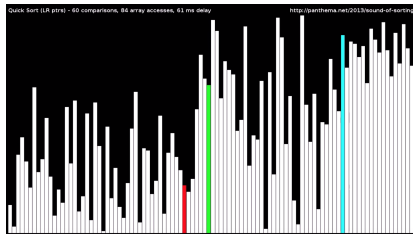


# Ordenamiento usando QuickSort

## Computer Science

**CS1100 - Introducción a Ciencia de la Computación**

# Por qué usar QuickSort?



Barras no ordenadas



Barras ordenadas no  
descendente

<https://www.youtube.com/watch?v=8hEyhs>

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer el tiempo de de ejecución y el espacio de memoria requerido por el algoritmo QuickSort.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer el tiempo de de ejecución y el espacio de memoria requerido por el algoritmo QuickSort.
- Desarrollar el algoritmo de particionamiento.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer el tiempo de de ejecución y el espacio de memoria requerido por el algoritmo QuickSort.
- Desarrollar el algoritmo de particionamiento.
- Desarrollar el paradigma de dividir y conquistar.

# Logro de la Sesión

Al finalizar esta sesión, estarás en la capacidad de:

- Conocer el tiempo de de ejecución y el espacio de memoria requerido por el algoritmo QuickSort.
- Desarrollar el algoritmo de particionamiento.
- Desarrollar el paradigma de dividir y conquistar.
- Utilizar QuickSort para ordenar una lista de numeros enteros.

# Tiempo de ejecución y Memoria utilizada

Tiempo de ejecución	QuickSort
Mejor de los casos	$\mathcal{O}(n \log n)$
Promedio	$\mathcal{O}(n \log n)$
Peor de los casos	$\mathcal{O}(n^2)$

Memoria	QuickSort
Mejor de los casos	$\mathcal{O}(\log n)$
Peor de los casos	$\mathcal{O}(n)$

# Definición

## Cómo funciona QuickSort?

- En QuickSort escogemos un elemento del array que lo llamamos pivot y particionamos el array de tal manera que todos los elementos menores al pivot estan a la izquierda y los mayores a su derecha.



# Definición

## Cómo funciona QuickSort?

- En QuickSort escogemos un elemento del array que lo llamamos pivot y particionamos el array de tal manera que todos los elementos menores al pivot estan a la izquierda y los mayores a su derecha.
- El particionamiento se realiza eficientemente a través de una serie de intercambios.

# Definición

## Cómo funciona QuickSort?

- En QuickSort escogemos un elemento del array que lo llamamos pivot y particionamos el array de tal manera que todos los elementos menores al pivot estan a la izquierda y los mayores a su derecha.
- El particionamiento se realiza eficientemente a través de una serie de intercambios.
- El resultado de particionar el array repetidamente alrededor de un elemento pivot hace que el array eventualmente se ordene.

# Definición

## Cómo funciona QuickSort?

- En QuickSort escogemos un elemento del array que lo llamamos pivot y particionamos el array de tal manera que todos los elementos menores al pivot están a la izquierda y los mayores a su derecha.
- El particionamiento se realiza eficientemente a través de una serie de intercambios.
- El resultado de particionar el array repetidamente alrededor de un elemento pivot hace que el array eventualmente se ordene.
- Sin embargo, no es garantizado que el elemento pivot sea la media del array y esa es la razón de que en el peor de los casos el algoritmo tiene un tiempo de ejecución de  $\mathcal{O}(n^2)$ .

# Ejemplo

Input

Q U I C K S O R T

shuffle

K R A T E L E P U

pivot

partition

E C A I E K L P U

No mayores

sort left

A C E E I K L P U

sort right

A C E E I K L M O

result

A C E E I K L M O

# Algoritmo de Particionamiento - Pivot Izquierdo

```
1  def partition(A, low, high):
2      i = low
3      /** Elegimos el primer elemento del
4          array como pivot */
5      pivot = A[low]
6      for j in range(low + 1, high + 1):
7          if A[j] <= pivot:
8              i = i+1
9              if i != j:
10                 A[i], A[j] = A[j], A[i]
11             A[i], A[low] = A[low], A[i]
12             w = i
13     return w
```



# Algoritmo de Particionamiento - Pivot Derecho

```
1  def partition(A, low, high):
2      i = high
3      /** Elegimos el ultimo elemento del array
4          como pivot */
5      pivot = A[high]
6      for j in range(low, high):
7          if A[j] < pivot:
8              A[i], A[j] = A[j], A[i]
9              i = i+1
10         A[i], A[high] = A[high], A[i]
11         w = i
12     return w
```

# Algoritmo QuickSort

## Utilizamos:

- Paradigma dividir y conquistar
- Recursividad

```
1 def quicksort(A, low, high):  
2     if low < high:  
3         w = partition(A, low, high)  
4         quicksort(A, low, w-1)  
5         quicksort(A, w+1, high)
```

# Algoritmo de Particionamiento - Indice medio

```
1  def partition(A, left, right):
2      middle = (left + right)/2
3      pivot = A[middle]
4      while left <= right:
5          /** Encontrar el elemento en la
6              izquierda que debería estar en la
7              derecha */
8          while A[left] < pivot:
9              left = left + 1
10             /** Encontrar el elemento en la
11                 derecha que debería estar en la
12                 izquierda */
13             while A[right] > pivot:
14                 right = right - 1
15             /** Intercambiar elementos, y mover
```





# Algoritmo QuickSort

```
1 def quicksort(A, left, right):  
2     index = partition(A, left, right)  
3     /** Ordenar en la mitad izquierda */  
4     if left < index - 1:  
5         quicksort(A, left, index - 1)  
6  
7     /** Ordenar en la mitad derecha */  
8     if index < right:  
9         quicksort(A, index, right)
```

# Ejercicio 1

## Enunciado

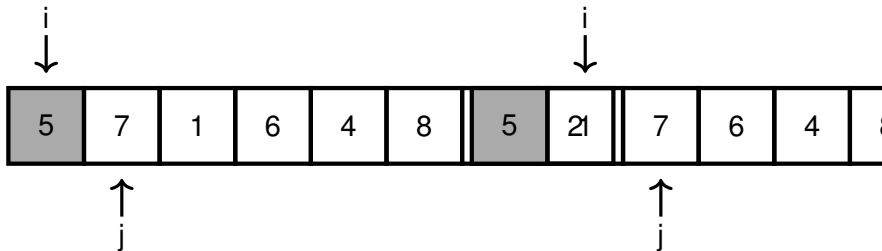
Escribir un programa en Python, que ordene una lista de numeros utilizando QuickSort con particionamiento de pivot izquierdo.

- $A = [5, 7, 1, 6, 4, 8, 3, 2]$

# Ejercicio 1

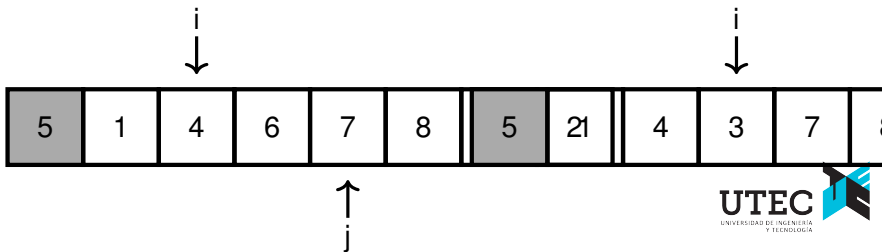
(a)

(b)



(c)

(d)



# Ejercicio 1

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

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

2	1	4	3
---	---	---	---

1	2	4	3
---	---	---	---

4	3
---	---

3	4
---	---

# Evaluación

## Trabajo Individual

■ [www.hackerrank.com/semana13b-mabisrror](http://www.hackerrank.com/semana13b-mabisrror)

# Cierre

En esta sesión aprendiste:

- Desarrollar el algoritmo de ordenamiento QuickSort

# Cierre

En esta sesión aprendiste:

- Desarrollar el algoritmo de ordenamiento QuickSort
- Desarrollar el algoritmo de particionamiento y sus diferentes formas

# Cierre

En esta sesión aprendiste:

- Desarrollar el algoritmo de ordenamiento QuickSort
- Desarrollar el algoritmo de particionamiento y sus diferentes formas
- Utilizar el paradigma de dividir y conquistar



# Cierre

## En esta sesión aprendiste:

- Desarrollar el algoritmo de ordenamiento QuickSort
- Desarrollar el algoritmo de particionamiento y sus diferentes formas
- Utilizar el paradigma de dividir y conquistar
- Conocer el tiempo de ejecución y memoria requerida en el algoritmo QuickSort.