

**INSTITUTO POLITÉCNICO NACIONAL**  
Unidad Profesional Interdisciplinaria de Ingeniería  
Campus Zacatecas.

**Materia:**  
Análisis y Diseño de Algoritmos

**Práctica 02: Implementación y Evaluación del  
Algoritmo Quicksort.**

**Docente:**  
M. en C. Erika Sánchez-Femat

**Nombre del alumno:**  
Dalia Naomi García Macías

**Fecha de entrega:**  
23 de Octubre de 2023

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo de la Práctica</b>	<b>3</b>
2.1. Implementación del algoritmo QuickSort . . . . .	3
2.2. Generación de casos de prueba . . . . .	5
2.3. Medición del tiempo . . . . .	6
2.4. Visualización de datos . . . . .	7
2.5. Modificación del pivote . . . . .	8
<b>3. Código completo</b>	<b>9</b>
<b>4. Resultados (Capturas)</b>	<b>12</b>
<b>5. Conclusiones</b>	<b>16</b>
<b>6. Referencias</b>	<b>16</b>

# 1. Introducción

Desde que existe la ciencia de la computación, uno de los mayores problemas con los que los ingenieros se encontraban en su día a día, era el de ordenar listas de elementos. Por su causa, diversos algoritmos de ordenación fueron desarrollados a lo largo de los años y siempre existió un intenso debate entre los desarrolladores sobre cual de todos los algoritmos de ordenación era el más rápido.

El debate finalizó abruptamente en 1960 cuando Sir Charles Antony Richard Hoare, nativo de Sri Lanka y ganador del premio Turing en 1980, desarrolló el algoritmo de ordenación QuickSort casi por casualidad mientras ideaba la forma de facilitar la búsqueda de palabras en el diccionario.

El algoritmo QuickSort, en español método de ordenamiento rápido, será el que se analizará durante la práctica, basa en la técnica de "divide y vencerás" por la que en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.

## 2. Desarrollo de la Práctica

### 2.1. Implementación del algoritmo QuickSort

Como bien especificamos en el documento teórico de esta práctica, el algoritmo QuickSort es un código que nos ayuda a ordenar elementos de un arreglo. Esto lo hace con la técnica divide y vencerás. Consiste en ir dividiendo el arreglo original tomando un número que llamaremos pivote, número que tomará como referencia para dividir el arreglo original. Luego de tener el pivote definido, comparar este número con todos los demás elementos, de forma que los números menores a este pivote los colocará en un arreglo que pondrá del lado izquierdo del pivote. Por el contrario, los números mayores al pivote los pondrá en otro arreglo al lado derecho del pivote, y el pivote lo dejaremos solo en un arreglo. Después de haber hecho esto, repetiremos este paso con cada uno de los arreglos que se vayan generando, hasta que quede cada numero separado en un arreglo para, por último, juntar todos esos números que ya estarán ordenados.

El algoritmo lo decidí dividir en funciones, una función diferente según cada especificación de la práctica. Primero, una función que me diera el tamaño de manera aleatoria de los arreglos; luego, creé una función que generara números aleatorios y los metiera dentro de un arreglo, esto según el tamaño que recibía de la primera función. Y, finalmente, una tercera función que crea una lista con todos los arreglos generados que salen de la función anterior. Finalmente, metí uno por uno los arreglos a la función que contendrá como tal el código principal de ordenamiento (QuickSort).

Listing 1: Funcion que define los tamanos del arreglo

```
1 #Funcion que genera el tama o de el arreglo
2 def Tam():
3     #Genera el tama o aleatoriamente
4     t=random.randrange(1,10)
5     return t
```

Listing 2: Funcion que genera cada arreglo segun el tamaño del arreglo

```
1 #Funcion que genera el arreglo
2 def GenA(n):
```

```

3 #Llena el arreglo aleatoriamente
4 A=[random.randrange(1,20) for i in range(n)]
5 return A

```

Listing 3: Funcion que guarda los 100 arreglos

```

1 #Funcion que genera una lista donde se guardan los 100 arreglos
2 def Arreglos():
3     #Genera la lista vacia
4     a=[]
5     #Hace que nos genere tantos arreglos como queramos
6     for i in range(100):
7         #Va agregando a la lista cada arreglo generado
8         a.append(GenA(Tam()))
9     return a

```

Aparte de las funciones anteriores, sabemos que dentro del algoritmo QuickSort, una parte importante es el pivote, el cual se calculó sacando la media del arreglo, así que para esto también creé una función aparte, la cual luego se manda llamar a la función principal QuickSort.

Listing 4: Funcion que genera la media de cada arreglo

```

1 def Media(x):
2     #Usa sum, nos ayuda a sumar todos los elementos de un arreglo
3     #La doble linea redondea la division a un numero entero
4     #Len calcula la longitud del arreglo
5     m=sum(x)//len(x)
6     return m

```

Ahora, dado que nuestro algoritmo debía ser recursivo, como caso base establecí que sería cuando el tamaño de los arreglos sea 1, y como caso general, hice un for. Este for es el que separa los números en los arreglos correspondientes al compararlos con el número que se toma como pivote, y se regresan a la función los nuevos arreglos que creamos llamados 'izq' y 'der', esto aparte de 'p', el arreglo que solo contiene el número que tomamos como pivote.

Listing 5: Funcion del QuickSort

```

1 def QuickSort(x):
2     #Se define el caso base, que dice que cuando un arreglo solo tenga
        un elemento pues ya estara ordenado
3     if len(x) <=1:
4         return x
5     #Se define el pivote, que ya se calculo anteriormente en la
        funcion media
6     m=Media(x)
7     p=m
8     #Se definen los arreglos donde clasificaremos los elementos luego
        de las comparaciones
9     izq=[]
10    der=[]
11    igual=[]
12    #Cilco que asigna cada valor del arreglo original a los arreglos
        resultantes de la divison

```

```

13     for e in x:
14         if e<p:
15             izq.append(e)
16         elif e>p:
17             der.append(e)
18         else:
19             #Se crea tambien un arreglo igual para el caso en que haya
20             #numeros repetidos o iguales al pivote
21             igual.append(e)
22     #Se definen los dos arreglos en una variable al momento de
23     #volverlas a llamar
24     izq_ordenado=QuickSort(izq)
25     der_ordenado=QuickSort(der)
26     #Se regresan los arreglos resultantes de las divisiones del
27     #arreglo original y se concatenan
28     return izq_ordenado + igual + der_ordenado

```

Listing 6: Implementacion de las funciones en el main

```

1 print("\n\t\t\tQuickSort")
2 n=Tam()
3 #Lista que contiene cada arreglo
4 x=Arreglos()
5
6 #Imprime cada elemento de la lista
7 print("\n-> Arreglos generados aleatoriamente")
8 Tama os=[]
9 for i in x:
10     print(f"    El arreglo {x.index(i)} es: {i}")
11     Tama os.append(len(i))
12
13 #Manda cada arreglo a la funcion QuickSort para que ordene de uno por
14 #uno
15 print("\n-> Arreglos ordenados")
16 ArregloOrdenado=[]
17 for i in x:
18     m=Media(i)
19     #Cada arreglo ordenado lo guarda en una variable
20     ordenado=QuickSort(i)
21     ArregloOrdenado.append(ordenado)
22     #Imprime cada arreglo uno por uno ordenado
23     print(f"    El arreglo {x.index(i)} ordenado es: {ordenado}")

```

## 2.2. Generación de casos de prueba

Para esta sección o este punto, fue entonces que creé las funciones, de modo que en la función 'lista', que guarda los arreglos generados, el límite fuera cien y generara así 100 arreglos con tamaños y números aleatorios.

A continuación mostraré el código y un ejemplo de su funcionamiento:

Listing 7: Funcion que guarda los 100 arreglos generados aleatoriamente

```

1 #Funcion que genera una lista donde se guardan los 100 arreglos
2 def Arreglos():
3     #Genera la lista vacia
4     a=[]
5     #Hace que nos genere tantos arreglos como queramos
6     for i in range(100):
7         #Va agregando a la lista cada arreglo generado
8         a.append(GenA(Tam()))
9     return a

```

### 2.3. Medición del tiempo

Para el caso de la medición del tiempo, usé una función que calculaba el tiempo de ejecución cada vez que se ejecutaba la función QuickSort, ya que lo que nos importaba era cuánto se tardaba en ordenar cada arreglo. Lo que mandaba la función en cada caso era el tiempo que se calculaba por arreglo, y al final, para no perder el tiempo de cada arreglo, lo almacenaba en una lista llamada 'Tiempos', dado que todos estos tiempos se usarían después.

Listing 8: Funcion que calcula el tiempo de ejecucion

```

1 #Funcion que genera una lista donde se guardan los 100 arreglos
2 #Funcion para calcular el tiempo de ejecucion de una funcion
3 def TiempoEjecucion(func, *args):
4     #Tomamos el tiempo de ejecucion inicial
5     inicio=time.time()
6     #Llama a la funcion de la que queremos el tiempo de ejecucion
7     func(*args)
8     #Tomamos el tiempo de ejecucion final
9     fin=time.time()
10    #Calculamos el tiempo de ejecucion real
11    TiempoReal=fin-inicio
12    #Regresamos como resultado de la funcion el tiempo de ejecucion
13    return TiempoReal

```

Listing 9: Calculo del tiempo de ejecucion en el main

```

1 print("\n-> Tiempos de ejecucion")
2 Tiempos=[]
3 Resultados=[]
4 for i in x:
5     #Cada tiempo de ejecucion lo guardaremos en la variable tiempo
6     Tiempo=TiempoEjecucion(QuickSort,i)
7     #Imprime el tiempo de ejecucion de cada arreglo generado
8     print(f"    Tiempo tardado en ordenar el arreglo {x.index(i)}: {
9         Tiempo}")
10    #Se guarda cada tiempo de ejecucion en un arreglo
11    Tiempos.append(Tiempo)
12    Resultados.append((Tiempo, i, QuickSort(i)))

```

## 2.4. Visualización de datos

Para la visualización de datos, se usó la biblioteca Matplotlib de Python, donde se mandaron como datos la lista que creé con los tiempos de ejecución, además de poner nombres a los ejes y al gráfico.

Listing 10: Calculo del tiempo de ejecución en el main

```
1 plt.xlabel("Tamaño del arreglo (n)")
2 plt.ylabel("Tiempo de ejecución (segundos)")
3 plt.title("Gráfico del algoritmo QuickSort")
4
5 # Usamos n como el tamaño del arreglo en el eje X y Tiempos en el eje
6 # Y
7 plt.plot(Tiempos)
8 plt.show()
```

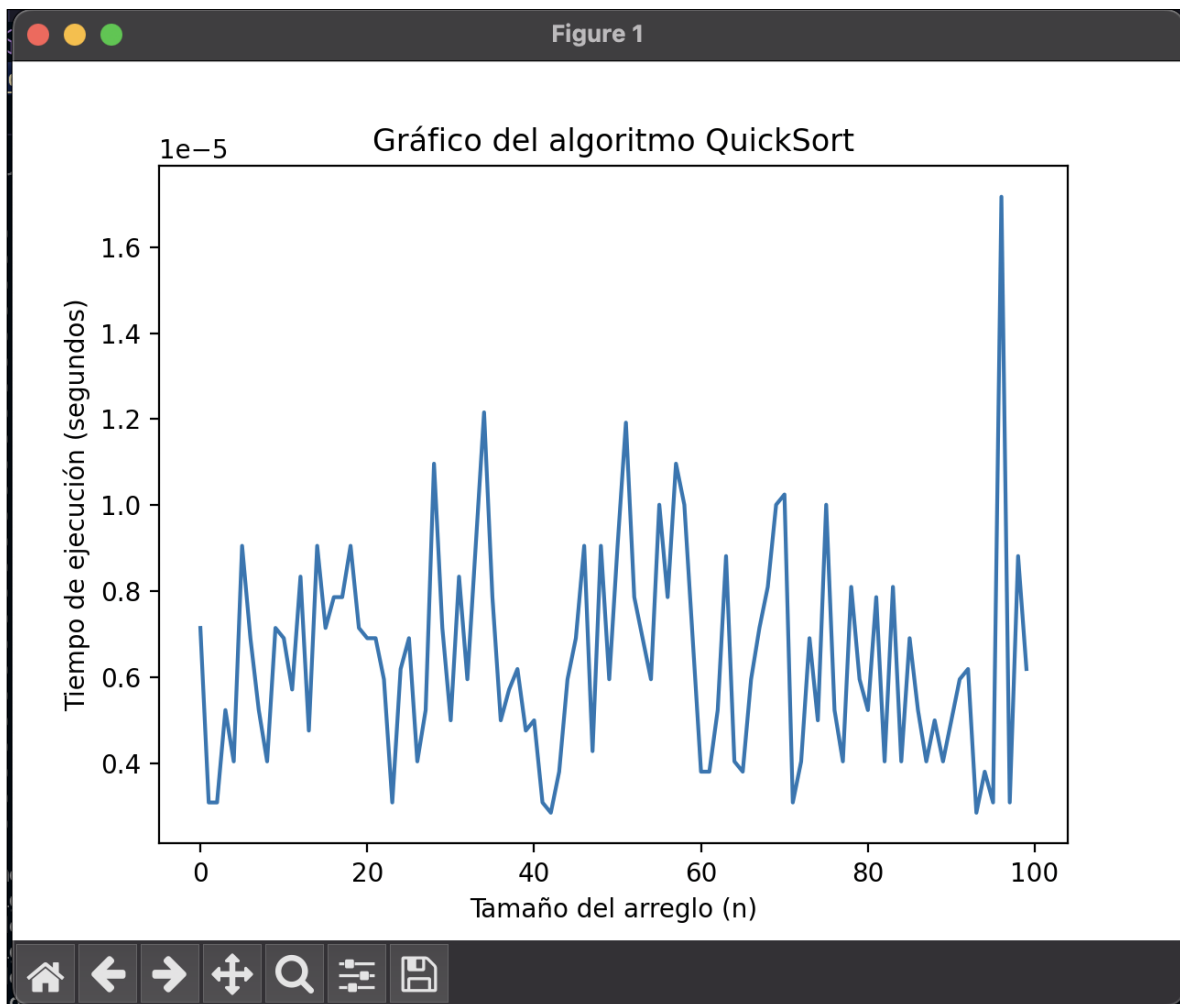


Figura 1: Gráfico de los casos de prueba

Además de esto, al final se muestran los arreglos desordenados y ordenados de los 3 casos de prueba que tuvieron menor tiempo de ejecución y los 3 casos que tardaron más tiempo. Esto se logra

creando una lista con todos los tiempos de ejecución y sus posiciones, para luego ordenarlas de forma ascendente, de esta forma imprimir los tres primeros arreglos y los tres últimos.

Listing 11: Cálculo del tiempo de ejecución en el main

```
1 Resultados = sorted(Resultados, key=lambda x: x[0])
2
3 print("\n-> Arreglos con menor tiempo de ejecución")
4 for i, (tiempo, arreglo_original, arreglo_ordenado) in enumerate(
5     Resultados[:3], 1):
6     print(f"    El arreglo original {x.index(arreglo_original)} es: {
7         arreglo_original}")
8     print(f"    El arreglo ordenado {x.index(arreglo_ordenado)} es: {
9         arreglo_ordenado}")
10
11 print("\n-> Arreglos con mayor tiempo de ejecución")
12 for i, (tiempo, arreglo_original, arreglo_ordenado) in enumerate(
13     Resultados[-3:], 1):
14     print(f"    El arreglo original {x.index(arreglo_original)} es: {
15         arreglo_original}")
16     print(f"    El arreglo ordenado {x.index(arreglo_ordenado)} es: {
17         arreglo_ordenado}")
```

## 2.5. Modificación del pivote

Para esta parte del código, lo único que hice fue dejar de usar la función que utilizaba aparte para calcular la media del arreglo y puse que usara como pivote el primer elemento del arreglo, por lo tanto solo se modificó la función de QuickSort, quedando así:

Listing 12: Función QuickSort con la modificación del pivote

```
1 def QuickSort(x):
2     #Se define el caso base, que dice que cuando un arreglo solo tenga
3     #un elemento pues ya estará ordenado
4     if len(x) <= 1:
5         return x
6     #Se define el pivote como el primer número del arreglo
7     p = x[0]
8     #Se definen los arreglos donde clasificaremos los elementos luego
9     #de las comparaciones
10    izq = []
11    der = []
12    igual = []
13    #Ciclo que asigna cada valor del arreglo original a los arreglos
14    #resultantes de la división
15    for e in x:
16        if e < p:
17            izq.append(e)
18        elif e > p:
19            der.append(e)
20        else:
```



```

18         #Se crea tambien un arreglo igual para el caso en que haya
           numeros repetidos
19         #0 iguales al pivote
           igual.append(e)
20
21     #Se definen los dos arreglos en una variable al momento de
           volverlas a llamar
22     izq_ordenado=QuickSort(izq)
23     der_ordenado=QuickSort(der)
24     #Se regresan los arreglos resultantes de las divisiones del
           arreglo original
25     return izq_ordenado + igual + der_ordenado

```

Para calcular una muestra de que los arreglo igual se siguen ordenando, hice un aprueba con 5 arreglo para no volver a poner capturas de 100, pero igual funciona para 100.

```

QuickSort

-> Arreglos generados aleatoriamente
El arreglo 0 es: [16, 16, 10, 18, 8, 15, 17, 14, 12, 3]
El arreglo 1 es: [12, 13, 5, 14, 7, 19, 18, 7, 13, 8]
El arreglo 2 es: [4, 2, 12, 16, 3, 6, 19, 19, 15, 1, 15, 13]
El arreglo 3 es: [19, 12, 19, 1, 9, 11, 12, 18, 2, 10, 17, 6, 11, 10]
El arreglo 4 es: [19, 8, 17, 6, 15, 16, 4, 2, 7, 19, 2]

-> Arreglos ordenados
El arreglo 0 ordenado es: [3, 8, 10, 12, 14, 15, 16, 16, 17, 18]
El arreglo 1 ordenado es: [5, 7, 7, 8, 12, 13, 13, 14, 18, 19]
El arreglo 2 ordenado es: [1, 2, 3, 4, 6, 12, 13, 15, 15, 16, 19, 19]
El arreglo 3 ordenado es: [1, 2, 6, 9, 10, 10, 11, 11, 12, 12, 17, 18, 19, 19]
El arreglo 4 ordenado es: [2, 2, 4, 6, 7, 8, 15, 16, 17, 19, 19]

-> Tiempos de ejecucion
Tiempo tardado en ordenar el arreglo 0: 6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 1: 6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 2: 6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 3: 1.0013580322265625e-05
Tiempo tardado en ordenar el arreglo 4: 6.9141387939453125e-06

-> Arreglos con menor tiempo de ejecución
El arreglo original 0 es: [16, 16, 10, 18, 8, 15, 17, 14, 12, 3]
El arreglo ordenado 0 es: [3, 8, 10, 12, 14, 15, 16, 16, 17, 18]
El arreglo original 1 es: [12, 13, 5, 14, 7, 19, 18, 7, 13, 8]
El arreglo ordenado 1 es: [5, 7, 7, 8, 12, 13, 13, 14, 18, 19]
El arreglo original 2 es: [4, 2, 12, 16, 3, 6, 19, 19, 15, 1, 15, 13]
El arreglo ordenado 2 es: [1, 2, 3, 4, 6, 12, 13, 15, 15, 16, 19, 19]

-> Arreglos con mayor tiempo de ejecución
El arreglo original 2 es: [4, 2, 12, 16, 3, 6, 19, 19, 15, 1, 15, 13]
El arreglo ordenado 2 es: [1, 2, 3, 4, 6, 12, 13, 15, 15, 16, 19, 19]
El arreglo original 4 es: [19, 8, 17, 6, 15, 16, 4, 2, 7, 19, 2]
El arreglo ordenado 4 es: [2, 2, 4, 6, 7, 8, 15, 16, 17, 19, 19]
El arreglo original 3 es: [19, 12, 19, 1, 9, 11, 12, 18, 2, 10, 17, 6, 11, 10]
El arreglo ordenado 3 es: [1, 2, 6, 9, 10, 10, 11, 11, 12, 12, 17, 18, 19, 19]
2023-10-24 21:53:43.133 Python[3540:227572] WARNING: Secure coding is automatically e

```

Figura 2: Prueba de la modificacion del pivote.

### 3. Codigo completo

Listing 13: Código completo

```

1 #Libreria para el tiempo
2 import time
3 #Libreria para poner datos aleatorios
4 import random
5 #Libreria para crear la grafica
6 import matplotlib.pyplot as plt
7
8 #Funcion que genera el tama o de el arreglo
9 def Tam():
10     #Genera el tama o aleatoriamente
11     t=random.randrange(5,15)
12     return t
13
14 #Funcion que genera el arreglo
15 def GenA(n):
16     #Llena el arreglo aleatoriamente
17     A=[random.randrange(1,20) for i in range(n)]
18     return A
19
20 #Funcion que genera una lista donde se guardan los 100 arreglos
21 def Arreglos():
22     #Genera la lista vacia
23     a=[]
24     #Hace que nos genere tantos arreglos como queramos
25     for i in range(100):
26         #Va agregando a la lista cada arreglo generado
27         a.append(GenA(Tam()))
28     return a
29
30 def Media(x):
31     #Usa sum, nos ayuda a sumar todos los elementos de un arreglo
32     #La doble linea redondea la division a un numero entero
33     #Len calcula la longitud del arreglo
34     m=sum(x)//len(x)
35     return m
36
37 def QuickSort(x):
38     #Se define el caso base, que dice que cuando un arreglo solo tenga
39     #un elemento pues ya estara ordenado
40     if len(x)<=1:
41         return x
42     #Se define el pivote, que ya se calculo anteriormente en la
43     #funcion media
44     m=Media(x)
45     p=m
46     #Se definen los arreglos donde clasificaremos los elementos luego
47     #de las comparaciones
48     izq=[]
49     der=[]
50     igual=[]

```

```

48     #Cilco que asigna cada valor del arreglo original a los arreglos
        resultantes de la division
49     for e in x:
50         if e<p:
51             izq.append(e)
52         elif e>p:
53             der.append(e)
54         else:
55             #Se crea tambien un arreglo igual para el caso en que haya
                numeros repetidos
56             #0 iguales al pivote
57             igual.append(e)
58     #Se definen los dos arreglos en una variable al momento de
        volverlas a llamar
59     izq_ordenado=QuickSort(izq)
60     der_ordenado=QuickSort(der)
61     #Se regresan los arreglos resultantes de las divisiones del
        arreglo original
62     return izq_ordenado + igual + der_ordenado
63
64 #Funcion para calcular el tiempo de ejecucion de una funcion
65 def TiempoEjecucion(func, *args):
66     #Tomamos el tiempo de ejecucion inicial
67     inicio=time.time()
68     #Llama a la funcion de la que queremos el tiempo de ejecucion
69     func(*args)
70     #Tomamos el tiempo de ejecucion final
71     fin=time.time()
72     #Calculamos el tiempo de ejecucion real
73     TiempoReal=fin-inicio
74     #Regresamos como resultado de la funcion el tiempo de ejecucion
75     return TiempoReal
76
77 print("\n\t\t\tQuickSort")
78 n=Tam()
79 #Lista que contiene cada arreglo
80 x=Arreglos()
81
82 #Imprime cada elemento de la lista
83 print("\n-> Arreglos generados aleatoriamente")
84 Tama os=[]
85 for i in x:
86     print(f"    El arreglo {x.index(i)} es: {i}")
87     Tama os.append(len(i))
88
89 #Manda cada arreglo a la funcion QuickSort para que ordene de uno por
    uno
90 print("\n-> Arreglos ordenados")
91 ArregloOrdenado=[]
92 for i in x:
93     m=Media(i)
94     #Cada arreglo ordenado lo guarda en una variable

```

```

95     ordenado=QuickSort(i)
96     ArregloOrdenado.append(ordenado)
97     #Imprime cada arreglo uno por uno ordenado
98     print(f"    El arreglo {x.index(i)} ordenado es: {ordenado}")
99
100 #Manda cada ejecucion de la funcion QuickSort
101 print("\n-> Tiempos de ejecucion")
102 Tiempos=[]
103 Resultados=[]
104 for i in x:
105     #Cada tiempo de ejecucion lo guardaremos en la variable tiempo
106     Tiempo=TiempoEjecucion(QuickSort,i)
107     #Imprime el tiempo de ejecucion de cada arreglo generado
108     print(f"    Tiempo tardado en ordenar el arreglo {x.index(i)}: {
109         Tiempo}")
109     #Se guarda cada tiempo de ejecucion en un arreglo
110     Tiempos.append(Tiempo)
111     Resultados.append((Tiempo, i, QuickSort(i)))
112
113 Resultados = sorted(Resultados, key=lambda x: x[0])
114
115 print("\n-> Arreglos con menor tiempo de ejecuci n")
116 for i, (tiempo, arreglo_original, arreglo_ordenado) in enumerate(
117     Resultados[:3], 1):
118     print(f"    El arreglo original {x.index(arreglo_original)} es: {
119         arreglo_original}")
119     print(f"    El arreglo ordenado {x.index(arreglo_original)} es: {
120         arreglo_ordenado}")
120
121 print("\n-> Arreglos con mayor tiempo de ejecuci n")
122 for i, (tiempo, arreglo_original, arreglo_ordenado) in enumerate(
123     Resultados[-3:], 1):
124     print(f"    El arreglo original {x.index(arreglo_original)} es: {
125         arreglo_original}")
125     print(f"    El arreglo ordenado {x.index(arreglo_original)} es: {
126         arreglo_ordenado}")
126
127 plt.xlabel('Tamaño del arreglo (n)')
128 plt.ylabel('Tiempo de ejecuci n (segundos)')
129 plt.title('Gráfico del algoritmo QuickSort')
130
131 # Usamos n como el tamaño del arreglo en el eje X y Tiempos en el eje
132     Y
133 plt.plot(Tiempos)
134 plt.show()

```

## 4. Resultados (Capturas)

```

QuickSort
-> Arreglos generados aleatoriamente
El arreglo 0 es: [13, 19, 8, 6, 15, 1, 19, 2, 2]
El arreglo 1 es: [12, 8]
El arreglo 2 es: [10, 10, 14, 9]
El arreglo 3 es: [6, 13, 1, 3]
El arreglo 4 es: [14, 18, 19, 2, 5, 3, 18, 7]
El arreglo 5 es: [12, 14, 9, 17, 10]
El arreglo 6 es: [1, 16, 14, 4, 6]
El arreglo 7 es: [15, 2, 7, 16, 17]
El arreglo 8 es: [11, 16, 11, 14, 10, 14, 18, 14]
El arreglo 9 es: [18]
El arreglo 10 es: [2, 5, 10, 13, 6, 5, 12, 13]
El arreglo 11 es: [8, 7, 1, 4]
El arreglo 12 es: [3, 4, 3, 7, 7, 12, 12, 15, 10]
El arreglo 13 es: [1, 7]
El arreglo 14 es: [7, 14, 4, 5, 17, 2]
El arreglo 15 es: [9, 11, 5, 11, 13, 15, 11, 13, 18]
El arreglo 16 es: [17, 14, 13, 16, 1]
El arreglo 17 es: [6, 15, 2, 1]
El arreglo 18 es: [19, 14, 3, 13, 2, 13, 4, 13]
El arreglo 19 es: [10]
El arreglo 20 es: [18, 12, 11, 16, 8]
El arreglo 21 es: [10, 5, 6, 13, 6]
El arreglo 22 es: [13, 3, 4, 11, 11, 7, 19, 1]
El arreglo 23 es: [13, 19]
El arreglo 24 es: [6, 15, 13, 14, 18, 8]
El arreglo 25 es: [10, 18, 6]
El arreglo 26 es: [1, 8, 13, 5, 12, 8, 9]
El arreglo 27 es: [9, 17, 7, 8, 15, 8, 11, 7]
El arreglo 28 es: [17, 3, 9]
El arreglo 29 es: [15, 7, 18, 12, 12, 11, 10]
El arreglo 30 es: [4, 1, 15]
El arreglo 31 es: [5, 7, 12, 7, 8, 13, 1, 18]
El arreglo 32 es: [6, 7, 13, 18]
El arreglo 33 es: [15, 19, 12, 1, 4, 1]
El arreglo 34 es: [3, 7, 19, 3, 11, 2, 15, 10]
El arreglo 35 es: [6, 15, 4, 9, 19]
El arreglo 36 es: [4, 19, 1, 17]

```

Figura 3: Casos de prueba (Parte 1)

```

El arreglo 37 es: [13, 10, 10, 2, 12, 17, 19, 11, 3, 18, 9, 17, 12, 8]
El arreglo 38 es: [15, 2, 8, 13, 6, 1, 10]
El arreglo 39 es: [14, 6, 5, 16, 12, 18, 18, 5, 5, 6, 16, 8, 1]
El arreglo 40 es: [13, 7, 3, 9, 4, 4, 2]
El arreglo 41 es: [4, 15, 17, 8, 5, 9, 11, 9, 6, 13, 13]
El arreglo 42 es: [14, 10, 8, 8, 8]
El arreglo 43 es: [1, 5, 11, 7, 18, 11, 6, 18, 16, 16, 5]
El arreglo 44 es: [14, 2, 5, 5, 6]
El arreglo 45 es: [8, 1, 13, 11, 9, 3, 6, 14, 4, 19]
El arreglo 46 es: [6, 4, 13, 13, 6, 10, 3, 16]
El arreglo 47 es: [4, 6, 7, 16, 15, 8]
El arreglo 48 es: [12, 18, 8, 4, 17]
El arreglo 49 es: [5, 3, 3, 14, 16, 8, 16, 5, 9, 3, 18, 12, 5, 17]
El arreglo 50 es: [13, 7, 18, 3, 8, 14, 5, 8, 17, 15, 12, 4]
El arreglo 51 es: [5, 7, 12, 8, 17, 14, 17, 1, 17, 14]
El arreglo 52 es: [11, 5, 3, 12, 8, 18, 11, 18, 13, 4, 19, 1, 11]
El arreglo 53 es: [1, 19, 1, 1, 17, 10, 8, 14, 11, 14, 13, 8]
El arreglo 54 es: [15, 2, 15, 5, 2, 8, 11, 4, 10, 6, 15, 7, 17, 3]
El arreglo 55 es: [6, 18, 4, 4, 7, 15, 7, 14, 8, 13, 3]
El arreglo 56 es: [18, 1, 13, 12, 14]
El arreglo 57 es: [14, 16, 13, 14, 12]
El arreglo 58 es: [7, 16, 12, 8, 10, 19]
El arreglo 59 es: [6, 18, 2, 9, 18, 9, 18, 6, 6]
El arreglo 60 es: [13, 12, 14, 14, 7, 2, 19, 15, 11]
El arreglo 61 es: [16, 17, 2, 7, 7, 15, 2, 9, 13, 6, 7, 18, 4]
El arreglo 62 es: [13, 9, 2, 2, 5, 14, 4, 12, 7, 14, 2, 8]
El arreglo 63 es: [9, 10, 15, 4, 6, 15, 11]
El arreglo 64 es: [12, 6, 13, 1, 2, 18, 5]
El arreglo 65 es: [14, 4, 4, 10, 6, 6, 2, 8, 4]
El arreglo 66 es: [16, 19, 2, 19, 19]
El arreglo 67 es: [11, 6, 18, 9, 9, 6, 6, 15, 2, 9, 17, 15, 13]
El arreglo 68 es: [3, 3, 9, 8, 15, 11, 7]
El arreglo 69 es: [3, 3, 15, 6, 17, 19, 1, 2, 3, 6, 17, 9, 12]
El arreglo 70 es: [5, 3, 5, 6, 9, 16, 6, 4, 1, 11, 13]
El arreglo 71 es: [18, 19, 13, 2, 13, 10, 11, 6, 5, 10, 17, 10, 11, 6]
El arreglo 72 es: [7, 18, 8, 3, 4]
El arreglo 73 es: [19, 9, 19, 1, 13, 10, 9, 13, 4, 16, 15, 14, 3]
El arreglo 74 es: [15, 15, 12, 3, 16, 7, 4, 6, 5, 12]
El arreglo 75 es: [18, 3, 13, 7, 6, 15, 18, 19]
El arreglo 76 es: [3, 4, 1, 7, 2]
El arreglo 77 es: [12, 7, 12, 9, 10, 2, 6, 1, 14, 3, 5]
El arreglo 78 es: [4, 15, 4, 19, 9, 19, 14, 11, 7, 13, 3, 17, 12, 6]
El arreglo 79 es: [14, 6, 11, 19, 17, 17, 2, 5]

```

Figura 4: Casos de prueba (Parte 2)

```

El arreglo 80 es: [1, 4, 13, 7, 10, 13, 19, 11]
El arreglo 81 es: [1, 1, 12, 4, 17, 8, 9, 8]
El arreglo 82 es: [17, 13, 3, 10, 15, 5, 15, 19, 12]
El arreglo 83 es: [12, 3, 15, 17, 17, 12, 6, 10, 11]
El arreglo 84 es: [14, 3, 8, 13, 15, 10, 18, 14, 11, 1]
El arreglo 85 es: [19, 11, 8, 7, 5, 13, 11, 5, 12, 15]
El arreglo 86 es: [14, 2, 5, 5, 16, 8, 11, 11, 13, 10, 16, 15, 13]
El arreglo 87 es: [15, 4, 3, 11, 3, 17, 10, 17]
El arreglo 88 es: [13, 5, 7, 14, 13, 8, 4, 14, 7, 16, 8, 13, 14, 15]
El arreglo 89 es: [6, 14, 16, 17, 13, 18, 10, 15, 14]
El arreglo 90 es: [16, 10, 11, 1, 10, 5, 14, 14, 10, 16, 15]
El arreglo 91 es: [3, 6, 9, 3, 16, 19]
El arreglo 92 es: [11, 3, 5, 5, 15]
El arreglo 93 es: [16, 16, 12, 5, 13, 6, 9, 12]
El arreglo 94 es: [12, 3, 1, 18, 15]
El arreglo 95 es: [7, 15, 18, 9, 5, 18, 3, 6, 5, 14, 2, 6]
El arreglo 96 es: [11, 1, 7, 15, 17, 3, 8, 9, 19, 8, 13, 2, 2, 13]
El arreglo 97 es: [4, 5, 12, 9, 9, 8, 1, 14, 1, 5, 3, 8]
El arreglo 98 es: [10, 13, 14, 14, 9, 2, 13, 9]
El arreglo 99 es: [14, 9, 17, 7, 4, 12]

```

Figura 5: Casos de prueba (Parte 3)

```

-> Arreglos ordenados
El arreglo 0 ordenado es: [1, 4, 5, 7, 10, 13, 19, 19]
El arreglo 1 ordenado es: [3, 9, 11, 15, 15, 16]
El arreglo 2 ordenado es: [5, 11, 13, 13, 14, 15]
El arreglo 3 ordenado es: [6, 6, 6, 9, 12, 14, 15]
El arreglo 4 ordenado es: [2, 7, 9, 18, 18]
El arreglo 5 ordenado es: [1, 2, 4, 7, 8, 10, 10, 10, 11, 13, 14, 17]
El arreglo 6 ordenado es: [1, 2, 5, 6, 8, 10, 11, 15, 18]
El arreglo 7 ordenado es: [2, 2, 4, 5, 14, 15]
El arreglo 8 ordenado es: [3, 13, 15, 16, 17]
El arreglo 9 ordenado es: [4, 4, 5, 7, 13, 14, 15, 15, 16, 18]
El arreglo 10 ordenado es: [1, 3, 3, 5, 7, 9, 12, 16, 16]
El arreglo 11 ordenado es: [6, 7, 10, 17, 17, 17, 18, 18, 19, 19]
El arreglo 12 ordenado es: [1, 4, 4, 4, 6, 7, 7, 11, 13, 16, 16, 18]
El arreglo 13 ordenado es: [1, 2, 3, 10, 13, 17, 19]
El arreglo 14 ordenado es: [3, 5, 7, 9, 9, 11, 12, 14, 16, 17, 18]
El arreglo 15 ordenado es: [2, 3, 3, 4, 5, 6, 9, 16, 17, 17, 18, 19]
El arreglo 16 ordenado es: [6, 7, 8, 9, 9, 10, 11, 11, 12, 14, 14, 15, 16]
El arreglo 17 ordenado es: [1, 3, 4, 4, 4, 8, 9, 10, 10, 11, 13, 14, 14, 16]
El arreglo 18 ordenado es: [1, 5, 7, 8, 8, 10, 11, 12, 14, 14, 15, 18, 19]
El arreglo 19 ordenado es: [1, 2, 3, 6, 9, 12, 13, 14, 17, 17]
El arreglo 20 ordenado es: [1, 7, 10, 11, 14, 14, 16, 17, 17, 19]
El arreglo 21 ordenado es: [2, 4, 6, 8, 12, 13, 17, 19]
El arreglo 22 ordenado es: [3, 6, 6, 7, 7, 9, 16, 18, 19]
El arreglo 23 ordenado es: [6, 7, 11, 14, 19]
El arreglo 24 ordenado es: [6, 10, 12, 15, 17, 17, 19, 19]
El arreglo 25 ordenado es: [3, 3, 4, 5, 8, 9, 10, 12, 14, 19]
El arreglo 26 ordenado es: [2, 3, 5, 6, 9, 10]
El arreglo 27 ordenado es: [1, 1, 6, 9, 12, 17]
El arreglo 28 ordenado es: [1, 3, 4, 6, 7, 7, 8, 13, 13, 16, 17, 17, 18]
El arreglo 29 ordenado es: [3, 4, 9, 9, 11, 12, 13, 15, 19, 19]
El arreglo 30 ordenado es: [1, 12, 13, 13, 15, 16, 17, 19]
El arreglo 31 ordenado es: [5, 5, 6, 7, 7, 12, 12, 14, 15, 17]
El arreglo 32 ordenado es: [1, 1, 2, 3, 6, 10, 17]
El arreglo 33 ordenado es: [1, 2, 3, 4, 10, 12, 13, 13, 14, 14, 17, 17]
El arreglo 34 ordenado es: [2, 4, 4, 9, 10, 10, 12, 14, 15, 16, 17, 18, 18]
El arreglo 35 ordenado es: [1, 3, 4, 4, 10, 10, 12, 16, 17, 19]
El arreglo 36 ordenado es: [11, 11, 15, 16, 17, 18]
El arreglo 37 ordenado es: [3, 3, 9, 14, 14, 16, 17]
El arreglo 38 ordenado es: [1, 6, 7, 8, 9, 12, 13, 19, 19]
El arreglo 39 ordenado es: [6, 6, 8, 11, 11, 13, 14]
El arreglo 40 ordenado es: [1, 6, 8, 8, 8, 9, 9, 13, 14, 14, 16]
El arreglo 41 ordenado es: [5, 6, 8, 11, 11]
El arreglo 42 ordenado es: [1, 1, 3, 6, 12, 17]
El arreglo 43 ordenado es: [1, 3, 11, 18, 19]
El arreglo 44 ordenado es: [3, 3, 3, 4, 9, 12, 16, 17, 17, 19]
El arreglo 45 ordenado es: [1, 1, 3, 4, 6, 7, 10, 10, 11, 13]
El arreglo 46 ordenado es: [1, 2, 4, 6, 6, 8, 12, 12, 12, 16, 17, 17]
El arreglo 47 ordenado es: [2, 3, 5, 8, 12]
El arreglo 48 ordenado es: [4, 4, 6, 6, 8, 10, 12, 13, 14, 14, 15, 17, 19]
El arreglo 49 ordenado es: [3, 4, 9, 12, 13, 15, 16, 17]
El arreglo 50 ordenado es: [5, 6, 11, 11, 12, 13, 13, 14, 14, 15, 15, 18, 18]
El arreglo 51 ordenado es: [1, 2, 6, 6, 8, 10, 13, 14, 14, 16, 17, 18, 18, 18]
El arreglo 52 ordenado es: [4, 5, 7, 7, 10, 10, 10, 11, 12, 12, 14, 18]
El arreglo 53 ordenado es: [3, 5, 5, 7, 8, 10, 10, 10, 15, 16, 18]
El arreglo 54 ordenado es: [1, 4, 8, 12, 12, 15, 15, 15, 17, 17, 18]
El arreglo 55 ordenado es: [2, 3, 6, 7, 7, 8, 8, 9, 9, 10, 12, 15, 16]
El arreglo 56 ordenado es: [4, 6, 7, 9, 11, 11, 11, 12, 12, 12, 14, 15, 16]
El arreglo 57 ordenado es: [2, 3, 7, 9, 12, 14, 14, 16, 16, 19]
El arreglo 58 ordenado es: [4, 5, 5, 6, 7, 7, 7, 10, 11, 11, 12, 13, 14, 14]
El arreglo 59 ordenado es: [1, 2, 2, 8, 12, 12, 14, 15, 16, 19]
El arreglo 60 ordenado es: [1, 7, 10, 11, 15, 17]
El arreglo 61 ordenado es: [4, 8, 10, 13, 14]
El arreglo 62 ordenado es: [1, 4, 4, 5, 6, 12, 17, 19]
El arreglo 63 ordenado es: [6, 7, 7, 7, 7, 7, 14, 14, 17, 18, 18]
El arreglo 64 ordenado es: [2, 10, 12, 15, 18, 19]
El arreglo 65 ordenado es: [3, 3, 12, 12, 19]
El arreglo 66 ordenado es: [3, 4, 11, 13, 14, 15, 15, 19]
El arreglo 67 ordenado es: [3, 6, 6, 7, 7, 10, 12, 14, 14]
El arreglo 68 ordenado es: [2, 4, 5, 10, 10, 12, 13, 16, 16, 19]
El arreglo 69 ordenado es: [2, 2, 3, 6, 7, 8, 9, 10, 11, 11, 16, 17, 17]
El arreglo 70 ordenado es: [2, 5, 7, 7, 8, 8, 10, 11, 12, 12, 14, 15, 18]
El arreglo 71 ordenado es: [13, 17, 17, 18, 19]
El arreglo 72 ordenado es: [3, 6, 8, 8, 15, 16]
El arreglo 73 ordenado es: [1, 4, 4, 5, 7, 8, 13, 14, 14, 16, 17]
El arreglo 74 ordenado es: [1, 5, 6, 8, 9, 16, 17, 19]
El arreglo 75 ordenado es: [2, 2, 8, 10, 13, 13, 13, 14, 16, 16, 19, 19]
El arreglo 76 ordenado es: [5, 7, 12, 16, 18, 18, 19]
El arreglo 77 ordenado es: [1, 2, 3, 7, 17, 18]
El arreglo 78 ordenado es: [1, 1, 3, 7, 7, 12, 13, 14, 15, 18, 19]
El arreglo 79 ordenado es: [1, 7, 9, 11, 11, 15, 16]

```

Figura 6: Arreglos ordenados (Parte 1)

Figura 7: Arreglos ordenados (Parte 2)

```

El arreglo 79 ordenado es: [1, 7, 9, 11, 11, 15, 16]
El arreglo 80 ordenado es: [2, 5, 6, 11, 14, 16, 17, 19]
El arreglo 81 ordenado es: [5, 5, 5, 9, 10, 11, 14, 14, 15, 17, 17, 18]
El arreglo 82 ordenado es: [1, 8, 11, 11, 19]
El arreglo 83 ordenado es: [2, 2, 2, 4, 5, 7, 7, 9, 9, 14, 19]
El arreglo 84 ordenado es: [2, 3, 11, 18, 19]
El arreglo 85 ordenado es: [4, 6, 6, 6, 9, 10, 11, 15, 16, 18, 19]
El arreglo 86 ordenado es: [1, 6, 8, 9, 11, 19, 19, 19]
El arreglo 87 ordenado es: [4, 7, 10, 13, 15, 16]
El arreglo 88 ordenado es: [1, 3, 8, 9, 11, 13, 15, 17]
El arreglo 89 ordenado es: [2, 3, 4, 9, 12, 13, 14, 16]
El arreglo 90 ordenado es: [2, 7, 7, 9, 11, 15, 17, 18, 19]
El arreglo 91 ordenado es: [3, 4, 11, 12, 13, 13, 15, 17]
El arreglo 92 ordenado es: [2, 2, 3, 6, 10, 11, 12, 18, 19]
El arreglo 93 ordenado es: [8, 8, 9, 10, 12, 13]
El arreglo 94 ordenado es: [3, 9, 13, 13, 14, 18]
El arreglo 95 ordenado es: [4, 4, 6, 9, 13]
El arreglo 96 ordenado es: [1, 2, 2, 3, 4, 5, 5, 7, 9, 9, 10, 15]
El arreglo 97 ordenado es: [9, 13, 15, 16, 18]
El arreglo 98 ordenado es: [1, 2, 2, 2, 10, 10, 11, 14, 14, 15, 17, 18]
El arreglo 99 ordenado es: [3, 4, 6, 7, 15, 17, 17, 18]

```

Figura 8: Arreglos ordenados (Parte 3)



-> Tiempos de ejecucion			
Tiempo tardado en ordenar el arreglo 0:	7.152557373046875e-06	Tiempo tardado en ordenar el arreglo 41:	3.0994415283203125e-06
Tiempo tardado en ordenar el arreglo 1:	3.0994415283203125e-06	Tiempo tardado en ordenar el arreglo 42:	2.86102294921875e-06
Tiempo tardado en ordenar el arreglo 2:	3.0994415283203125e-06	Tiempo tardado en ordenar el arreglo 43:	3.814697265625e-06
Tiempo tardado en ordenar el arreglo 3:	5.245208740234375e-06	Tiempo tardado en ordenar el arreglo 44:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 4:	4.0531158447265625e-06	Tiempo tardado en ordenar el arreglo 45:	6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 5:	9.059906005859375e-06	Tiempo tardado en ordenar el arreglo 46:	9.059906005859375e-06
Tiempo tardado en ordenar el arreglo 6:	6.9141387939453125e-06	Tiempo tardado en ordenar el arreglo 47:	4.291534423828125e-06
Tiempo tardado en ordenar el arreglo 7:	5.245208740234375e-06	Tiempo tardado en ordenar el arreglo 48:	9.059906005859375e-06
Tiempo tardado en ordenar el arreglo 8:	4.0531158447265625e-06	Tiempo tardado en ordenar el arreglo 49:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 9:	7.152557373046875e-06	Tiempo tardado en ordenar el arreglo 50:	9.059906005859375e-06
Tiempo tardado en ordenar el arreglo 10:	6.9141387939453125e-06	Tiempo tardado en ordenar el arreglo 51:	1.1920928955078125e-05
Tiempo tardado en ordenar el arreglo 11:	5.7220458984375e-06	Tiempo tardado en ordenar el arreglo 52:	7.867813110351562e-06
Tiempo tardado en ordenar el arreglo 12:	8.344650268554688e-06	Tiempo tardado en ordenar el arreglo 53:	6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 13:	4.76837158203125e-06	Tiempo tardado en ordenar el arreglo 54:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 14:	9.059906005859375e-06	Tiempo tardado en ordenar el arreglo 55:	1.0013580322265625e-05
Tiempo tardado en ordenar el arreglo 15:	7.152557373046875e-06	Tiempo tardado en ordenar el arreglo 56:	7.867813110351562e-06
Tiempo tardado en ordenar el arreglo 16:	7.867813110351562e-06	Tiempo tardado en ordenar el arreglo 57:	1.0967254638671875e-05
Tiempo tardado en ordenar el arreglo 17:	7.867813110351562e-06	Tiempo tardado en ordenar el arreglo 58:	1.0013580322265625e-05
Tiempo tardado en ordenar el arreglo 18:	9.059906005859375e-06	Tiempo tardado en ordenar el arreglo 59:	6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 19:	7.152557373046875e-06	Tiempo tardado en ordenar el arreglo 60:	3.814697265625e-06
Tiempo tardado en ordenar el arreglo 20:	6.9141387939453125e-06	Tiempo tardado en ordenar el arreglo 61:	3.814697265625e-06
Tiempo tardado en ordenar el arreglo 21:	6.9141387939453125e-06	Tiempo tardado en ordenar el arreglo 62:	5.245208740234375e-06
Tiempo tardado en ordenar el arreglo 22:	5.9604644775390625e-06	Tiempo tardado en ordenar el arreglo 63:	8.821487426757812e-06
Tiempo tardado en ordenar el arreglo 23:	3.0994415283203125e-06	Tiempo tardado en ordenar el arreglo 64:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 24:	6.198883056640625e-06	Tiempo tardado en ordenar el arreglo 65:	3.814697265625e-06
Tiempo tardado en ordenar el arreglo 25:	6.9141387939453125e-06	Tiempo tardado en ordenar el arreglo 66:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 26:	4.0531158447265625e-06	Tiempo tardado en ordenar el arreglo 67:	7.152557373046875e-06
Tiempo tardado en ordenar el arreglo 27:	5.245208740234375e-06	Tiempo tardado en ordenar el arreglo 68:	8.106231689453125e-06
Tiempo tardado en ordenar el arreglo 28:	1.0967254638671875e-05	Tiempo tardado en ordenar el arreglo 69:	1.0013580322265625e-05
Tiempo tardado en ordenar el arreglo 29:	7.152557373046875e-06	Tiempo tardado en ordenar el arreglo 70:	1.0251998901367188e-05
Tiempo tardado en ordenar el arreglo 30:	5.0067901611328125e-06	Tiempo tardado en ordenar el arreglo 71:	3.0994415283203125e-06
Tiempo tardado en ordenar el arreglo 31:	8.344650268554688e-06	Tiempo tardado en ordenar el arreglo 72:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 32:	5.9604644775390625e-06	Tiempo tardado en ordenar el arreglo 73:	6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 33:	9.059906005859375e-06	Tiempo tardado en ordenar el arreglo 74:	5.0067901611328125e-06
Tiempo tardado en ordenar el arreglo 34:	1.2159347534179688e-05	Tiempo tardado en ordenar el arreglo 75:	1.0013580322265625e-05
Tiempo tardado en ordenar el arreglo 35:	7.867813110351562e-06	Tiempo tardado en ordenar el arreglo 76:	5.245208740234375e-06
Tiempo tardado en ordenar el arreglo 36:	5.0067901611328125e-06	Tiempo tardado en ordenar el arreglo 77:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 37:	5.7220458984375e-06	Tiempo tardado en ordenar el arreglo 78:	8.106231689453125e-06
Tiempo tardado en ordenar el arreglo 38:	6.198883056640625e-06	Tiempo tardado en ordenar el arreglo 79:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 39:	4.76837158203125e-06	Tiempo tardado en ordenar el arreglo 80:	5.245208740234375e-06
Tiempo tardado en ordenar el arreglo 40:	5.0067901611328125e-06	Tiempo tardado en ordenar el arreglo 81:	7.867813110351562e-06
		Tiempo tardado en ordenar el arreglo 82:	4.0531158447265625e-06
		Tiempo tardado en ordenar el arreglo 83:	8.106231689453125e-06

Figura 9: Tiempos de ejecución (Parte 1)

Figura 10: Tiempos de ejecución (Parte 2)

Tiempo tardado en ordenar el arreglo 84:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 85:	6.9141387939453125e-06
Tiempo tardado en ordenar el arreglo 86:	5.245208740234375e-06
Tiempo tardado en ordenar el arreglo 87:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 88:	5.0067901611328125e-06
Tiempo tardado en ordenar el arreglo 89:	4.0531158447265625e-06
Tiempo tardado en ordenar el arreglo 90:	5.0067901611328125e-06
Tiempo tardado en ordenar el arreglo 91:	5.9604644775390625e-06
Tiempo tardado en ordenar el arreglo 92:	6.198883056640625e-06
Tiempo tardado en ordenar el arreglo 93:	2.86102294921875e-06
Tiempo tardado en ordenar el arreglo 94:	3.814697265625e-06
Tiempo tardado en ordenar el arreglo 95:	3.0994415283203125e-06
Tiempo tardado en ordenar el arreglo 96:	1.71661376953125e-05
Tiempo tardado en ordenar el arreglo 97:	3.0994415283203125e-06
Tiempo tardado en ordenar el arreglo 98:	8.821487426757812e-06
Tiempo tardado en ordenar el arreglo 99:	6.198883056640625e-06

Figura 11: Tiempos de ejecución (Parte 3)

```

-> Arreglos con menor tiempo de ejecución
El arreglo original 42 es: [12, 6, 3, 1, 17, 1]
El arreglo ordenado 42 es: [1, 1, 3, 6, 12, 17]
El arreglo original 93 es: [12, 10, 9, 13, 8, 8]
El arreglo ordenado 93 es: [8, 8, 9, 10, 12, 13]
El arreglo original 1 es: [15, 16, 15, 11, 3, 9]
El arreglo ordenado 1 es: [3, 9, 11, 15, 15, 16]

-> Arreglos con mayor tiempo de ejecución
El arreglo original 51 es: [1, 2, 16, 18, 10, 17, 6, 14, 18, 14, 8, 13, 6, 18]
El arreglo ordenado 51 es: [1, 2, 6, 6, 8, 10, 13, 14, 14, 16, 17, 18, 18, 18]
El arreglo original 34 es: [2, 10, 4, 14, 12, 9, 17, 16, 10, 18, 18, 4, 15]
El arreglo ordenado 34 es: [2, 4, 4, 9, 10, 10, 12, 14, 15, 16, 17, 18, 18]
El arreglo original 96 es: [7, 9, 4, 5, 2, 1, 15, 3, 5, 10, 9, 2]
El arreglo ordenado 96 es: [1, 2, 2, 3, 4, 5, 5, 7, 9, 9, 10, 15]
2023-10-24 20:56:15 744 Python[3082:202534] WARNING: Secure coding is automatically enabl

```

Figura 12: Arreglos con mas y menos tiempo de ejecución

## 5. Conclusiones

Gracias a la práctica, analizamos y comprendimos a fondo el funcionamiento del algoritmo Quick-Sort. Del mismo modo, entendimos por qué es más rápido que los demás, dado que a pesar de que proporcionábamos distintos tamaños de algoritmos, el tiempo de ejecución no cambiaba de manera drástica, y el tiempo, a pesar de ser mayor que el de los arreglos con menos elementos, tampoco aumentaba demasiado.

También, en lo personal, practiqué mucho más la sintaxis de Python y comprendí de mejor manera las funciones y la utilidad de estas, y se me hizo mucho más fácil usar funciones que haberlo hecho todo dentro del ámbito del código.

## 6. Referencias

- [https://www.udb.edu.sv/udb\\_files/recursos\\_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf](https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf)
- <https://numerentur.org/quicksort/>
- <https://www.genbeta.com/desarrollo/implementando-el-algoritmo-quicksort>
- [http://aniei.org.mx/paginas/uam/CursoAA/curso\\_aa\\_19.html#:~:text=El%20mejor%20caso%20de%20quick,lista%20original%20en%20dos%20listas.](http://aniei.org.mx/paginas/uam/CursoAA/curso_aa_19.html#:~:text=El%20mejor%20caso%20de%20quick,lista%20original%20en%20dos%20listas.)