

... previamente en IIC2133

# 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



divide and conquer...

# 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

# 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  $A[0, \dots, n-1]$ , índices  $i, f$

**output**:  $\emptyset$

QuickSort ( $A, i, f$ ):

```
1  if  $i \leq f$  :  
2       $p \leftarrow \text{Partition}(A, i, f)$   
3      Quicksort( $A, i, p-1$ )  
4      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  $A[0, \dots, n-1]$ , índices  $i, f$

**output**: Índice del pivote aleatorio en la secuencia ordenada

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ 
2   $p \leftarrow A[x]$ 
3   $A[x] \rightleftharpoons A[f]$ 
4   $j \leftarrow i$ 
5  for  $k = i \dots f - 1$  :
6      if  $A[k] < p$  :
7           $A[j] \rightleftharpoons A[k]$ 
8           $j \leftarrow j + 1$ 
9   $A[j] \rightleftharpoons A[f]$ 
10 return  $j$ 
```

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

# Quick Sort

- Atributos generales
  - $O(n^2)$
  - $O(1)$  en memoria
  - Inventado por Sir Tony Hoare - 1959
  - Aplica Dividir para Conquistar
  - ¿Estable?
  - ¿Envenenable?

¿Cuándo usar Quick Sort?

# QuickSort y sus propiedades

Clase 05

IIC 2133 - Sección 2

Prof. Mario Droguett

# Sumario

**Introducción**

Correctitud

Complejidad

Cierre

## Versión *in place* de Partition

**input** : Arreglo  $A[0, \dots, n-1]$ , índices  $i, f$

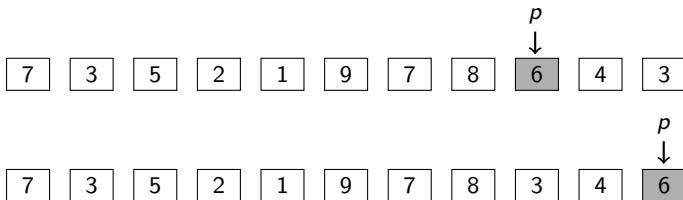
**output**: Índice del pivote aleatorio en la secuencia ordenada

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

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ 
2   $p \leftarrow A[x]$ 
3   $A[x] \rightleftharpoons A[f]$ 
4   $j \leftarrow i$ 
5  for  $k = i \dots f - 1$  :
6      if  $A[k] < p$  :
7           $A[j] \rightleftharpoons A[k]$ 
8           $j \leftarrow j + 1$ 
9   $A[j] \rightleftharpoons A[f]$ 
10 return  $j$ 
```

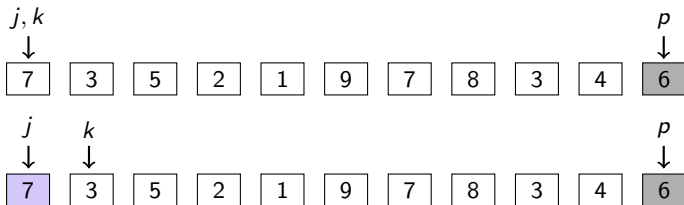
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



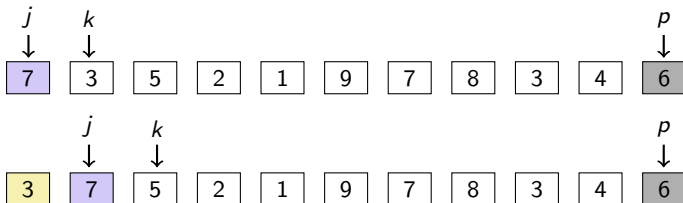
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



Partition ( $A, i, f$ ):

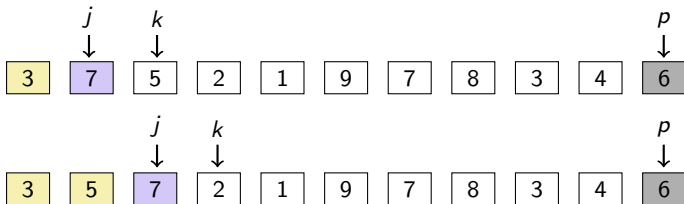
```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```





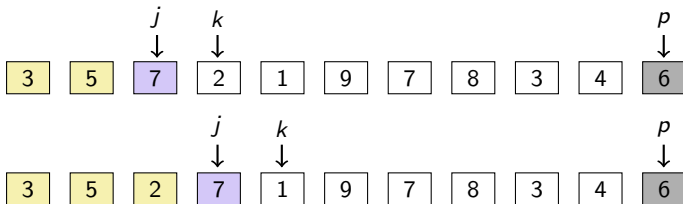
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



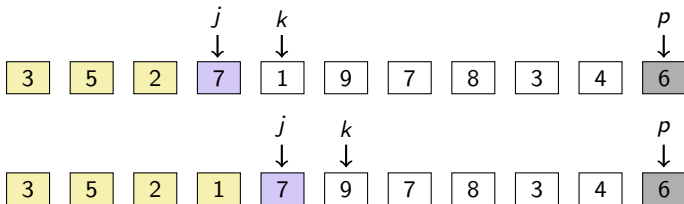
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



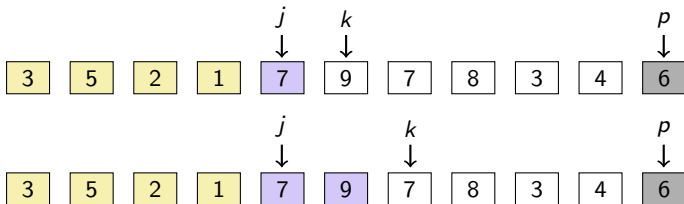
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



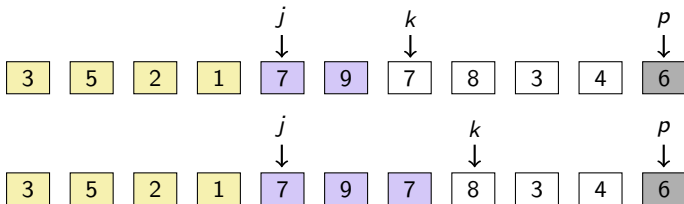
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



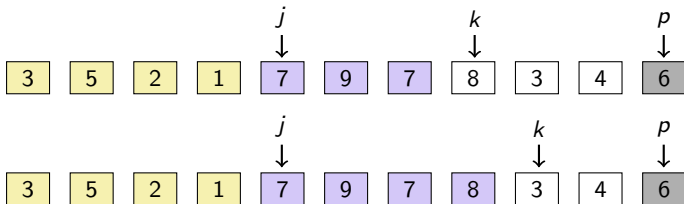
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



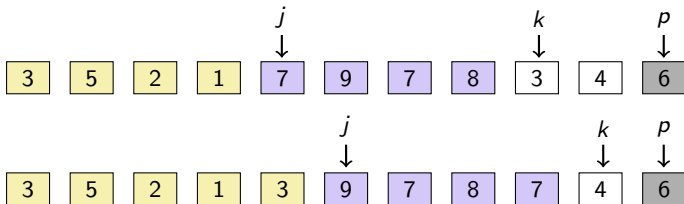
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



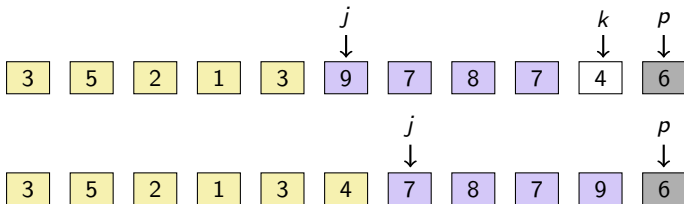
Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



Partition ( $A, i, f$ ):

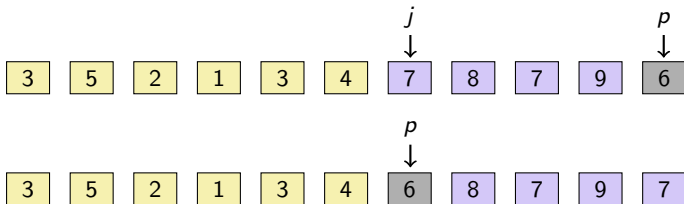
```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```





Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ ;   $p \leftarrow A[x]$ 
2   $A[x] \rightleftharpoons A[f]$ 
3   $j \leftarrow i$ 
4  for  $k = i \dots f - 1$  :
5      if  $A[k] < p$  :
6           $A[j] \rightleftharpoons A[k]$ 
7           $j \leftarrow j + 1$ 
8   $A[j] \rightleftharpoons A[f]$ 
9  return  $j$ 
```



# Ordenar con Partition

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  $A[0, \dots, n-1]$ , índices  $i, f$

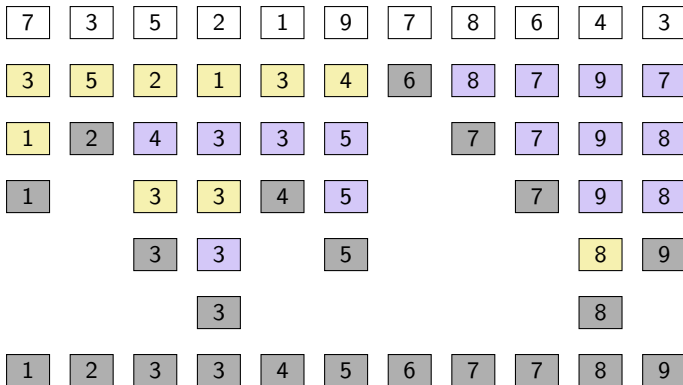
**output:**  $\emptyset$

QuickSort ( $A, i, f$ ):

```
1  if  $i \leq f$  :  
2       $p \leftarrow \text{Partition}(A, i, f)$   
3      Quicksort( $A, i, p-1$ )  
4      Quicksort( $A, p+1, f$ )
```

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

## Quicksort: Ejemplo de ejecución



# Objetivos de la clase

- ☐ Demostrar correctitud de Partition
- ☐ Demostrar correctitud de Quicksort
- ☐ Determinar la complejidad de tiempo de Quicksort
- ☐ Comprender posibles mejoras para Quicksort

# Sumario

Introducción

**Correctitud**

Complejidad

Cierre

# Correctitud de Partition

## Ejercicio

Demuestre que Partition es correcto

**input** : Arreglo  $A[0, \dots, n-1]$ , índices  $i, f$

**output**: Índice del pivote aleatorio en la secuencia ordenada

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  $\{i, \dots, f\}$ 
2   $p \leftarrow A[x]$ 
3   $A[x] \rightleftharpoons A[f]$ 
4   $j \leftarrow i$ 
5  for  $k = i \dots f - 1$  :
6      if  $A[k] < p$  :
7           $A[j] \rightleftharpoons A[k]$ 
8           $j \leftarrow j + 1$ 
9   $A[j] \rightleftharpoons A[f]$ 
10 return  $j$ 
```

# Correctitud de Partition



# Correctitud de Partition

Demostración (finitud)

# Correctitud de Partition

## Demostración (finitud)

Dado que para la llamada `Partition(A, i, f)` se itera el **for**  $f - 1 - i$  veces, haciendo a lo más un intercambio en cada iteración, el **for** es finito. Los pasos adicionales son trivialmente finitos. □

# Correctitud de Partition

## Demostración (finitud)

Dado que para la llamada `Partition( $A, i, f$ )` se itera el **for**  $f - 1 - i$  veces, haciendo a lo más un intercambio en cada iteración, el **for** es finito. Los pasos adicionales son trivialmente finitos. □

## Demostración (posición del pivote)

# Correctitud de Partition

## Demostración (finitud)

Dado que para la llamada `Partition(A, i, f)` se itera el **for**  $f - 1 - i$  veces, haciendo a lo más un intercambio en cada iteración, el **for** es finito. Los pasos adicionales son trivialmente finitos. □

## Demostración (posición del pivote)

Probaremos que el pivote se encuentra en su posición ordenada en la secuencia.

# Correctitud de Partition

## Demostración (finitud)

Dado que para la llamada `Partition(A, i, f)` se itera el **for**  $f - 1 - i$  veces, haciendo a lo más un intercambio en cada iteración, el **for** es finito. Los pasos adicionales son trivialmente finitos. □

## Demostración (posición del pivote)

Probaremos que el pivote se encuentra en su posición ordenada en la secuencia. Para esto, dado que en la línea final se pone el pivote en la posición  $j$ , demostraremos la siguiente propiedad para un pivote  $p$  dado

# Correctitud de Partition

## Demostración (finitud)

Dado que para la llamada `Partition(A, i, f)` se itera el **for**  $f - 1 - i$  veces, haciendo a lo más un intercambio en cada iteración, el **for** es finito. Los pasos adicionales son trivialmente finitos. □

## Demostración (posición del pivote)

Probaremos que el pivote se encuentra en su posición ordenada en la secuencia. Para esto, dado que en la línea final se pone el pivote en la posición  $j$ , demostraremos la siguiente propiedad para un pivote  $p$  dado

**P(n)**     $:=$     Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
                 contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
                 solo elementos mayores o iguales a  $p$

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

### 1. Caso base. **P(1)**



# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

1. **Caso base.** **P(1)** es el estado de  $A[i \dots k - 1]$  luego de la primera iteración. Tenemos dos casos

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

1. **Caso base.** **P(1)** es el estado de  $A[i \dots k - 1]$  luego de la primera iteración. Tenemos dos casos
  - Si  $A[i] < p$ ,

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

1. **Caso base.** **P(1)** es el estado de  $A[i \dots k - 1]$  luego de la primera iteración. Tenemos dos casos
  - Si  $A[i] < p$ , como  $i = j$ , se mantiene  $A[i]$  en su posición y  $j, k$  aumentan en 1. Luego,  $A[i \dots j - 1]$  tiene solo el elemento  $A[i]$ , el cual es menor a  $p$  y  $A[j \dots k - 1]$  no tiene elementos pues  $j = k$ .

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)** := Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
solo elementos mayores o iguales a  $p$

1. **Caso base.** **P(1)** es el estado de  $A[i \dots k - 1]$  luego de la primera iteración. Tenemos dos casos
  - Si  $A[i] < p$ , como  $i = j$ , se mantiene  $A[i]$  en su posición y  $j, k$  aumentan en 1. Luego,  $A[i \dots j - 1]$  tiene solo el elemento  $A[i]$ , el cual es menor a  $p$  y  $A[j \dots k - 1]$  no tiene elementos pues  $j = k$ .
  - Si  $A[i] \geq p$ ,

# Correctitud de Partition

## Demostración (posición del pivote)

**P(n)**    :=    Luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$   
                 contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$   
                 solo elementos mayores o iguales a  $p$

1. **Caso base.** **P(1)** es el estado de  $A[i \dots k - 1]$  luego de la primera iteración. Tenemos dos casos
  - Si  $A[i] < p$ , como  $i = j$ , se mantiene  $A[i]$  en su posición y  $j, k$  aumentan en 1. Luego,  $A[i \dots j - 1]$  tiene solo el elemento  $A[i]$ , el cual es menor a  $p$  y  $A[j \dots k - 1]$  no tiene elementos pues  $j = k$ .
  - Si  $A[i] \geq p$ , no se aumenta  $j$  y sí  $k$ , por lo tanto  $A[i \dots j - 1]$  considera 0 elementos y  $A[j \dots k - 1]$  tiene un elemento mayor o igual a  $p$ .

# Correctitud de Partition

Demostración (posición del pivote)

## 2. **Hipótesis inductiva (H.I.)**

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos



# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ ,

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ , este elemento se pone en la posición  $j$  antes de aumentarla en 1. Por **H.I.** todos los anteriores a él también son menores a  $p$ .

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ , este elemento se pone en la posición  $j$  antes de aumentarla en 1. Por **H.I.** todos los anteriores a él también son menores a  $p$ . Además, como los datos de  $A[j \dots k - 1]$  eran mayores o iguales a  $p$  antes de aumentar los índices, con el intercambio se mantiene la condición.

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ , este elemento se pone en la posición  $j$  antes de aumentarla en 1. Por **H.I.** todos los anteriores a él también son menores a  $p$ . Además, como los datos de  $A[j \dots k - 1]$  eran mayores o iguales a  $p$  antes de aumentar los índices, con el intercambio se mantiene la condición.
- Si  $A[k] \geq p$ ,

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ , este elemento se pone en la posición  $j$  antes de aumentarla en 1. Por **H.I.** todos los anteriores a él también son menores a  $p$ . Además, como los datos de  $A[j \dots k - 1]$  eran mayores o iguales a  $p$  antes de aumentar los índices, con el intercambio se mantiene la condición.
- Si  $A[k] \geq p$ , no se aumenta  $j$  y no se hacen intercambios. Por **H.I.** los elementos en  $A[i \dots j - 1]$  son menores a  $p$

# Correctitud de Partition

## Demostración (posición del pivote)

2. **Hipótesis inductiva (H.I.)** Suponemos se cumple luego de la  $n$ -ésima iteración,  $A[i \dots j - 1]$  contiene solo elementos menores a  $p$  y  $A[j \dots k - 1]$  solo mayores o iguales.

Al término de la  $(n + 1)$ -ésima iteración, nuevamente pueden haber ocurrido dos casos

- Si  $A[k] < p$ , este elemento se pone en la posición  $j$  antes de aumentarla en 1. Por **H.I.** todos los anteriores a él también son menores a  $p$ . Además, como los datos de  $A[j \dots k - 1]$  eran mayores o iguales a  $p$  antes de aumentar los índices, con el intercambio se mantiene la condición.
- Si  $A[k] \geq p$ , no se aumenta  $j$  y no se hacen intercambios. Por **H.I.** los elementos en  $A[i \dots j - 1]$  son menores a  $p$  y como  $A[k]$  al igual que los elementos de  $A[j \dots k - 1]$  son mayores o iguales, luego de aumentar  $k$  el rango  $A[j \dots k - 1]$  sigue cumpliendo la propiedad. □

# Correctitud de Quicksort

# Correctitud de Quicksort

## Ejercicio

Demuestre que Quicksort es correcto

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

**output**:  $\emptyset$

QuickSort ( $A, i, f$ ):

```
1   if  $i \leq f$  :  
2        $p \leftarrow \text{Partition}(A, i, f)$   
3       QuickSort( $A, i, p-1$ )  
4       QuickSort( $A, p+1, f$ )
```



# Correctitud de Quicksort

Demostración (finitud)

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

Ahora, notamos que cada llamado recursivo de Quicksort se hace a una instancia de tamaño estrictamente menor al original.

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

Ahora, notamos que cada llamado recursivo de Quicksort se hace a una instancia de tamaño estrictamente menor al original.

En efecto, la llamada  $\text{Quicksort}(A, i, f)$  establece el *tamaño* de la instancia analizada como

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

Ahora, notamos que cada llamado recursivo de Quicksort se hace a una instancia de tamaño estrictamente menor al original.

En efecto, la llamada  $\text{Quicksort}(A, i, f)$  establece el *tamaño* de la instancia analizada como

$$d = f - i$$

Tenemos dos casos según el valor de  $d$

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

Ahora, notamos que cada llamado recursivo de Quicksort se hace a una instancia de tamaño estrictamente menor al original.

En efecto, la llamada  $\text{Quicksort}(A, i, f)$  establece el *tamaño* de la instancia analizada como

$$d = f - i$$

Tenemos dos casos según el valor de  $d$

- Si  $d < 0$ ,

# Correctitud de Quicksort

## Demostración (finitud)

Para demostrar que Quicksort termina, primero usamos el hecho de que Partition termina y por ello la línea 2 toma una cantidad finita de pasos.

Ahora, notamos que cada llamado recursivo de Quicksort se hace a una instancia de tamaño estrictamente menor al original.

En efecto, la llamada  $\text{Quicksort}(A, i, f)$  establece el *tamaño* de la instancia analizada como

$$d = f - i$$

Tenemos dos casos según el valor de  $d$

- Si  $d < 0$ , Quicksort termina sin hacer más llamados recursivos

# Correctitud de Quicksort

## Demostración (finitud)

El otro caso es

- Si  $d \geq 0$ ,



# Correctitud de Quicksort

## Demostración (finitud)

El otro caso es

- Si  $d \geq 0$ , se hacen dos llamados recursivos de tamaños respectivos

# Correctitud de Quicksort

## Demostración (finitud)

El otro caso es

- Si  $d \geq 0$ , se hacen dos llamados recursivos de tamaños respectivos

$$d_1 = p - 1 - i < d \quad d_2 = f - p - 1 < d \quad \text{para } i \leq p \leq f$$

Es decir, con cada llamado recursivo  $d$  disminuye al menos en 1.

# Correctitud de Quicksort

## Demostración (finitud)

El otro caso es

- Si  $d \geq 0$ , se hacen dos llamados recursivos de tamaños respectivos

$$d_1 = p - 1 - i < d \quad d_2 = f - p - 1 < d \quad \text{para } i \leq p \leq f$$

Es decir, con cada llamado recursivo  $d$  disminuye al menos en 1.

Con ambos casos, tenemos que en cada llamado se reduce al menos en 1 el tamaño de la instancia y el algoritmo termina al llegar a tamaño 0.

# Correctitud de Quicksort

## Demostración (finitud)

El otro caso es

- Si  $d \geq 0$ , se hacen dos llamados recursivos de tamaños respectivos

$$d_1 = p - 1 - i < d \quad d_2 = f - p - 1 < d \quad \text{para } i \leq p \leq f$$

Es decir, con cada llamado recursivo  $d$  disminuye al menos en 1.

Con ambos casos, tenemos que en cada llamado se reduce al menos en 1 el tamaño de la instancia y el algoritmo termina al llegar a tamaño 0.

Como el llamado inicial es  $\text{Quicksort}(A, 0, n - 1)$ , la profundidad máxima de la recursión es  $n - 1$  y por lo tanto, el algoritmo termina. □

# Correctitud de Quicksort

Demostración (ordenación)

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar.

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.



# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort( $A, i, f$ )` verifica si hay elementos en el rango  $i \dots f$

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort( $A, i, f$ )` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ ,

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort( $A, i, f$ )` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ , no hay elementos que puedan ser escogidos como pivote

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort( $A, i, f$ )` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ , no hay elementos que puedan ser escogidos como pivote
- Si  $i \leq f$  el rango es de  $f - i + 1 > 0$  elementos,

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort( $A, i, f$ )` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ , no hay elementos que puedan ser escogidos como pivote
- Si  $i \leq f$  el rango es de  $f - i + 1 > 0$  elementos, se escoge un pivote y se realizan dos llamados cuyos rangos suman  $f - i - 1$  elementos

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort(A, i, f)` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ , no hay elementos que puedan ser escogidos como pivote
- Si  $i \leq f$  el rango es de  $f - i + 1 > 0$  elementos, se escoge un pivote y se realizan dos llamados cuyos rangos suman  $f - i - 1$  elementos

Dado que en cada llamado no vacío se escoge un pivote, los llamados siempre reducen su tamaño y el algoritmo solo deja de hacer recursión cuando el llamado es vacío, todo elemento es escogido pivote en algún llamado.

# Correctitud de Quicksort

## Demostración (ordenación)

Debido a que probamos la correctitud de `Partition` y que `Quicksort` divide en sub-problemas con `Partition`, esta demostración será más sencilla.

Sabemos que `Partition` ubica correctamente el pivote al terminar. Basta demostrar que todo elemento de  $A[0 \dots, n - 1]$  es elegido como pivote en algún llamado de `Quicksort`.

El llamado `Quicksort(A, i, f)` verifica si hay elementos en el rango  $i \dots f$

- Si  $i > f$ , no hay elementos que puedan ser escogidos como pivote
- Si  $i \leq f$  el rango es de  $f - i + 1 > 0$  elementos, se escoge un pivote y se realizan dos llamados cuyos rangos suman  $f - i - 1$  elementos

Dado que en cada llamado no vacío se escoge un pivote, los llamados siempre reducen su tamaño y el algoritmo solo deja de hacer recursión cuando el llamado es vacío, todo elemento es escogido pivote en algún llamado. Como `Partition` es correcto, todo elemento queda ordenado. □

# Sumario

Introducción

Correctitud

**Complejidad**

Cierre



# Quicksort

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

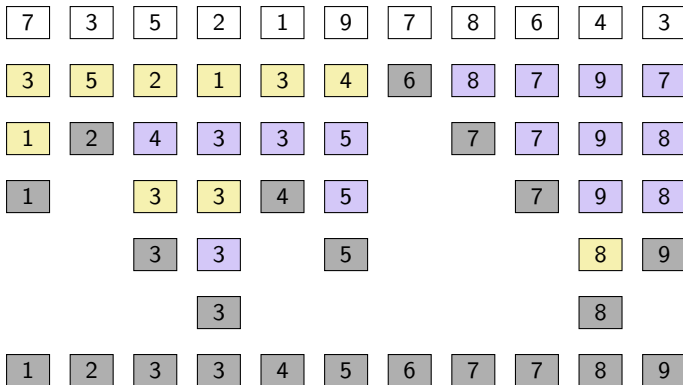
**output**:  $\emptyset$

QuickSort ( $A, i, f$ ):

```
1  if  $i \leq f$  :  
2       $p \leftarrow \text{Partition}(A, i, f)$   
3      QuickSort( $A, i, p-1$ )  
4      QuickSort( $A, p+1, f$ )
```

¿Cuál es la complejidad de Quicksort en los distintos casos?

## Quicksort: Ejemplo de ejecución



# Complejidad de tiempo de Quicksort

# Complejidad de tiempo de Quicksort

Tal como con `Median`, la complejidad depende de la elección del pivote: es **probabilística**

# Complejidad de tiempo de Quicksort

Tal como con `Median`, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$



# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$

Peor caso

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$

Peor caso

- Partition genera sub-secuencias de tamaño 0 y  $n - 1$

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$

Peor caso

- Partition genera sub-secuencias de tamaño 0 y  $n - 1$
- Ecuación de recurrencia  $T(n) = n + T(n - 1)$

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

## Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$

## Peor caso

- Partition genera sub-secuencias de tamaño 0 y  $n - 1$
- Ecuación de recurrencia  $T(n) = n + T(n - 1)$
- Complejidad resultante:  $\mathcal{O}(n^2)$

# Complejidad de tiempo de Quicksort

Tal como con Median, la complejidad depende de la elección del pivote: es **probabilística**

Mejor caso

- Partition genera sub-secuencias del *mismo* tamaño
- Ecuación de recurrencia  $T(n) = n + 2T(n/2)$  (como en MergeSort)
- Complejidad  $\mathcal{O}(n \log(n))$

Peor caso

- Partition genera sub-secuencias de tamaño 0 y  $n - 1$
- Ecuación de recurrencia  $T(n) = n + T(n - 1)$
- Complejidad resultante:  $\mathcal{O}(n^2)$

¿Qué hay del caso promedio?

## Caso promedio de Quicksort

# Caso promedio de Quicksort

En nuestro análisis de **caso promedio** supondremos que el pivote queda en cualquiera de las  $n$  posiciones con igual probabilidad

# Caso promedio de Quicksort

En nuestro análisis de **caso promedio** supondremos que el pivote queda en cualquiera de las  $n$  posiciones con igual probabilidad

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```



# Caso promedio de Quicksort

En nuestro análisis de **caso promedio** supondremos que el pivote queda en cualquiera de las  $n$  posiciones con igual probabilidad

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```

Contaremos el número de comparaciones de la línea 5 de Partition, pues mide cuántas iteraciones debe hacer este método

# Caso promedio de Quicksort

En nuestro análisis de **caso promedio** supondremos que el pivote queda en cualquiera de las  $n$  posiciones con igual probabilidad

```
Partition ( $A, i, f$ ):  
1    $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2    $A[x] \rightleftharpoons A[f]$   
3    $j \leftarrow i$   
4   for  $k = i \dots f - 1$  :  
5       if  $A[k] < p$  :  
6            $A[j] \rightleftharpoons A[k]$   
7            $j \leftarrow j + 1$   
8    $A[j] \rightleftharpoons A[f]$   
9   return  $j$ 
```

Contaremos el número de comparaciones de la línea 5 de Partition, pues mide cuántas iteraciones debe hacer este método

Definimos

$$C(n) := \begin{array}{l} \# \text{ comp. } A[k] < p \\ \text{en Quicksort} \end{array}$$

# Caso promedio de Quicksort

En nuestro análisis de **caso promedio** supondremos que el pivote queda en cualquiera de las  $n$  posiciones con igual probabilidad

```
Partition ( $A, i, f$ ):  
1    $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2    $A[x] \rightleftharpoons A[f]$   
3    $j \leftarrow i$   
4   for  $k = i \dots f - 1$  :  
5       if  $A[k] < p$  :  
6            $A[j] \rightleftharpoons A[k]$   
7            $j \leftarrow j + 1$   
8    $A[j] \rightleftharpoons A[f]$   
9   return  $j$ 
```

Contaremos el número de comparaciones de la línea 5 de Partition, pues mide cuántas iteraciones debe hacer este método

Definimos

$$C(n) := \begin{array}{l} \# \text{ comp. } A[k] < p \\ \text{en Quicksort} \end{array}$$

Encontraremos una ecuación de recurrencia para  $C(n)$

## Caso promedio de Quicksort

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
    $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$

## Caso promedio de Quicksort

```
Partition ( $A, i, f$ ):  
1    $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2    $A[x] \rightleftharpoons A[f]$   
3    $j \leftarrow i$   
4   for  $k = i \dots f - 1$  :  
5       if  $A[k] < p$  :  
6            $A[j] \rightleftharpoons A[k]$   
7            $j \leftarrow j + 1$   
8    $A[j] \rightleftharpoons A[f]$   
9   return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$
- Supongamos que Partition termina retornando  $q$ , i.e. deja al pivote en  $A[q]$

## Caso promedio de Quicksort

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
    $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$
- Supongamos que Partition termina retornando  $q$ , i.e. deja al pivote en  $A[q]$
- Las llamadas recursivas de Quicksort se harán sobre los

## Caso promedio de Quicksort

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$
- Supongamos que Partition termina retornando  $q$ , i.e. deja al pivote en  $A[q]$
- Las llamadas recursivas de Quicksort se harán sobre los
  - $q$  elementos a la izq

# Caso promedio de Quicksort

Partition ( $A, i, f$ ):

```
1   $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2   $A[x] \rightleftharpoons A[f]$   
3   $j \leftarrow i$   
4  for  $k = i \dots f - 1$  :  
5      if  $A[k] < p$  :  
6           $A[j] \rightleftharpoons A[k]$   
7           $j \leftarrow j + 1$   
8   $A[j] \rightleftharpoons A[f]$   
9  return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$
- Supongamos que Partition termina retornando  $q$ , i.e. deja al pivote en  $A[q]$
- Las llamadas recursivas de Quicksort se harán sobre los
  - $q$  elementos a la izq
  - $n - q - 1$  elementos a la der



## Caso promedio de Quicksort

```
Partition ( $A, i, f$ ):  
1    $x \leftarrow$  índice aleatorio en  
     $\{i, \dots, f\}; \quad p \leftarrow A[x]$   
2    $A[x] \rightleftharpoons A[f]$   
3    $j \leftarrow i$   
4   for  $k = i \dots f - 1$  :  
5       if  $A[k] < p$  :  
6            $A[j] \rightleftharpoons A[k]$   
7            $j \leftarrow j + 1$   
8    $A[j] \rightleftharpoons A[f]$   
9   return  $j$ 
```

- En la primera llamada se hacen  $n - 1$  comparaciones, pues  $i = 0$  y  $f = n - 1$
- Supongamos que Partition termina retornando  $q$ , i.e. deja al pivote en  $A[q]$
- Las llamadas recursivas de Quicksort se harán sobre los
  - $q$  elementos a la izq
  - $n - q - 1$  elementos a la der
- Los llamados recursivos aportan  $C(q) + C(n - q - 1)$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 +$$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 + \frac{1}{n} \sum_{q=0}^{n-1} (C(q) + C(n - q - 1))$$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 + \frac{1}{n} \sum_{q=0}^{n-1} (C(q) + C(n - q - 1))$$

Además, notando que

$$\sum_{q=0}^{n-1} C(n - q - 1) =$$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 + \frac{1}{n} \sum_{q=0}^{n-1} (C(q) + C(n - q - 1))$$

Además, notando que

$$\sum_{q=0}^{n-1} C(n - q - 1) = C(n - 1) + C(n - 2) + \dots + C(0)$$

## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 + \frac{1}{n} \sum_{q=0}^{n-1} (C(q) + C(n - q - 1))$$

Además, notando que

$$\sum_{q=0}^{n-1} C(n - q - 1) = C(n - 1) + C(n - 2) + \dots + C(0) = \sum_{q=0}^{n-1} C(q)$$



## Caso promedio de Quicksort

Con lo anterior, las comparaciones cuando el pivote queda en  $A[q]$  son

$$n - 1 + C(q) + C(n - q - 1)$$

Para ver el caso promedio, ponderamos para todos los  $q$  posibles obteniendo

$$C(n) = n - 1 + \frac{1}{n} \sum_{q=0}^{n-1} (C(q) + C(n - q - 1))$$

Además, notando que

$$\sum_{q=0}^{n-1} C(n - q - 1) = C(n - 1) + C(n - 2) + \dots + C(0) = \sum_{q=0}^{n-1} C(q)$$

obtenemos la regla simplificada

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

# Caso promedio de Quicksort

Para la ecuación de recurrencia

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

tenemos dos casos base

# Caso promedio de Quicksort

Para la ecuación de recurrencia

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

tenemos dos casos base

- $C(0) = 0,$

# Caso promedio de Quicksort

Para la ecuación de recurrencia

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

tenemos dos casos base

- $C(0) = 0$ , pues corresponde al caso base de Quicksort

# Caso promedio de Quicksort

Para la ecuación de recurrencia

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

tenemos dos casos base

- $C(0) = 0$ , pues corresponde al caso base de Quicksort
- $C(1) = 0$ ,

# Caso promedio de Quicksort

Para la ecuación de recurrencia

$$C(n) = n - 1 + \frac{2}{n} \sum_{q=0}^{n-1} C(q)$$

tenemos dos casos base

- $C(0) = 0$ , pues corresponde al caso base de Quicksort
- $C(1) = 0$ , pues Partition no itera si hay un solo elemento

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$nC(n) = n(n-1) + 2 \sum_{q=0}^{n-1} C(q)$$

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$nC(n) = n(n-1) + 2 \sum_{q=0}^{n-1} C(q)$$

$$(n-1)C(n-1) = (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)$$



## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$nC(n) = n(n-1) + 2 \sum_{q=0}^{n-1} C(q)$$

$$(n-1)C(n-1) = (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)$$

Restando ambas ecuaciones obtenemos

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$nC(n) = n(n-1) + 2 \sum_{q=0}^{n-1} C(q)$$

$$(n-1)C(n-1) = (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)$$

Restando ambas ecuaciones obtenemos

$$nC(n) = (n+1)C(n-1) + 2n - 2$$

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$nC(n) = n(n-1) + 2 \sum_{q=0}^{n-1} C(q)$$

$$(n-1)C(n-1) = (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)$$

Restando ambas ecuaciones obtenemos

$$nC(n) = (n+1)C(n-1) + 2n - 2$$

Dividimos por  $n(n+1)$  y comenzamos una serie de inecuaciones

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$\begin{aligned}nC(n) &= n(n-1) + 2 \sum_{q=0}^{n-1} C(q) \\ (n-1)C(n-1) &= (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)\end{aligned}$$

Restando ambas ecuaciones obtenemos

$$nC(n) = (n+1)C(n-1) + 2n - 2$$

Dividimos por  $n(n+1)$  y comenzamos una serie de inecuaciones

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)}$$

## Caso promedio de Quicksort

Amplificamos por  $n$  la recurrencia y la reescribimos para  $n - 1$

$$\begin{aligned}nC(n) &= n(n-1) + 2 \sum_{q=0}^{n-1} C(q) \\ (n-1)C(n-1) &= (n-1)(n-2) + 2 \sum_{q=0}^{n-2} C(q)\end{aligned}$$

Restando ambas ecuaciones obtenemos

$$nC(n) = (n+1)C(n-1) + 2n - 2$$

Dividimos por  $n(n+1)$  y comenzamos una serie de inecuaciones

$$\begin{aligned}\frac{C(n)}{n+1} &= \frac{C(n-1)}{n} + \frac{2}{n+1} - \frac{2}{n(n+1)} \\ &\leq \frac{C(n-1)}{n} + \frac{2}{n+1}\end{aligned}$$

## Caso promedio de Quicksort

$$\frac{C(n)}{n+1} \leq \frac{C(n-1)}{n} + \frac{2}{n+1}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1}\end{aligned}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}\end{aligned}$$



## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots\end{aligned}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1}\end{aligned}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1} \\ &\leq \sum_{k=1}^n \frac{2}{k}\end{aligned}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1} \\ &\leq \sum_{k=1}^n \frac{2}{k} \leq \int_1^n \frac{2}{x} dx\end{aligned}$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1} \\ &\leq \sum_{k=1}^n \frac{2}{k} \leq \int_1^n \frac{2}{x} dx = 2 \log(n)\end{aligned}$$

Concluimos que una cota superior para el número de comparaciones es

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1} \\ &\leq \sum_{k=1}^n \frac{2}{k} \leq \int_1^n \frac{2}{x} dx = 2 \log(n)\end{aligned}$$

Concluimos que una cota superior para el número de comparaciones es

$$C(n) \leq 2(n+1) \log(n)$$

## Caso promedio de Quicksort

$$\begin{aligned}\frac{C(n)}{n+1} &\leq \frac{C(n-1)}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\leq \frac{C(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\ &\dots \\ &\leq \frac{C(1)}{2} + \sum_{k=2}^n \frac{2}{k+1} \\ &\leq \sum_{k=1}^n \frac{2}{k} \leq \int_1^n \frac{2}{x} dx = 2 \log(n)\end{aligned}$$

Concluimos que una cota superior para el número de comparaciones es

$$C(n) \leq 2(n+1) \log(n)$$

Quicksort es  $\mathcal{O}(n \log(n))$  en el caso promedio

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort



# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote
  - Dado  $A$ , escogemos 3 elementos  $A[k_1], A[k_2], A[k_3]$

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote
  - Dado  $A$ , escogemos 3 elementos  $A[k_1], A[k_2], A[k_3]$
  - En  $\mathcal{O}(1)$  encontramos la mediana entre ellos



# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote
  - Dado  $A$ , escogemos 3 elementos  $A[k_1], A[k_2], A[k_3]$
  - En  $\mathcal{O}(1)$  encontramos la mediana entre ellos
- Particionar la secuencia en 3 sub-secuencias: menores, iguales y mayores

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote
  - Dado  $A$ , escogemos 3 elementos  $A[k_1], A[k_2], A[k_3]$
  - En  $\mathcal{O}(1)$  encontramos la mediana entre ellos
- Particionar la secuencia en 3 sub-secuencias: menores, iguales y mayores
  - Mejora para caso con datos repetidos

# Mejoras en Quicksort

- Para sub-secuencias pequeñas (e.g.  $n \leq 20$ ) podemos usar InsertionSort
  - A pesar de no tener una complejidad mejor
  - Eso es solo cuando hablamos asintóticamente
  - En la práctica, en instancias pequeñas tiene mejor desempeño
  - No olvidar que Quicksort es **recursivo**... eso cuesta recursos
- Usar la mediana de tres elementos como pivote
  - *Informar* la elección de pivote
  - Dado  $A$ , escogemos 3 elementos  $A[k_1], A[k_2], A[k_3]$
  - En  $\mathcal{O}(1)$  encontramos la mediana entre ellos
- Particionar la secuencia en 3 sub-secuencias: menores, iguales y mayores
  - Mejora para caso con datos repetidos
  - Evita que Partition particione innecesariamente sub-secuencias en que todos los valores son iguales

# 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	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n \log(n))$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
Heap Sort	?	?	?	?

# Sumario

Introducción

Correctitud

Complejidad

**Cierre**

# Objetivos de la clase

# Objetivos de la clase

- ☐ Demostrar correctitud de Partition

# Objetivos de la clase

- ☐ Demostrar correctitud de Partition
- ☐ Demostrar correctitud de Quicksort



# Objetivos de la clase

- ☐ Demostrar correctitud de Partition
- ☐ Demostrar correctitud de Quicksort
- ☐ Determinar la complejidad de tiempo de Quicksort

# Objetivos de la clase

- ☐ Demostrar correctitud de Partition
- ☐ Demostrar correctitud de Quicksort
- ☐ Determinar la complejidad de tiempo de Quicksort
- ☐ Comprender posibles mejoras para Quicksort