Ordenación interna

(Parte I)

Prof. Cristian Cappo

Marzo/2018

En esta clase

- Conceptos
  - Definición
  - Algoritmos y clasificación
  - Rendimiento
  - Inversiones
- Algoritmos
  - Inserción
  - ShellSort
  - Quicksort

Ordenación
Conceptos

- Conceptos fundamentales

La ordenación es el proceso de tomar una colección de *objetos ordenables*(a_1, a_2, a_3, .. , a_n) y retornar la misma colección pero reubicados (a'_1, a'_2, a'_3, .. , a'_n) de tal forma que

$$a'_1 < a'_2 < a'_3 < .. < a'_n$$

Objetos ordenables: siempre que exista posibilidad de compararlos y aplicar operadores relacionales < , >, == .
Ejemplos: números, nombres, fechas, etc.

Ordenación
Conceptos

- Muchos problemas se basan en operaciones de ordenación.

  A veces la eficiencia de la solución a esos problemas queda determinada por el método de ordenación utilizado (*ver problema 8.1 en* [WEISS2000])

- Operación más estudiada en computación.
- Se han ideado muchos algoritmos que lo resuelven.

Ordenación Interna:

Realizada enteramente en memoria principal (RAM)

Ordenación
Conceptos

- Estudiaremos algoritmos que operan bajo la premisa de que los elementos pueden compararse.

  La mayoría de los algoritmos asumen esto, y se denominan ***Algoritmos de Ordenación basados en comparación***

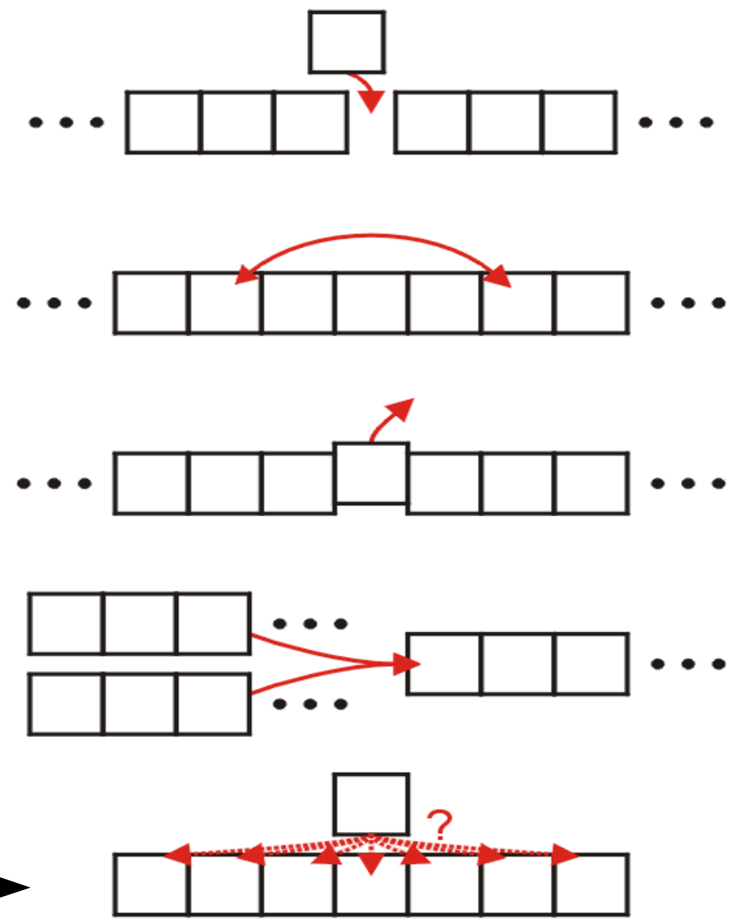
  Existen otros algoritmos que no necesitan realizar comparaciones y también los veremos en la siguiente clase.

Ordenación
Conceptos

- Memoria adicional para los algoritmos:
  - **In-Place** : no existe un consumo adicional de memoria por tanto su costo espacial esta en *O(1)*. Esto es: solo lo necesario para variables locales.
    - Inserción, Selección, Burbuja, ShellSort, QuickSort, HeapSort
  - **Not-in-place**: requieren memoria adicional en relación a la cantidad de elementos a ordenar. Por ejemplo un arreglo auxiliar, lo que sería un costo espacial *O(n)*.
    - MergeSort, RadixSort

Ordenación
Conceptos

- Podemos clasificar los algoritmos por las operaciones que hacen
  - Inserción
  - Extracción
  - Selección
  - Mezcla
  - Distribución



Ordenación
Rendimiento

- El rendimiento de los algoritmos suelen estar en una de estas tres posibles opciones

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

- En general examinaremos los escenarios para cada caso: peor, promedio y mejor
- El tiempo de ejecución de cada algoritmo cambia radicalmente de acuerdo al escenario.

¿Por qué $O(n)$ es el mínimo? y $O(n^2)$ el máximo?

Ordenación
Rendimiento

- Algunos algoritmos de ordenación agrupados por su rendimiento

Cuadráticos **$O(n^2)$**	**Sub-Cuadráticos** **$O(n^2) > T(n) > O(n)$**	**Lineales** **$O(n)$** *Asumen algunas características de los datos*
Burbuja Selección Inserción	*O(n log n)* - QuickSort - Montículo (HeapSort) - Mezcla (MergeSort) ShellSort $O(n^{3/2})$	Cubetas (BinSort) Residuos (RadixSort) *Estos no se basan en comparación*

Ordenación
Conceptos

Inversiones

Considere las siguientes listas de tamaño 20

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

¿Cuál es el grado de "desorden" de estas listas?

Ordenación
Conceptos

Definiendo *inversión*

La primera solo requiere unos pocos ajustes

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

1 12 16 25 26 33 35 42 45 56 58 67 74 75 81 83 86 88 95 99

Ordenación
Conceptos

Definiendo inversión

La segunda tiene dos entradas bien desordenadas

1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

1 17 21 23 24 27 32 35 42 45 47 57 66 69 70 76 85 87 95 99

sin embargo, muchas entradas (13) ya están ordenadas.

Ordenación
Conceptos

Definiendo inversión.

La tercera lista podemos decir que es significativamente desordenada.

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

1 10 12 16 20 22 26 31 38 44 48 79 80 81 84 87 92 95 96 99

Ordenación
Conceptos

Inversión

Dada una lista de *n* números, hay

$$\binom{n}{2} = \frac{n(n-1)}{2}$$ par de números

Por ejemplo, la lista (1, 3, 5, 4, 2, 6) contiene los siguientes 15 pares

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
	(3, 5)	(3, 4)	(3, 2)	(3, 6)
		(5, 4)	(5, 2)	(5, 6)
			(4, 2)	(4, 6)
				(2, 6)

Ordenación
Conceptos

Inversión

Los siguientes pares ya están ordenados

(1, 3)	**(1, 5)**	**(1, 4)**	**(1, 2)**	**(1, 6)**
(3, 5)	**(3, 4)**	(3, 2)	**(3, 6)**	
		(5, 4)	(5, 2)	**(5, 6)**
			(4, 2)	**(4, 6)**
				(2, 6)

Ordenación
Conceptos

Inversión

Los siguientes pares son inversiones

(1, 3)	(1, 5)	(1, 4)	(1, 2)	(1, 6)
(3, 5)	(3, 4)	**(3, 2)**	(3, 6)	
		(5, 4)	**(5, 2)**	(5, 6)
			(4, 2)	(4, 6)
				(2, 6)

Ordenación
Conceptos

- Considere un conjunto de elementos `S`
- Una permutación de los elementos de `S` es una reordenación de sus elementos
- Ej. dado `S = {0,1,2,3}`, las `4! = 24` permutaciones posibles son

0 1 2 3	0 1 3 2	0 2 1 3	0 2 3 1	0 3 1 2	0 3 2 1
1 0 2 3	1 0 3 2	1 2 0 3	1 2 3 0	0 3 1 2	0 3 2 1
2 0 1 3	2 0 3 1	2 1 0 3	2 1 3 0	2 3 0 1	2 3 1 0
3 0 1 2	3 0 2 1	3 1 0 2	3 1 2 0	3 2 0 1	3 2 1 0

Ordenación
Conceptos

Definición formal de Inversión

Dada una permutación de ***n*** elementos

$p_0, p_1, p_2, .. , p_{n-1}$

una inversión es definida como un par de entradas que están intercambiadas

Esto es, (p_i, p_j) forma una inversión si:

$\boldsymbol{i < j}$ pero $\boldsymbol{p_i > p_j}$

Ordenación
Conceptos

Así por ejemplo, la permutación

1, 3, 5, 4, 2, 6

contiene cuatro inversiones

(3,2), (5,4), (5,2), (4,2)

Intercambiando dos entradas se puede:

- Remover una inversión ó
- Introducir una nueva inversión

Ordenación
Conceptos

Número de inversiones

- Hay $\binom{n}{2} = \frac{n(n-1)}{2}$ par de números en cualquier conjunto de *n* objetos (recordar combinación de n elementos en grupos de 2 elementos) $C_2^n = \frac{n!}{2!(n-2)!}$
- Consecuentemente cada par puede:
  - Ser parte del conjunto ordenado de pares
  - Ser parte del conjunto de inversiones.
- Para una ordenación aleatoria, se espera aproximadamente que la mitad de todos los pares sean inversiones $\frac{1}{2}\binom{n}{2} = \frac{n(n-1)}{4} = \mathbf{O}(n^2)$

Ordenación
Conceptos

Por ejemplo la siguiente lista de 56 elementos

261 548 3 923 195 973 289 237 57 299 594 928 515 55

507 351 262 797 788 442 97 798 227 127 474 825 7 182

929 852 504 485 45 98 538 476 175 374 523 800 19 901

349 947 613 265 844 811 636 859 81 270 697 563 976 539

tiene 655 inversiones y 885 pares ordenados. La fórmula predice 770 inversiones.

¿Es una lista relativamente aleatorizada?

Ordenación
Conceptos

Considere estas listas de 20 elementos cada una:

1 16 12 26 25 35 33 58 45 42 56 67 83 75 74 86 81 88 99 95

1 17 21 42 24 27 32 35 45 47 57 23 66 69 70 76 87 85 95 99

22 20 81 38 95 84 99 12 79 44 26 87 96 10 48 80 1 31 16 92

Hay 190 pares y se espera 95 inversiones

- La primera tiene 13 inversiones
- La segunda tiene 13 inversiones
- La tercera tiene 100 inversiones

¿Cual es la mejor "aleatorizada"?

Ordenación
Conceptos

- El número de inversiones brinda la idea del trabajo máximo que el algoritmo de ordenación debe hacer.

- Así... podemos tener una cota máxima para un algoritmo de ordenación basado en comparación..

  es decir, $O(n^2)$

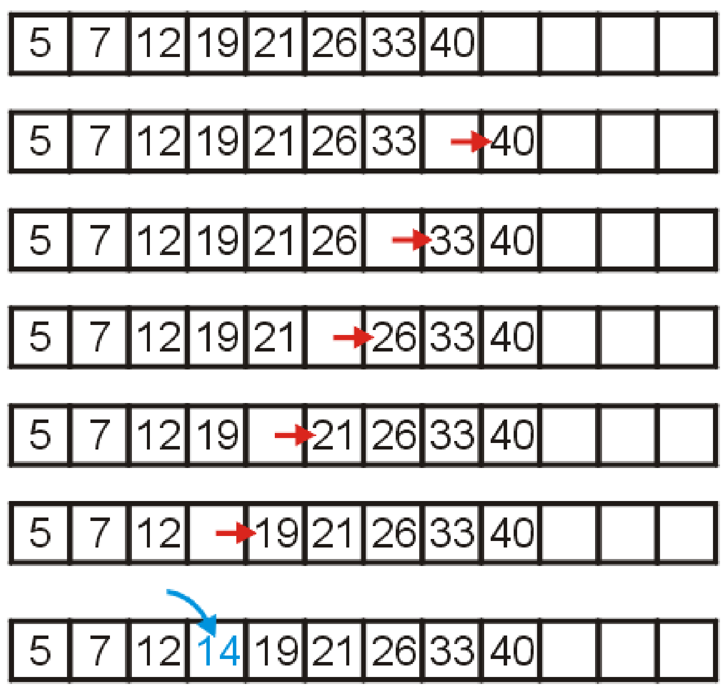
Ordenación
InsertSort

- El algoritmo:
  - Para cualquier lista desordenada
    - Trata el primer elemento como una lista ordenada de tamaño k=1
  - Entonces, dada una lista de tamaño k
    - Insertar el (k+1)ésimo ítem en la lista desordenada dentro de la ordenada.
    - La lista ordenada es ahora de tamaño k+1

Ordenación
InsertSort

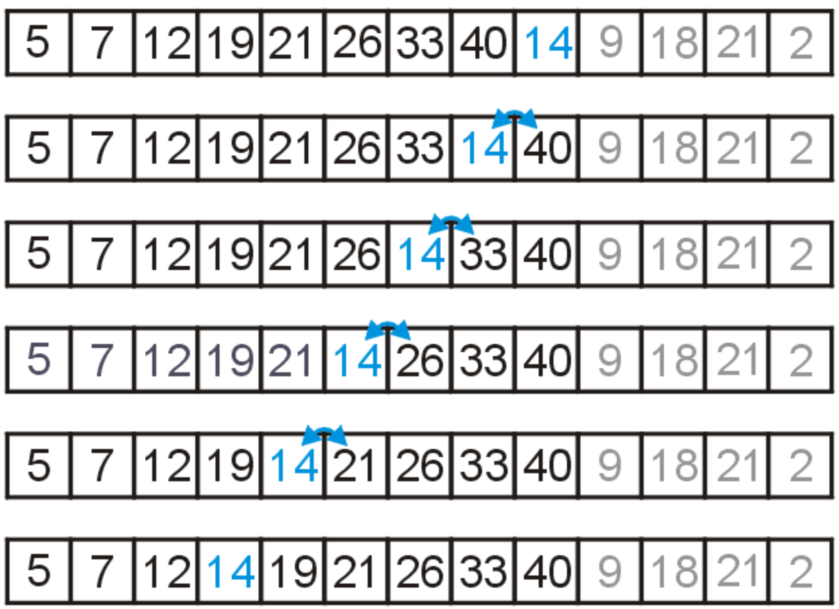
- Ejemplo, suponga que queremos insertar 14 en la siguiente lista ya ordenada (k=8)

5	7	12	19	21	26	33	40				

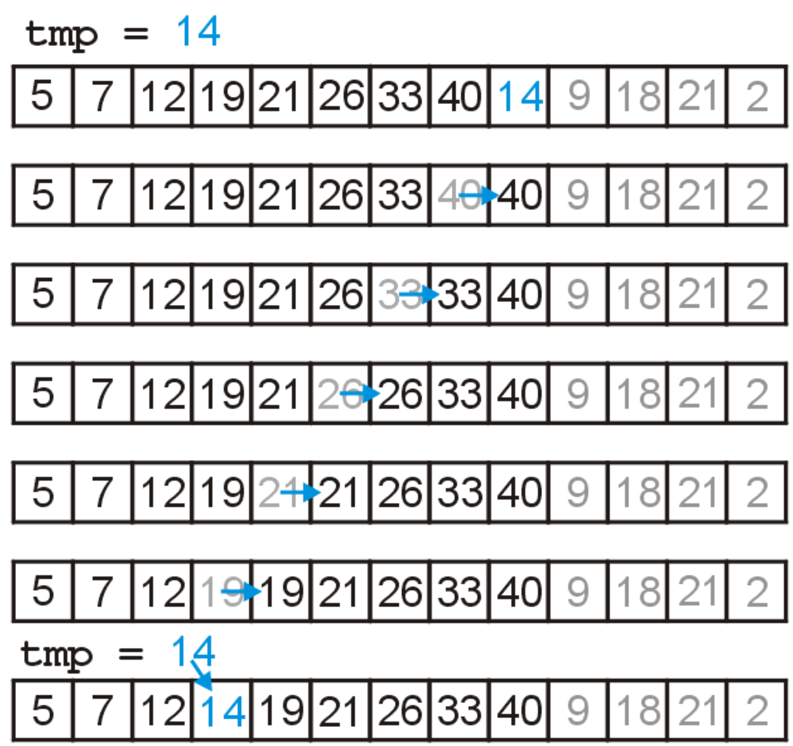


Ordenación
InsertSort

Usando intercambios en cada paso



Usando una variable temporal



Ordenación
InsertSort

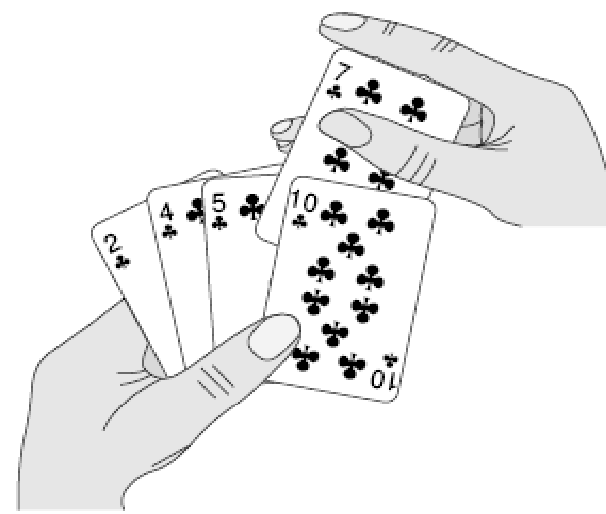
```
int key, i;

for ( int j=1; j < A.length; j++ ) {

   key = A[j];
   i   = j-1 ;

   while ( i >= 0 && A[i] > key ) {
      A[i+1] = A[i];
      i = i - 1 ;
   }

   A[i+1] = key;
}
```



Ordenación
InsertSort

Resumen de tiempos de ejecución

Caso	Tiempo	Comentario
Peor	$\Theta(n^2)$	Orden inverso
Medio	$\mathbf{O}(d+n)$	$d = \mathrm{O}(n^2)$ (d = *inversiones*)
Mejor	$\Theta(n)$	Muy pocas inversiones (por ejemplo ya esta ordenado)

Ordenación
InsertSort

- Ventajas
  - Simple de implementar
  - Para tamaños pequeños es bastante rápido, en promedio ***(¿porqué?)***
- Desventajas
  - Para tamaños grandes es bastante ineficiente en comparación con los algoritmos subcuadráticos

Ordenación
ShellSort

- Propuesta por Donald Shell en 1959
- La idea: *incrementos decrecientes* evitando gran cantidad de movimientos.
- Es básicamente la misma idea que InsertSort
- Uno puede utilizar los incrementos que le parezca con la **condición** de que el incremento final sea 1 (como en InsertSort).
- Tiempo de ejecución sub-cuadrático ($O(n^{3/2})$).
  Algunas cotas son mejores con incrementos diferentes

Ordenación
ShellSort

```
public void shellsort ( int [] a ) {
  for ( int gap = a.length / 2; gap > 0 ;
        gap = gap == 2 ? 1 : (int) (gap/2.2) )

    for (int i = gap; i < a.length; i++ )
    {
        int key = a[i];
        int j   = i;

        while ( j >= gap && key < a[j-gap] ) {
            a[j] = a[j-gap];
            j    = j- gap;
        }
        a[j] = key;
    }
}
```

InsertSort

Ordenación
ShellSort

Secuencia de decrementos producida por el algoritmo mostrado en la lámina anterior.

Ejemplo *n = 1000000*

```
500000
227272
103305
46956
21343
9701
4409
2004
910
413
187
85
38
17
7
3
1
```

Ordenación
Ejemplo de ShellSort

Ordenar : *18, 32, 12, 5, 38, 33, 16, 2*

Por simplicidad usamos incremento de n/2

Incremento 4:

1				2				3				4

18	32	12	5	38	33	16	2

Paso 1) solo miramos el 18 y 38, no hay intercambio

Paso 2) solo vemos el 32 y 33, no hay intercambio

Paso 3) solo observamos el 12 y 16, no hay intercambio

Paso 4) vemos el **5** y **2**, y existe intercambio.

Ordenación
Ejemplo de ShellSort

Resultado con intercambio de 4: *18, 32, 12,* ***2****, 38, 33, 16,* ***5***

Incremento 2:

1	**2**						
18	**32**	**12**	**2**	**38**	**33**	**16**	**5**

Paso 1) mirar el **18, 12, 38, 16** , hay intercambios
Resultado: **12**, 32, **16**, 2, **18**, 33, **38**, 5

Paso 2) mirar el **32, 2, 33, 5**, hay intercambios
Resultado: 12, **2**, 16, **5**, 18, **32**, 38, **33**

Ordenación
Ejemplo de ShellSort

Resultado con intercambio de 2: *12*, ***2***, *16*, ***5***, *18*, ***32***, *38*, ***33***

Incremento 1:

1

12 2 16 5 18 32 38 33

Paso único) mirar todos los elementos (InsertSort normal)

Resultado final : *2, 5, 12, 16, 18, 32, 33, 38*

Ordenación

QuickSort

- Inventado por C.A. Hoare en 1962
- Su tiempo de ejecución en promedio es O($n \log n$)
- Su peor caso esta en $O(n^2)$, aunque con algunas modificaciones puede evitarse.

Ordenación

QuickSort

- Por ejemplo, dada la siguiente lista

  80 38 95 84 99 10 79 44 26 87 96 12 43 81 3

- Podemos seleccionar el elemento central, 44, y ordenar el resto en dos grupos, los menores a 44 y los mayores a 44.

  38 10 26 12 43 3   44   80 95 84 99 79 87 96 81

- Si nosotros cada sublista, la volvemos a ordenar entonces podemos tener ordenado todo el arreglo.

Ordenación
QuickSort

- Uno de los principales puntos en este algoritmo: *selección del pivot*

Mostramos la mediana de tres

80	38	95	84	99	10	79	44	26	87	96	12	43	81	3

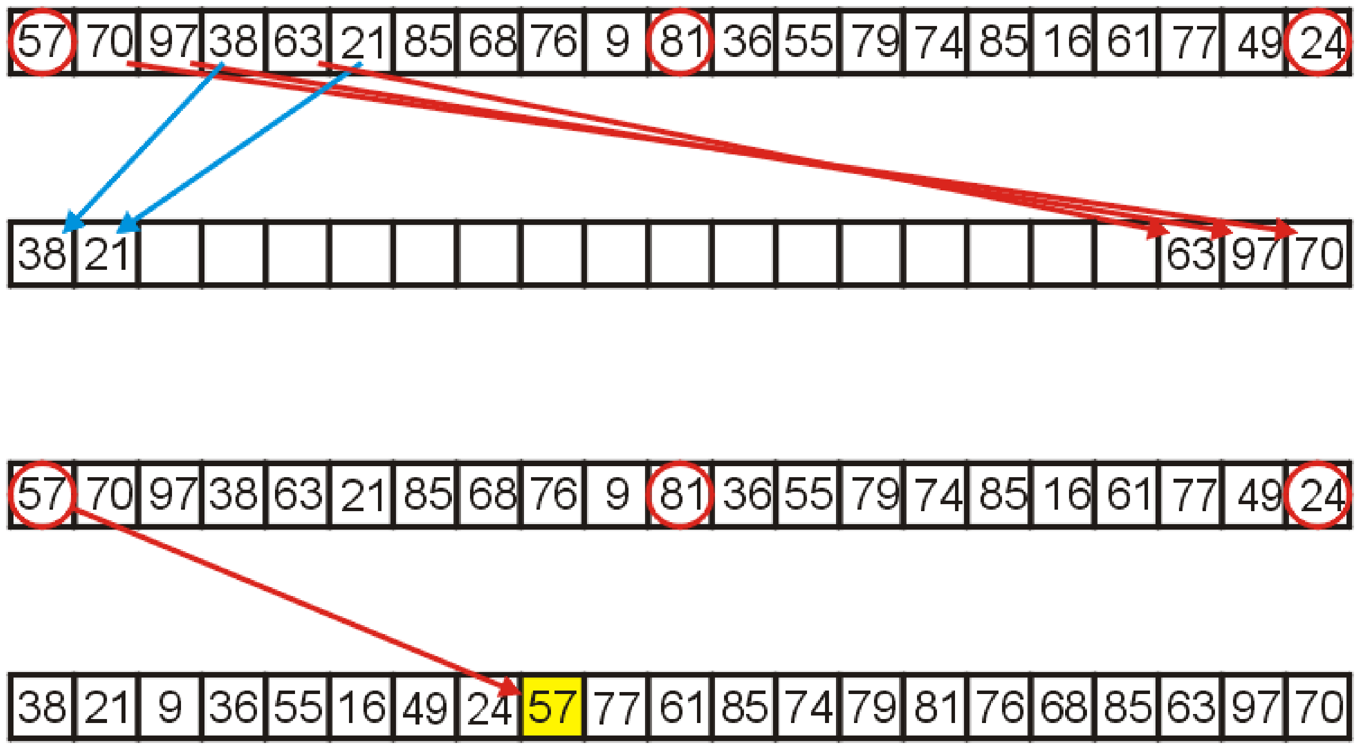
38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

38	10	26	12	43	3	44	80	95	84	99	79	87	96	81

Ordenación
QuickSort

- Ejemplo (pivot = 57) (con auxiliar)



Ordenación
QuickSort

- Ejemplo completo

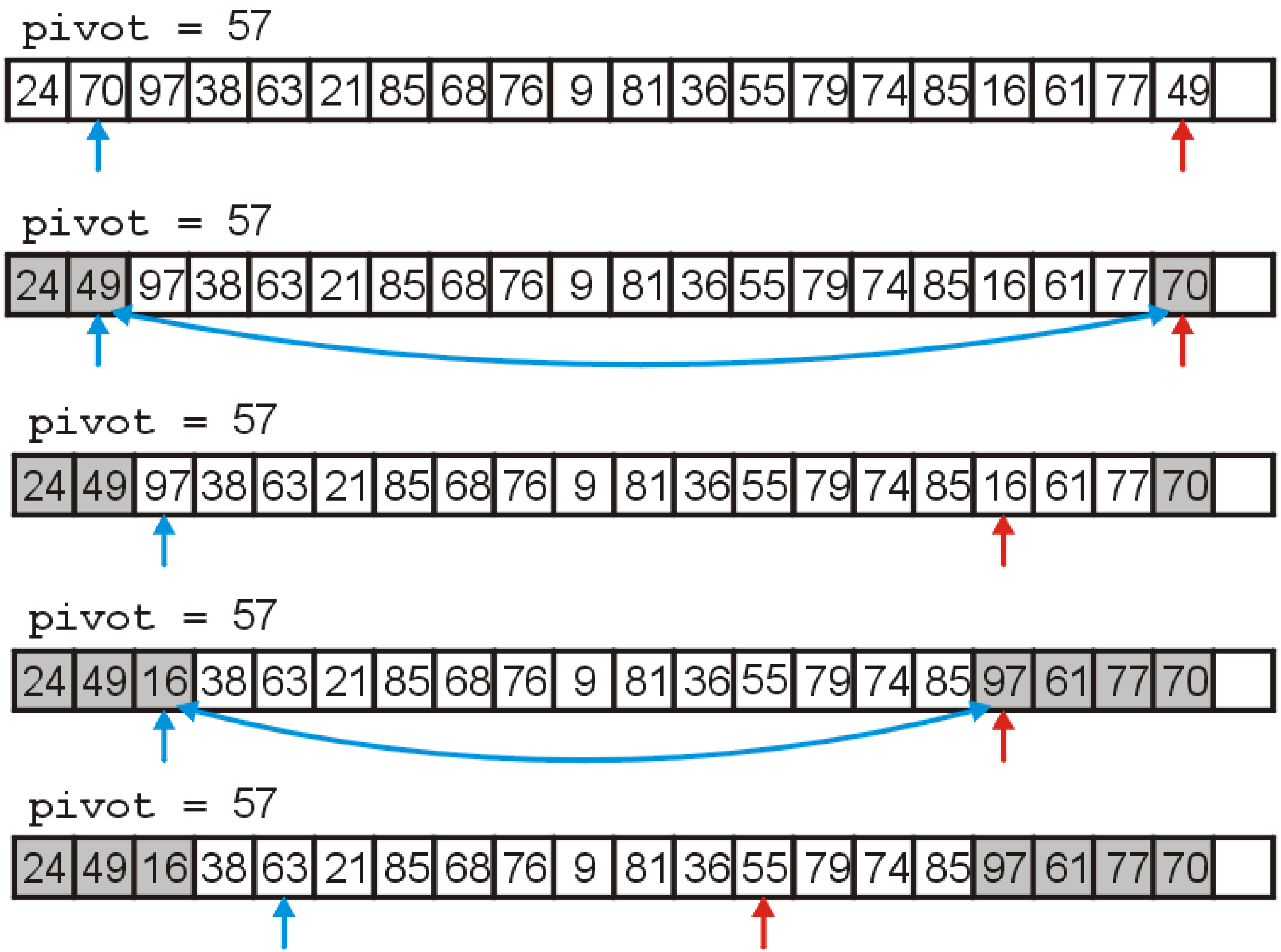
`pivot =`

57	70	97	38	63	21	85	68	76	9	81	36	55	79	74	85	16	61	77	49	24

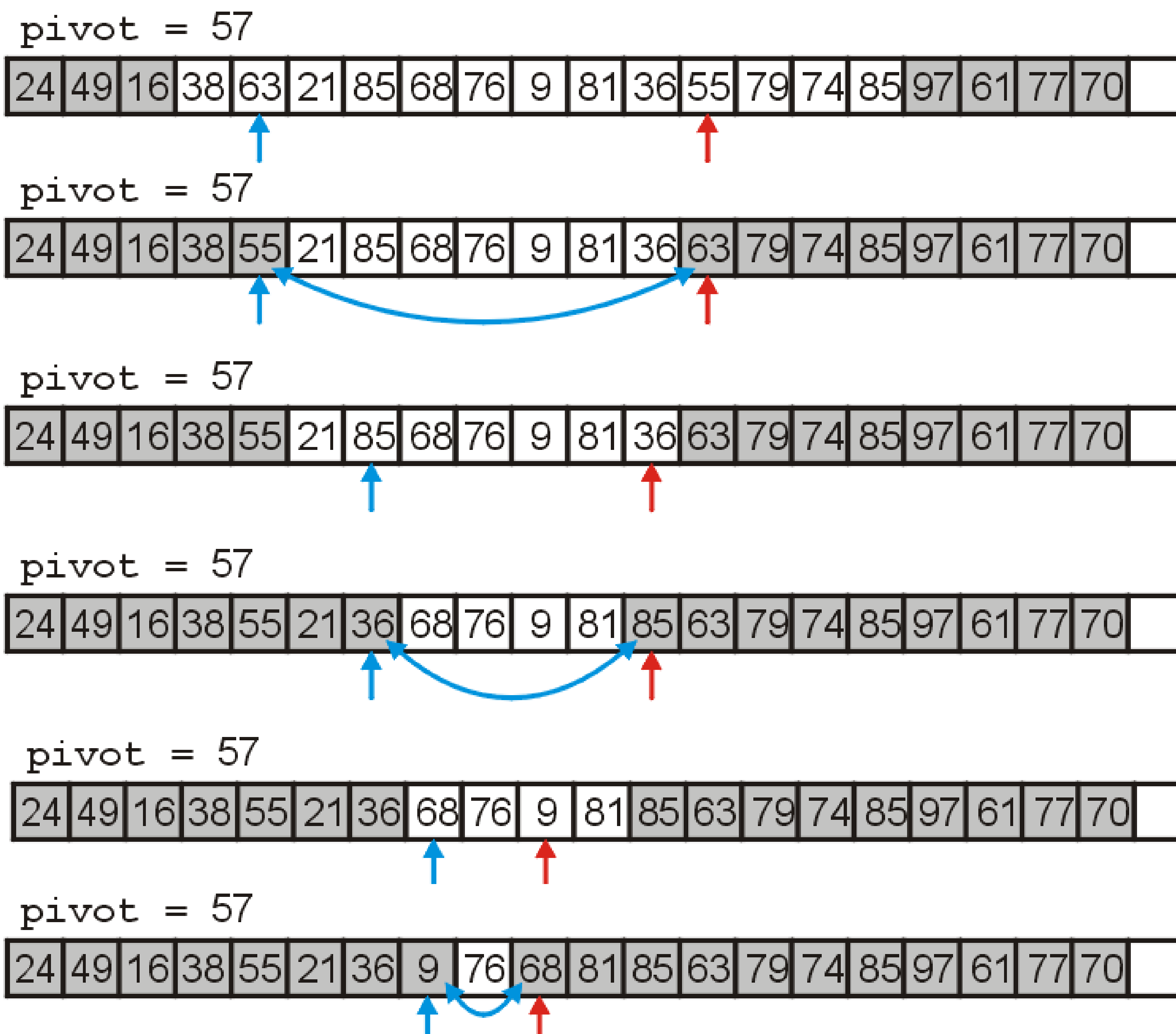
`pivot =` 57

24	70	97	38	63	21	85	68	76	9	81	36	55	79	74	85	16	61	77	49	

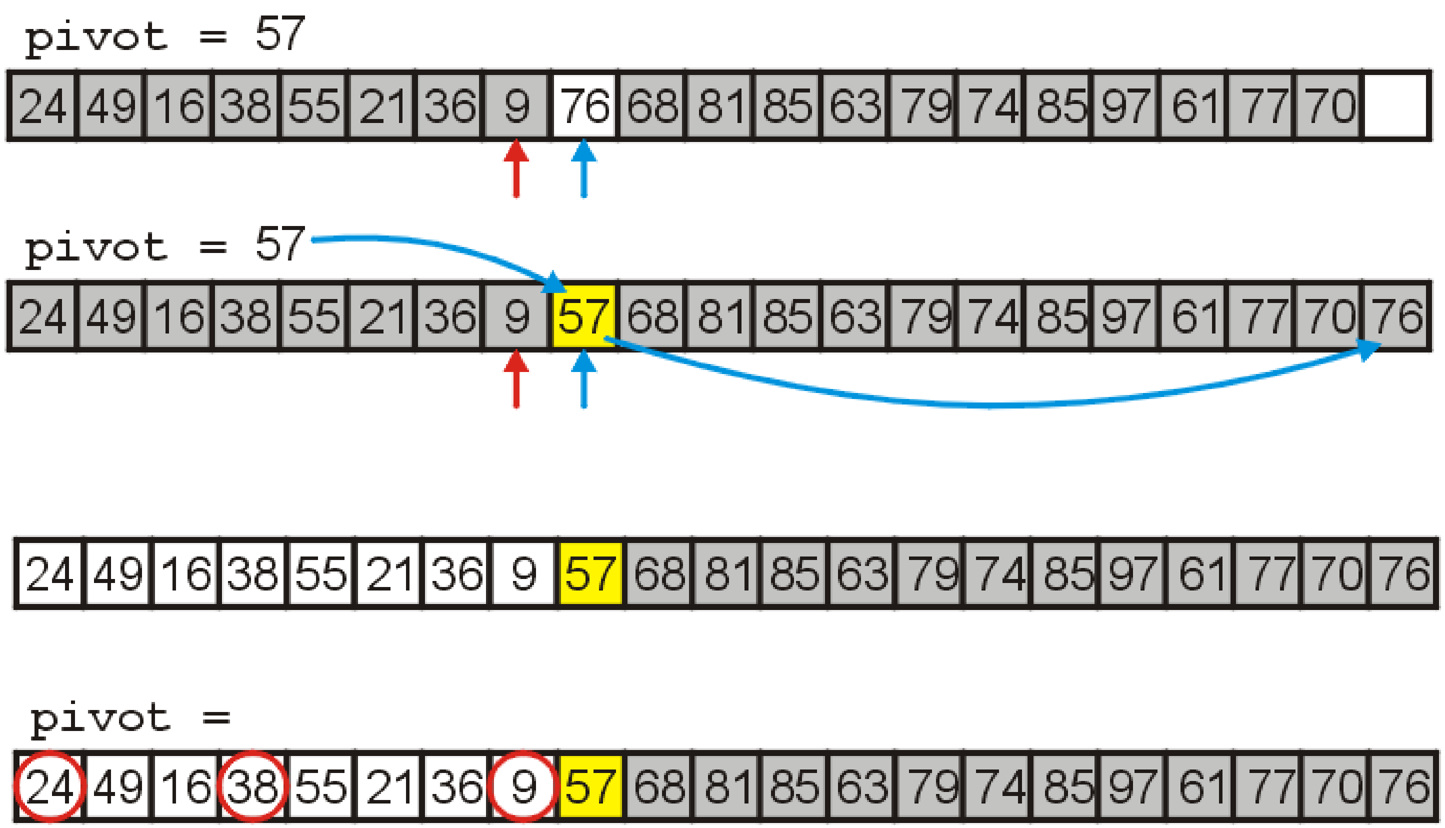
Ordenación
QuickSort



Ordenación
QuickSort



Ordenación
QuickSort



Ordenación
QuickSort
Implementación simple

```
QUICKSORT(A, p, r )
if p < r
   then q ← PARTITION(A, p, r )
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r )

Llamada inicial : QUICKSORT(A, 1, n).
```

a) Pruebe el algoritmo con la secuencia.
{4,8,10,9,6,2,1,7}
De qué lado quedan los menores y de que lado los mayores? Muestre el resultado luego de la primera llamada a PARTITION

b) Determine el tiempo de este algoritmo, Indique su ecuación de recurrencia y Luego calcule su cota

```
PARTITION(A, p, r )
  x ← A[r ] // estrategia muy simple, no usa mediana de tres, toma el último
  i ← p − 1
  for j ← p to r − 1
    do if A[ j ] ≤ x
       then i ← i + 1
            intercambiar A[i ] ↔ A[ j ]
  intercambiar A[i + 1] ↔ A[r ]

return i + 1
```

Ordenación
QuickSort

Peor caso

- Ocurre cuando el arreglo esta totalmente desbalanceado
- Se tiene 0 elementos en un subarreglo y n-1 en otro.

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Theta(n^2)
\end{aligned}
$$

Ordenación
QuickSort

Mejor caso

- Ocurre cuando el arreglo esta totalmente balanceado
- Se tiene n/2 elementos en cada subarreglo

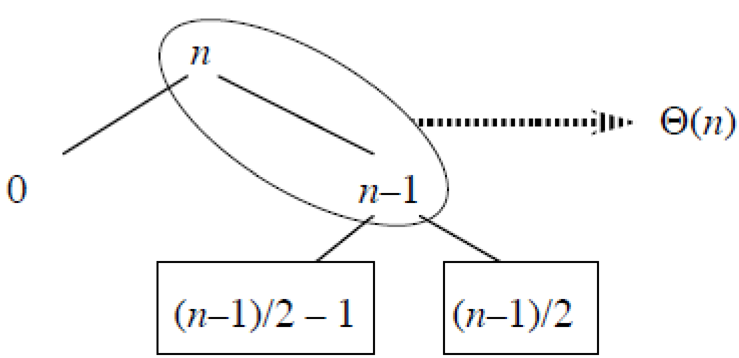
$$
\begin{aligned}
T(n) &= 2T(n/2) + \Theta(n) \\
&= \Theta(n \lg n)
\end{aligned}
$$

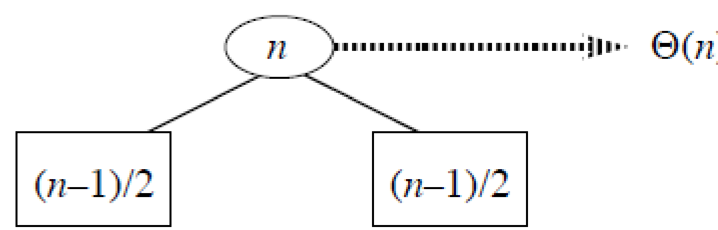
Ordenación
Quicksort

El caso promedio esta más relacionado al mejor caso que al peor caso.

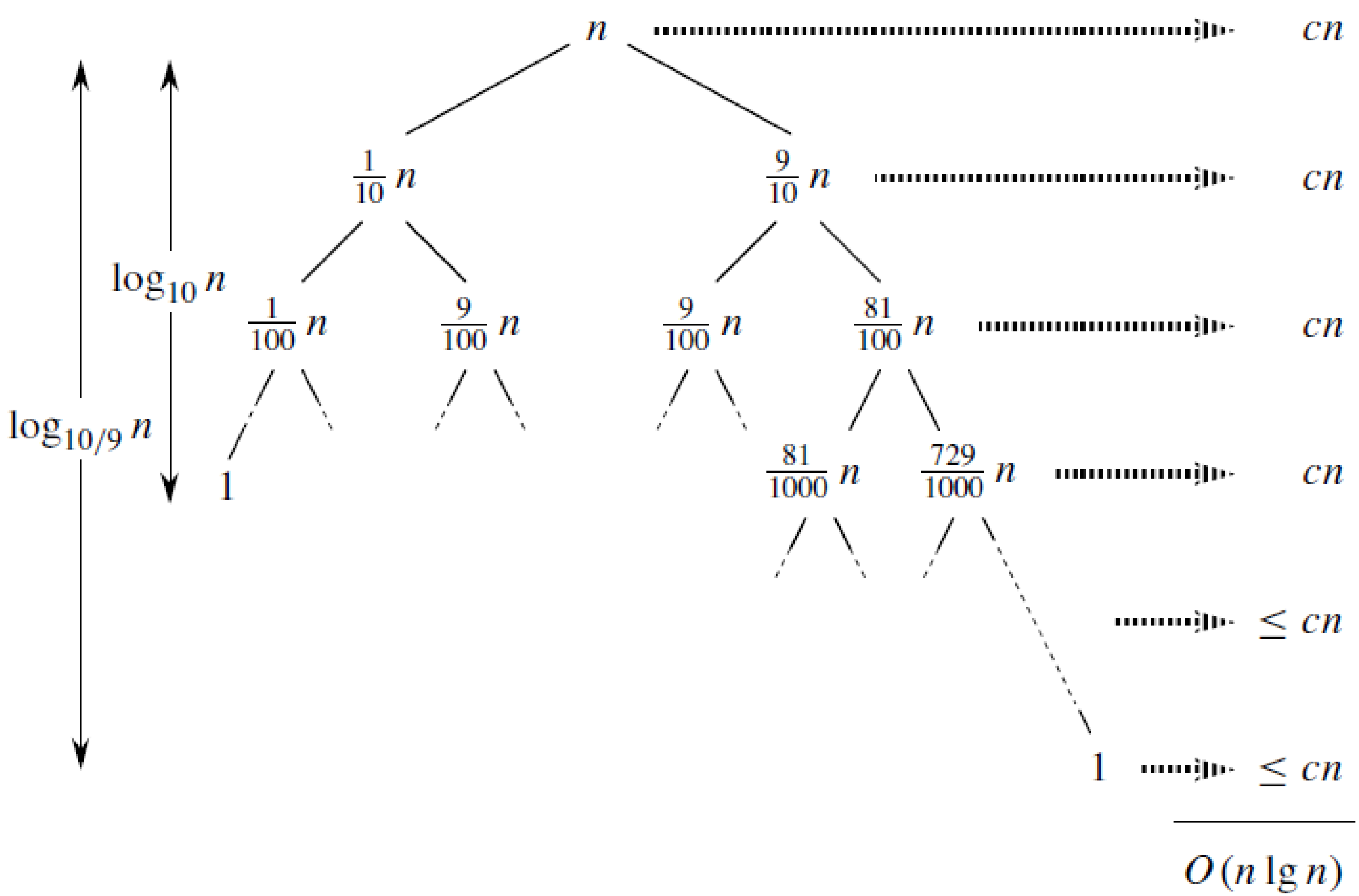
Imaginar que la partición produzca arreglos en relación 9-1

$$
\begin{aligned}
T(n) &= T(9n/10) + T(n/10) + \Theta(n) \\
&= \Theta(n \lg n)
\end{aligned}
$$





Ordenación
Quicksort



Ordenación
QuickSort

Caso	Tiempo de ejecución	Comentarios
Peor	$\mathbf{O}(n^2)$	Mala selección de pivot o entrada con sesgo
Promedio	$\mathbf{O}(n \lg(n))$	
Mejor	$\mathbf{O}(n \lg(n))$	Es la misma cota que el caso promedio

Ordenación
QuickSort

DEMO de Código

A continuación y en la próxima clase

- Demostración de cota mínima para algoritmos de ordenación por comparación
- MergeSort
- HeapSort
- CountingSort
- RadixSort
- BucketSort