Fundamentos de análisis y diseño de algoritmos

Ordenamiento en tiempo lineal

Ordenamiento en tiempo lineal

Counting sort

Radix sort

Bucket sort

Ordenamiento en tiempo lineal

Insertionsort, MergeSort, Heapsort,y Quicksort son algoritmos por comparaciones

La cota mínima de cualquier algoritmo por comparaciones es $\Omega(n\lg n)$

No es posible bajar esa cota si se utilizann comparaciones

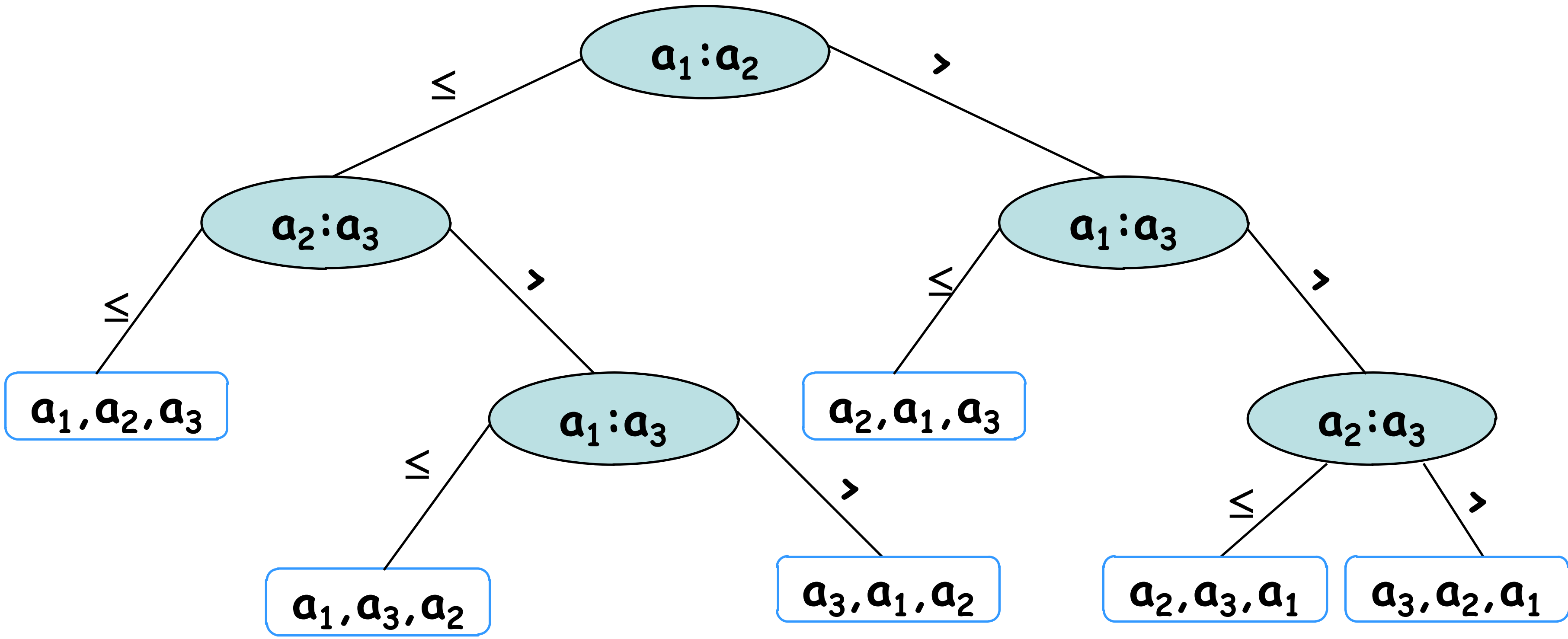
Ordenamiento en tiempo lineal

Otras estrategias:

- Counting sort
- Radix sort
- Bucket sort

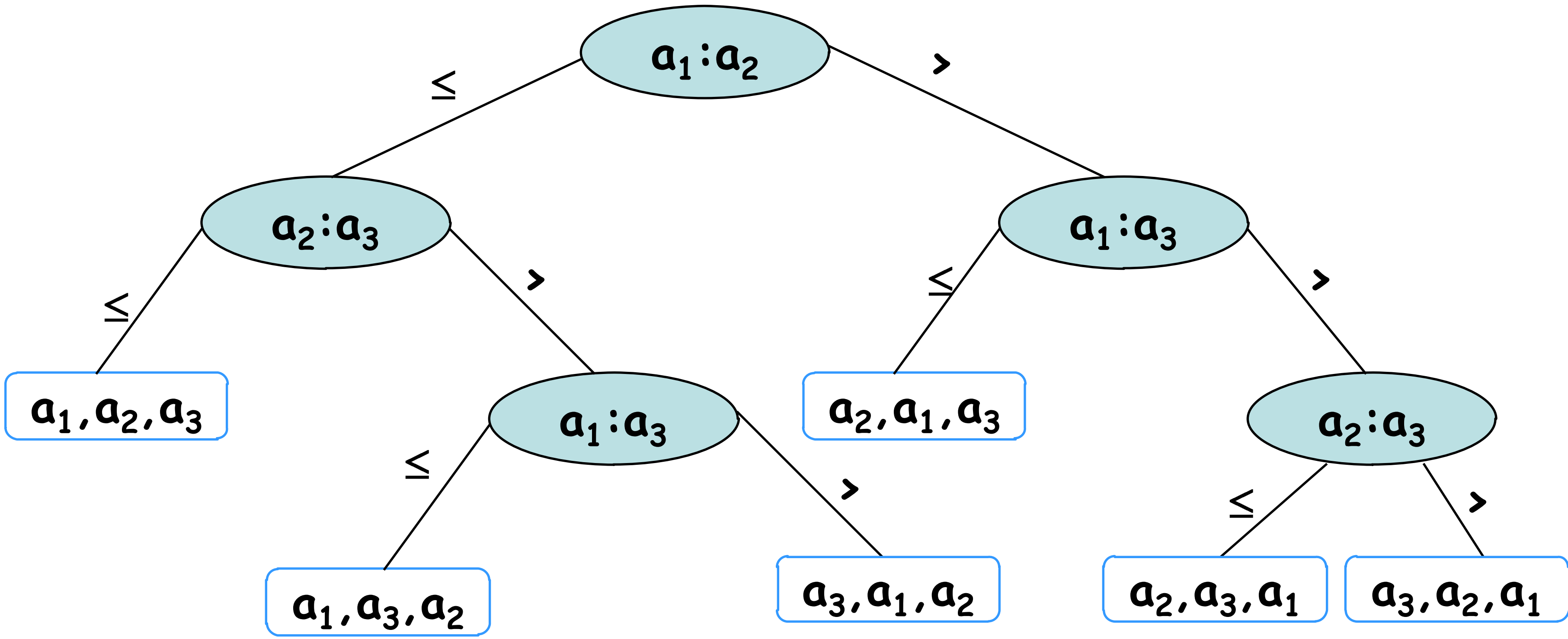
Ordenamiento en tiempo lineal

Cota inferior para ordenar por comparaciones



Ordenamiento en tiempo lineal

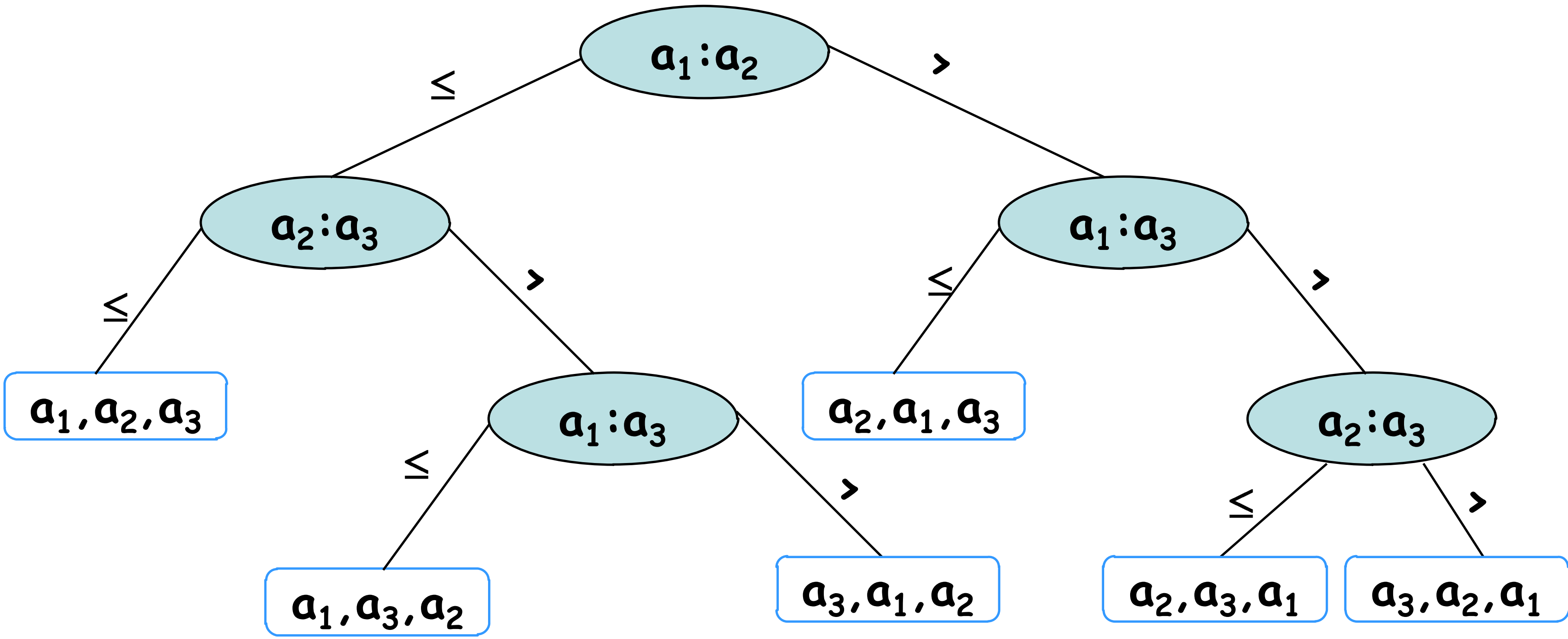
Cota inferior para ordenar por comparaciones



Para ordenar 3 elementos, se tienen 3! posibles caminos

Ordenamiento en tiempo lineal

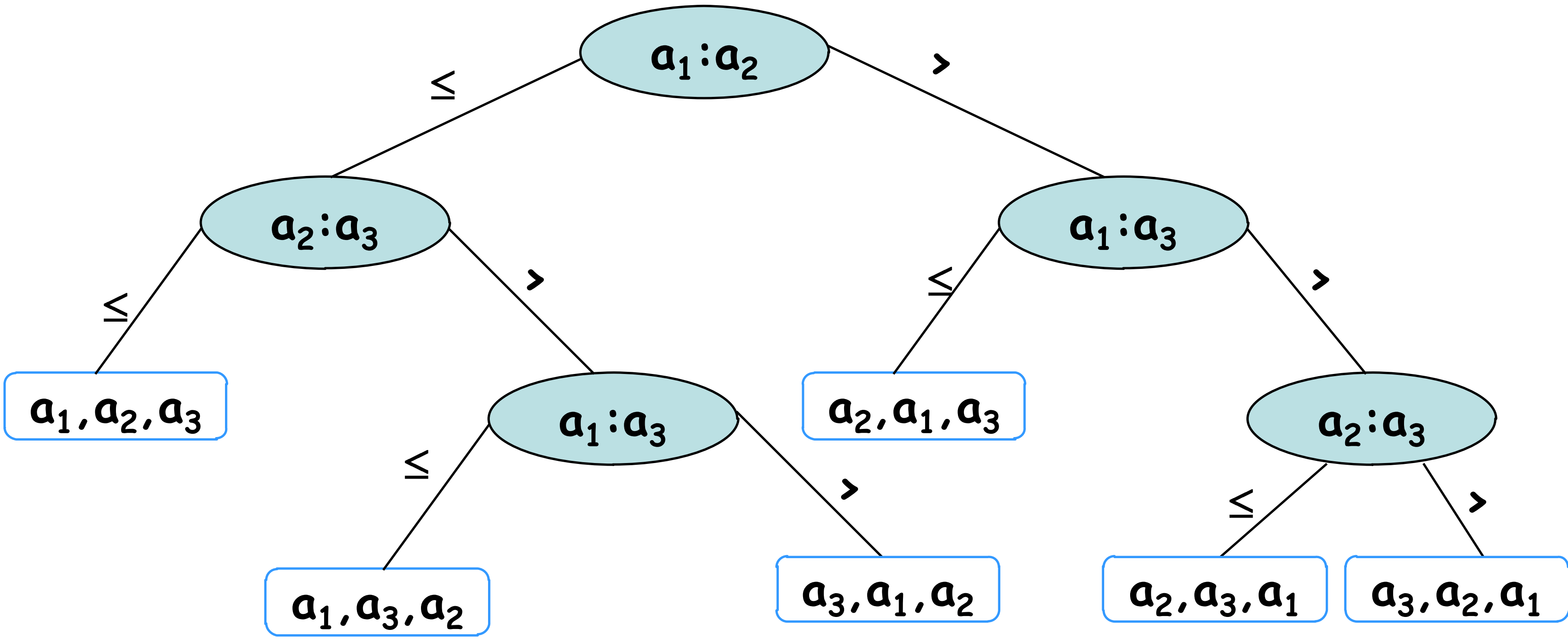
Cota inferior para ordenar por comparaciones



La ejecución de un algoritmo de ordenamiento corresponde a encontrar un camino desde la raiz hasta una hoja

Ordenamiento en tiempo lineal

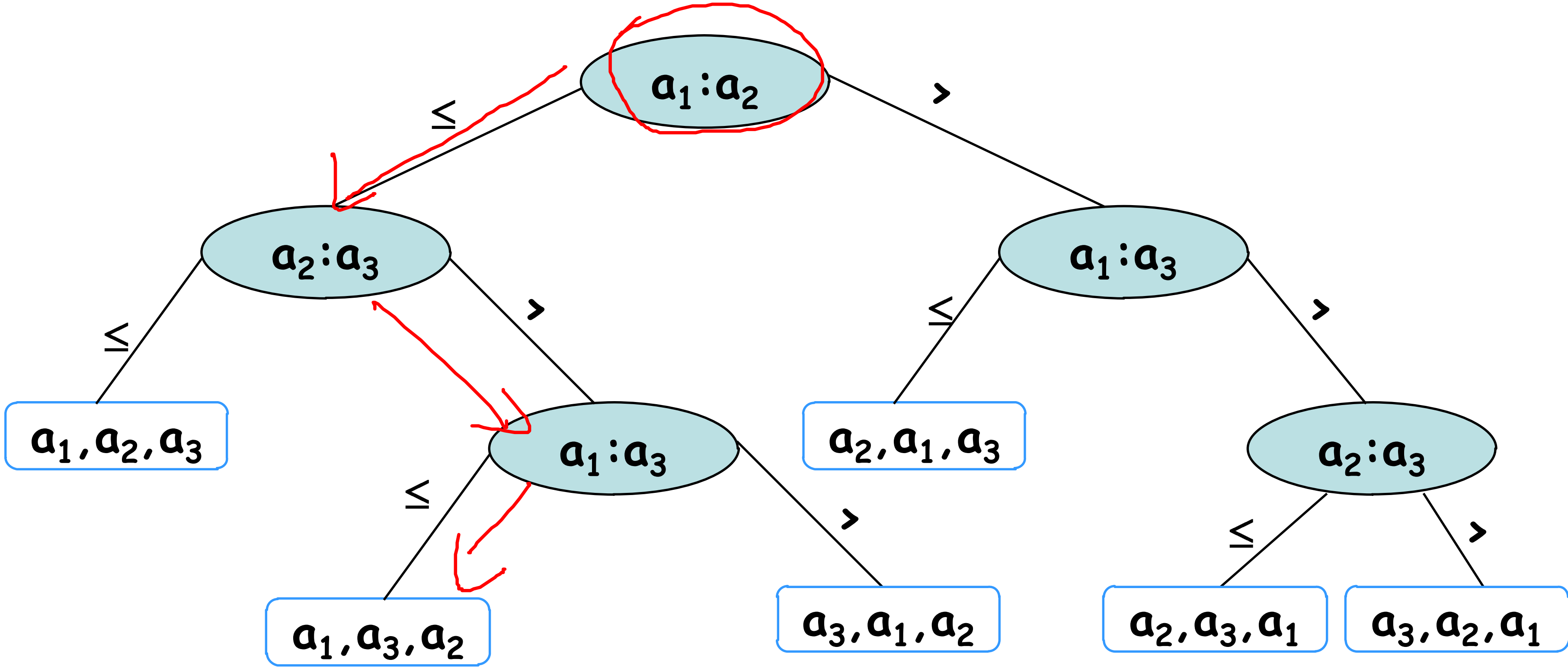
Cota inferior para ordenar por comparaciones



¿Cuántas comparaciones se deben realizar? depende de la altura del árbol

Ordenamiento en tiempo lineal

Cota inferior para ordenar por comparaciones



¿Cuál es el peor caso para un algoritmo de ordenamiento?
Está representado por el camino más larga de raíz a hoja

Ordenamiento en tiempo lineal

Teorema: Cualquier árbol de decisión que ordene n elementos tiene altura $\Omega(n\lg n)$

n! permutaciones → n! hojas

Además, un árbol binario de altura h no tiene más de 2^h hojas

$n! \leq 2^h$

$h \geq \lg(n!)$, se conoce que $n! > (n/e)^n$

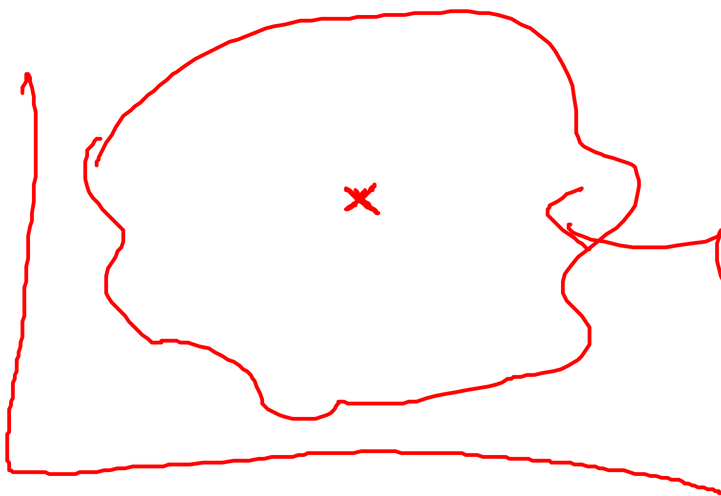
$h \geq \lg (n/e)^n$

$= n\lg n - n\lg e$

$= \Omega(n\lg n)$

$(a_1, a_2, \ldots, a_n)$

$(n)(n-1)(n-2)\ldots(1) = n!$



$O(n!) \sim NP$

Merge Quick

Insert

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

P

Ordenamiento en tiempo lineal

Counting sort

Asume que cada uno de los n elementos a ordenar es un número entero entre 1 y k. $\boxed{k=O(n)}$

Idea: contar, para cada elemento x, los elementos que son menores que x

Por ejemplo, si para x se tiene que el conteo es 0, significa que no hay elementos menores que x, por lo que x es el menor

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

- A[1..n]: datos de entrada
- B[1..n]: mantiene la salida — ordenado
- C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

¿Cuánto vale k?

$k = 6$

8

6

k es $O(n)$

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

¿Cuánto vale k?

Los números están entre 1 y 6. por lo que k=6

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B								

	1	2	3	4	5	6
C						

1 . . . K

¿Cuánto vale k?

Los números están entre 1 y 6. por lo que k=6

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B								

	1	2	3	4	5	6
C	2	0	2	3	0	1

Se completa C contando las veces que aparece cada i de C en A

C indica la cantidad de números iguales a i

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B								

	1	2	3	4	5	6
C	2	0	2	3	0	1

Se busca ahora que C indica la cantidad de números menores o iguales a i

for i←2 to k

do C[i] ← C[i]+C[i-1]

1	2	3			
2	2	4	7	7	8

Ordenamiento en tiempo lineal

Counting sort

Se utilizan 3 arreglos:

A[1..n]: datos de entrada

B[1..n]: mantiene la salida

C[1..k]: mantiene los conteos

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B								

	1	2	3	4	5	6
C	2	2	4	7	7	8

Se busca ahora que C indica la cantidad de números <u>menores o iguales a i</u>

C[1]=2 indica que hay 2 números menores o iguales a 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1					4	

	1	2	3	4	5	6
C	2	2	4	7	7	8

Considere A[8], cuántos elementos son menores o iguales a 4?

C[A[8]]=7 $\quad c[4]=7$

$-C[A[7]] = C[1] = 2$

Se coloca el número A[8]=4 en la séptima posición de B

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
						4	

C

1	2	3	4	5	6
2	2	4	6	7	8

6

Considere A[8], cuántos elementos son menores o iguales a 4?

C[A[8]]=7

Se coloca el número A[8]=4 en la séptima posición de B

Se disminuye el C[A[8]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B							4	

	1	2	3	4	5	6
C	2	2	4	6	7	8

Considere A[7], cuántos elementos son menores o iguales a 1?

C[A[7]]=2

Se coloca el número A[7]=1 en la posición 2 de B

Se disminuye el C[A[7]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1					4	

	1	2	3	4	5	6
C	1	2	4	6	7	8

Considere A[7], cuántos elementos son menores o iguales a 1?

C[A[7]]=2

Se coloca el número A[7]=1 en la posición 2 de B

Se disminuye el C[A[7]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1				4	4	

	1	2	3	4	5	6
C	1	2	4	6	7	8

5

Considere A[6], qué cambios ocurren en los arreglos?

C[A[6]]

C[4] = 6

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1					4	

	1	2	3	4	5	6
C	1	2	4	6	7	8

Considere A[6], cuántos elementos son menores o iguales a 4?

C[A[6]]=6

Se coloca el número A[6]=4 en la posición 6 B

Se disminuye el C[A[6]] en 1

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
	1				4	4	

C

1	2	3	4	5	6
1	2	4	5	7	8

Considere A[6], cuántos elementos son menores o iguales a 4?

C[A[6]]=6

Se coloca el número A[6]=4 en la posición 6 de B

Se disminuye el C[A[6]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B	1	1	3	3	4	4	4	6

	1	2	3	4	5	6
C	~~1~~	2	~~4~~	~~5~~	7	~~8~~
	0		~~3~~ 2	4		7

Considere A[5], cuántos elementos son menores o iguales a 3?

C[A[5]]=4

Se coloca el número A[5]=3 en la posición 4 de B

Se disminuye el C[A[5]] en 1

1) Encontrar el k O(n)

2) Llenar el arreglo C, que contiene el conteo de los elementos que son iguales a la posición i

3) Iterar c, de tal forma que en la posición i que el conteo de los que son menores o iguales i

4) Comenzar a iterar desde la ultima posición de A hasta la primera, llenado el arreglo B

O(n)
O(k)
O(n)
O(n)

O(n)

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1		3		4	4	

	1	2	3	4	5	6
C	1	2	3	5	7	8

Considere A[5], cuántos elementos son menores o iguales a 3?

C[A[5]]=4

Se coloca el número A[5]=3 en la posición 4 de B

Se disminuye el C[A[5]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B		1		3		4	4	

	1	2	3	4	5	6
C	1	2	3	5	7	8

Considere A[4], cuántos elementos son menores o iguales a 1?

C[A[4]]=1

Se coloca el número A[4]=1 en la posición 1 de B

Se disminuye el C[A[4]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B	1	1		3		4	4	

	1	2	3	4	5	6
C	0	2	3	5	7	8

Considere A[4], cuántos elementos son menores o iguales a 1?

C[A[4]]=1

Se coloca el número A[4]=1 en la posición 1 de B

Se disminuye el C[A[4]] en 1

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
1	1		3		4	4	

C

1	2	3	4	5	6
0	2	3	5	7	8

Considere A[3], cuántos elementos son menores o iguales a 4?

C[A[3]]=5

Se coloca el número A[3]=4 en la posición 5 de B

Se disminuye el C[A[3]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B	1	1		3	4	4	4	

	1	2	3	4	5	6
C	0	2	3	4	7	8

Considere A[3], cuántos elementos son menores o iguales a 4?

C[A[3]]=5

Se coloca el número A[3]=4 en la posición 5 de B

Se disminuye el C[A[3]] en 1

Ordenamiento en tiempo lineal

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B	1	1		3	4	4	4	

	1	2	3	4	5	6
C	0	2	3	4	7	8

Considere A[2], cuántos elementos son menores o iguales a 6?

C[A[2]]=8

Se coloca el número A[2]=6 en la posición 8 de B

Se disminuye el C[A[2]] en 1

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
1	1		3	4	4	4	6

C

1	2	3	4	5	6
0	2	3	4	7	7

Considere A[2], cuántos elementos son menores o iguales a 6?

C[A[2]]=8

Se coloca el número A[2]=6 en la posición 8 de B

Se disminuye el C[A[2]] en 1

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
1	1		3	4	4	4	6

C

1	2	3	4	5	6
0	2	3	4	7	7

Considere A[1], cuántos elementos son menores o iguales a 3?

C[A[1]]=3

Se coloca el número A[1]=3 en la posición 3 de B

Se disminuye el C[A[1]] en 1

Ordenamiento en tiempo lineal

A

1	2	3	4	5	6	7	8
3	6	4	1	3	4	1	4

B

1	2	3	4	5	6	7	8
1	1	3	3	4	4	4	6

C

1	2	3	4	5	6
0	2	2	4	7	7

Considere A[1], cuántos elementos son menores o iguales a 3?

C[A[1]]=3

Se coloca el número A[1]=3 en la posición 3 de B

Se disminuye el C[A[1]] en 1

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
     do C[i]←0
  for j←1 to length[A]
     do C[A[j]]←C[A[j]]+1
  for j←2 to k
     do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
     do B[C[A[j]]]←A[j]
        C[A[j]]←C[A[j]]-1
```

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
     do C[i]←0
  for j←1 to length[A]
     do C[A[j]]←C[A[j]]+1
  for j←2 to k
     do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
     do B[C[A[j]]]←A[j]
         C[A[j]]←C[A[j]]-1
```

$F(n) \leq C g(n)$

Aplique el algoritmo

Counting sort

$K=5$

A

3	2	1	5

B

1	2	3	5

C

1	2	3	4	5
1	1	1	0	1

0

~~1~~ 0	~~2~~ 1	~~3~~ 2	3	~~4~~ 3

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
    do C[i]←0
  for j←1 to length[A]
    do C[A[j]]←C[A[j]]+1
  for j←2 to k
    do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
    do B[C[A[j]]]←A[j]
       C[A[j]]←C[A[j]]-1
```

Aplique el algoritmo

Counting sort

K = 9

A

9	1	1	4	7

B

1	1	4	7	9

C

1								9
2	0	0	1	0	0	1	0	1

1	2	3	4	5	6	7		
~~2~~	2	2	~~3~~	3	3	~~4~~	4	~~5~~

~~1~~ 0 2 3 4

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
     do C[i]←0
  for j←1 to length[A]
     do C[A[j]]←C[A[j]]+1
  for j←2 to k
     do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
     do B[C[A[j]]]←A[j]
        C[A[j]]←C[A[j]]-1
```

Aplique el algoritmo

Counting sort

5	3	3	1	2	7	4

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
     do C[i]←0
  for j←1 to length[A]
     do C[A[j]]←C[A[j]]+1
  for j←2 to k
     do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
     do B[C[A[j]]]←A[j]
        C[A[j]]←C[A[j]]-1
```

Analice la complejidad

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
    do C[i]←0
  for j←1 to length[A]
    do C[A[j]]←C[A[j]]+1
  for j←2 to k
    do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
    do B[C[A[j]]]←A[j]
       C[A[j]]←C[A[j]]-1
```

Analice la complejidad

$n = 100$

$k = 100^2$

$k >> n$

$O(k)$

$O(n)$

$O(k)$

$O(n)$

$k = O(n^2)$

T(n)=O(2n+2k), como k=O(n)

T(n)=O(n)

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
  for i←1 to k
     do C[i]←0
  for j←1 to length[A]
     do C[A[j]]←C[A[j]]+1
  for j←2 to k
     do C[i]←C[i] + C[i-1]
  for j←length[A] downto 1
     do B[C[A[j]]]←A[j]
         C[A[j]]←C[A[j]]-1
```

Estabilidad

Counting sort es estable, números con el mismo valor, aparecen en el mismo orden en el arreglo de salida

Algunos algoritmos pueden necesitar esta propiedad en el ordenamiento, radix bucket sort lo hacen

Ordenamiento en tiempo lineal

RADIX-SORT

Ordena n números enteros de d dígitos, ordenando del menos al más significativo

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

n

d O(n)

Ordenamiento en tiempo lineal

RADIX-SORT(A,d)

for i←1 to d

do *sort array A on digit i* O(n)

Se asume que los n elementos del arreglo tienen d dígitos donde el dígito 1 es el menos significativo

Complejidad de O(n+k) donde k es el rango de valores que puede tomar cada digito, como k=O(n), T(n)=O(n)

Ordenamiento en tiempo lineal

BUCKET-SORT

Se asume que la entrada es un conjunto de n números reales en el intervalo [0,1)

Idea

- Dividir el intervalo [0,1) en n subintervalos de igual tamaño (Buckets) y distribuir los n números de entrada en los buckets
- Ordenar cada bucket por inserción y luego concatenar los resultados en cada bucket

Tabla Hash

Ordenamiento en tiempo lineal

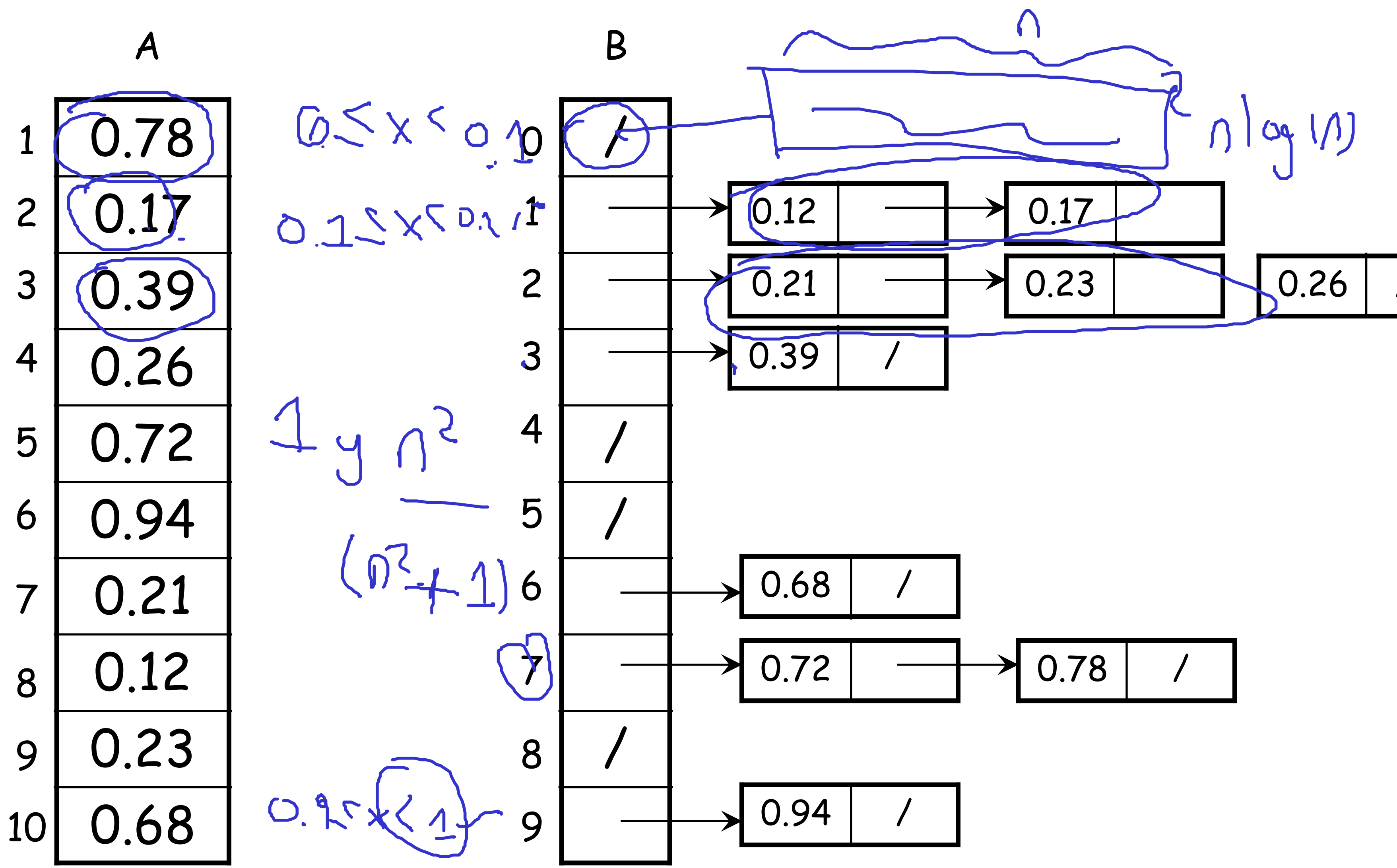
BUCKET-SORT

Como la distribución es uniforme, se espera que los buckets tengan aproximadamente la misma cantidad de elementos

Entrada: $A[1..n]$, donde $0 \leq A[i] < 1$

Se necesita además un arreglo $B[0..n-1]$ de lista enlazadas

Ordenamiento en tiempo lineal



Ordenamiento en tiempo lineal

BUCKET-SORT(A)

n←length[A]

for i←1 to n

do *insert A[i] into list B[⌊nA[i]⌋]*

for i←0 to n-1

do *sort list B[i] with insertion sort*

concatenate the lists B[0], B[1], …, B[n-1]

Ordenamiento en tiempo lineal

BUCKET-SORT(A)

n←length[A]

for i←1 to n

do *insert A[i] into list B[⌊nA[i]⌋]*

for i←0 to n-1

do *sort list B[i] with insertion sort*

concatenate the lists B[0], B[1], …, B[n-1]

Aplicando análisis de esperanza se puede obtener que:

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) = \Theta(n)$$

Ordenamiento en tiempo linea

1) Ordenamiento por comparaciones: Espacio de búsqueda: n!, algoritmos O(n^2) y O(nlog(n))
CUalquier algoritmo que trbaje comparaciones no puede bajar de O(nlog(n))

2) Ordenamiento por otras estrategias, tiempo lineal O(n)

Couting sort: El mayor no puede ser más O(n). Arreglo A: Arreglo inicial, B: SOlución C: Conteo de los menores o igual. Miramos el valor de A, que es la posición del valor en C, y el valor C nos la posición B

Radix SOrt: Ordenar de acuerdo a los dígitos, menos significativo al mas significativo

Bucket Sort: Numero en el intervalo [0, 1), partimos en n rangos. N rangos van a una tabla Hash
Los numeros deben estar uniformemente distribuidos, es decir, no existen preferidos. Puede ordenar por ejemplo entre 1 y n^2, normalizo [0 y 1)

En resumen, los algoritmo en tiempo lineal sólo funcionan en situaciones muy especificas.