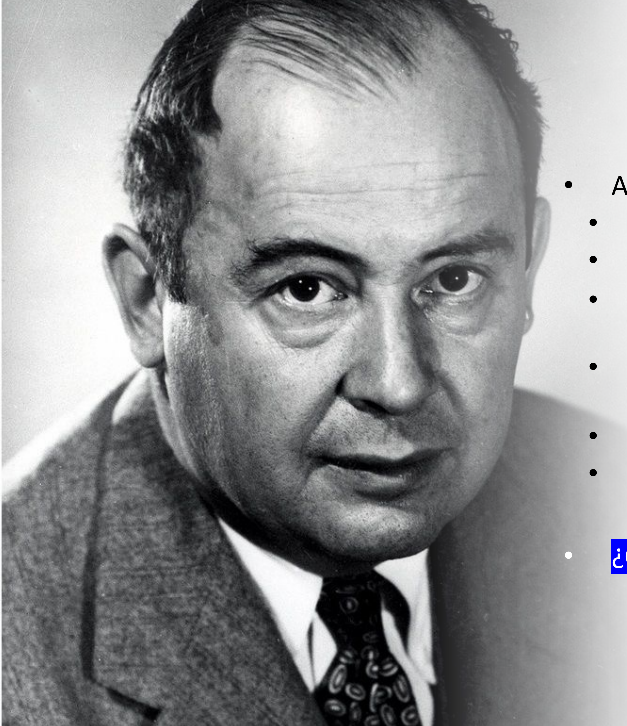
… previamente en IIC2133

El algoritmo MergeSort

A continuación tenemos el pseudocódigo del algoritmo recursivo `MergeSort`

input : Secuencia A
output: Secuencia ordenada B

```
MergeSort (A):
    if |A| = 1 : return A
    Dividir A en mitades A_1 y A_2
    B_1 ← MergeSort(A_1)
    B_2 ← MergeSort(A_2)
    B ← Merge(B_1, B_2)
    return B
```



Merge Sort

- Atributos generales
  - O(n log n)
  - O(n) en memoria
  - Inventado por John von Neumann - 1945
  - Aplica Dividir para Conquistar
  - Estable (*)
  - Cintas magnéticas (**)

- ¿Cuándo usar Merge Sort?

Complejidad de algoritmos de ordenación

Resumimos los resultados de complejidad por caso hasta el momento

Algoritmo	Mejor caso	Caso promedio	Peor caso	Memoria adicional
Selection Sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Insertion Sort	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Merge Sort	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n)$
Quick Sort	?	?	?	?
Heap Sort	?	?	?	?

Notemos la mejora en tiempo con `MergeSort`
a cambio de memoria adicional

QuickSort

Clase 04

IIC 2133 coordinado

Prof. Yadran Eterovic, Mario Droguett, Eduardo Bustos

Sumario

Introducción

Particiones

Quicksort

Cierre

Dividir para conquistar


divide and conquer...

Dividir para conquistar

La estrategia sigue los siguientes pasos

Dividir para conquistar

La estrategia sigue los siguientes pasos

1. Dividir el problema original en dos (o más) **sub-problemas** del mismo tipo

Dividir para conquistar

La estrategia sigue los siguientes pasos

1. Dividir el problema original en dos (o más) **sub-problemas** del mismo tipo
2. Resolver **recursivamente** cada sub-problema

Dividir para conquistar

La estrategia sigue los siguientes pasos

1. Dividir el problema original en dos (o más) **sub-problemas** del mismo tipo
2. Resolver **recursivamente** cada sub-problema
3. Encontrar solución al problema original **combinando** las soluciones a los sub-problemas

Búsqueda binaria

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

input : Secuencia $A[0, \ldots, n-1]$, elemento x, índices i, f
output: Índice $m \in \{0, \ldots, n-1\}$ o -1

`BinarySearch` (A, x, i, f)**:**

    if $f < i$ **: return** -1
    $m \leftarrow \left\lfloor \dfrac{i+f}{2} \right\rfloor$
    if $A[m] = x$ **: return** m
    if $A[m] > x$ **:**
        return `BinarySearch` $(A, x, i, m-1)$
    return `BinarySearch` $(A, x, m+1, f)$

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

BSearch (A, x, i, f)**:**

    if $f < i$ **: return** -1

    $m \leftarrow \left\lfloor \dfrac{i+f}{2} \right\rfloor$

    if $A[m] = x$ **: return** m

    if $A[m] > x$ **:**

        return BSearch $(A, x, i, m-1)$

    return BSearch $(A, x, m+1, f)$

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

```
BSearch (A, x, i, f):
1   if f < i : return −1
2   m ← ⌊(i + f) / 2⌋
3   if A[m] = x : return m
4   if A[m] > x :
5       return BSearch (A, x, i, m − 1)
6   return BSearch (A, x, m + 1, f)
```

- La división en subproblemas se hace escogiendo un **pivote** central (línea 2)

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

BSearch (A, x, i, f)**:**

1 **if** $f < i$ **: return** -1

2 $m \leftarrow \left\lfloor \frac{i+f}{2} \right\rfloor$

3 **if** $A[m] = x$ **: return** m

4 **if** $A[m] > x$ **:**

5 **return** BSearch $(A, x, i, m-1)$

6 **return** BSearch $(A, x, m+1, f)$

- La división en subproblemas se hace escogiendo un **pivote** central (línea 2)
- La naturaleza de la búsqueda hace que sea necesario resolver **solo uno de los subproblemas**

Búsqueda binaria

El algoritmo de **búsqueda binaria** también está basado en la estrategia dividir para conquistar

`BSearch` (A, x, i, f):

1 **if** $f < i$ **: return** -1

2 $m \leftarrow \left\lfloor \dfrac{i+f}{2} \right\rfloor$

3 **if** $A[m] = x$ **: return** m

4 **if** $A[m] > x$ **:**

5 **return** `BSearch` $(A, x, i, m-1)$

6 **return** `BSearch` $(A, x, m+1, f)$

- La división en subproblemas se hace escogiendo un **pivote** central (línea 2)
- La naturaleza de la búsqueda hace que sea necesario resolver **solo uno de los subproblemas**
- Solución al problema es exactamente la solución al llamado recursivo

Un problema para inspirarnos

Un problema para inspirarnos

- Consideremos el problema de procesar un gran conjunto de coordenadas en 2D

Un problema para inspirarnos

- Consideremos el problema de procesar un gran conjunto de coordenadas en 2D
- Para repartir la carga, se reparten los datos en zonas rectangulares

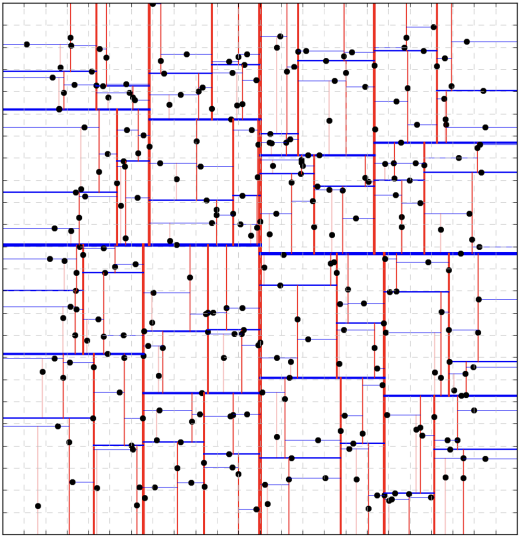
Un problema para inspirarnos

- Consideremos el problema de procesar un gran conjunto de coordenadas en 2D
- Para repartir la carga, se reparten los datos en zonas rectangulares
- La idea es que los rectángulos **particionen** el espacio 2D

Un problema para inspirarnos

- Consideremos el problema de procesar un gran conjunto de coordenadas en 2D
- Para repartir la carga, se reparten los datos en zonas rectangulares
- La idea es que los rectángulos **particionen** el espacio 2D
- Además queremos que cada zona tenga la misma cantidad de datos

Un problema para inspirarnos



Objetivos de la clase

Objetivos de la clase

- ☐ Comprender la estrategia de elegir pivote para particionar una secuencia

Objetivos de la clase

- ☐ Comprender la estrategia de elegir pivote para particionar una secuencia
- ☐ Identificar cuándo tal estrategia encuentra la mediana

Objetivos de la clase

- ☐ Comprender la estrategia de elegir pivote para particionar una secuencia
- ☐ Identificar cuándo tal estrategia encuentra la mediana
- ☐ Comprender y analizar el algoritmo `Partition`

Objetivos de la clase

- ☐ Comprender la estrategia de elegir pivote para particionar una secuencia
- ☐ Identificar cuándo tal estrategia encuentra la mediana
- ☐ Comprender y analizar el algoritmo `Partition`
- ☐ Comprender el uso de `Partition` en `Median` para encontrar el elemento central

Objetivos de la clase

- Comprender la estrategia de elegir pivote para particionar una secuencia
- Identificar cuándo tal estrategia encuentra la mediana
- Comprender y analizar el algoritmo `Partition`
- Comprender el uso de `Partition` en `Median` para encontrar el elemento central
- Comprender el uso de `Partition` para ordenar secuencias

Sumario

Introducción

Particiones

Quicksort

Cierre

El desafío de encontrar la mediana

El desafío de encontrar la mediana

- En el contexto de una secuencia de valores

El desafío de encontrar la mediana

- En el contexto de una secuencia de valores
- ¿Cómo encontramos la mediana?

El desafío de encontrar la mediana

- En el contexto de una secuencia de valores
- ¿Cómo encontramos la mediana?
- Si ordenamos la secuencia: es la mitad...

El desafío de encontrar la mediana

- En el contexto de una secuencia de valores
- ¿Cómo encontramos la mediana?
- Si ordenamos la secuencia: es la mitad...
- ¿Podemos hacerlo sin recurrir a ordenar los datos?

El desafío de encontrar la mediana

Definición

Dada una secuencia **ordenada** de n valores $a_0, a_1, \ldots, a_{n-1}$, llamaremos **mediana** al valor M_e dado por

El desafío de encontrar la mediana

Definición

Dada una secuencia **ordenada** de n valores $a_0, a_1, \ldots, a_{n-1}$, llamaremos **mediana** al valor M_e dado por

- Si n es impar,

$$M_e = a_{\lfloor n/2 \rfloor}$$

El desafío de encontrar la mediana

Definición

Dada una secuencia **ordenada** de n valores $a_0, a_1, \ldots, a_{n-1}$, llamaremos **mediana** al valor M_e dado por

- Si n es impar,

$$M_e = a_{\lfloor n/2 \rfloor}$$

- Si n es par,

$$M_e = \frac{a_{n/2} + a_{n/2-1}}{2}$$

El desafío de encontrar la mediana

Definición

Dada una secuencia **ordenada** de n valores $a_0, a_1, \ldots, a_{n-1}$, llamaremos **mediana** al valor M_e dado por

- Si n es impar,

$$M_e = a_{\lfloor n/2 \rfloor}$$

- Si n es par,

$$M_e = \frac{a_{n/2} + a_{n/2-1}}{2}$$

La mediana es tal que en la secuencia hay la
misma cantidad de elementos mayores y menores a ella

Primera estrategia

Primera estrategia

Dado un elemento de la secuencia al cual llamaremos **pivote**

Primera estrategia

Dado un elemento de la secuencia al cual llamaremos **pivote**

1. ¿El pivote es la mediana?

Primera estrategia

Dado un elemento de la secuencia al cual llamaremos **pivote**

1. ¿El pivote es la mediana?
2. Si no, ¿la mediana es mayor o menor al pivote?

Primera estrategia

Dado un elemento de la secuencia al cual llamaremos **pivote**

1. ¿El pivote es la mediana?
2. Si no, ¿la mediana es mayor o menor al pivote?

Si escogemos el pivote de manera aleatoria, ¿cómo respondemos a estas preguntas?

Primera estrategia

Dado un elemento de la secuencia al cual llamaremos **pivote**

1. ¿El pivote es la mediana?
2. Si no, ¿la mediana es mayor o menor al pivote?

Si escogemos el pivote de manera aleatoria, ¿cómo respondemos a estas preguntas?

Necesitamos el número de elementos menores y mayores que el pivote

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

0	1	2	3	4	5	6	7	8	9	10
7	3	5	2	1	9	7	8	6	4	3

y el **pivote** aleatorio $\boxed{6}$

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

y el **pivote** aleatorio $\boxed{6}$

- Elementos menores:

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

y el **pivote** aleatorio $\boxed{6}$

- Elementos menores: $\boxed{3}$ $\boxed{5}$ $\boxed{2}$ $\boxed{1}$ $\boxed{4}$ $\boxed{3}$

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

y el **pivote** aleatorio $\boxed{6}$

- Elementos menores: $\boxed{3}$ $\boxed{5}$ $\boxed{2}$ $\boxed{1}$ $\boxed{4}$ $\boxed{3}$
- Elementos mayores:

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

y el **pivote** aleatorio 6

- Elementos menores: 3 5 2 1 4 3
- Elementos mayores: 7 9 7 8

Primera estrategia...

Ejemplo

Dada la siguiente secuencia de datos (de largo impar)

7	3	5	2	1	9	7	8	6	4	3
0	1	2	3	4	5	6	7	8	9	10

y el **pivote** aleatorio $\boxed{6}$

- Elementos menores: $\boxed{3}$ $\boxed{5}$ $\boxed{2}$ $\boxed{1}$ $\boxed{4}$ $\boxed{3}$
- Elementos mayores: $\boxed{7}$ $\boxed{9}$ $\boxed{7}$ $\boxed{8}$

Con esta información, ¿es 6 la mediana? ¿dónde se ubica esta?

Primera estrategia... recursiva

Primera estrategia... recursiva

Moviendo los elementos según si son mayores o menores que el pivote,

Primera estrategia... recursiva

Moviendo los elementos según si son mayores o menores que el pivote,

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Moviendo los elementos según si son mayores o menores que el pivote,

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Como la posición de la mediana debiera ser $\lfloor 11/2 \rfloor = 5$ y el pivote quedó en la posición 6, revisamos el tramo a la izquierda del pivote

Primera estrategia... recursiva

Moviendo los elementos según si son mayores o menores que el pivote,

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Como la posición de la mediana debiera ser $\lfloor 11/2 \rfloor = 5$ y el pivote quedó en la posición 6, revisamos el tramo a la izquierda del pivote

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Tomando aleatoriamente el pivote $\boxed{2}$

Primera estrategia... recursiva

Tomando aleatoriamente el pivote $\boxed{2}$

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Tomando aleatoriamente el pivote $\boxed{2}$

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores:

Primera estrategia... recursiva

Tomando aleatoriamente el pivote $\boxed{2}$

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: $\boxed{1}$

Primera estrategia… recursiva

Tomando aleatoriamente el pivote 2

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 1
- Elementos mayores:

Primera estrategia... recursiva

Tomando aleatoriamente el pivote 2

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 1
- Elementos mayores: 3 5 4 3

Primera estrategia... recursiva

Tomando aleatoriamente el pivote $\boxed{2}$

3	5	2	1	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 1
- Elementos mayores: 3 5 4 3

Reordenamos el tramo analizado para ver si el pivote es la mediana

Primera estrategia... recursiva

Ubicamos los elementos según el pivote 2, obteniendo

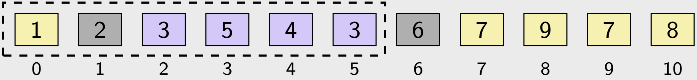
Primera estrategia... recursiva

Ubicamos los elementos según el pivote 2, obteniendo

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Ubicamos los elementos según el pivote 2, obteniendo



Nuevamente, como la posición del pivote es 1, nos concentramos recursivamente en el sub-tramo derecho del último pivote

Primera estrategia... recursiva

Ubicamos los elementos según el pivote 2, obteniendo

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Nuevamente, como la posición del pivote es 1, nos concentramos recursivamente en el sub-tramo derecho del último pivote

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Con el pivote $\boxed{5}$

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Con el pivote $\boxed{5}$

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores:

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 3 4 3

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 3 4 3
- Elementos mayores:

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 3 4 3
- Elementos mayores: $\varnothing$

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 3 4 3
- Elementos mayores: $\varnothing$

Reubicamos los elementos para ver la posición del pivote actual

Primera estrategia... recursiva

Con el pivote 5

1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: 3 4 3
- Elementos mayores: $\varnothing$

Reubicamos los elementos para ver la posición del pivote actual

1	2	3	4	3	5	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Primera estrategia... recursiva

Con el pivote $\boxed{5}$

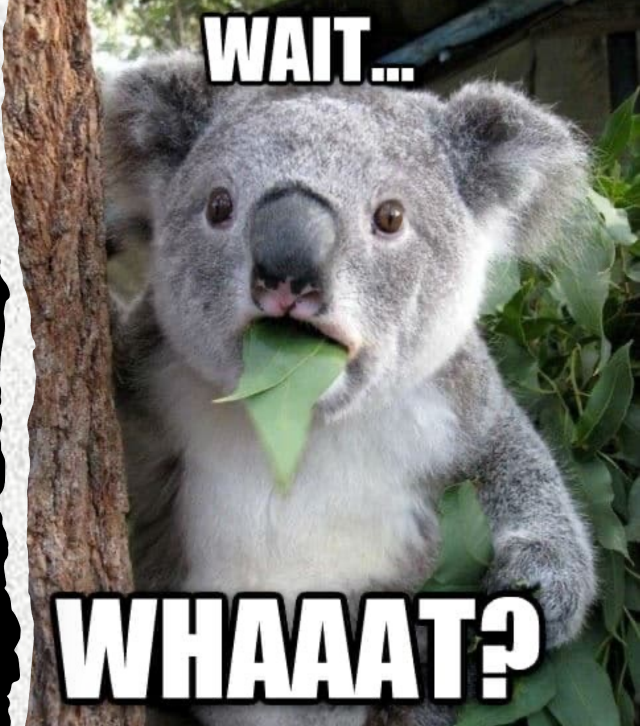
1	2	3	5	4	3	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

- Elementos menores: $\boxed{3}$ $\boxed{4}$ $\boxed{3}$
- Elementos mayores: $\varnothing$

Reubicamos los elementos para ver la posición del pivote actual

1	2	3	4	3	5	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

Como el pivote está en la posición $\lfloor n/2 \rfloor = \lfloor 11/2 \rfloor = 5$, **es la mediana**


WAIT...
WHAAAT?

Particiones

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Luego, si reubicamos los elementos de la secuencia de forma que

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Luego, si reubicamos los elementos de la secuencia de forma que

- Primero se colocan los elementos de m

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Luego, si reubicamos los elementos de la secuencia de forma que

- Primero se colocan los elementos de m
- Luego se coloca p

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Luego, si reubicamos los elementos de la secuencia de forma que

- Primero se colocan los elementos de m
- Luego se coloca p
- Finalmente se colocan los elementos de M

Particiones

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p **particiona** la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

Luego, si reubicamos los elementos de la secuencia de forma que

- Primero se colocan los elementos de m
- Luego se coloca p
- Finalmente se colocan los elementos de M

es claro que p está en su posición ordenada

Partition

Partition

Supongamos que no hay elementos repetidos en la secuencia

Partition

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \dots, n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

$\texttt{Partition}\ (A, i, f)$**:**
    $p \leftarrow$ pivote aleatorio en $A[i, f]$
    $m, M \leftarrow$ secuencias vacías
    Insertar p en M
    for $x \in A[i, f]$ **:**
        if $x < p$ **:** Insertar x en m
        elif $x > p$ **:** Insertar x en M
    $A[i, f] \leftarrow \texttt{Concat}(m, M)$
    return $i + |m|$

Partition

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \dots, n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A, i, f)**:**

    $p \leftarrow$ pivote aleatorio en $A[i, f]$
    $m, M \leftarrow$ secuencias vacías
    Insertar p en M
    for $x \in A[i, f]$ **:**
        if $x < p$ **:** Insertar x en m
        elif $x > p$ **:** Insertar x en M
    $A[i, f] \leftarrow \texttt{Concat}(m, M)$
    return $i + |m|$

Notemos que `Partition` retorna y además reordena los elementos de A

Partition

Supongamos que no hay elementos repetidos en la secuencia

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

Partition

Supongamos que no hay elementos repetidos en la secuencia

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

- Se ubican los menores, el pivote y luego los mayores

Partition

Supongamos que no hay elementos repetidos en la secuencia

`Partition` (A, i, f)**:**

  $p \leftarrow$ pivote aleatorio en $A[i, f]$
  $m, M \leftarrow$ secuencias vacías
  Insertar p en M
  for $x \in A[i, f]$ **:**
    if $x < p$ **:** Insertar x en m
    elif $x > p$ **:** Insertar x en M
  $A[i, f] \leftarrow \texttt{Concat}(m, M)$
  return $i + |m|$

- Se ubican los menores, el pivote y luego los mayores
- i: # elems. a la izq. de $A[i]$

Partition

Supongamos que no hay elementos repetidos en la secuencia

$\texttt{Partition}\ (A, i, f)$**:**

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
    if $x < p$ **:** Insertar x en m
    elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow \texttt{Concat}(m, M)$
return $i + |m|$

- Se ubican los menores, el pivote y luego los mayores
- i: # elems. a la izq. de $A[i]$
- $|m|$: # menores a p en $A[i, f]$

Partition

Supongamos que no hay elementos repetidos en la secuencia

`Partition` (A, i, f):

- $p \leftarrow$ pivote aleatorio en $A[i, f]$
- $m, M \leftarrow$ secuencias vacías
- Insertar p en M
- **for** $x \in A[i, f]$ **:**
  - **if** $x < p$ **:** Insertar x en m
  - **elif** $x > p$ **:** Insertar x en M
- $A[i, f] \leftarrow$ `Concat`(m, M)
- **return** $i + |m|$

- Se ubican los menores, el pivote y luego los mayores
- i: # elems. a la izq. de $A[i]$
- $|m|$: # menores a p en $A[i, f]$
- Por lo tanto, se retorna la posición correcta de p

Partition

Supongamos que no hay elementos repetidos en la secuencia

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
    if $x < p$ **:** Insertar x en m
    elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow \texttt{Concat}(m, M)$
return $i + |m|$

- Se ubican los menores, el pivote y luego los mayores
- i: # elems. a la izq. de $A[i]$
- $|m|$: # menores a p en $A[i, f]$
- Por lo tanto, se retorna la posición correcta de p

Su retorno es la cantidad de elementos a la izquierda de p en la secuencia ordenada

Complejidad de `Partition`

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

Complejidad de `Partition`

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$

$m, M \leftarrow$ secuencias vacías

Insertar p en M

for $x \in A[i, f]$ **:**

  if $x < p$ **:** Insertar x en m

  elif $x > p$ **:** Insertar x en M

$A[i, f] \leftarrow \texttt{Concat}(m, M)$

return $i + |m|$

Las inserciones al final de la EDD

Complejidad de `Partition`

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
  if $x < p$ **:** Insertar x en m
  elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow \texttt{Concat}(m, M)$
return $i + |m|$

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas

Complejidad de `Partition`

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x

Complejidad de `Partition`

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

Complejidad de `Partition`

`Partition` (A, i, f):

    $p \leftarrow$ pivote aleatorio en $A[i, f]$
    $m, M \leftarrow$ secuencias vacías
    Insertar p en M
    for $x \in A[i, f]$ **:**
        if $x < p$ **:** Insertar x en m
        elif $x > p$ **:** Insertar x en M
    $A[i, f] \leftarrow \texttt{Concat}(m, M)$
    return $i + |m|$

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

La concatenación

Complejidad de `Partition`

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
  if $x < p$ **:** Insertar x en m
  elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow \texttt{Concat}(m, M)$
return $i + |m|$

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

La concatenación

- $\mathcal{O}(1)$ en listas

Complejidad de `Partition`

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
  if $x < p$ **:** Insertar x en m
  elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow$ `Concat`(m, M)
return $i + |m|$

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

La concatenación

- $\mathcal{O}(1)$ en listas
- $\mathcal{O}(n)$ en arreglos

Complejidad de `Partition`

```
Partition (A, i, f):
    p ← pivote aleatorio en A[i, f]
    m, M ← secuencias vacías
    Insertar p en M
    for x ∈ A[i, f] :
        if x < p : Insertar x en m
        elif x > p : Insertar x en M
    A[i, f] ← Concat(m, M)
    return i + |m|
```

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

La concatenación

- $\mathcal{O}(1)$ en listas
- $\mathcal{O}(n)$ en arreglos

Además, usa memoria adicional $\mathcal{O}(n)$ para mantener las secuencias m, M

Complejidad de `Partition`

`Partition` (A, i, f):

$p \leftarrow$ pivote aleatorio en $A[i, f]$
$m, M \leftarrow$ secuencias vacías
Insertar p en M
for $x \in A[i, f]$ **:**
  if $x < p$ **:** Insertar x en m
  elif $x > p$ **:** Insertar x en M
$A[i, f] \leftarrow$ `Concat`(m, M)
return $i + |m|$

Las inserciones al final de la EDD

- $\mathcal{O}(1)$ en arreglos y listas
- Se hace una por elemento x
- Total: $\mathcal{O}(n)$

La concatenación

- $\mathcal{O}(1)$ en listas
- $\mathcal{O}(n)$ en arreglos

Además, usa memoria adicional $\mathcal{O}(n)$ para mantener las secuencias m, M

Más adelante veremos una versión *in place* de `Partition`

Median

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \ldots, n-1]$
output: Elemento en posición central de A

```
Median (A):
    i ← 0
    f ← n − 1
    x ← Partition(A, i, f)
    while x no es el centro :
        if x < centro : i ← x + 1
        else: f ← x − 1
        x ← Partition(A, i, f)
    return A[x]
```

Median

Supongamos que no hay elementos repetidos en la secuencia

`Median` (A)**:**

  $i \leftarrow 0$

  $f \leftarrow n - 1$

  $x \leftarrow \texttt{Partition}(A, i, f)$

  while x *no es el centro* **:**

    if $x <$ *centro* **:** $i \leftarrow x + 1$

    else: $f \leftarrow x - 1$

    $x \leftarrow \texttt{Partition}(A, i, f)$

  return $A[x]$

Median

Supongamos que no hay elementos repetidos en la secuencia

`Median` (A):

- $i \leftarrow 0$
- $f \leftarrow n - 1$
- $x \leftarrow \texttt{Partition}(A, i, f)$
- **while** x *no es el centro* **:**
  - **if** $x <$ *centro* **:** $i \leftarrow x + 1$
  - **else:** $f \leftarrow x - 1$
  - $x \leftarrow \texttt{Partition}(A, i, f)$
- **return** $A[x]$

- x: # datos menores al pivote

Median

Supongamos que no hay elementos repetidos en la secuencia

`Median` (A):

$i \leftarrow 0$
$f \leftarrow n - 1$
$x \leftarrow \texttt{Partition}(A, i, f)$
while x *no es el centro* **:**
    if $x <$ *centro* **:** $i \leftarrow x + 1$
    else: $f \leftarrow x - 1$
    $x \leftarrow \texttt{Partition}(A, i, f)$
return $A[x]$

- x: # datos menores al pivote
- `Median` entrega el elemento en la posición central ordenada

Median

Supongamos que no hay elementos repetidos en la secuencia

`Median` (A):

$i \leftarrow 0$
$f \leftarrow n - 1$
$x \leftarrow \texttt{Partition}(A, i, f)$
while x *no es el centro* **:**
    if $x <$ *centro* **:** $i \leftarrow x + 1$
    else: $f \leftarrow x - 1$
    $x \leftarrow \texttt{Partition}(A, i, f)$
return $A[x]$

- x: # datos menores al pivote
- `Median` entrega el elemento en la posición central ordenada
- Para n impar, `Median`(A) corresponde a la mediana

Median

Supongamos que no hay elementos repetidos en la secuencia

`Median` (A)**:**
    $i \leftarrow 0$
    $f \leftarrow n - 1$
    $x \leftarrow \texttt{Partition}(A, i, f)$
    while x *no es el centro* **:**
        if $x < centro$ **:** $i \leftarrow x + 1$
        else: $f \leftarrow x - 1$
        $x \leftarrow \texttt{Partition}(A, i, f)$
    return $A[x]$

- x: # datos menores al pivote
- `Median` entrega el elemento en la posición central ordenada
- Para n impar, `Median`(A) corresponde a la mediana
- Para n par, `Median`(A) es uno de los dos elementos centrales

Median

Supongamos que no hay elementos repetidos en la secuencia

```
Median (A):
    i ← 0
    f ← n − 1
    x ← Partition(A, i, f)
    while x no es el centro :
        if x < centro : i ← x + 1
        else: f ← x − 1
        x ← Partition(A, i, f)
    return A[x]
```

- x: # datos menores al pivote
- `Median` entrega el elemento en la posición central ordenada
- Para n impar, `Median`(A) corresponde a la mediana
- Para n par, `Median`(A) es uno de los dos elementos centrales

Podemos parametrizar `Median` para encontrar el k-ésimo estadístico de orden

Median (versión parametrizada)

`Median` (versión parametrizada)

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \ldots, n-1]$, índice $k \in \{0, \ldots, n-1\}$
output: k-ésimo estadístico de orden de A

`Median` (A, k)**:**
    $i \leftarrow 0$
    $f \leftarrow n-1$
    $x \leftarrow \texttt{Partition}(A, i, f)$
    while $x \neq k$ **:**
        if $x < k$ **:** $i \leftarrow x+1$
        else: $f \leftarrow x-1$
        $x \leftarrow \texttt{Partition}(A, i, f)$
    return $A[x]$

Median (versión parametrizada)

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \ldots, n-1]$, índice $k \in \{0, \ldots, n-1\}$
output: k-ésimo estadístico de orden de A

`Median` (A, k):

$i \leftarrow 0$
$f \leftarrow n - 1$
$x \leftarrow \texttt{Partition}(A, i, f)$
while $x \neq k$ **:**
  if $x < k$ **:** $i \leftarrow x + 1$
  else: $f \leftarrow x - 1$
  $x \leftarrow \texttt{Partition}(A, i, f)$
return $A[x]$

- `Median`$(A, 0)$ entrega el menor elemento de A

`Median` (versión parametrizada)

Supongamos que no hay elementos repetidos en la secuencia

input : Secuencia $A[0, \ldots, n-1]$, índice $k \in \{0, \ldots, n-1\}$
output: k-ésimo estadístico de orden de A

`Median` (A, k)**:**

$i \leftarrow 0$
$f \leftarrow n - 1$
$x \leftarrow \texttt{Partition}(A, i, f)$
while $x \neq k$ **:**
  if $x < k$ **:** $i \leftarrow x + 1$
  else: $f \leftarrow x - 1$
  $x \leftarrow \texttt{Partition}(A, i, f)$
return $A[x]$

- `Median`$(A, 0)$ entrega el menor elemento de A
- `Median`$(A, n-1)$ entrega el mayor elemento de A

`Median` y dividir para conquistar


divide and conquer...

`Median` y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

`Median` y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)

Median y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)
2. Naturaleza del problema hace que solo sea necesario resolver uno de los dos sub-problemas

Median y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)
2. Naturaleza del problema hace que solo sea necesario resolver uno de los dos sub-problemas
   - Similar al caso de búsqueda binaria

`Median` y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)
2. Naturaleza del problema hace que solo sea necesario resolver uno de los dos sub-problemas
    - Similar al caso de búsqueda binaria
    - Resolvemos el sub-problema correspondiente a la zona donde debiera estar la mediana

`Median` y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)
2. Naturaleza del problema hace que solo sea necesario resolver uno de los dos sub-problemas
   - Similar al caso de búsqueda binaria
   - Resolvemos el sub-problema correspondiente a la zona donde debiera estar la mediana
   - Se puede cambiar el enfoque recursivo por uno iterativo

Median y dividir para conquistar

El algoritmo `Median` también aplica la estrategia dividir para conquistar

1. División del problema en dos sub-problemas (`Partition`)
2. Naturaleza del problema hace que solo sea necesario resolver uno de los dos sub-problemas
    - Similar al caso de búsqueda binaria
    - Resolvemos el sub-problema correspondiente a la zona donde debiera estar la mediana
    - Se puede cambiar el enfoque recursivo por uno iterativo
3. Solución al problema original se obtiene de la solución al sub-problema

Complejidad de `Median`

Discusión

¿Cuál es la complejidad de `Median`?

- ¿Importa el orden de A?
- ¿Depende de algo distinto?
- ¿Cuáles serían su mejor y peor caso?

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$
- Como solo se debe tomar un pivote: Total: $\mathcal{O}(n)$

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$
- Como solo se debe tomar un pivote: Total: $\mathcal{O}(n)$

El **peor caso**

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$
- Como solo se debe tomar un pivote: Total: $\mathcal{O}(n)$

El **peor caso**

- Que todos los datos menos la mediana sean escogidos como pivote

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$
- Como solo se debe tomar un pivote: Total: $\mathcal{O}(n)$

El **peor caso**

- Que todos los datos menos la mediana sean escogidos como pivote
- Para cada pivote hay que separar menores y mayores $\mathcal{O}(n)$

Complejidad de `Median`

La complejidad de `Median` depende la elección del pivote: es **probabilística**

El **mejor caso**

- Escoger la mediana como primer pivote
- Esto significa separar los demás elementos en m y M
- Eso toma $\mathcal{O}(n)$ por pivote $\mathcal{O}(n)$
- Como solo se debe tomar un pivote: Total: $\mathcal{O}(n)$

El **peor caso**

- Que todos los datos menos la mediana sean escogidos como pivote
- Para cada pivote hay que separar menores y mayores $\mathcal{O}(n)$
- Como solo se deben tomar $\mathcal{O}(n)$ pivotes: Total: $\mathcal{O}(n^2)$

Ordenar con `Partition`


WAIT...
WHAAAT?

Ordenar con `Partition`

Luego de cada ejecución de `Partition`, el pivote queda en su posición ordenada

1	2	3	4	3	5	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

1	2	3	5	4
0	1	2	3	4

Ordenar con `Partition`

Luego de cada ejecución de `Partition`, el pivote queda en su posición ordenada

1	2	3	4	3	5	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

1	2	3	5	4
0	1	2	3	4

Además, las dos sub-secuencias están "*del lado correcto*" del pivote

Ordenar con `Partition`

Luego de cada ejecución de `Partition`, el pivote queda en su posición ordenada

1	2	3	4	3	5	6	7	9	7	8
0	1	2	3	4	5	6	7	8	9	10

1	2	3	5	4
0	1	2	3	4

Además, las dos sub-secuencias están "*del lado correcto*" del pivote

¿Cómo usar esto para ordenar?

Sumario

Introducción

Particiones

Quicksort

Cierre

Ordenar con `Partition`

Nuevamente podemos usar la estrategia dividir para conquistar


divide and conquer…

Ordenar con `Partition`

Nuevamente podemos usar la estrategia dividir para conquistar

1. División del problema en dos sub-problemas con `Partition`
2. Aplicar recursivamente `Partition` para cada sub-secuencia
3. No necesitamos combinar nada: la secuencia queda ordenada

Llamaremos **Quicksort** a este algoritmo

Quicksort

input : Secuencia $A[0, \ldots, n-1]$, índices i, f
output: $\varnothing$

```
QuickSort (A, i, f):
    if i ≤ f :
        p ← Partition(A, i, f)
        Quicksort(A, i, p − 1)
        Quicksort(A, p + 1, f)
```

Quicksort

input : Secuencia $A[0, \ldots, n-1]$, índices i, f

output: $\varnothing$

```
QuickSort (A, i, f):
    if i ≤ f :
        p ← Partition(A, i, f)
        Quicksort(A, i, p − 1)
        Quicksort(A, p + 1, f)
```

El llamado inicial es `Quicksort`$(A, 0, n-1)$

Quicksort: Ejemplo de ejecución

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	E	**E**	T	I	N	G	O	X	S	M	P	L	R
A	A	**E**												
A	**A**													
A														
				L	I	N	G	O	P	M	**R**	X	T	S
				L	I	G	**M**	O	P	N				
				G	I	L								
				—	I	**L**								
				—	**I**									
								N	P	O				
								—	**O**	P				
								—	—	**P**				
												S	T	X
												—	T	**X**
												—	**T**	
A	**A**	**E**	**E**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**R**	**S**	**T**	**X**

En este ejemplo, el pivote es siempre el elemento que está en el extremo derecho del subarreglo, es decir, *A*[*f*]; al terminar *Partition*, el pivote queda en la posición que se muestra en rojo

Complejidad de memoria de `Quicksort`

En términos de memoria

- La complejidad de `Quicksort` depende solo de `Partition`
- Vimos una versión $\mathcal{O}(n)$ de `Partition`
- Es posible contar con una versión *in place*

Para la versión *in place* usaremos arreglos

- Haremos intercambios dentro del arreglo
- El truco será partir poniendo el pivote al final

Versión *in place* de `Partition`

input : Arreglo $A[0, \ldots, n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A, i, f)**:**

$\quad x \leftarrow$ índice aleatorio en $\{i, \ldots, f\}$
$\quad p \leftarrow A[x]$
$\quad A[x] \rightleftarrows A[f]$
$\quad j \leftarrow i$
$\quad$ **for** $k = i \ldots f - 1$ **:**
$\qquad$ **if** $A[k] < p$ **:**
$\qquad\quad A[j] \rightleftarrows A[k]$
$\qquad\quad j \leftarrow j + 1$
$\quad A[j] \rightleftarrows A[f]$
$\quad$ **return** j

Con este cambio, `Quicksort` usa memoria adicional $\mathcal{O}(1)$

Análisis de `Quicksort`

Próxima clase demostraremos

- correctitud
- complejidad de tiempo
- En particular, que en el caso promedio es $\mathcal{O}(n \log(n))$

Partiremos de la base que `Partition` es correcto

Ejercicio (propuesto)

Demuestre que `Partition` es correcto

Quick Sort

- Atributos generales
  - O(¿?)
  - O(1) en memoria
  - Inventado por Sir Tony Hoare - 1959
  - Aplica Dividir para Conquistar
  - ¿Estable?
  - ¿Envenenable?

¿Cuándo usar Quick Sort?

Sumario

Introducción

Particiones

Quicksort

Cierre

Objetivos de la clase

- ☐ Comprender la estrategia de elegir pivote para particionar una secuencia
- ☐ Identificar cuándo tal estrategia encuentra la mediana
- ☐ Comprender y analizar el algoritmo `Partition`
- ☐ Comprender el uso de `Partition` en `Median` para encontrar el elemento central
- ☐ Comprender el uso de `Partition` para ordenar secuencias