(n) = c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^J c2$$

Diagram illustrating the resolution of nested summations from innermost to outermost:

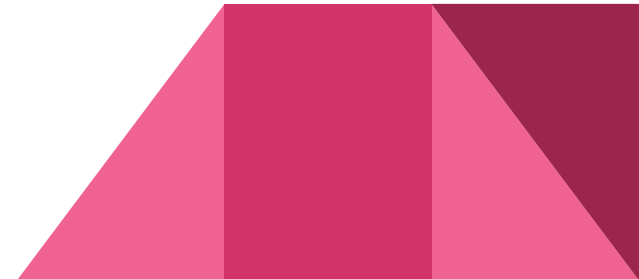
- 1ro (innermost): $\sum_{k=1}^J c2$ (green box)
- 2do: $\sum_{j=i+1}^n$ (red box)
- 3ro (outermost): $\sum_{i=1}^{n-1}$ (blue box)



A tener en cuenta...

$$\sum_{k=1}^j c2 = c1 + c2 + c2 + \dots + c2 = j * c2 \text{ (porque sumamos } c2 \text{ } j\text{-veces)}$$

$$\sum_{j=1}^i j = 1 + 2 + 3 + 4 + 5 + \dots + i = i(i + 1) / 2 \text{ (aplico fórmula)}$$



Resolución

```
public static void uno (int n)  {
    int  i, j, k ;
    int [] [] a, b, c;
    a = new int [n] [n];
    b = new int [n] [n];
    c = new int [n] [n];
    for ( i=1; i<=n-1; i++) {
        for ( j=i+1; j<=n; j++) {
            for ( k=1; k<=j; k++)  {
                c[i][j] = c[i][j]+ a[i][j]*b[i][j];
            }
        }
    }
}
```

$$T(n) = c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^j c2 = c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n (j * c2) =$$

$$= c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n (j * c2)$$

$$= c1 + c2 * \left(\sum_{i=1}^{n-1} \sum_{j=i+1}^n j \right)$$

$$= c1 + c2 * \sum_{i=1}^{n-1} \left(\sum_{j=1}^n j - \sum_{j=1}^i j \right)$$

$$= c1 + c2 * \sum_{i=1}^{n-1} \left(\frac{n * (n + 1)}{2} - \frac{i * (i + 1)}{2} \right)$$

$$= c1 + c2 * \sum_{i=1}^{n-1} \left(\frac{n * (n+1)}{2} - \frac{i * (i+1)}{2} \right)$$

$$= c1 + \frac{c2}{2} \sum_{i=1}^{n-1} n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} i * (i+1)$$

$$= c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} (i^2 + i)$$

$$= c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} (i^2) - \frac{c2}{2} \sum_{i=1}^{n-1} (i)$$

$$c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \left(\frac{(n-1) * n (2(n-1) + 1)}{6} \right) - \frac{c2}{2} \left(\frac{(n-1)n}{2} \right)$$

Encontrar el T(n)

$$T(n) = c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^j c2 = c1 + \sum_{i=1}^{n-1} \sum_{j=i+1}^n (j * c2) = c1 + c2 * \left(\sum_{i=1}^{n-1} \sum_{j=i+1}^n j \right) =$$

$$T(n) = c1 + c2 * \sum_{i=1}^{n-1} \left(\sum_{j=1}^n j - \sum_{j=1}^i j \right) = c1 + c2 * \sum_{i=1}^{n-1} \left(\frac{n * (n+1)}{2} - \frac{i * (i+1)}{2} \right) =$$

$$T(n) = c1 + \frac{c2}{2} \sum_{i=1}^{n-1} n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} i * (i+1) = c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} (i^2 + i) =$$

$$c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \sum_{i=1}^{n-1} (i^2) - \frac{c2}{2} \sum_{i=1}^{n-1} (i) =$$

$$c1 + \frac{c2}{2} (n-1) * n * (n+1) - \frac{c2}{2} \left(\frac{(n-1) * n (2(n-1) + 1)}{6} \right) - \frac{c2}{2} \left(\frac{(n-1)n}{2} \right)$$

Puntos claves

- Traducir constantes e iteraciones correctamente.
- Respetar los límites de las iteraciones al traducirlas a sumatorias (respetando las variables).
- Prestar atención a si dentro de una sumatoria **se hace referencia a la variable índice**.
- Tener presente que las equivalencias para la suma de los n primeros números naturales **comienza en 1** y no en un número arbitrario.

