

Agenda

Análisis de algoritmos

- Introducción al concepto $T(n)$
 - Tiempo, entrada, peor caso, etc.
- Cálculo del $T(n)$
 - En algoritmos iterativos
 - En algoritmos recursivos
- Notación Big-Oh
 - Definición y ejemplos
 - Reglas (suma, producto)
- Ejemplo de optimización de algoritmos

Análisis de algoritmos

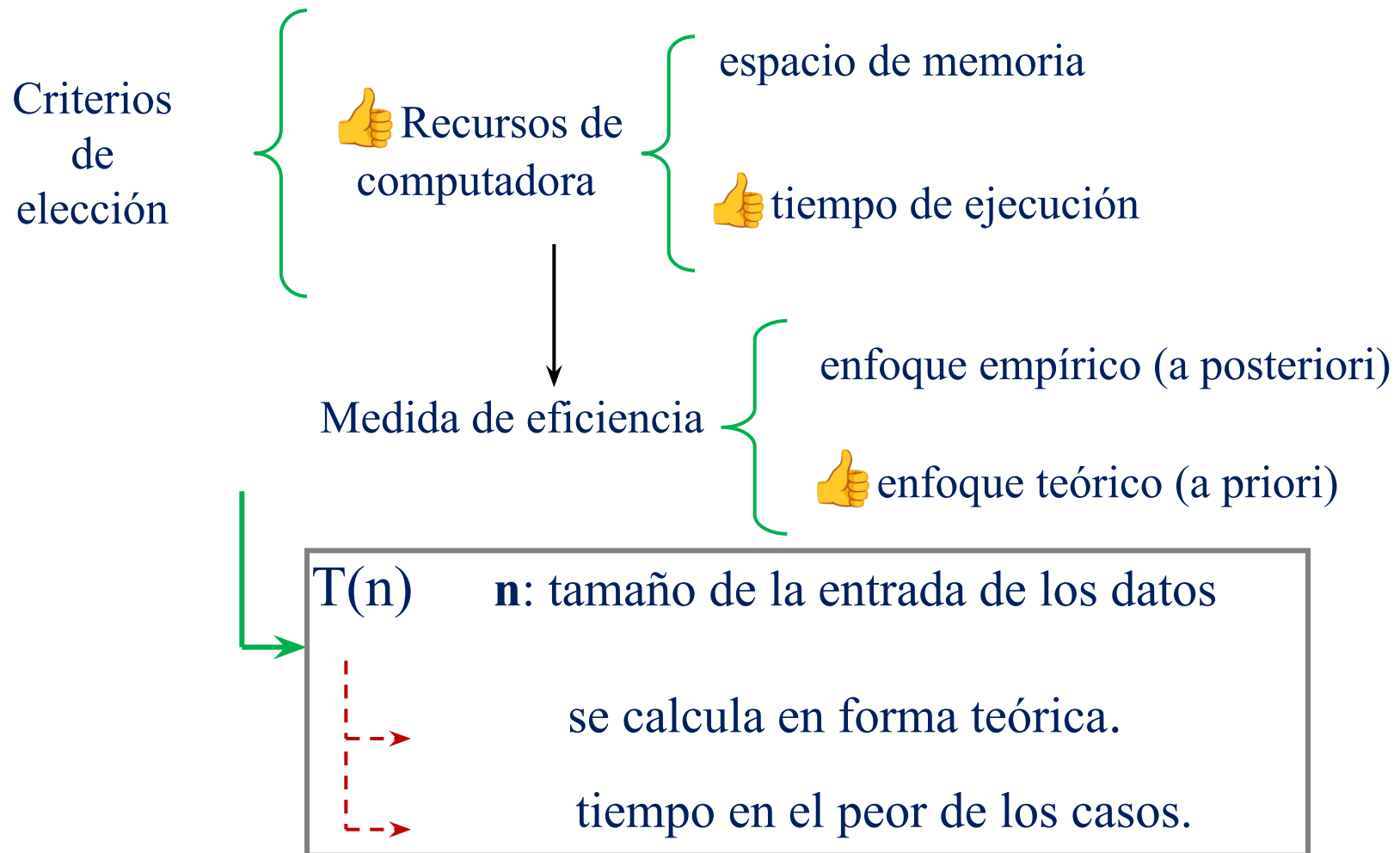
- *Nos permite comparar algoritmos en forma independiente de una plataforma en particular*
- *Mide la eficiencia de un algoritmo, dependiendo del tamaño de la entrada*

Análisis de algoritmos

Pasos a seguir:

- Caracterizar los datos de entrada del algoritmo
- Identificar las operaciones abstractas, sobre las que se basa el algoritmo
- Realizar un análisis matemático, para encontrar los valores de las cantidades del punto anterior

Introducción al concepto $T(n)$



Introducción al concepto $T(n)$

Adivinar número - Búsqueda lineal o binaria-

<https://es.khanacademy.org/computing/computer-science/algorithms/intro-to-algorithms/a/a-guessing-game>

Hemos analizado la búsqueda lineal y la búsqueda binaria al contar el número máximo de intentos que necesitamos hacer.

Pero lo que en realidad queremos saber es *cuánto tiempo tardan* estos algoritmos.

Estamos interesados en el *tiempo*, no sólo en la cantidad máxima de *intentos*.

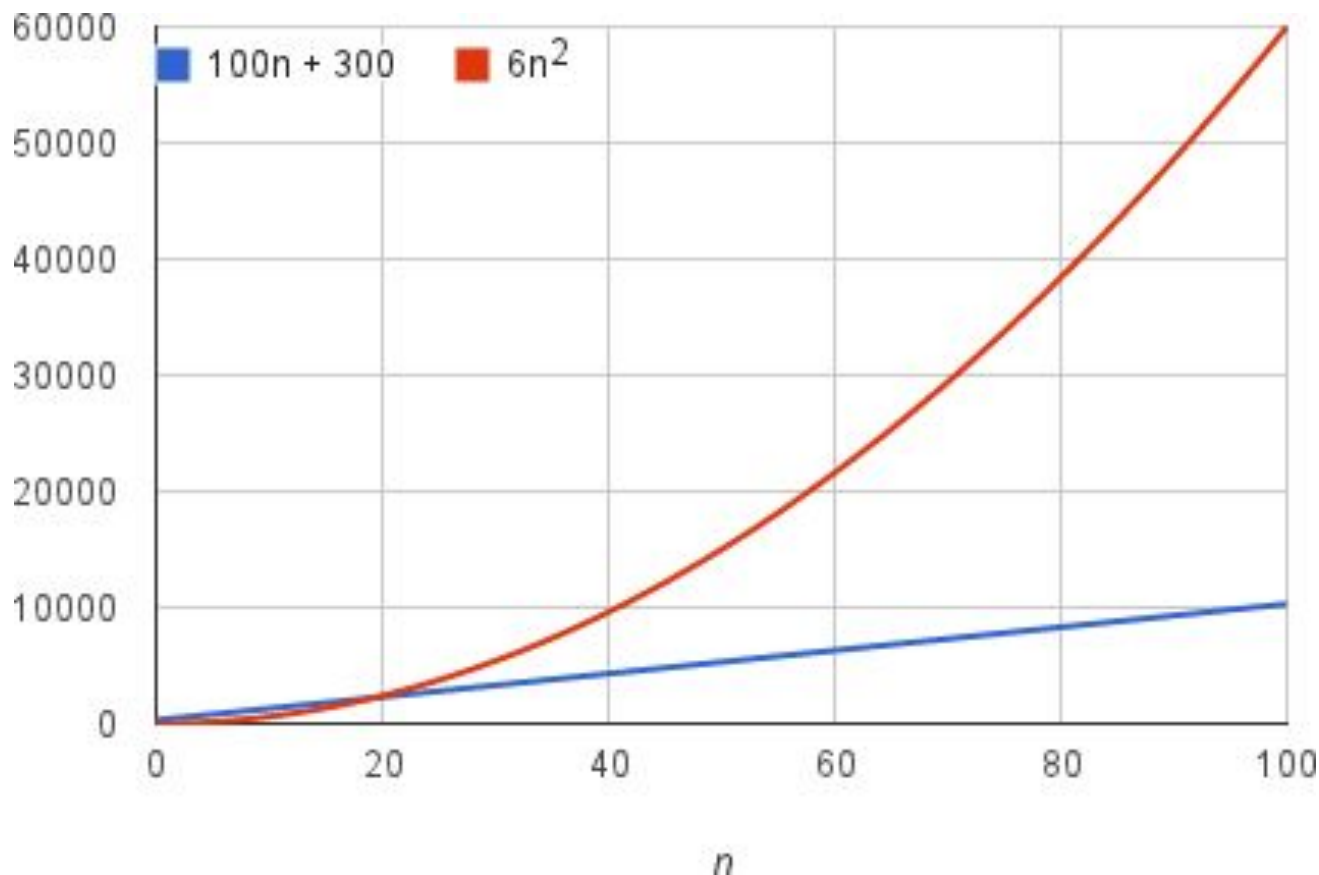
Introducción al concepto $T(n)$

Debemos enfocarnos en cuán rápido crece una función $T(n)$ respecto al tamaño de la entrada. A esto lo llamamos la **tasa o velocidad de crecimiento** del tiempo de ejecución.

Por ejemplo, supongamos que un algoritmo, que corre con una entrada de tamaño n , tarda $6n^2+100n+300$ instrucciones de máquina. El término $6n^2$ se vuelve más grande que el resto de los términos, $100n+300$ una vez que n se hace suficientemente grande, 20 en este caso.

Introducción al concepto $T(n)$

Gráfica que muestra los valores de $6n^2$ y de $100n+300$ para valores de n de 0 a 100:



Introducción al concepto $T(n)$

Al descartar los términos menos significativos y los coeficientes constantes, podemos enfocarnos en la parte importante del tiempo de ejecución de un algoritmo, su tasa o velocidad de crecimiento, sin involucrarnos en detalles que complican nuestro entendimiento.

Cuando descartamos los coeficientes constantes y los términos menos significativos, usamos **notación asintótica**.

Cuadro comparativo del tiempo para diferentes funciones

Costo		$n=10^3$	Tiempo	$n=10^6$	Tiempo
Logarítmico	$\log_2(n)$	10	10 segundos	20	20 segundos
Lineal	n	10^3	16 minutos	10^6	11 días
Cuadrático	n^2	10^6	11 días	10^{12}	30.000 años

Orden de ejecución del algoritmo

↓
Cantidad de operaciones
↓
Tiempo total del algoritmo

$n = 10^3$

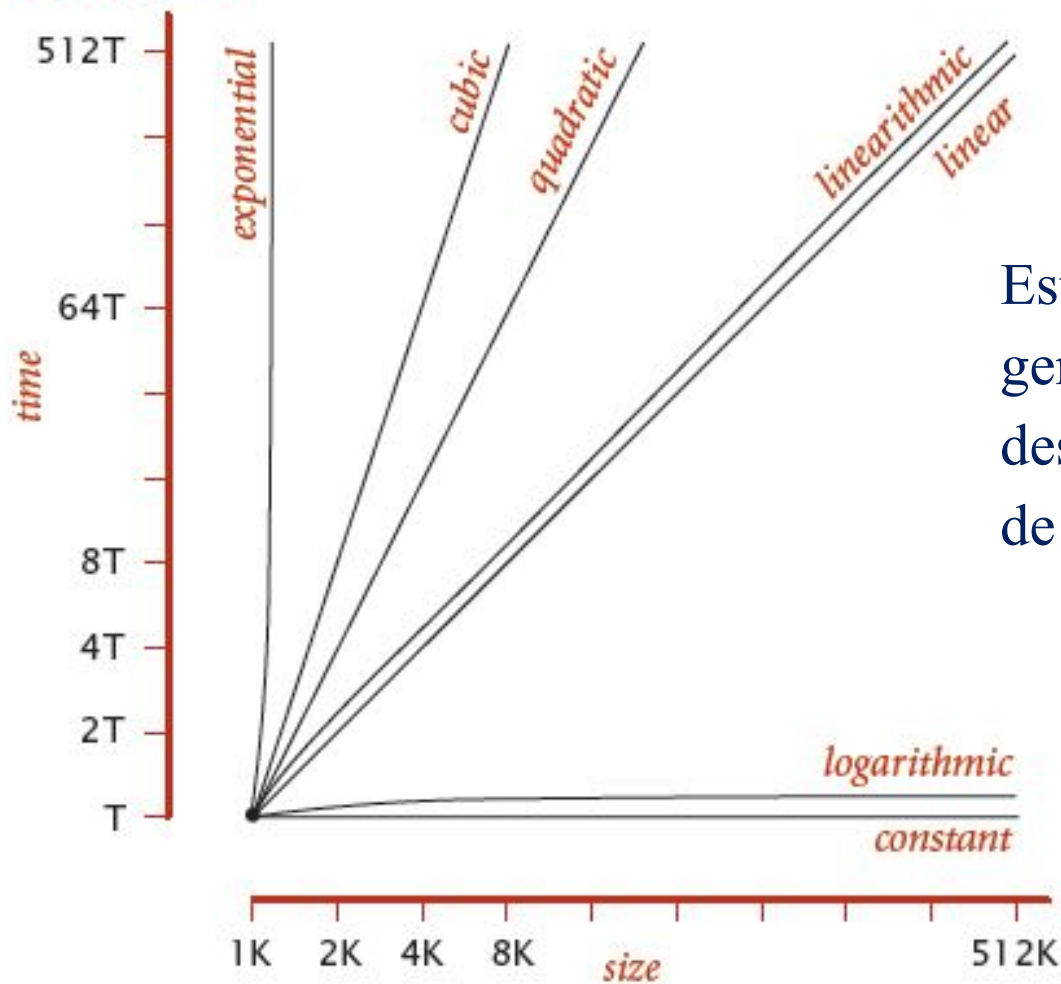
↓
Cantidad de operaciones
↓
Tiempo total del algoritmo

$n = 10^6$

Algunas funciones

Ordenadas en forma creciente	Nombre
1	Constante
$\log n$	Logaritmo
n	Lineal
$n \log n$	$n \log n$
n^2	Cuadrática
n^3	Cúbica
$c^n \quad c > 1$	Exponencial

Algunas funciones



Este conjunto de funciones en general es suficiente para describir la tasa de crecimiento de los algoritmos típicos

Problema

Considerando que un algoritmo requiere $f(n)$ operaciones para resolver un problema y la computadora procesa 100 operaciones por segundo.

Si $f(n)$ es:

a.- $\log_{10} n$

b.- \sqrt{n}

Determine el tiempo en segundos requerido por el algoritmo para resolver un problema de tamaño $n=10000$.

Problema

Suponga que Ud. tiene un algoritmo ALGO-1 con un tiempo de ejecución exacto de $10n^2$. ¿En cuánto se hace más lento ALGO-1 cuando el tamaño de la entrada n aumenta:.....?

- a.- El doble
- b.- El triple

Cálculo del Tiempo de Ejecución

Estructuras
de
Control

- Secuencia
- Condicional:
 - *if/else*
 - *switch*
- Iteración:
 - *for*
 - *while*
 - *do-while*

Cálculo del Tiempo de Ejecución

Condicional:

a) **if** (*boolean expression*) {
 statement(s)
 }

b) **if** (*boolean expression*) {
 statement(s)
 } **else** {
 statement(s)
 }

Cálculo del Tiempo de Ejecución

Condicional:

c) **switch** (*integer expression*) {
 case *integer expression* : *statement(s)* ; **break**;
 ...
 case *integer expression* : *statement(s)* ; **break**;
 default : *statement(s)* ; **break**;
}

Cálculo del Tiempo de Ejecución

Iteración:


- a)* **for** (*initialization; termination; increment*) {
 statement(s)
 }
- b)* **while** (*boolean expression*) {
 statement(s)
 }
- c)* **do** {
 statement(s)
 } **while** (*boolean expression*);

Cálculo del Tiempo de Ejecución

Iteración:

a) For

Viene como
parámetro

```
int sum = 0;   
int [] a = new int [n];  
for (int i = 1; i <= n ; i++ )  
    sum += a[i];
```


Cálculo del Tiempo de Ejecución

Iteración:

a) For

Viene como
parámetro

```
int sum = 0;  
int [] a = new int [n];  
for (int i = 1; i <= n ; i++ )  
    sum += a[i];
```



$$T(n) = cte_1 + \sum_{i=1}^n cte_2 =$$

$$= cte_1 + n * cte_2$$


$$\Rightarrow O(n)$$

Cálculo del Tiempo de Ejecución

a) **For**

Viene como
parámetro

```
int sum = 0;  
int [] a = new int [n][n];  
for (int i = 1; i <= n ; i++) {  
    for (int j = 1; j <= n ; j++)  
        sum += a[i][j];  
}
```




Cálculo del Tiempo de Ejecución

a) **For**

Viene como
parámetro

```
int sum = 0;  
int [] a = new int [n][n];  
for (int i = 1; i <= n ; i++) {  
    for (int j = 1; j <= n ; j++)  
        sum += a[i][j];  
}
```



$$T(n) = cte_1 + \sum_{i=1}^n \sum_{j=1}^n cte_2 =$$

$$= cte_1 + n * n * cte_2$$

$$\Rightarrow O(n^2)$$

Cálculo del Tiempo de Ejecución

a) **For**

Viene como
parámetro



```
int [] a = new int [n];  
int [] s = new int [n];  
for ( int i =1; i<= n ; i++ )  
    s[i] = 0;  
for ( int i =1; i<= n ; i++ ) {  
    for (int j =1; j<= i ; j++)  
        s[i] += a[j];  
}
```

Cálculo del Tiempo de Ejecución

a) **For**

Viene como
parámetro



```
int [] a = new int [n];  
int [] s = new int [n];  
for ( int i =1; i<= n ; i++ )  
    s[i] = 0;  
for ( int i =1; i<= n ; i++ ) {  
    for (int j =1; j<= i ; j++)  
        s[i] += a[j];  
}
```

$$\begin{aligned} T(n) &= cte_1 + \sum_{i=1}^n cte_2 + \\ &+ \sum_{i=1}^n \sum_{j=1}^i cte_3 = \\ &= cte_1 + n * cte_2 + \\ &cte_3 * \sum_{i=1}^n i = \dots \end{aligned}$$

$$\Rightarrow O(n^2)$$

Cálculo del Tiempo de Ejecución

□ Iteración:

b) **While**

```
int x= 0;  
int i = 1;  
while ( i <= n) {  
    x = x + 1;  
    i = i + 2;  
}
```


Cálculo del Tiempo de Ejecución

□ Iteración:

b) While

```
int x= 0;  
int i = 1;  
while ( i <= n) {  
    x = x + 1;  
    i = i + 2;  
}
```

$$\begin{aligned} T(n) &= cte_1 + \sum_{i=1}^{(n+1)/2} cte_2 = \\ &= cte_1 + cte_2/2 * (n+1) \\ &\Rightarrow O(n) \end{aligned}$$

Cálculo del Tiempo de Ejecución

□ Iteración :

b) While

int x= 1;

while (x < n)

*x = 2 *x;*

Cálculo del Tiempo de Ejecución

□ Iteración :

b) While

int x= 1;

while (x < n)

*x = 2 *x;*

$$T(n) = cte_1 + cte_2 * \log(n)$$

$$\Rightarrow O(\log(n))$$

Aclaración:

Si n es potencia de 2: realiza $\log(n)$ iteraciones

Si n no es potencia de 2: realiza $\log(n) + 1$ iteraciones

Cálculo del Tiempo de Ejecución

Ejercicio

¿Cuál es la expresión correcta respecto al tiempo de ejecución del siguiente segmento de código?

```
for (i = 0; i < n; i++)  
    for (j = 1; j < n; j+=n/2)  
        x = x + 1;
```

Cálculo del Tiempo de Ejecución

Ejercicio

¿Cuál es la expresión correcta respecto al tiempo de ejecución del siguiente segmento de código?

```
for (i = 0; i < n; i++)  
    for (j = 1; j < n; j+=n/2)  
        x = x + 1;
```

- (a) $O(\sqrt{n})$ (b) $O(n)$ (c) $O(n \log n)$ (d) $O(n^2)$ (e) $O(n^3)$

Cálculo del Tiempo de Ejecución

Ejercicio

Considere el siguiente fragmento de código:

```
int count = 0;
int N = a.length;
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        a[j]++;
```

Suponga que tarda 1 seg cuando $N=3500$,
¿cuánto tardará *aproximadamente* para $N=35000$? Justifique su respuesta.

Cálculo del Tiempo de Ejecución

Ejemplo:

```
private void imparesypares(int n){  
    int x=0; int y=0;  
  
    for (int i=1;i<=n;i++)  
        if (esImpar(i))  
            for (int j=i;j<=n;j++)  
                x++;  
        else  
            for (int j=1;j<=i;j++)  
                y++;  
}
```

```
public boolean esImpar(int unNumero){  
    if (unNumero%2 != 0)  
        return true;  
    else  
        return false;  
}
```

Cálculo del Tiempo de Ejecución

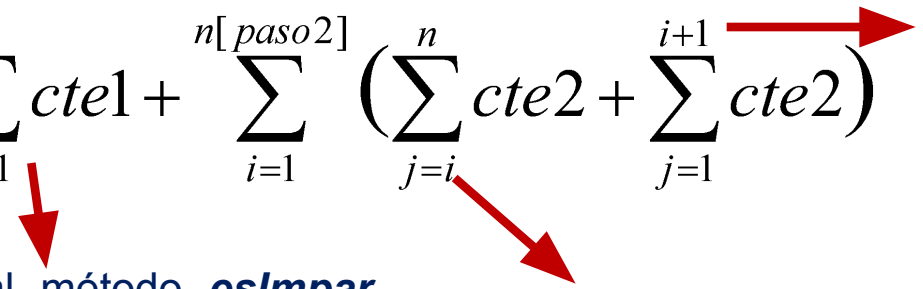
Ejemplo (cont.):

Desarrollo de la función $T(n)$ del método *imparesypares*

- Asumiendo valor de “n” par.
- El método *esImpar* tiene todas sentencias constantes

$$T_{esImpar}(n) = cte1$$

- El método *imparesypares* tiene un loop en el que: en cada iteración se llama al método *esImpar* y la mitad de las veces se ejecuta uno de los *for* (para valores de “i” impares) y la mitad restante el otro *for* (para valores de “i” pares)

$$T(n) = \sum_{i=1}^n cte1 + \sum_{i=1}^{n[paso2]} \left(\sum_{j=i}^n cte2 + \sum_{j=1}^{i+1} cte2 \right)$$


Es la llamada al método *esImpar*, que se ejecuta para todos los valores de “i”

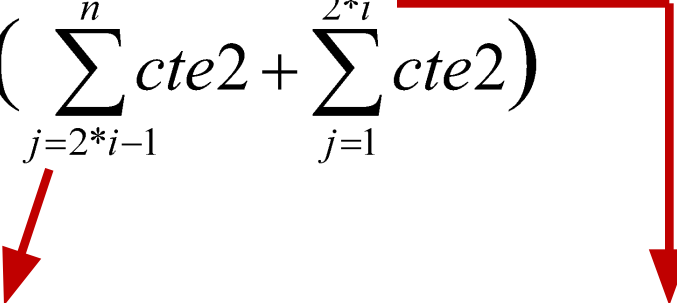
Valores de “i” impares

Valores pares dados por el siguiente a los impares “i”

Cálculo del Tiempo de Ejecución

Ejemplo (cont.):

Desarrollo de la función $T(n)$ del método *imparesypares*

$$T(n) = \sum_{i=1}^n cte1 + \sum_{i=1}^{n/2} \left(\sum_{j=2*i-1}^n cte2 + \sum_{j=1}^{2*i} cte2 \right)$$


Como “ i ” ahora toma valores consecutivos entre 1 y $n/2$, entonces se hace un cambio de variable para seguir tomándose valores impares y pares en cada loop.

Cálculo del Tiempo de Ejecución

Ejemplo (cont.):

Resolviendo la función $T(n)$

$$T(n) = \sum_{i=1}^n cte1 + \sum_{i=1}^{n/2} \left(\sum_{j=2*i-1}^n cte2 + \sum_{j=1}^{2*i} cte2 \right)$$

$$T(n) = cte1 * n + \sum_{i=1}^{n/2} cte2 * (n - 2*i + 1 + 1 + 2*i - 1 + 1) =$$

$$= cte1 * n + cte2 * (n + 2) * n / 2$$

$$= cte1 * n + cte2 / 2 * n^2 + cte2 * n$$

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

Ejercicio

```
private int ejercicio3(int n) {  
    int p=0; int j=1;  
  
    for (int i=1; i<=n; i++)  
        if (esImpar(i))  
            j:=j*2;  
        else  
            for (int k=1; k<=j; k++)  
                p:= p+1;  
    return p;  
}
```

```
public boolean esImpar(int unNumero){  
    if (unNumero%2 != 0)  
        return true;  
    else  
        return false;  
}
```

Ejercicio

```
0  i = 0; j = 0;
1  while(i < 1000)
2      for( int k = i; k <= n; k++ ) {
3          i++;
4          j++; }
5  for( int p = 0; p < n*n; p++ )
6      for( int q = 0; q < p; q++ )
7          j--;
```

1. ¿Con qué valor termina la variable i ?
2. ¿Cuántas veces se ejecuta la sentencia 3?
 - a. $O(n)$
 - b. $O(n^2)$
 - c. $O(n^3)$
 - d. $O(n^4)$
 - e. Ninguna de las anteriores