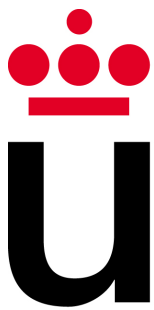
Tema 3.1: Introducción a la eficiencia de algoritmos

Diseño y Análisis de Algoritmos



Universidad
Rey Juan Carlos

Contenidos

Eficiencia de Algoritmos

- ¿Qué recursos necesitan los algoritmos?
  - En espacio (memoria)
  - En tiempo (número de operaciones)
  - Otros:
    - Ancho de banda

Eficiencia de Algoritmos

- ¿Por qué estudiamos la eficiencia de algoritmos, cuando parece que hay otras características del software más importantes?

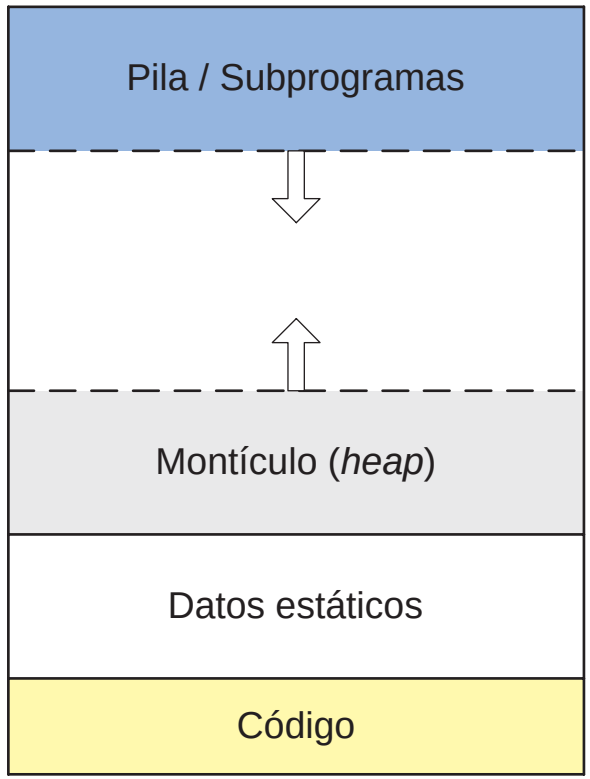
- Amigabilidad
- Buen estilo de programación
- Comentarios
- Corrección
- Escalabilidad
- Funcionalidad
- Mantenibilidad
- Modularidad
- Rendimiento
- Robustez
- Seguridad
- Simplicidad
- Tiempo de programación
- ...

- La eficiencia nos va a ayudar a alcanzar el resto de características

Modelo de memoria

- Código del algoritmo
- Datos estáticos
  - Variables globales
- Llamadas a subprogramas
  - Parámetros
  - Variables locales
- Memoria dinámica

- Modelo de memoria



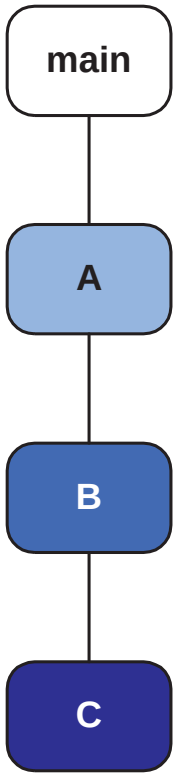
Llamadas a subprogramas

```
1  main(...){
2  ...
3  A(..)
4  ...
5  }
6
7  A(...){
8  ...
9  B(..);
10 ...
11 }
12
```

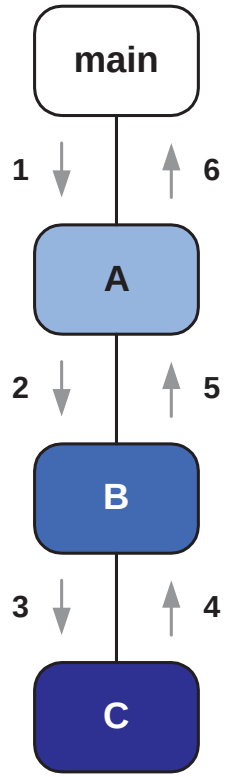
```
13 B(...){
14 ...
15 C(..);
16 ...
17 }
18
19 C(...){
20 ...
21 }
```

Árbol de llamadas

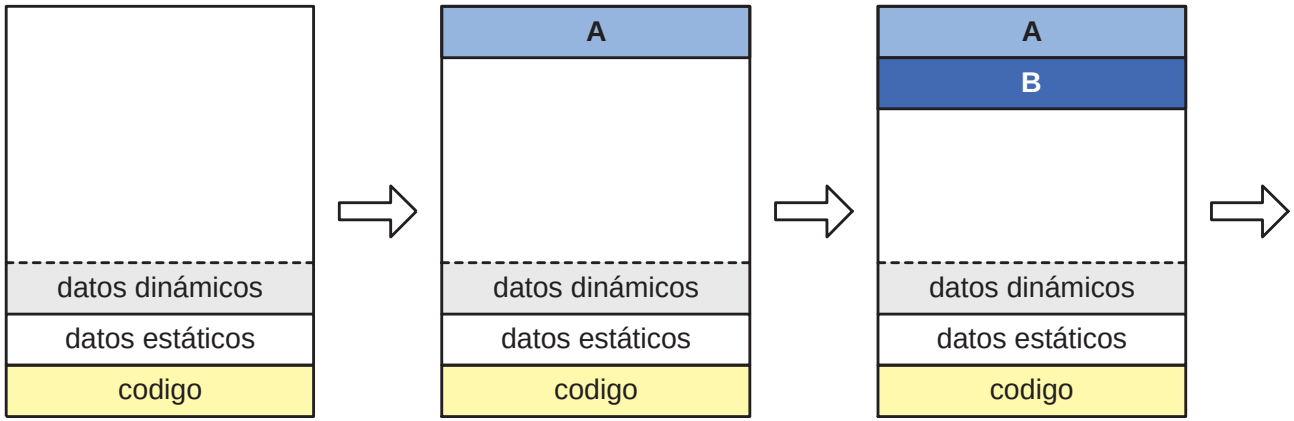
- Árbol de llamadas

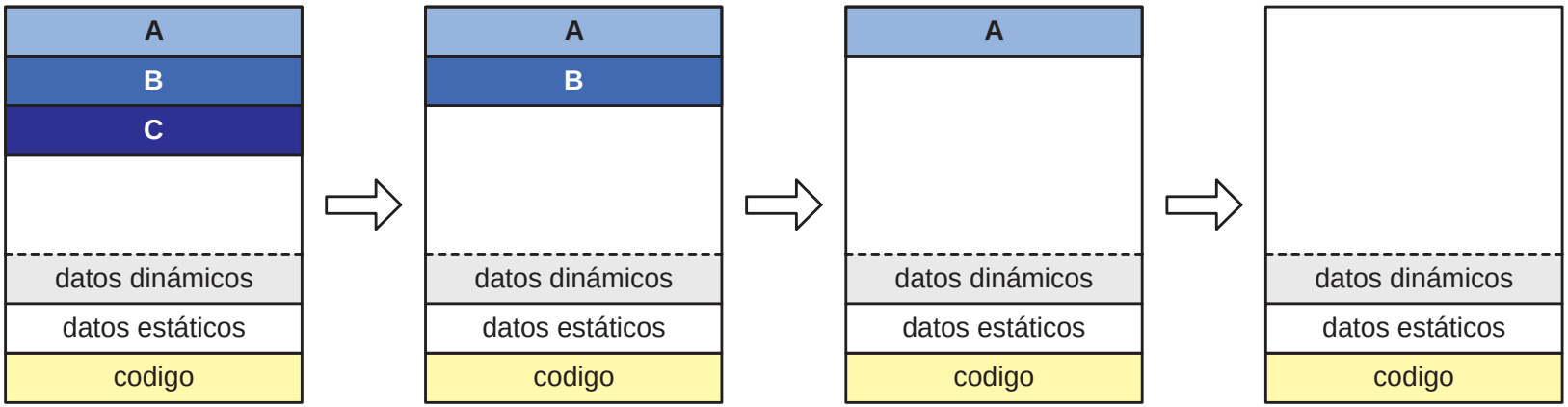


- Proceso de llamadas



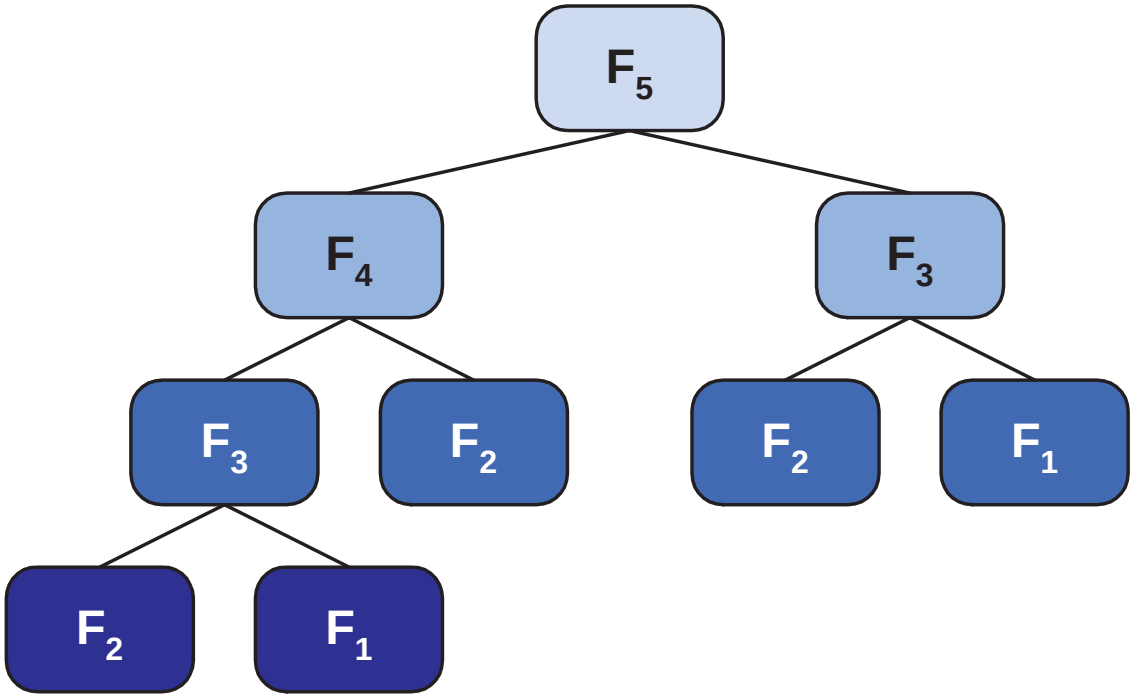
Estado de la pila





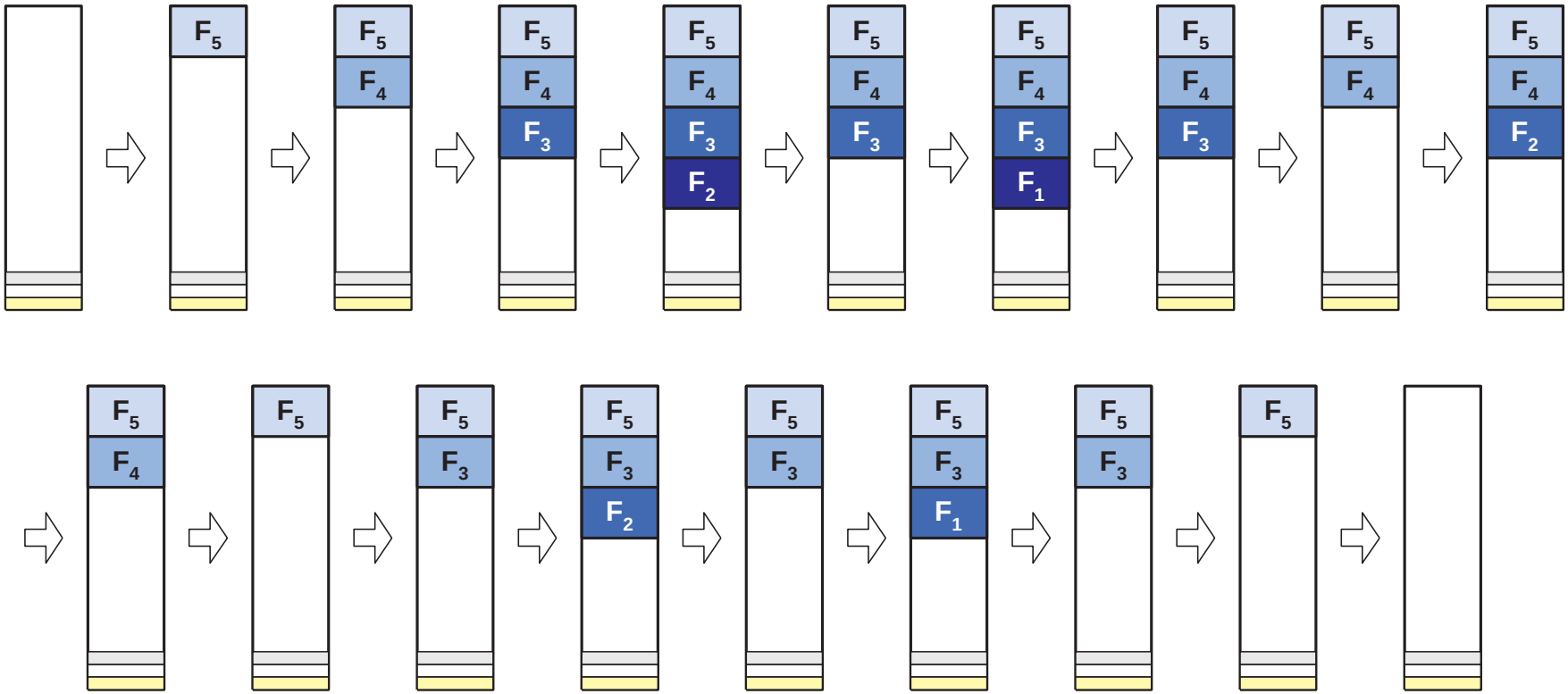
Recursividad

- Caso recursivo: $F_{n+2} = F_{n+1} + F_n$
- Casos base $F_1 = F_2 = 1$



- Observación: ¿Por qué los nodos de un mismo nivel aparecen con el mismo color, a pesar de corresponder a llamadas con diferentes parámetros?

Estado de la pila



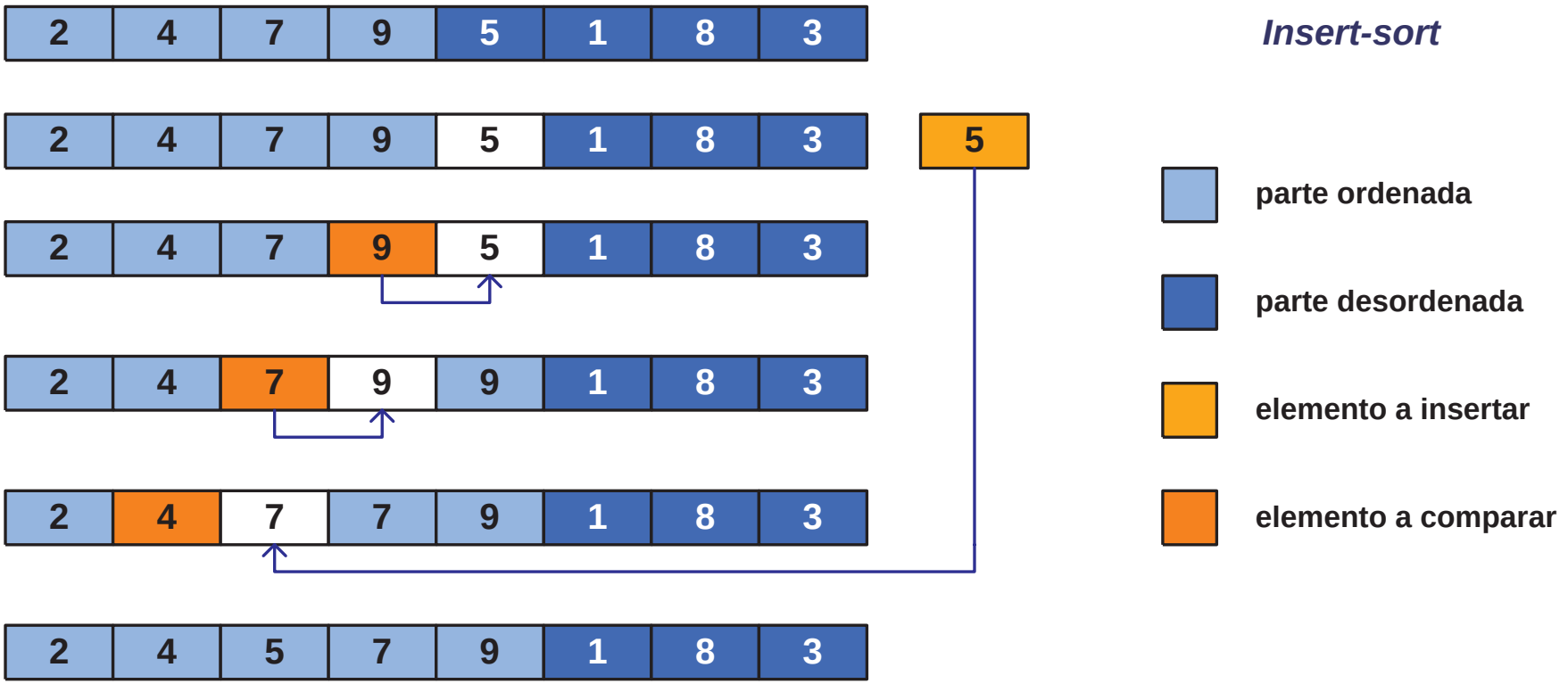
$$\text{Espacio}(n) \in \Theta(n)$$

Llamadas a subprogramas

- La complejidad depende de la profundidad del árbol de llamadas
- Para ilustrar esto se han coloreado los nodos de un mismo nivel en el árbol con el mismo color
  - Las llamadas de un mismo color aparecen en el mismo nivel en la pila, al igual que en el árbol
- La complejidad también depende de los recursos que consuma cada subprograma ejecutado (cada llamada)
  - Parámetros de los subprogramas
  - Variables locales a los subprogramas

Un ejemplo: Insert-sort

- Algoritmo de ordenación por inserción directa (*insert-sort*)



Un ejemplo: Insert-sort

- Pseudocódigo y coste por línea

	Insert-sort(A)	**Coste**	**Nº de veces**
1	FOR j=2 to length(A)	C_1	n
2	val = A[j]	C_2	$n-1$
3	// Inserta A[j] en la secuencia		
	// ordenada A[1..j-1]	0	$n-1$
4	i = j-1	C_4	$n-1$
5	WHILE (i>0) y (A[i]>val)	C_5	$\sum_{j=2}^{n} t_j$
6	A[i+1] = A[i]	C_6	$\sum_{j=2}^{n}(t_j-1)$
7	i = i-1	C_7	$\sum_{j=2}^{n}(t_j-1)$
8	A[i+1] = val	C_8	$n-1$

- n = length(A)
- t_j = nº de veces que se repite el bucle de la línea 5, para un determinado valor de j

Un ejemplo: Insert-sort

- Ignoramos el coste concreto de cada operación básica
- Cada línea I tardará un determinado tiempo o coste, que denotamos por C_I
- La función de coste o tiempo, en función del tamaño del vector n es:

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n} t_j +$$

$$C_6 \sum_{j=2}^{n} (t_j - 1) + C_7 \sum_{j=2}^{n} (t_j - 1) + C_8(n-1)$$

- t_j depende del vector de entrada particular

Un ejemplo: Insert-sort

- Mejor caso (vector ordenado de menor a mayor)
- $t_j = 1$, para todo j

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5(n-1) + C_8(n-1)$$

$$= (C_1 + C_2 + C_4 + C_5 + C_8)n - (C_2 + C_4 + C_5 + C_8)$$

$$= K_1 n + K_2 \quad \in \quad \Theta(n)$$

- El orden de $T(n)$ es lineal
- ¡Para el orden, el valor de las constantes no importa!

Un ejemplo: Insert-sort

- Peor caso (vector ordenado de mayor a menor)
- $t_j = j$ (valor máximo para cada j)
- En primer lugar observamos que:

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n} (j-1) = \frac{n(n+1)}{2} - 1 - (n-1) = \frac{n(n-1)}{2}$$

Un ejemplo: Insert-sort

- Sustituyendo:

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\left(\frac{n(n+1)}{2} - 1\right) +$$

$$C_6\left(\frac{n(n-1)}{2}\right) + C_7\left(\frac{n(n-1)}{2}\right) + C_8(n-1) =$$

$$= \left(\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2}\right) n^2 + \left(C_1 + C_2 + C_4 + \frac{C_5}{2} - \frac{C_6}{2} - \frac{C_7}{2} + C_8\right) n -$$

$$(C_2 + C_4 + C_5 + C_8) = K_1 n^2 + K_2 n + K_3 \quad \in \quad \Theta(n^2)$$

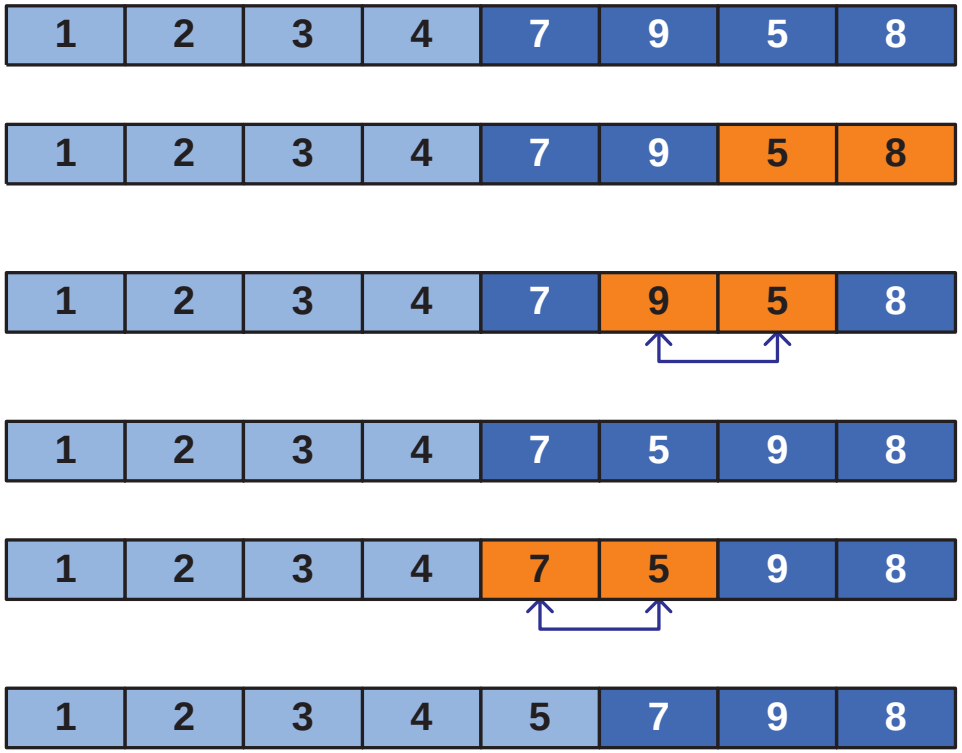
- El orden de $T(n)$ es cuadrático

Caso medio frente a caso peor

- En ocasiones se puede calcular el caso medio, pero nos centraremos en el estudio de caso peor debido a:
  - El caso peor representa un cota superior para cualquier entrada, dándonos una garantía de que el algoritmo no tardará más
  - El caso peor suele ocurrir con frecuencia
    - En una búsqueda, ocurre cuando el elemento buscado no se encuentra
  - El tiempo medio suele ser tan malo como el peor en términos asintóticos
    - Si en el *insert-sort* hay que insertar el elemento hasta la posición $j/2$, el tiempo también sale cuadrático

Otro ejemplo: Bubble-sort

- Algoritmo de ordenación "burbuja" (*bubble-sort*)



Otro ejemplo: Bubble-sort

- Pseudocódigo

```
1  void bubbleSort(int []v)
2  {
3    for(int i=0; i<v.length-1; i++)       // for 1
4    {
5      for(int j=v.length-1; j>i; j--)     // for 2
6      {
7        if(v[j-1]>v[j])
8        {
9          int aux = v[j-1];
10         v[j-1] = v[j];
11         v[j] = aux;
12       }
13     }
14   }
15 }
```

Operaciones de un bucle de n iteraciones

- 1 inicialización
- n comparaciones
- Tiempo de ejecutar el cuerpo del bucle n veces
- n incrementos
- 1 última comparación para salir del bucle

$$T_{\text{bucle}} = 1_{\text{inic.}} + \sum^{n}(1_{\text{comp.}} + T_{\text{cuerpo}} + 1_{\text{incr.}}) + 1_{\text{última comp.}}$$

Tiempo en el mejor caso

- 1ª forma: simplemente contando operaciones
  - For 1:

$$1_{\text{inic.}} + (n-1)_{\text{incr.}} + (n-1)_{\text{comp.}} + 1_{\text{última comp.}} = 2n$$

  - For 2:

$$(n-1)_{\text{inic.}} + \big((n-1)+(n-2)+\ldots+1\big)_{\text{comp.}} + (n-1)_{\text{última comp.}}$$

$$\big((n-1)+(n-2)+\ldots+1\big)_{\text{decr.}} + \big((n-1)+(n-2)+\ldots+1\big)_{\text{comp. del IF}}$$

  - Sumando todo:

$$T_{\text{mejor}}(n) = 2n + 2(n-1) + 3\frac{n(n-1)}{2} = \frac{3}{2}n^2 + \frac{5}{2}n - 2 \quad \in \quad \Theta(n^2)$$

Tiempo en el mejor caso

- 2ª forma: usando la fórmula del bucle

$$T_{\text{mejor}}(n) = 1 + \sum_{i=0}^{n-2} \left[1 + 1 + \sum_{j=n-1}^{i+1} (1+1+1) + 1 + 1 \right] + 1$$

  - For 1:

$$T_{\text{mejor}}(n) = 1_{\text{inic.}} + \sum_{i=0}^{n-2} [1_{\text{comp.}} + T_{\text{For 2}} + 1_{\text{incr.}}] + 1_{\text{última comp.}}$$

  - For 2:

$$T_{\text{mejor}}(n) = 1_{\text{inic.}} + \sum_{j=n-1}^{i+1} [1_{\text{comp.}} + 1_{\text{comp. IF}} + 1_{\text{incr.}}] + 1_{\text{última comp.}}$$

Tiempo en el mejor caso

- Simplificando:

$$T_{\text{mejor}}(n) = 2 + \sum_{i=0}^{n-2} \left[4 + \sum_{j=n-1}^{i+1} 3 \right]$$

- El sumatorio interno suma $(n - i - 1)$ veces. Por tanto:

$$T_{\text{mejor}}(n) = 2 + 4(n-1) + \sum_{i=0}^{n-2} 3(n-i-1) =$$

$$2 + 4n - 4 + 3n(n-1) - 3\frac{(n-1)(n-2)}{2} - 3(n-1)$$

$$T_{\text{mejor}}(n) = \frac{3}{2}n^2 + \frac{5}{2}n - 2 \quad \in \quad \Theta(n^2)$$