Divide y Vencerás

José Ortiz Bejar

Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo

Febrero 28, 2022

Contenido

- Mezcla (Unión de conjuntos)
- Búsqueda
- Ordenamiento
- Multiplicación
- Computar la *mediana*

Mezcla (Unión de conjuntos)

Mezcla (Unión de conjuntos)

- **Input:** Dos secuencias $A = \langle a_1, a_2, \ldots, a_n \rangle$ y $B = \langle b_1, b_2, \ldots, b_m \rangle$

  Output: Una secuencia $X = \langle x_1, x_2, \ldots, x_\ell \rangle$ tal que

Mezcla (Unión de conjuntos)

- **Input:** Dos secuencias $A = \langle a_1, a_2, \ldots, a_n \rangle$ y $B = \langle b_1, b_2, \ldots, b_m \rangle$

  Output: Una secuencia $X = \langle x_1, x_2, \ldots, x_\ell \rangle$ tal que

  - cada elemento en A aparece una vez en X
  - cada elemento en B aparece una vez en X
  - todos los elementos en X aparecen en A o en B o en ambos

Mezcla (Unión de conjuntos)

- **Input:** Dos secuencias $A = \langle a_1, a_2, \ldots, a_n \rangle$ y $B = \langle b_1, b_2, \ldots, b_m \rangle$

  Output: Una secuencia $X = \langle x_1, x_2, \ldots, x_\ell \rangle$ tal que

  - cada elemento en A aparece una vez en X
  - cada elemento en B aparece una vez en X
  - todos los elementos en X aparecen en A o en B o en ambos

- **Ejemplo:**

  $A = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10 \rangle$

  $B = \langle 51, 21, 14, 15, 27, 31, 2 \rangle$

  $X =$

Mezcla (Unión de conjuntos)

- **Input:** Dos secuencias $A = \langle a_1, a_2, \ldots, a_n \rangle$ y $B = \langle b_1, b_2, \ldots, b_m \rangle$

  Output: Una secuencia $X = \langle x_1, x_2, \ldots, x_\ell \rangle$ tal que

  - cada elemento en A aparece una vez en X
  - cada elemento en B aparece una vez en X
  - todos los elementos en X aparecen en A o en B o en ambos

- **Ejemplo:**

  $A = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10 \rangle$

  $B = \langle 51, 21, 14, 15, 27, 31, 2 \rangle$

  $X = \langle 34, 7, 11, 31, 14, 51, 8, 21, 10, 15, 27, 2 \rangle$

Algoritmo de mezcla simple

- Estrategia

Algoritmo de mezcla simple

- Estrategia
  - iterar todas las posiciones i, primero la secuencia A, y después B
  - regresar a_i si a_i no está en $\langle a_1, a_2, \ldots, a_{i-1} \rangle$
  - regresar b_i si b_i no está aún en $\langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots b_{i-1} \rangle$

Algoritmo de mezcla simple

- Estrategia
  - iterar todas las posiciones i, primero la secuencia A, y después B
  - regresar a_i si a_i no está en $\langle a_1, a_2, \ldots, a_{i-1} \rangle$
  - regresar b_i si b_i no está aún en $\langle a_1, a_2, \ldots, a_n, b_1, b_2, \ldots b_{i-1} \rangle$

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 .. i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 .. i − 1], B[i])
6          output B[i]
```

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

sea $n = length(A) + length(B)$

$$T(n) = \sum_{i=1}^{length(A)} T_{\textbf{Find}}(i) + \sum_{i=1}^{length(B)} \left(T_{\textbf{Find}}(i) + T_{\textbf{Find}}(length(A))\right)$$

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

sea $n = length(A) + length(B)$

$$T(n) = \sum_{i=1}^{length(A)} T_{\textbf{Find}}(i) + \sum_{i=1}^{length(B)} \left(T_{\textbf{Find}}(i) + T_{\textbf{Find}}(length(A))\right)$$

$$T(n) = \sum_{i=1}^{n} T_{\textbf{Find}}(i)$$

- **Input:** una secuencia *A* y un valor *key*
  Output: true si *A* contiene *key*, o false en caso contrario

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en caso contrario

```
Find(A, key)
1  for i = 1 to length(A)
2      if A[i] == key
3          return true
4  return false
```

```
Find(A, begin, end, key)
1  for i = begin to end
2      if A[i] == key
3          return true
4  return false
```

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en caso contrario

```
Find(A, key)
1  for i = 1 to length(A)
2      if A[i] == key
3          return true
4  return false
```

```
Find(A, begin, end, key)
1  for i = begin to end
2      if A[i] == key
3          return true
4  return false
```

- La complejidad de **Find** es

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en caso contrario

```
Find(A, key)
1  for i = 1 to length(A)
2      if A[i] == key
3          return true
4  return false
```

```
Find(A, begin, end, key)
1  for i = begin to end
2      if A[i] == key
3          return true
4  return false
```

- La complejidad de **Find** es

$$T(n) = O(n)$$

Búsqueda en una lista

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en otro caso

Búsqueda en una lista

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en otro caso

```
FindInList(A, key)
1  item = first(A)
2  while item ≠ last(A)
3        if value(item) == key
4              return true
5        item = next(item)
6  return false
```

Búsqueda en una lista

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en otro caso

FindInList(A, key)

```
1  item = first(A)
2  while item ≠ last(A)
3        if value(item) == key
4              return true
5        item = next(item)
6  return false
```

- La complejidad de **FindInList** es

Búsqueda en una lista

- **Input:** una secuencia A y un valor *key*
  Output: true si A contiene *key*, o false en otro caso

FindInList(A, key)

```
1  item = first(A)
2  while item ≠ last(A)
3        if value(item) == key
4              return true
5        item = next(item)
6  return false
```

- La complejidad de **FindInList** es

$$T(n) = O(n)$$

Complejidad de MergeSimple

```
MergeSimple(A, B)
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

Complejidad de MergeSimple

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

$$T(n) = \sum_{i=1}^{n} T_{\textbf{Find}}(i)$$

Complejidad de MergeSimple

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

$$T(n) = \sum_{i=1}^{n} T_{\textbf{Find}}(i)$$

$$T(n) = \sum_{i=1}^{n} O(i) =$$

Complejidad de MergeSimple

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

$$T(n) = \sum_{i=1}^{n} T_{\textbf{Find}}(i)$$

$$T(n) = \sum_{i=1}^{n} O(i) = O\left(\frac{n(n+1)}{2}\right) =$$

Complejidad de MergeSimple

MergeSimple(A, B)

```
1  for i = 1 to length(A)
2      if not Find(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not Find(A, B[i]) and not Find(B[1 . . i − 1], B[i])
6          output B[i]
```

$$T(n) = \sum_{i=1}^{n} T_{\textbf{Find}}(i)$$

$$T(n) = \sum_{i=1}^{n} O(i) = O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

Búsqueda (2)

- **Input:** una secuencia *ordenada* A *key*
  Output: true si A contiene *key*, o false en cualquier otro caso

Búsqueda (2)

- **Input:** una secuencia *ordenada* A *key*

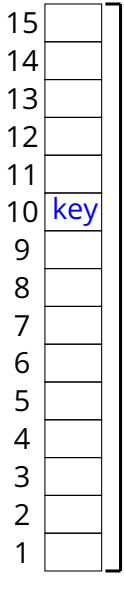
  Output: true si A contiene *key*, o false en cualquier otro caso

BinarySearch(A, key)

```
1   first = 1
2   last = length(A)
3   while first ≤ last
4       middle = ⌈(first + last)/2⌉
5       if A[middle] == key
6           return true
7       elseif first = last
8           return false
9       elseif A[middle] > key
10          last = middle − 1
11      else first = middle + 1
12  return false
```

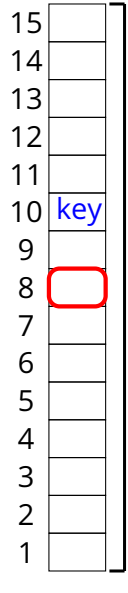
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



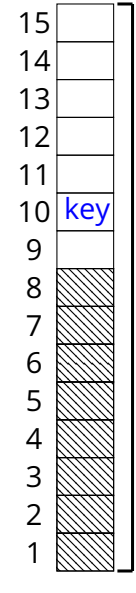
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



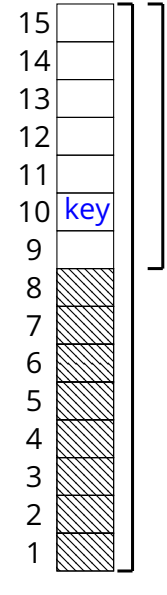
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



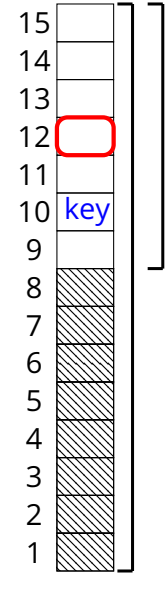
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



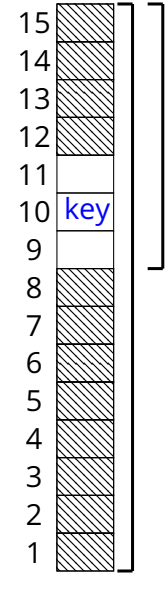
Búsqueda Binaria

```
BinarySearch(A, key)
1   first = 1
2   last = length(A)
3   while first ≤ last
4         middle = ⌈(first + last)/2⌉
5         if A[middle] == key
6               return true
7         elseif first = last
8               return false
9         elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



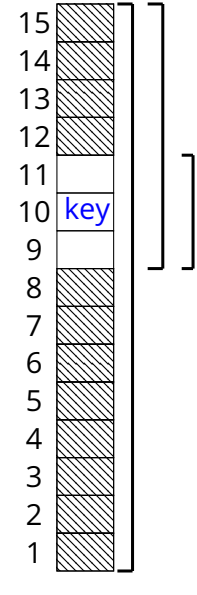
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



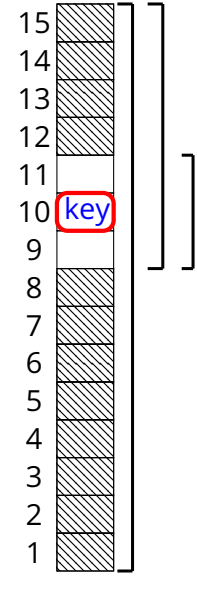
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



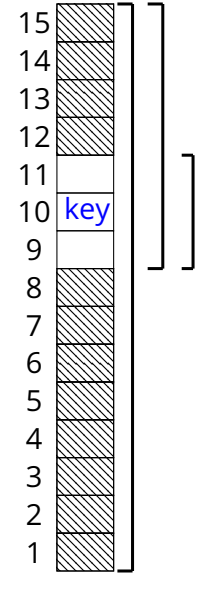
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



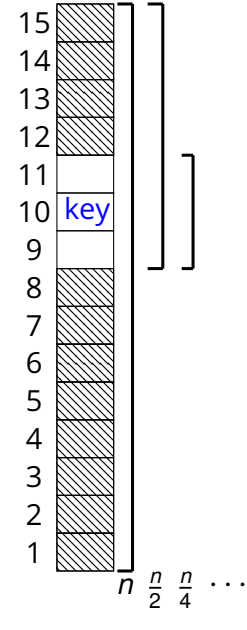
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



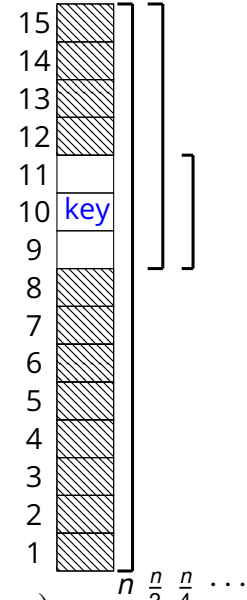
Búsqueda Binaria

```
BinarySearch(A, key)
 1  first = 1
 2  last = length(A)
 3  while first ≤ last
 4        middle = ⌈(first + last)/2⌉
 5        if A[middle] == key
 6              return true
 7        elseif first = last
 8              return false
 9        elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



Búsqueda Binaria

```
BinarySearch(A, key)
1   first = 1
2   last = length(A)
3   while first ≤ last
4         middle = ⌈(first + last)/2⌉
5         if A[middle] == key
6               return true
7         elseif first = last
8               return false
9         elseif A[middle] > key
10              last = middle − 1
11        else first = middle + 1
12  return false
```



$T(n) = O(\log n)$

Merging Sorted Sequences

- Un problema ligeramente diferente:

  Input: Dos secuencias*ordenadas* $A = \langle a_1, a_2, \ldots, a_n \rangle$ and $B = \langle b_1, b_2, \ldots, b_m \rangle$, donde $a_1 \leq a_2 \leq \ldots \leq a_n$ and $b_1 \leq b_2 \leq \ldots \leq b_m$

  Output: una secuencia $X = \langle x_1, x_2, \ldots, x_\ell \rangle$ tal que

  - todos los elementos en A aparecen una vez en X
  - todos los elementos en B aparecen en X
  - todos los elementos en X están en A en B o en ambos

Un mejor algoritmo de mezcla

```
MergeSimple2(A, B)
1  for i = 1 to length(A)
2      if not BinarySearch(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not BinarySearch(A, B[i])
6      and not BinarySearch(B[1 . . i − 1], B[i])
7          output B[i]
```

Un mejor algoritmo de mezcla

MergeSimple2(A, B)

```
1  for i = 1 to length(A)
2      if not BinarySearch(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not BinarySearch(A, B[i])
6      and not BinarySearch(B[1 . . i − 1], B[i])
7          output B[i]
```

$$T(n) = \sum_{i=1}^{n} O(\log i) =$$

Un mejor algoritmo de mezcla

MergeSimple2(A, B)

```
1  for i = 1 to length(A)
2      if not BinarySearch(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not BinarySearch(A, B[i])
6      and not BinarySearch(B[1 . . i − 1], B[i])
7          output B[i]
```

$$T(n) = \sum_{i=1}^{n} O(\log i) = O(n \log n)$$

Un mejor algoritmo de mezcla

MergeSimple2(A, B)

```
1  for i = 1 to length(A)
2      if not BinarySearch(A[1 . . i − 1], A[i])
3          output A[i]
4  for i = 1 to length(B)
5      if not BinarySearch(A, B[i])
6      and not BinarySearch(B[1 . . i − 1], B[i])
7          output B[i]
```

$$T(n) = \sum_{i=1}^{n} O(\log i) = O(n \log n)$$

Mejor que $O(n^2)$. ¿Se puede hacer aún mejor que $O(n \log n)$?

Un algoritmo de mezcla aún mejor

- *Intuición:* la secuencias A y B están *ordenadas*
  ej.
  $A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$
  $B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$

Un algoritmo de mezcla aún mejor

- *Intuición:* la secuencias A y B están *ordenadas*
  ej.

  $A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$

  $B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$

  de igual forma que en **BinarySearch** dados $x \in A$ y yB Se puede evitar buscar un elemento x si el *primer* elemento que se ve es $y > x$

Un algoritmo de mezcla aún mejor

- *Intuición:* la secuencias A y B están *ordenadas*
  ej.
  $A = \langle 3, 7, 12, 13, 34, 37, 70, 75, 80 \rangle$
  $B = \langle 1, 5, 6, 7, 34, 35, 40, 41, 43 \rangle$
  de igual forma que en **BinarySearch** dados $x \in A$ y yB Se puede evitar buscar un elemento x si el *primer* elemento que se ve es $y > x$

- Estrategia
  - iterar todas las posiciones i de A y j en B
  - regresar a_i y avanzar i si $a_i \leq b_j$ o j está al final de B
  - regresar b_j e incrementar j si $a_i \geq b_j$ o si i ha pasado el final de A

Algoritmo de mezcla

$A =$

3	7	12	13	34	37	70	75	80

$B =$

1	5	6	7	34	35	40	41	43

Algoritmo de mezcla

$A =$

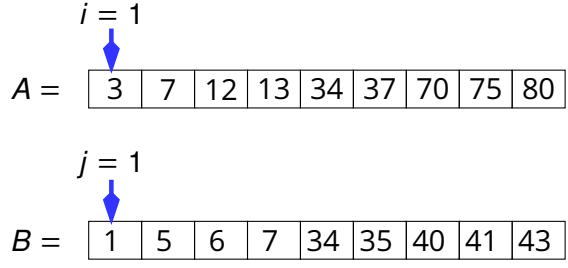
3	7	12	13	34	37	70	75	80

$B =$

1	5	6	7	34	35	40	41	43

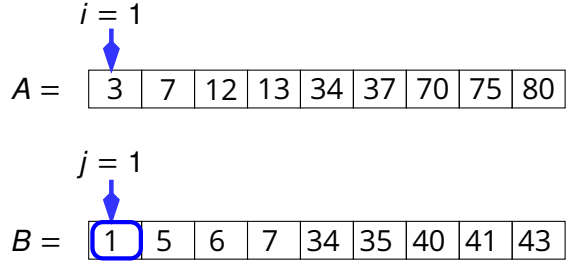
$C = AoB =$

Algoritmo de mezcla



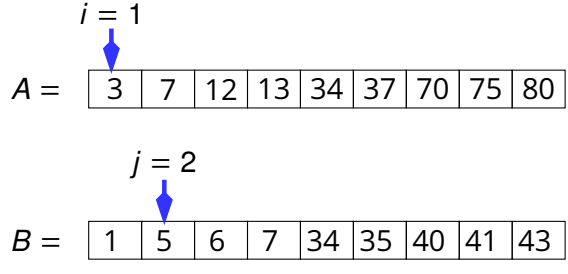
$C = AoB =$

Algoritmo de mezcla



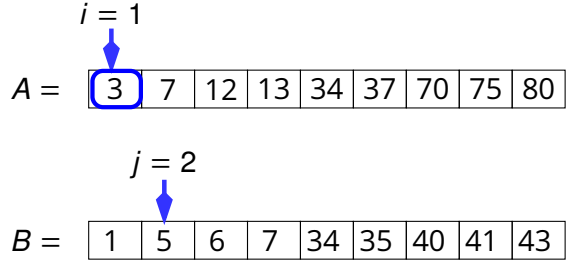
$C = AoB =$ 1

Algoritmo de mezcla



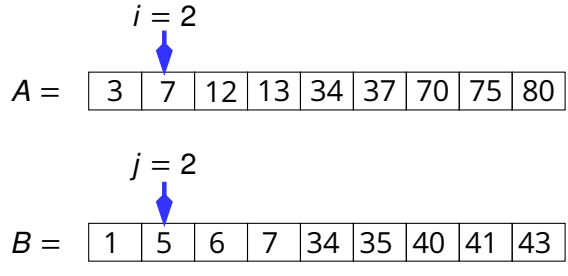
$C = AoB =$ 1

Algoritmo de mezcla



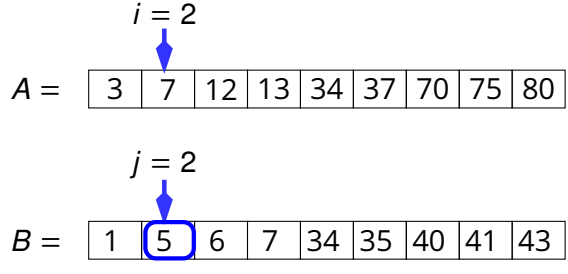
$C = AoB =$ 1 3

Algoritmo de mezcla



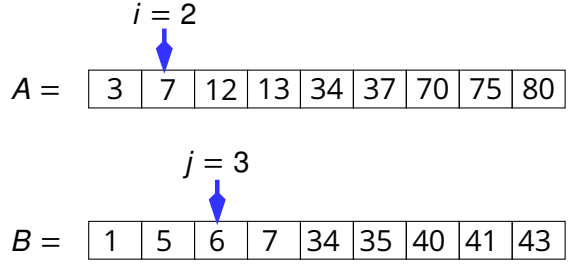
$C = AoB =$ 1 3

Algoritmo de mezcla



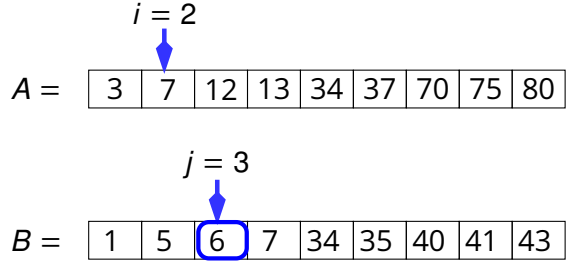
$C = AoB =$ 1 3 5

Algoritmo de mezcla



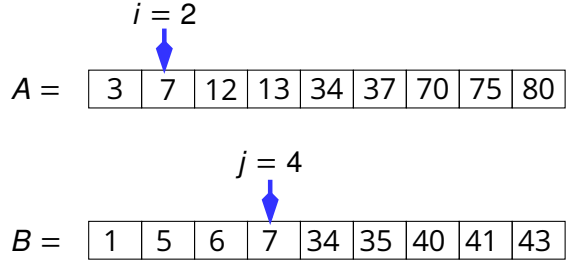
$C = AoB =$ 1 3 5

Algoritmo de mezcla



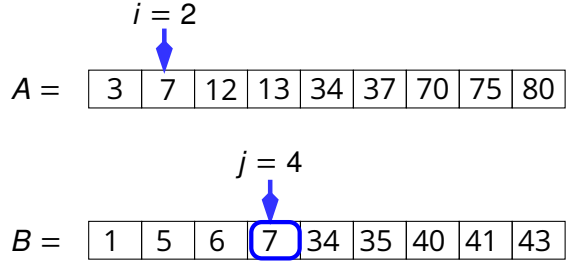
$C = AoB =$ 1 3 5 6

Algoritmo de mezcla



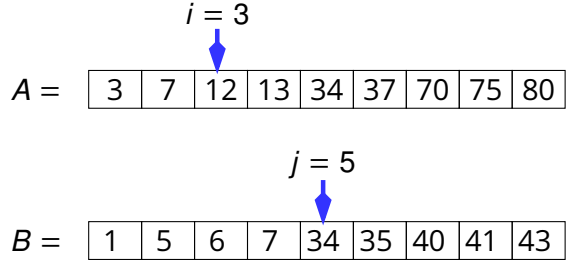
$C = AoB =$ 1 3 5 6

Algoritmo de mezcla



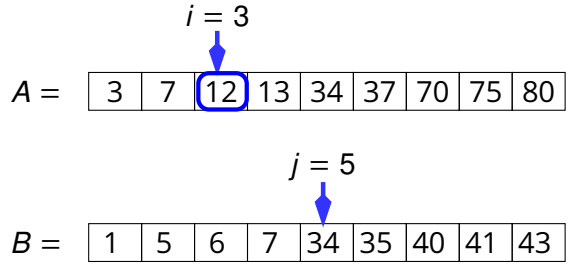
$C = AoB =$ 1 3 5 6 7

Algoritmo de mezcla



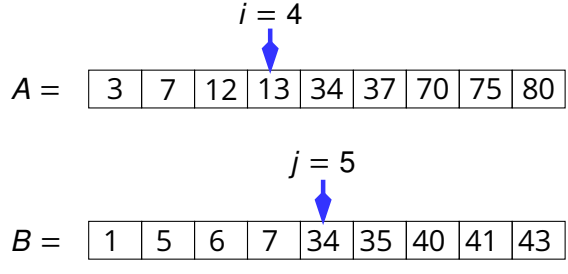
$C = AoB =$ 1 3 5 6 7

Algoritmo de mezcla



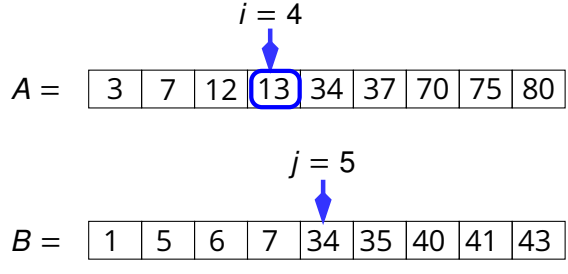
$C = AoB =$ 1 3 5 6 7 12

Algoritmo de mezcla



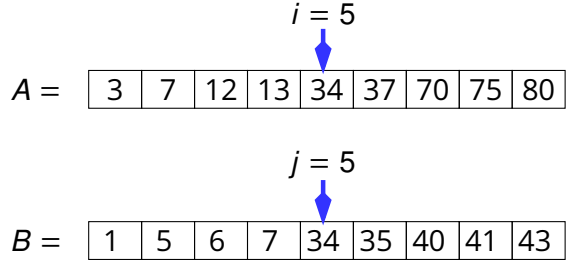
$C = AoB =$ 1 3 5 6 7 12

Algoritmo de mezcla



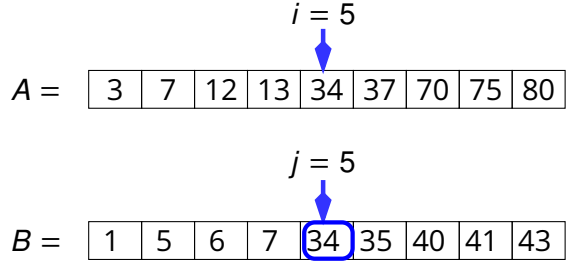
$C = AoB =$ 1 3 5 6 7 12 13

Algoritmo de mezcla



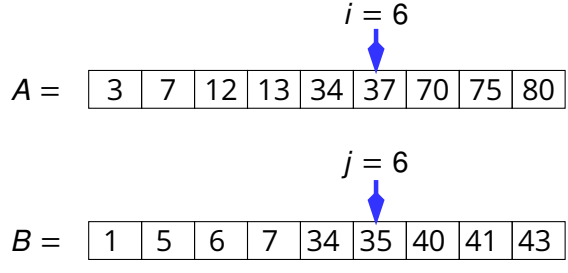
$C = AoB =$ 1 3 5 6 7 12 13

Algoritmo de mezcla



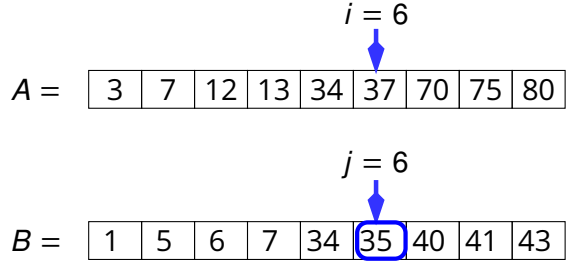
$C = AoB =$ 1 3 5 6 7 12 13 34

Algoritmo de mezcla



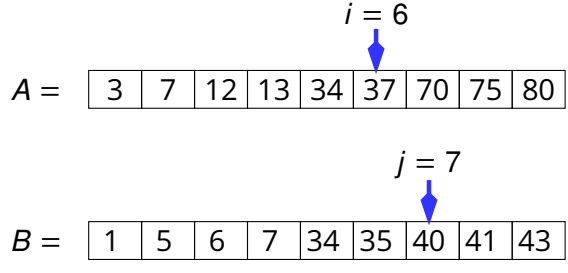
$C = AoB =$ 1 3 5 6 7 12 13 34

Algoritmo de mezcla



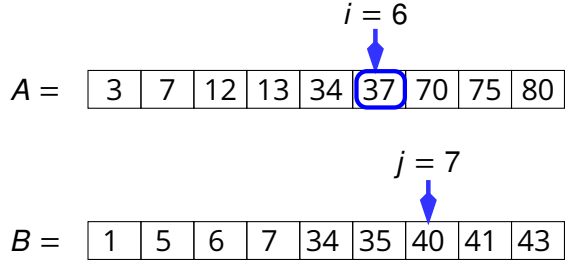
$C = AoB =$ 1 3 5 6 7 12 13 34 35

Algoritmo de mezcla



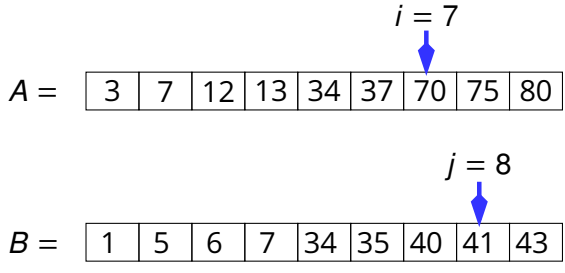
$C = AoB =$ 1 3 5 6 7 12 13 34 35

Algoritmo de mezcla



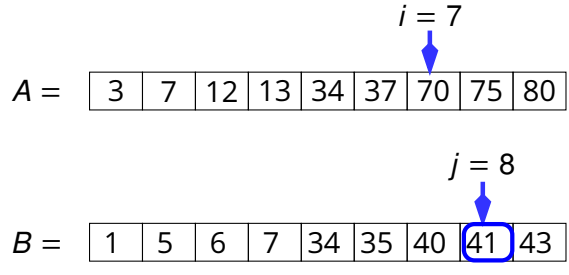
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37

Algoritmo de mezcla



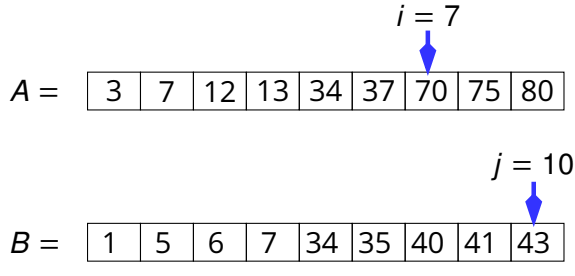
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37

Algoritmo de mezcla



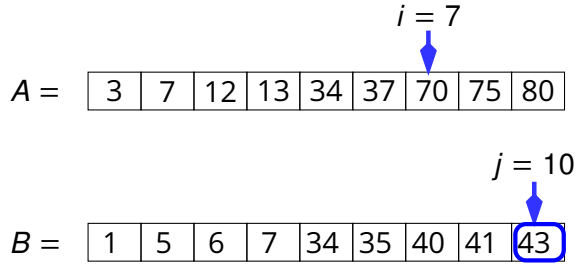
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41

Algoritmo de mezcla



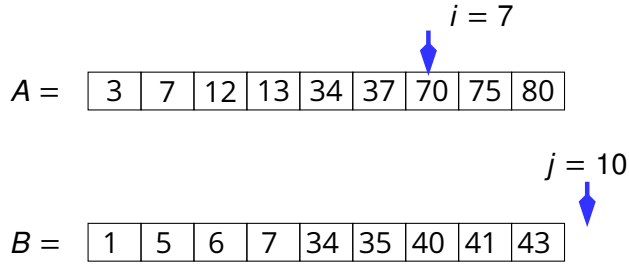
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41

Algoritmo de mezcla



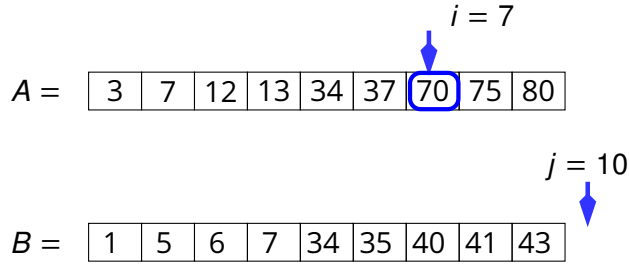
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43

Algoritmo de mezcla



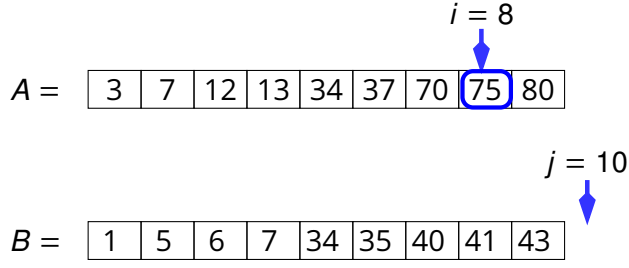
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43

Algoritmo de mezcla



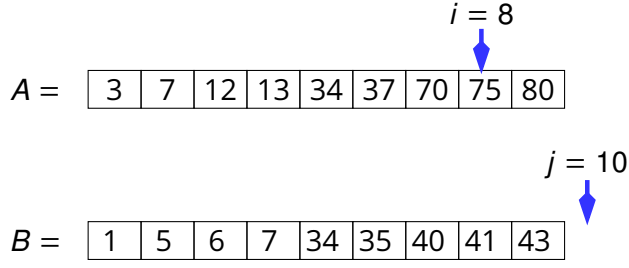
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70

Algoritmo de mezcla



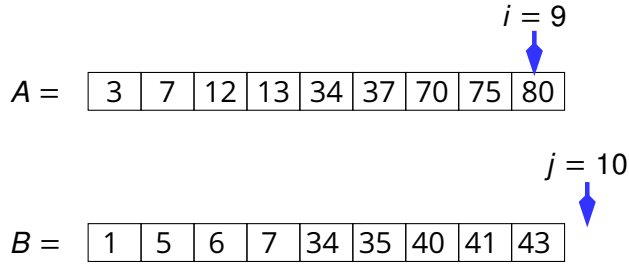
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70 75

Algoritmo de mezcla



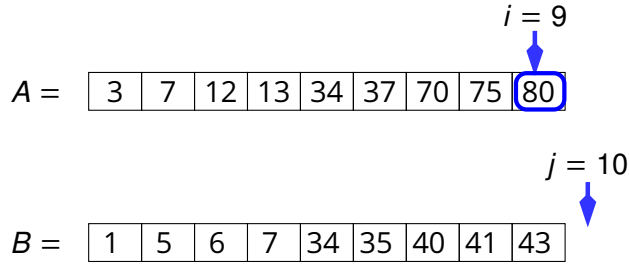
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70 75

Algoritmo de mezcla



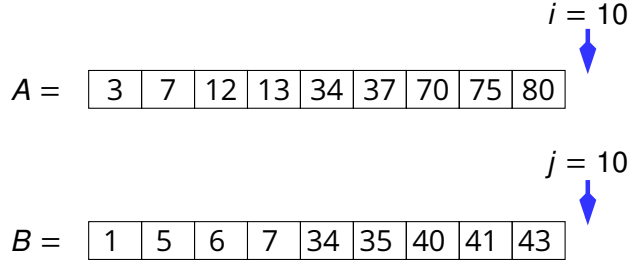
$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70 75

Algoritmo de mezcla



$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70 75 80

Algoritmo de mezcla



$C = AoB =$ 1 3 5 6 7 12 13 34 35 37 41 43 70 75 80

Algoritmo de Mezcla (2)

```
Merge(A, B)
 1  i, j = 1
 2  X = ∅
 3  while i ≤ length(A) or j ≤ length(B)
 4       if i > length(A)
 5            X = X ∘ B[j]          // appends B[j] to X
 6            j = j + 1
 7       elseif j > length(B)
 8            X = X ∘ A[i]
 9            i = i + 1
10       elseif A[i] < B[j]
11            X = X ∘ A[i]
12            i = i + 1
13       else X = X ∘ B[j]
14            j = j + 1
15  return X
```

Algoritmo de Mezcla (2)

```
Merge(A, B)
 1  i, j = 1
 2  X = ∅
 3  while i ≤ length(A) or j ≤ length(B)
 4      if i > length(A)
 5          X = X ∘ B[j]          // appends B[j] to X
 6          j = j + 1
 7      elseif j > length(B)
 8          X = X ∘ A[i]
 9          i = i + 1
10      elseif A[i] < B[j]
11          X = X ∘ A[i]
12          i = i + 1
13      else X = X ∘ B[j]
14          j = j + 1
15  return X
```

- Este algoritmo es incorrecto (Ejercicio: correjirlo)

Complejidad de Merge

```
Merge(A, B)
1  i, j = 1
2  X = ∅
3  while i ≤ length(A) or j ≤ length(B)
4       if i ≤ length(A) and (j > length(B) or A[i] < B[j])
5            X = X ∘ A[i]
6            i = i + 1
7       else X = X ∘ B[j]
8            j = j + 1
9  return X
```

Complejidad de Merge

```
Merge(A, B)
1  i, j = 1
2  X = ∅
3  while i ≤ length(A) or j ≤ length(B)
4       if i ≤ length(A) and (j > length(B) or A[i] < B[j])
5            X = X ∘ A[i]
6            i = i + 1
7       else X = X ∘ B[j]
8            j = j + 1
9  return X
```

$$T(n) = \Theta(n)$$

Complejidad de Merge

```
Merge(A, B)
1  i, j = 1
2  X = ∅
3  while i ≤ length(A) or j ≤ length(B)
4       if i ≤ length(A) and (j > length(B) or A[i] < B[j])
5            X = X ∘ A[i]
6            i = i + 1
7       else X = X ∘ B[j]
8            j = j + 1
9  return X
```

$$T(n) = \Theta(n)$$

- ¿Es posible hacerlo mejor?

Complejidad de Merge

```
Merge(A, B)
1  i, j = 1
2  X = ∅
3  while i ≤ length(A) or j ≤ length(B)
4       if i ≤ length(A) and (j > length(B) or A[i] < B[j])
5            X = X ∘ A[i]
6            i = i + 1
7       else X = X ∘ B[j]
8            j = j + 1
9  return X
```

$$T(n) = \Theta(n)$$

- ¿Es posible hacerlo mejor? No!

Complejidad de Merge

Merge(A, B)

```
1  i, j = 1
2  X = ∅
3  while i ≤ length(A) or j ≤ length(B)
4        if i ≤ length(A) and (j > length(B) or A[i] < B[j])
5              X = X ∘ A[i]
6              i = i + 1
7        else X = X ∘ B[j]
8              j = j + 1
9  return X
```

$$T(n) = \Theta(n)$$

- ¿Es posible hacerlo mejor? No!
  - debemos procesar $n = length(A) + length(B)$ elementos

Utilizar Merge

- Se tiene un procedimiento de mezcla con *complejidad lineal*.
  - mezclar secuencias *ordenadas*
  - *produce una secuencia ordenada*

Utilizar Merge

- Se tiene un procedimiento de mezcla con *complejidad lineal*.
  - mezclar secuencias *ordenadas*
  - *produce una secuencia ordenada*

- Por tanto podemos implementar un algoritmo de ordenamiento

Utilizar Merge

- Se tiene un procedimiento de mezcla con *complejidad lineal*.
  - mezclar secuencias *ordenadas*
  - *produce una secuencia ordenada*

- Por tanto podemos implementar un algoritmo de ordenamiento

- Idea
  - utilizar una variante de **Merge** que regreses *todos* elementos de las secuencias de entrada
    - e.i., sin remover los elementos repetidos
  - asumir que A esta compuesta de dos mitades, $A_L \circ A_R = A$, y que A_L y A_R están ya ordenadas

Utilizar Merge

- Se tiene un procedimiento de mezcla con *complejidad lineal*.
  - mezclar secuencias *ordenadas*
  - *produce una secuencia ordenada*
- Por tanto podemos implementar un algoritmo de ordenamiento
- Idea
  - utilizar una variante de **Merge** que regreses *todos* elementos de las secuencias de entrada
    - e.i., sin remover los elementos repetidos
  - asumir que A esta compuesta de dos mitades, $A_L \circ A_R = A$, y que A_L y A_R están ya ordenadas
  - utilizar **Merge** para combinar A_L y A_R en una secuencia ordenada.

Utilizar Merge

- Se tiene un procedimiento de mezcla con *complejidad lineal*.
  - mezclar secuencias *ordenadas*
  - *produce una secuencia ordenada*
- Por tanto podemos implementar un algoritmo de ordenamiento
- Idea
  - utilizar una variante de **Merge** que regreses *todos* elementos de las secuencias de entrada
    - e.i., sin remover los elementos repetidos
  - asumir que A esta compuesta de dos mitades, $A_L \circ A_R = A$, y que A_L y A_R están ya ordenadas
  - utilizar **Merge** para combinar A_L y A_R en una secuencia ordenada.
  - esto sugiere un algoritmo recursivo

Ordenamiento por mezcla (Merge Sort)

```
MergeSort(A)
1  if length(A) == 1
2        return A
3  m = ⌊length(A)/2⌋
4  A_L = MergeSort(A[1 . . m])
5  A_R = MergeSort(A[m + 1 . . length(A)])
6  return Merge(A_L, A_R)
```

Ordenamiento por mezcla (Merge Sort)

MergeSort(A)

```
1  if length(A) == 1
2        return A
3  m = ⌊length(A)/2⌋
4  A_L = MergeSort(A[1 . . m])
5  A_R = MergeSort(A[m + 1 . . length(A)])
6  return Merge(A_L, A_R)
```

- ¿Cuál es la complejidad de **MergeSort**?

Ordenamiento por mezcla (Merge Sort)

MergeSort(A)

```
1  if length(A) == 1
2        return A
3  m = ⌊length(A)/2⌋
4  A_L = MergeSort(A[1 . . m])
5  A_R = MergeSort(A[m + 1 . . length(A)])
6  return Merge(A_L, A_R)
```

- ¿Cuál es la complejidad de **MergeSort**?

$$T(n) = 2T(n/2) + O(n)$$

Ordenamiento por mezcla (Merge Sort)

```
MergeSort(A)
1  if length(A) == 1
2        return A
3  m = ⌊length(A)/2⌋
4  A_L = MergeSort(A[1 . . m])
5  A_R = MergeSort(A[m + 1 . . length(A)])
6  return Merge(A_L, A_R)
```

- ¿Cuál es la complejidad de **MergeSort**?

$$T(n) = 2T(n/2) + O(n)$$

$$T(n) = O(n \log n)$$

Divide y vencerás

- **MergeSort** ilustra una estrategia clásica de ***divide y vencerás***

Divide y vencerás

- **MergeSort** ilustra una estrategia clásica de ***divide y vencerás***

- *Estrategia general:* dato un problema P para un conjunto de datos A
  - ***dividir*** la entrada A en partes $A_1, A_2, \ldots, A_k$ con $|A_i| < |A| = n$
  - ***resolver*** el problema P para cada una de las k partes
  - ***combinar*** las soluciones parciales para obtener la solución para A.

Divide y vencerás

- **MergeSort** ilustra una estrategia clásica de ***divide y vencerás***

- *Estrategia general:* dato un problema P para un conjunto de datos A
  - ***dividir*** la entrada A en partes $A_1, A_2, \ldots, A_k$ con $|A_i| < |A| = n$
  - ***resolver*** el problema P para cada una de las k partes
  - ***combinar*** las soluciones parciales para obtener la solución para A.

- Análisis de Complejidad

$$T(n) = T_{\text{dividir}} + \sum_{i=1}^{k} T(|A_i|) + T_{\text{combinar}}$$

mas adelante analizaremos esta fórmula…

```
MergeR(A, B)
1  if length(A) == 0
2       return B
3  if length(B) == 0
4       return A
5  if A[1] < B[1]
6       return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

Un Merge Divide-Vencerás

MergeR(A, B)

```
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)

Un Merge Divide-Vencerás

```
MergeR(A, B)
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)
- La complejidad de **MergeR** es

```
MergeR(A, B)
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)
- La complejidad de **MergeR** es

$$T(n) = C_1 + T(n-1)$$

MergeR(A, B)

```
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)
- La complejidad de **MergeR** es

$$T(n) = C_1 + T(n-1) = C_1 n$$

MergeR(A, B)

```
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)
- La complejidad de **MergeR** es

$$T(n) = C_1 + T(n-1) = C_1 n = O(n)$$

- ¿Lo podemos hacer mejor?

Un Merge Divide-Vencerás

MergeR(A, B)

```
1  if length(A) == 0
2        return B
3  if length(B) == 0
4        return A
5  if A[1] < B[1]
6        return A[1] ∘ MergeR(A[2 . . length(A)], B)
7  else return B[1] ∘ MergeR(A, B[2 . . length(B)])
```

- Nuevamente, este ejercicio es incorrecto (Exejcio: repararlo.)
- La complejidad de **MergeR** es

$$T(n) = C_1 + T(n-1) = C_1 n = O(n)$$

- ¿Lo podemos hacer mejor? No! (pero ya lo sabiamos)

Multiplicación Divide-Vencerás

Multiplicación Divide-Vencerás

- Regresemos al ejemplo de la multiplicación...

- Regresemos al ejemplo de la multiplicación...

$x =$ X_L X_R y $y =$ Y_L Y_R

- Regresemos al ejemplo de la multiplicación...

  $x =$ | X_L | X_R | y $y =$ | Y_L | Y_R |

  los que implica $x = 2^{\ell/2}x_L + x_R$ and $y = 2^{\ell/2}y_L + y_R$, entonces...

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

  se redujo el problema a multiplicar dos números de ℓ bits aun problema de multiplicar *cuatro* numeros de $\ell/2$ bits...

- Regresemos al ejemplo de la multiplicación...

$x =$ | X_L | X_R | y $y =$ | Y_L | Y_R |

los que implica $x = 2^{\ell/2}x_L + x_R$ and $y = 2^{\ell/2}y_L + y_R$, entonces...

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

se redujo el problema a multiplicar dos números de ℓ bits aun problema de multiplicar *cuatro* numeros de $\ell/2$ bits...

$$T(\ell) = 4T(\ell/2) + O(\ell)$$

- Regresemos al ejemplo de la multiplicación...

$x =$ | X_L | X_R | y $y =$ | Y_L | Y_R |

los que implica $x = 2^{\ell/2}x_L + x_R$ and $y = 2^{\ell/2}y_L + y_R$, entonces...

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

se redujo el problema a multiplicar dos números de ℓ bits aun problema de multiplicar *cuatro* numeros de $\ell/2$ bits...

$$T(\ell) = 4T(\ell/2) + O(\ell)$$

$$T(\ell) = \Theta(\ell^2)$$

Divide-and-Conquer Multiplication (2)

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_L y_L + 2^{\ell/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

pero note que $x_L y_R + x_R y_L = (x_L + x_R)(y_R + y_L) - x_L y_L - x_R y_R$, so

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

pero note que $x_Ly_R + x_Ry_L = (x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R$, so

$$xy = 2^{\ell}x_Ly_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R) + x_Ry_R$$

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

pero note que $x_Ly_R + x_Ry_L = (x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R$, so

$$xy = 2^{\ell}x_Ly_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R) + x_Ry_R$$

solo 3 multiplicaciones : x_Ly_L, $(x_L + x_R)(y_R + y_L)$, and x_Ry_R

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

pero note que $x_Ly_R + x_Ry_L = (x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R$, so

$$xy = 2^{\ell}x_Ly_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R) + x_Ry_R$$

solo 3 multiplicaciones : x_Ly_L, $(x_L + x_R)(y_R + y_L)$, and x_Ry_R

$$T(\ell) = 3T(\ell/2) + O(\ell)$$

Divide-and-Conquer Multiplication (2)

- Nuevamente, se tiene

$$\begin{aligned} xy &= (2^{\ell/2}x_L + x_R)(2^{\ell/2}y_L + y_R) \\ &= 2^{\ell}x_Ly_L + 2^{\ell/2}(x_Ly_R + x_Ry_L) + x_Ry_R \end{aligned}$$

pero note que $x_Ly_R + x_Ry_L = (x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R$, so

$$xy = 2^{\ell}x_Ly_L + 2^{\ell/2}((x_L + x_R)(y_R + y_L) - x_Ly_L - x_Ry_R) + x_Ry_R$$

solo 3 multiplicaciones : x_Ly_L, $(x_L + x_R)(y_R + y_L)$, and x_Ry_R

$$T(\ell) = 3T(\ell/2) + O(\ell)$$

lo cual, lleva a un complejidad de

$$T(\ell) = O(\ell^{\log_2 3}) = O(\ell^{1.59})$$

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto?

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto? Sí

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto? Sí

- ¿Cuál es su complejidad?

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto? Sí

- ¿Cuál es su complejidad? $T(n) = T_{\textbf{MergeSort}}(n) = O(n \log n)$

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto? Sí

- ¿Cuál es su complejidad? $T(n) = T_{\textbf{MergeSort}}(n) = O(n \log n)$

- ¿Es posible hacerlo mejor?

Calcular la Mediana

- La *mediana* de una secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores que m y la mitad son mayores que m
  - ej., cual es la mediana de $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

- Idea: primero ordenar, después tomar el elemento en la mitad.

```
SimpleMedian(A)
1  X = MergeSort(A)
2  return X[⌊length(A)/2⌋]
```

- ¿Es correcto? Sí

- ¿Cuál es su complejidad? $T(n) = T_{\textbf{MergeSort}}(n) = O(n \log n)$

- ¿Es posible hacerlo mejor? Intentemos una estrategia *divide y vencerás*-...

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

- Generalización, los ***k-smallest*** elementos de una secuencia A es un valor $v \in A$ tal que exactamente k elementos de A son menores o iguales a v

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

- Generalización, los ***k-smallest*** elementos de una secuencia A es un valor $v \in A$ tal que exactamente k elementos de A son menores o iguales a v

  Ej.,

  - para $k = 1$, el mínimo de A

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

- Generalización, los ***k-smallest*** elementos de una secuencia A es un valor $v \in A$ tal que exactamente k elementos de A son menores o iguales a v

  Ej.,

  - para $k = 1$, el mínimo de A
  - para $k = \lfloor |A|/2 \rfloor$, la mediana de A

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

- Generalización, los ***k-smallest*** elementos de una secuencia A es un valor $v \in A$ tal que exactamente k elementos de A son menores o iguales a v

  Ej.,

  - para $k = 1$, el mínimo de A
  - para $k = \lfloor |A|/2 \rfloor$, la mediana de A
  - ¿Cuál sería el *6th-smallest* elemento en $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

Calculando la Mediana (2)

- La *mediana* de una a secuencia A es el valor $m \in A$ tal que la mitad de los valores en A son menores o igual a m

- Generalización, los ***k-smallest*** elementos de una secuencia A es un valor $v \in A$ tal que exactamente k elementos de A son menores o iguales a v

  Ej.,

  - para $k = 1$, el mínimo de A
  - para $k = \lfloor |A|/2 \rfloor$, la mediana de A
  - ¿Cuál sería el *6th-smallest* elemento en $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$?

    el 6th-smallest elemento de A—a.k.a. $select(A, 6)$—es 8

El *k*-ésimo elemento más pequeño (k-Smallest)

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

$$A_L = \langle 2, 4, 1 \rangle$$

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle$$

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

Ahora, donde está el 7th-smallest valor de A?

El k-ésimo elemento más pequeño (k-Smallest)

- Idea: Se divide la secuencia A en tres partes basados en *elegir un valor* $v \in A$
  - A_L contiene el conjunto de los elmentos tal que *menores que* v
  - A_v contiene el conjunto de los elementos que son *iguales a* v
  - A_R contiene el conjunto de los elemento que son of *mayores que* v

Ej., $A = \langle 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1 \rangle$
vamos a calcular el 7th-smallest valore en A
tomamos y dividimos usando por ejemplo $v = 5$

$$A_L = \langle 2, 4, 1 \rangle \quad A_v = \langle 5, 5 \rangle \quad A_R = \langle 36, 21, 8, 13, 11, 20 \rangle$$

Ahora, donde está el 7th-smallest valor de A?
¿Es el 2nd-smallest valor de A_R

k-Smallest (2)

Uitilizaremos $select(A, k)$ para denominar el denote el k-smallest elemento en A

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

k-Smallest (2)

Uitilizaremos $select(A, k)$ para denominar el denote el k-smallest elemento en A

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Calcular A_L, A_v, y A_R toma $O(n)$

k-Smallest (2)

Uitilizaremos $select(A, k)$ para denominar el denote el k-smallest elemento en A

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Calcular A_L, A_v, y A_R toma $O(n)$
- ¿Como escogemos v?

k-Smallest (2)

Uitilizaremos $select(A, k)$ para denominar el denote el k-smallest elemento en A

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Calcular A_L, A_v, y A_R toma $O(n)$
- ¿Como escogemos v?
- Idealmente, deberíamos elegir v para que $|A_L| \approx |A_R| \approx |A|/2$
  - idealmente deberíamos elegir $v = median(A)$, pero…

k-Smallest (2)

Uitilizaremos $select(A, k)$ para denominar el denote el k-smallest elemento en A

$$select(A, k) = \begin{cases} select(A_L, k) & \text{if } k \leq |A_L| \\ v & \text{if } |A_L| < k \leq |A_L| + |A_v| \\ select(A_R, k - |A_L| - |A_v|) & \text{if } k > |A_L| + |A_v| \end{cases}$$

- Calcular A_L, A_v, y A_R toma $O(n)$
- ¿Como escogemos v?
- Idealmente, deberíamos elegir v para que $|A_L| \approx |A_R| \approx |A|/2$
  - idealmente deberíamos elegir $v = median(A)$, pero...
- Elegimos *un elemento aleatorio de A*

Algoritmo de selección

Selection(A, k)

```
1   v = A[random(1 ... |A|)]
2   A_L, A_v, A_R = ∅
3   for i = 1 to |A|
4       if A[i] < v
5           A_L = A_L ∪ A[i]
6       elseif A[i] == v
7           A_v = A_v ∪ A[i]
8       else A_R = A_R ∪ A[i]
9   if k ≤ |A_L|
10      return Selection(A_L, k)
11  elseif k > |A_L| + |A_v|
12      return Selection(A_R, k − |A_L| − |A_v|)
13  else return v
```