COMPARACIÓN ENTRE QUICK SORT Y MERGE SORT

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

DAVIDSON ESFLEIDER SANCHEZ GORDILLO - 20231020183

DIEGO FELIPE DIAZ ROA - 20201020147

DANIA LIZETH GUZMÁN TRIVIOÑO - 20221020061

JOSHOA ALARCON SANCHEZ - 20221020013

SEPTIEMBRE 2025

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

FACULTAD DE INGENIERÍA

CIENCIAS DE LA COMPUTACIÓN I

SIMAR ENRIQUE HERRERA JIMENEZ

QUICK SORT

El algoritmo de ordenamiento de rápido o Quick sort consiste en elegir un elemento para que sirva como pivote, en este caso, el dato que se toma para el pivote siempre es el del medio. Luego se ordena el arreglo de tal manera que todos los elementos menores que el pivote estén a la izquierda y todos los elementos mayores que el pivote estén a la derecha.

Se llama al Quick sort de forma recursiva, teniendo en cuenta el pivote anterior para subdividir adecuadamente los arreglos izquierdo y derecho.

La complejidad de tiempo del caso promedio de Quick Sort es O(n logn) y la complejidad de tiempo del peor caso es O (n^2) dependiendo de la selección del elemento pivote, que divide el arreglo actual en dos sub arreglos.

MERGE SORT

Merge Sort es un algoritmo Divide y vencerás consiste en dividir el arreglo de entrada en dos mitades, luego Ordena la mitad izquierda y la mitad derecha usando el mismo algoritmo recurrente para después fusionar las dos mitades ordenadas. La mayor parte del algoritmo tiene dos matrices ordenadas, y tenemos que fusionarlas en un único arreglo ordenado.

Este algoritmo posee un orden de complejidad O(n logn) en todos los casos, ya sea peores, mejores o en promedio, lo que lo convierte en un algoritmo muy eficiente para conjuntos de datos grandes.

ANÁLISIS COMPARATIVO

Para la práctica de esta comparativa se usó el lenguaje Python y se escribieron los siguientes códigos:

- **Quick Sort**

```python
import random

array=[random.randint(0,100) for i in range(1000)]

n = 0

def quicksort(arr):
    global n

    if(len(arr)>1):
        piv=int(len(arr)/2)
        val=arr[piv]
        n += 6

        left=[i for i in arr if i<val]
        n += len(arr)

        mid=[i for i in arr if i==val]
        n += len(arr)

        right=[i for i in arr if i>val]
        n += len(arr)

        res=quicksort(left)+ mid + quicksort(right)
        n += 4
        return res
    else:
        n += 1
        return arr

print(quicksort(array), n)
```

- **Merge Sort**

```
import random

array=[random.randint(0,100) for i in range(1000)]
n = 0

def merge(left,right,compare):
    global n
    result = []
    i,j = 0,0
    n += 2

    while (i < len(left) and j < len(right)):
        if compare(left[i],right[j]):
            result.append(left[i])
            i += 1
            n += 4
        else:
            result.append(right[j])
            j += 1
            n += 4
        n += 5

    while (i < len(left)):
        result.append(left[i])
        i += 1
        n += 4
    while (j < len(right)):
        result.append(right[j])
        j += 1
        n += 4
    n += 1
    return result
```

```
def merge_sort(arr, compare = lambda x, y: x < y):
    global n

    if len(arr) < 2:
        n += 3
        return arr[:]
    else:
        middle = len(arr) // 2
        n += 3

        left = merge_sort(arr[:middle], compare)
        n += 2

        right = merge_sort(arr[middle:], compare)
        n += 2

        n += 2
        return merge(left, right, compare)

print(merge_sort(array), n)
```

Ambos códigos tienen la capacidad de imprimir el número de pasos realizado durante la ejecución del programa con un array de 1000 elementos aleatorios, comprendidos entre 1 y 100, por lo cual se hicieron 5 ejecuciones en cada programa y se obtuvieron los siguientes resultados para el número de pasos:

# de Ejecución	**1**	**2**	**3**	**4**	**5**
QUICK SORT	23117	26270	24221	22763	22808
MERGE SORT	98422	98447	98532	98412	98537

Como se puede observar en los resultados de los test, se puede ver que a pesar de que ambos algoritmos de ordenamiento tienen el mismo nivel de complejidad, a la hora de ejecutarlos se nota una clara diferencia de eficiencia, en cuanto a pasos, entre los dos, siendo el algoritmo Quick sort el más destacado en este caso de prueba.

Esto se debe a la diferencia entre los procesos de ordenamiento principalmente, ya que la cantidad de operaciones necesarias para dividir y organizar las listas es muy diferente entre los dos algoritmos. En el caso de Quicksort, el arreglo se separa en sublistas, que se comparan con un pivote, lo que implica recorrer el arreglo unas pocas veces, reduciendo así el número de pasos totales. Mientras que el Merge Sort realiza comparaciones elemento por elemento, además de crear y copiar subarreglos en cada división, lo que significativamente la cantidad de pasos contados.