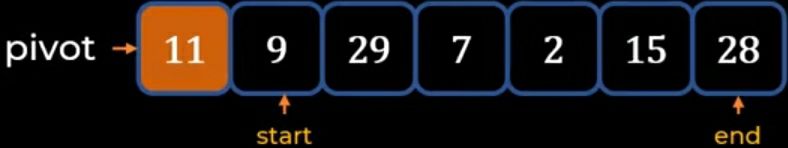
***HOARE PARTITION***

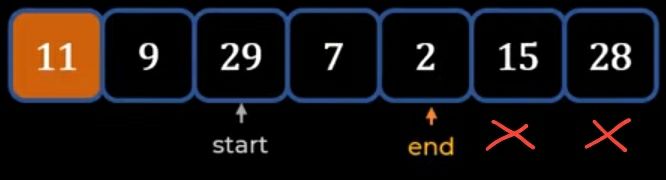
El ***PÍVOT*** se va a comparar con los elementos de ***START*** y ***END***:



***Start*** (¿el 9 es mayor que el 11?) : ¡¡¡NO!! (¿el 29 es mayor que el 11?) : ¡¡¡SI!!!



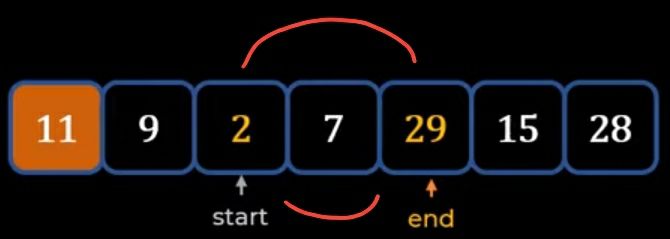
***End*** (¿el 28 es menor que 11?) ¡¡¡NO!! (¿el 15 es menor que 11?) ¡¡¡NO!!! (¿el 2 es menor que 11?) ¡¡¡SI!!



Debemos colocar el ***PIVOT*** en el medio de tal forma quede de la siguiente manera:

***(Elementos Que Son Menores Que El PIVOT) PIVOT (Elementos Que Son Mayores Que El PIVOT)***

Entonces para que eso se cumpla, en primer lugar, debemos intercambiar los valores de ***START*** y ***END***:



Entonces hacemos el proceso del comienzo de nuevo.

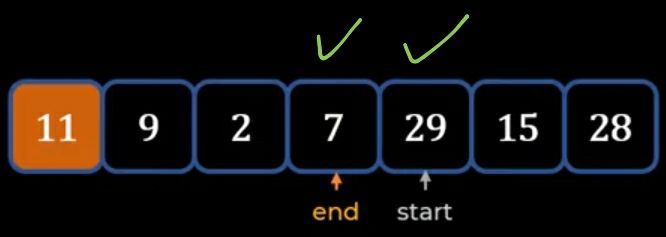
***Start*** (¿el 2 es mayor que el 11?) NO (¿el 7 es mayor que el 11?) NO (¿el 29 es mayor que el 11?) SI

***End*** (¿el 29 es menor que el 11?) NO



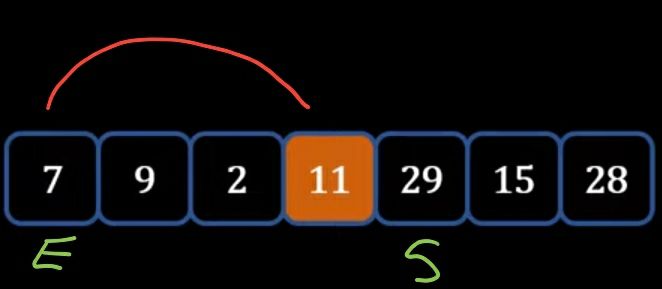
Y la idea es que ***START*** se seguirá moviendo hasta encontrar un numero mayor al ***PIVOT*** (11) en este caso lo encontró que es el 29.

Pero como no encontró el que era menor que el ***PIVOT*** (11) el ***END*** se seguirá moviendo, en este caso hasta 7, porque 7 si es menor que el ***PIVOT*** (11).



Entonces, el ***PIVOT*** (11) cambia de valor, cuando ***END*** pasa a través de ***START*** y de esta manera se cumple la regla de:

***(Elementos Que Son Menores Que El PIVOT) PIVOT (Elementos Que Son Mayores Que El PIVOT)***

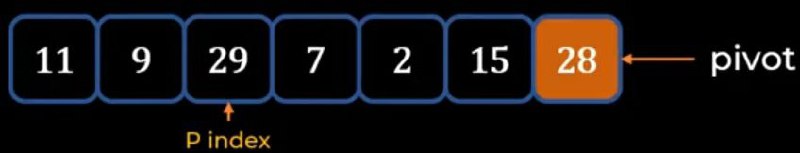


***LOUMUTO PARTITION***

El ultimo elemento es el ***PIVOT***, mas una variable llamada ***P INDEX*** el cual va a comparar los elementos con el ***PIVOT*** hasta encontrar un número que sea mayor que el ***PIVOT***.



***P INDEX*** (¿el 11 es mayor que el 28?) NO (¿el 9 es mayor que el 28?) NO (¿el 29 es mayor que el 28?) SI



Entonces se va a iniciar otro contador delante del ***P INDEX***, en este caso se llama ***I***, entonces se intercambian los valores.

El ***P INDEX*** = ***7***, e ***I*** = ***29***





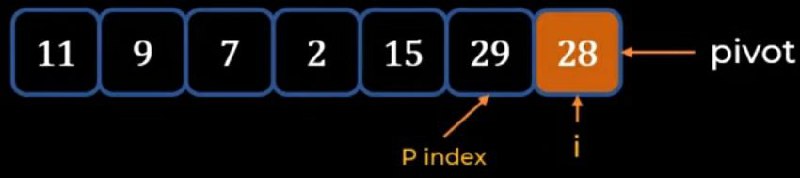
Pero como ***P INDEX*** se continúa ejecutando hasta encontrar elementos mayores que el ***PIVOT***, entonces pasara al ***29***, y el ***I*** seguirá buscando elementos menores que el ***PIVOT***, entonces pasara al ***2.***



Entonces vuelve a pasar lo mismo, intercambiamos valores.



Y seguirá el proceso, ***P INDEX*** será el ***29***, ***I*** será el ***15***, y volverán a cambiar. Hasta que el ***29*** este al lado del ***PIVOT***. Y ocurre lo mismo, intercambian sus valores.





Su complejidad es ***O(LOG n)*** en el peor de los casos, ***O(N2)***

En este caso vamos a codificar la parte de ***HOARE PARTITION.***

Esta parte corresponde a ***START*** y ***END.***

def partition(elements):

    pivot\_index = 0

    pivot = elements[pivot\_index] # PIVOT = 11

    start = pivot\_index + 1 # START = 9

    end = len(elements) – 1 # END = 28

    while elements[start] <= pivot: # mientras el 9 sera menor que 11, start se moverá 1 posicion.

        start += 1

    while elements[end] > pivot: # mientras el 28 sera mayor que el 11, end se moovera 1 posicion

        end -= 1

elements = [11,9,29,7,2,15,28]

el código de arriba se ira ejecutando hasta llegar al punto en el que, ***START (29)*** será mayor que el ***PIVOT (11)***, y que ***END (2)*** sea menor que el ***PIVOT (11)***.

Si vamos a la teoría, lo que sigue es hacer el ***SWAP*** o ***INTERCAMBIO DE VALORES***. Y que las variables de ***START*** y ***END*** se sigan moviendo.

Para eso programamos el ***INTERCAMBIO DE VALORES***.

def swap(start, end, elements): # cuando los dos WHILE dejen de ejecutarse se ejecuta esta función

    if start != end:

        tmp = elements[start] # 29

        elements[start] = elements[end] # 29 == 2

        elements[end] = tmp # 2 == 29

bien, ahora que ya tenemos la función para intercambiar los valores, en el siguiente FOR el código nos deja el PIVOT en el medio.

def swap(start, end, elements):

    if start != end:

        tmp = elements[start]

        elements[start] = elements[end]

        elements[end] = tmp

def partition(elements):

    pivot\_index = 0

    pivot = elements[pivot\_index]

    start = pivot\_index + 1

    end = len(elements) - 1

    while start < end:

        while elements[start] <= pivot:

            start += 1

        while elements[end] > pivot:

            end -= 1

        if start < end:

            swap(start, end, elements)

    swap(pivot\_index, end, elements)

def quick\_sort(elements):

    partition(elements)

elements = [11,9,29,7,2,15,28]

quick\_sort(elements)

print(elements)

ahora nos falta implementar la recursividad para llevar a cabo los demás intercambios con los demás números.