

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 utilizan comparaciones

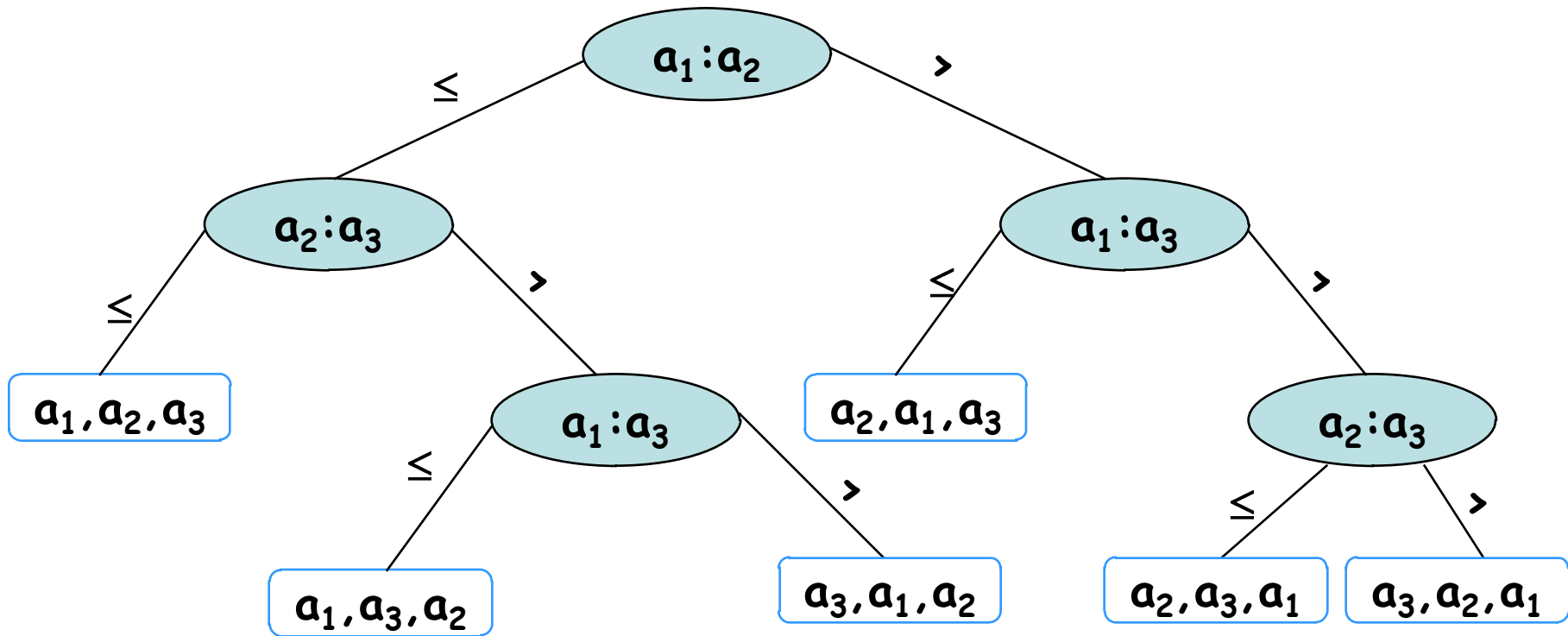
Ordenamiento en tiempo lineal

Otras estrategias:

- Counting sort
- Radix sort
- Bucket sort

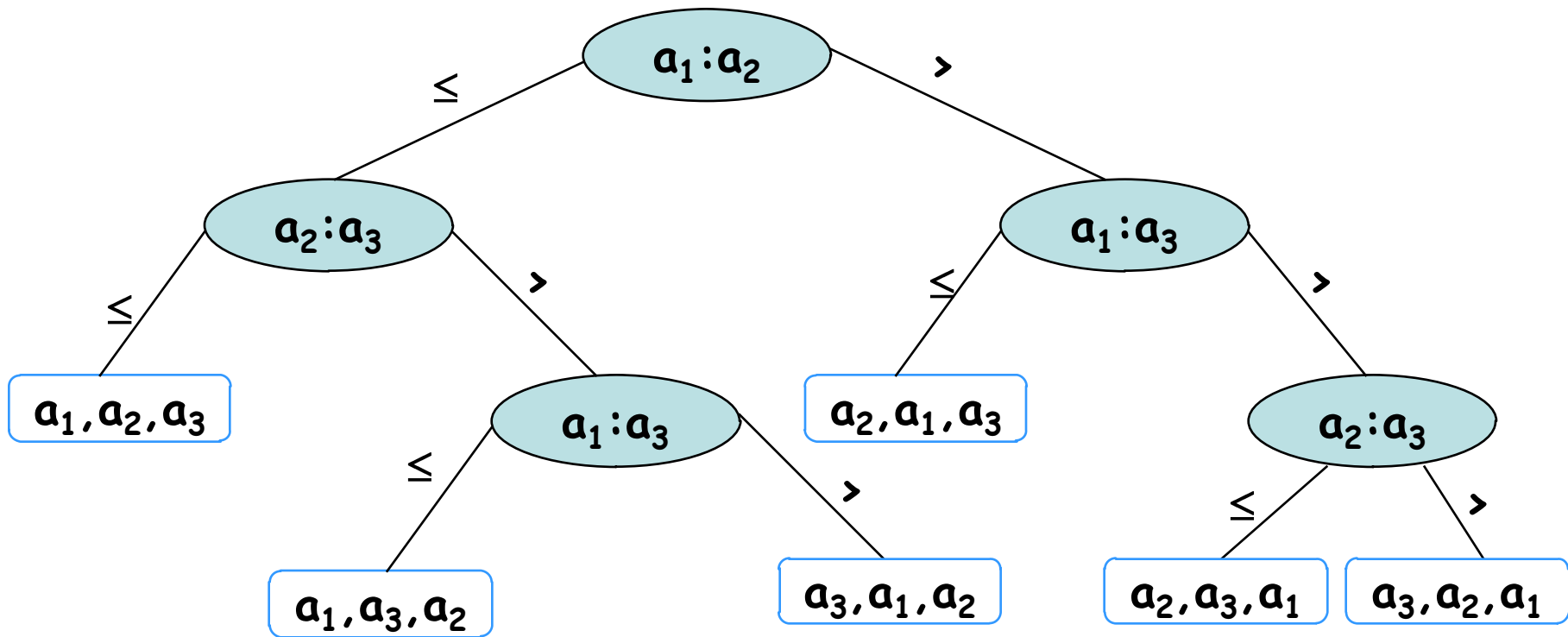
Ordenamiento en tiempo lineal

Cota inferior para ordenar por comparaciones



Ordenamiento en tiempo lineal

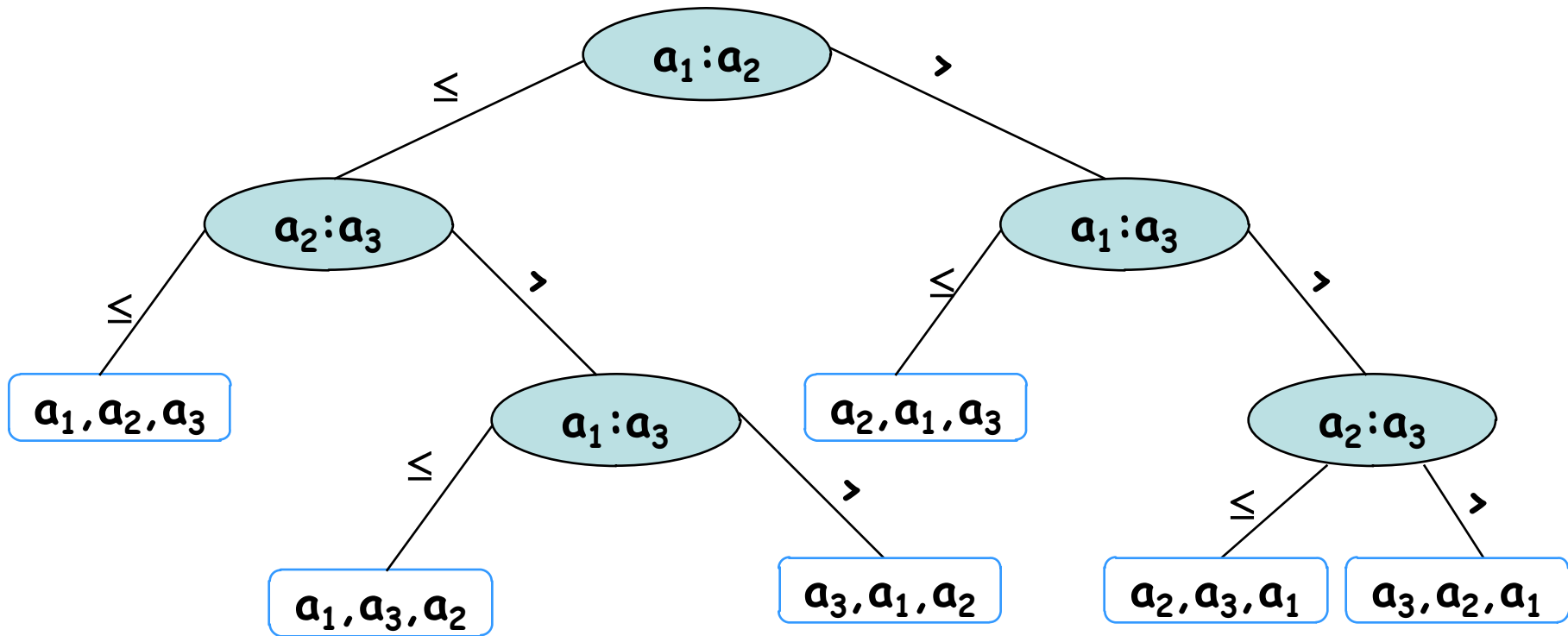
Cota inferior para ordenar por comparaciones



Para ordenar 3 elementos, se tienen $3!$ posibles caminos

Ordenamiento en tiempo lineal

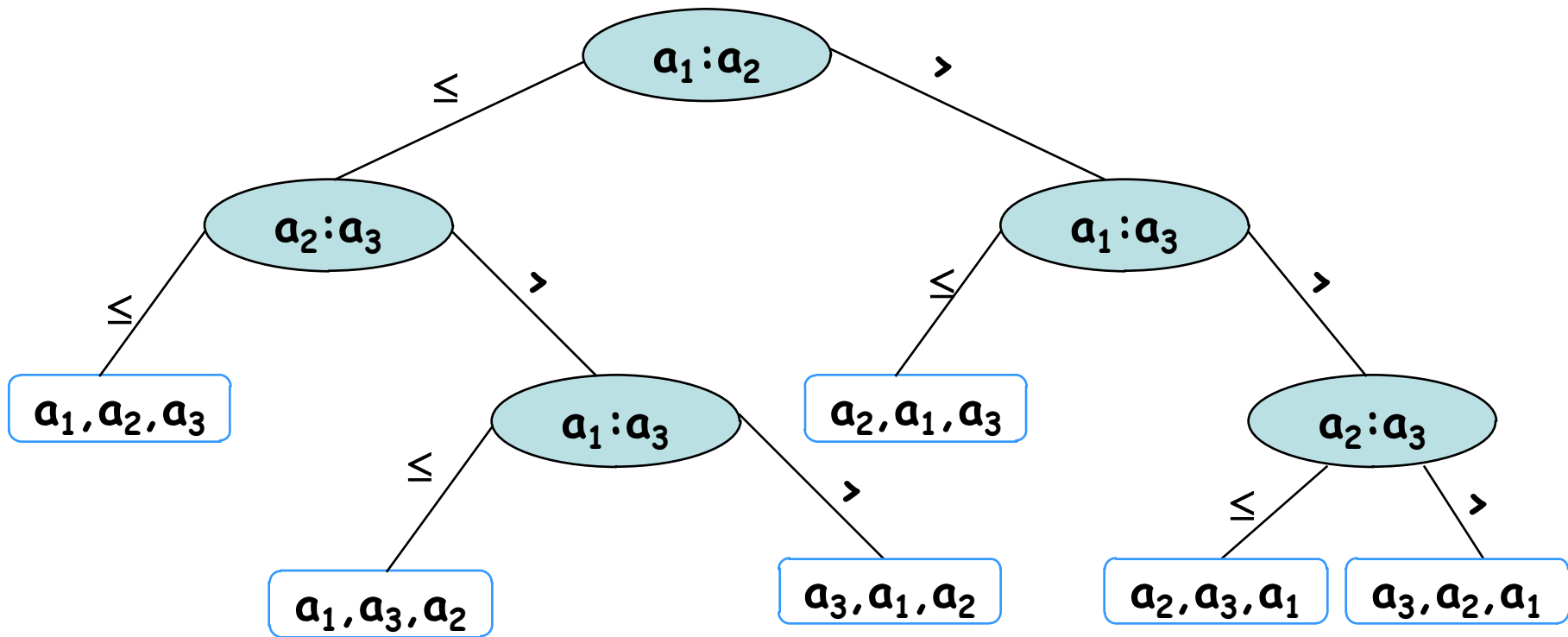
Cota inferior para ordenar por comparaciones



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

Ordenamiento en tiempo lineal

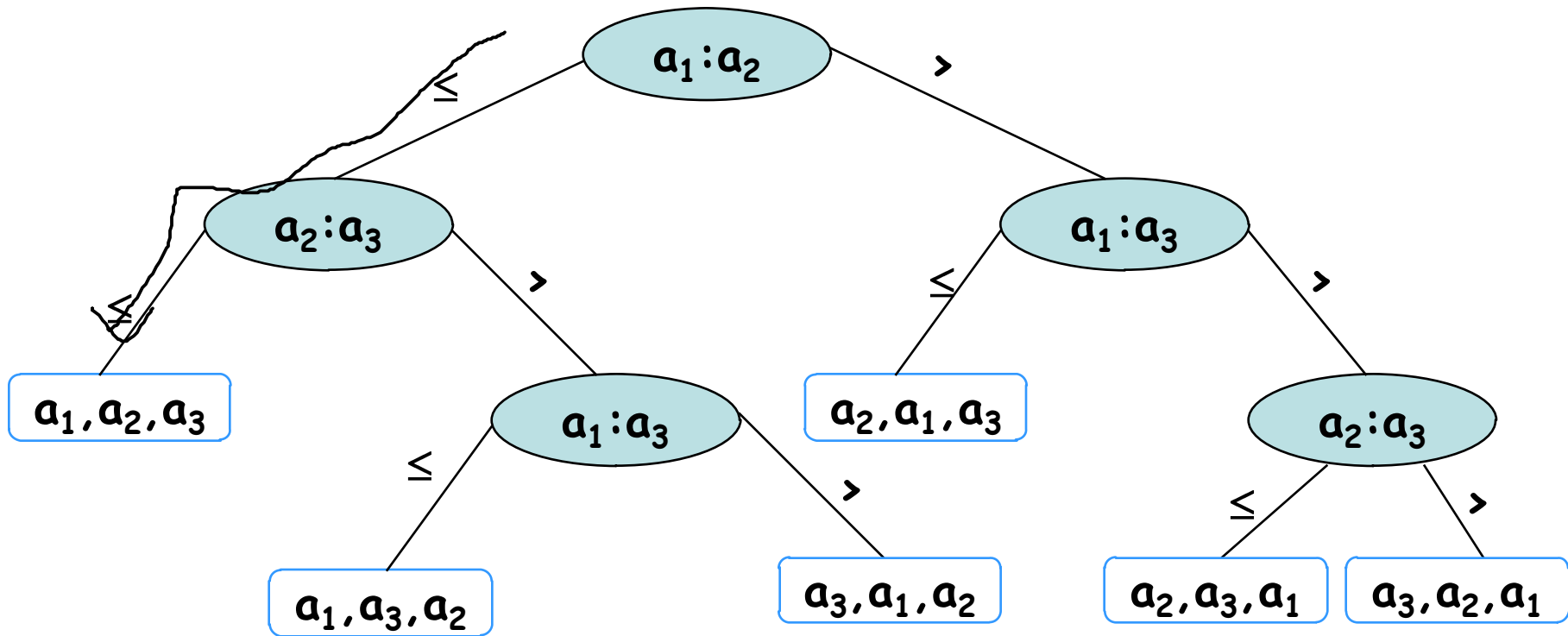
Cota inferior para ordenar por comparaciones



¿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 $\Omega(n \lg n)$ Si está balanceado

$n!$ permutaciones $\rightarrow n!$ 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 $n! > (n/e)^n$

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

$$= \underline{n \lg n - n \lg e}$$

$$= \Omega(n \lg n)$$

Ordenamiento en tiempo lineal

Counting sort

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

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

$C[1..k]$: mantiene los conteos

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

¿Cuánto vale k?

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								

¿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 $i \leftarrow 2$ to k

do $C[i] \leftarrow C[i] + C[i-1]$

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 menores o iguales a i

$C[1]=2$ 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	2	3	4	5	6
C	2	2	4	7	7	8

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

$C[A[8]]=7$

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

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 $A[8]$, 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

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

	1	2	3	4	5	6	7	8
B		1					4	

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

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

	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	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				4	4	

	1	2	3	4	5	6
C	1	2	4	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 $A[5]$, 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

	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 $A[3]$, 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

	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	6

	1	2	3	4	5	6
C	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

	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	6

	1	2	3	4	5	6
C	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

	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	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 $i \leftarrow 1$ to k

do $C[i] \leftarrow 0$

for $j \leftarrow 1$ to $\text{length}[A]$

do $C[A[j]] \leftarrow C[A[j]] + 1$

for $j \leftarrow 2$ to k

do $C[j] \leftarrow C[j] + C[j-1]$

for $j \leftarrow \text{length}[A]$ downto 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Quedamos aquí

1) k es $O(n)$

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[j] ← C[j] + C[j-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

3	2	1	5
---	---	---	---

A 3 2 1 5

K = 5

B 1 2 3 5

4 y K

C - - - - -

1) 1 1 1 0 1
1 2 3 4 5

2)

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

C : [A[i]]

i = 4, 3, 2, 1

Index de A

C : [5] = 4 ← index B

C : [A[3]] = C[1] = 1

C : [A[2]] = C[2] = 2

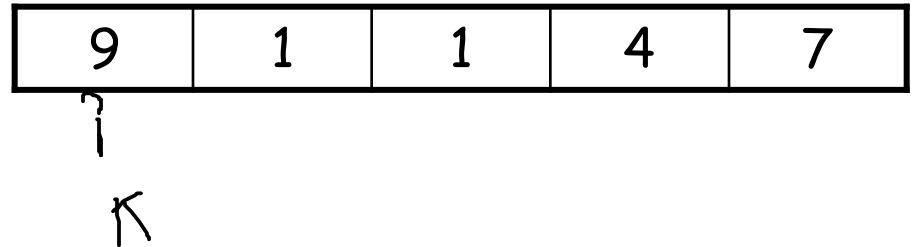
C : [A[1]] = C[3] = 3

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[j] ← C[j] + C[j-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 \neq S



9	1	1	4	7
---	---	---	---	---

A: ⑨ 1 1 4 7

B: 1 1 4 7 9

C: - - - - -

1) 2 0 0 1 0 0 1 0 1

2 2 2 3 3 3 4 4 5

2 2 2 3 3 3 3 4 5

2 2 2 2 3 3 3 4 5

1 2 2 2 3 3 3 4 5

0 2 2 2 3 3 3 4 5

0 2 2 2 3 3 3 4 4

$C[A[5]] =$

$C[7] = 4$

$C[4] = 3$

$C[1] = 2$

$C[1] = 1$

$C[9] = 3$

$K \neq O(n)$

$K = O(n \log n)$

$K = 5 \log_2(5)$

$K = 12$

$9 = 12$

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[j] ← C[j] + C[j-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 i ← 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 $i \leftarrow 1$ to k

do $C[i] \leftarrow 0$

for $j \leftarrow 1$ to $\text{length}[A]$

do $C[A[j]] \leftarrow C[A[j]] + 1$

for $j \leftarrow 2$ to k

do $C[j] \leftarrow C[j] + C[j-1]$

for $j \leftarrow \text{length}[A]$ downto 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Analice la complejidad

} $O(k)$

} $O(n)$

} $O(k)$

} $O(n)$

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

$T(n) = O(n)$

Ordenamiento en tiempo lineal

```
COUNTING-SORT(A,B,k)
```

```
  for  $i \leftarrow 1$  to  $k$ 
```

```
    do  $C[i] \leftarrow 0$ 
```

```
  for  $j \leftarrow 1$  to  $\text{length}[A]$ 
```

```
    do  $C[A[j]] \leftarrow C[A[j]] + 1$ 
```

```
  for  $j \leftarrow 2$  to  $k$ 
```

```
    do  $C[i] \leftarrow C[i] + C[i-1]$ 
```

```
  for  $j \leftarrow \text{length}[A]$  downto 1
```

```
    do  $B[C[A[j]]] \leftarrow A[j]$ 
```

```
       $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

Estabilidad

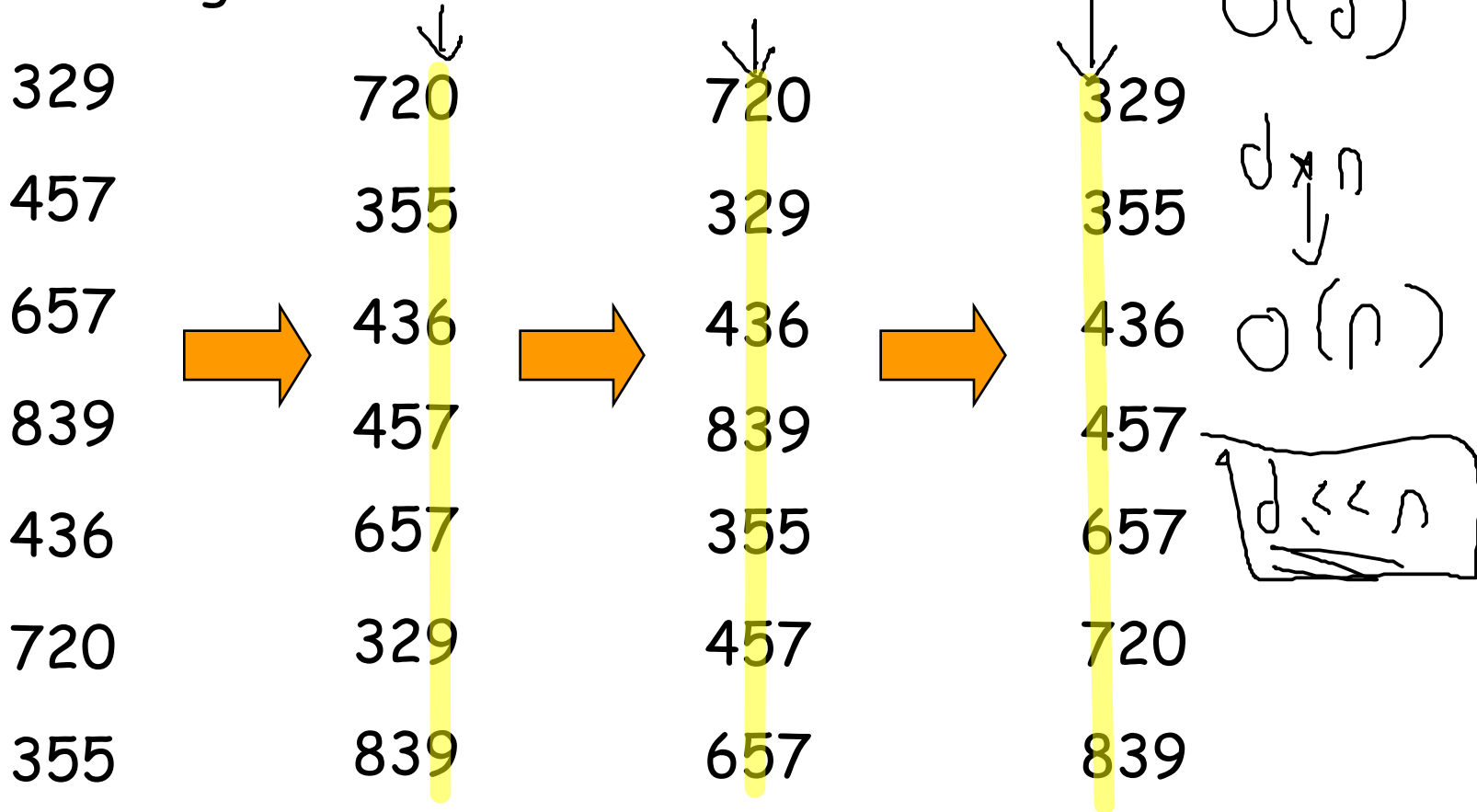
Counting sort es estable, números con el mismo valor, aparecen en el mismo orden en el arreglo de salida

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

427

657

839

436

720

355

329

355

427

436

657

720

839

329

427

X

Este algoritmo trabaja ordenando desde la menor cifra significativa a la mayor cifra significativa, si se intenta en otro orden va a fallar

Ordenamiento en tiempo lineal

RADIX-SORT(A, d)

for $i \leftarrow 1$ to d

do *sort array A on digit i*

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

Complejidad de $O(n \cdot k)$ donde k es el rango de valores que puede tomar cada dígito, como $k = O(n)$, $T(n) = O(n)$

$$\frac{d \leq n}{\underline{\underline{\quad}}}$$

Ordenamiento en tiempo lineal

BUCKET-SORT

Se asume que la entrada es un conjunto de n números reales en el intervalo $[0,1)$ *Si no + doc Normalizar*

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

Ordenamiento en tiempo lineal

BUCKET-SORT

$[0 \dots 1)$

No tenemos preferencias

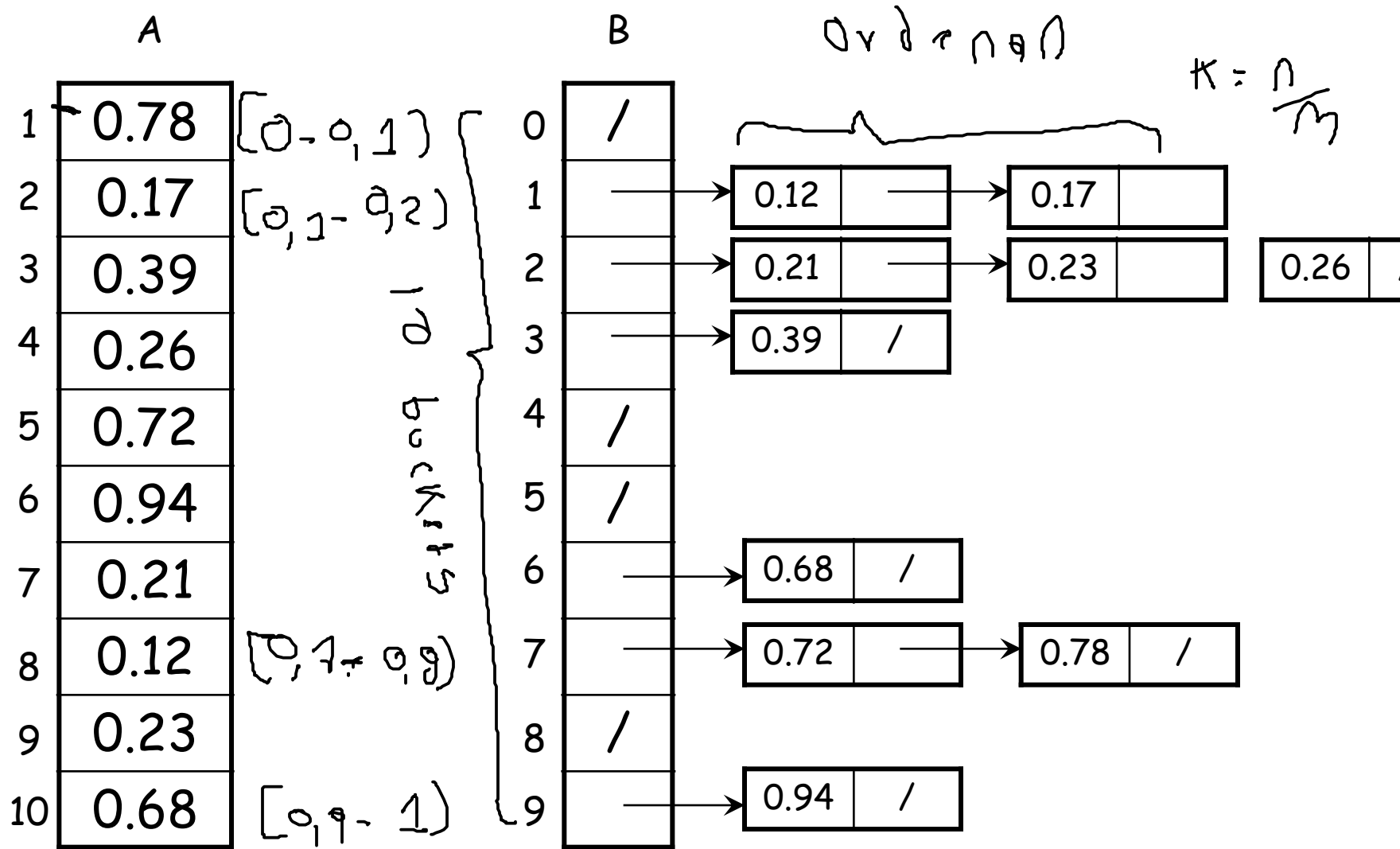
Como la distribución es uniforme, se espera que los buckets tengan aproximadamente la misma cantidad de elementos

Precondición

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)

$n \leftarrow \text{length}[A]$

for $i \leftarrow 1$ to n

 do *insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$*

for $i \leftarrow 0$ to $n-1$

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

concatenate the lists $B[0], B[1], \dots, B[n-1]$

Ordenamiento en tiempo lineal

BUCKET-SORT(A)

$n \leftarrow \text{length}[A]$

for $i \leftarrow 1$ to n

do *insert* $A[i]$ into list $B[\lfloor nA[i] \rfloor]$

for $i \leftarrow 0$ to $n-1$

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

concatenate the lists $B[0], B[1], \dots, B[n-1]$

Aplicando análisis de esperanza se puede obtener que:

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

1) Counting-sort

9) $Num \in \{1 \dots K\}$

b) K es $O(n)$

2) Radix-Sort

a) n n u m p , o

b) d d. g. to s d < c n

c) Se ordena cifra m p n s, significativamente,

3) Bucket Sort

9) Annunzio

d) $\eta \in V[0, 1]$

b) $n \in (0, 1)$

c) m b u k p t s

a) $n \in \text{NumProcs}$
 b) $n \in [0, 1)$
 c) $m \in \text{Buckets}$

d) $n \in U[0, 1)$
 \swarrow
 $d_{st} + \text{uniform}$
 $O(c+1) = O(n)$

$$O(n^2 + 1) = O(n^2)$$