

ALGORITMOS DE ORDENAMIENTO

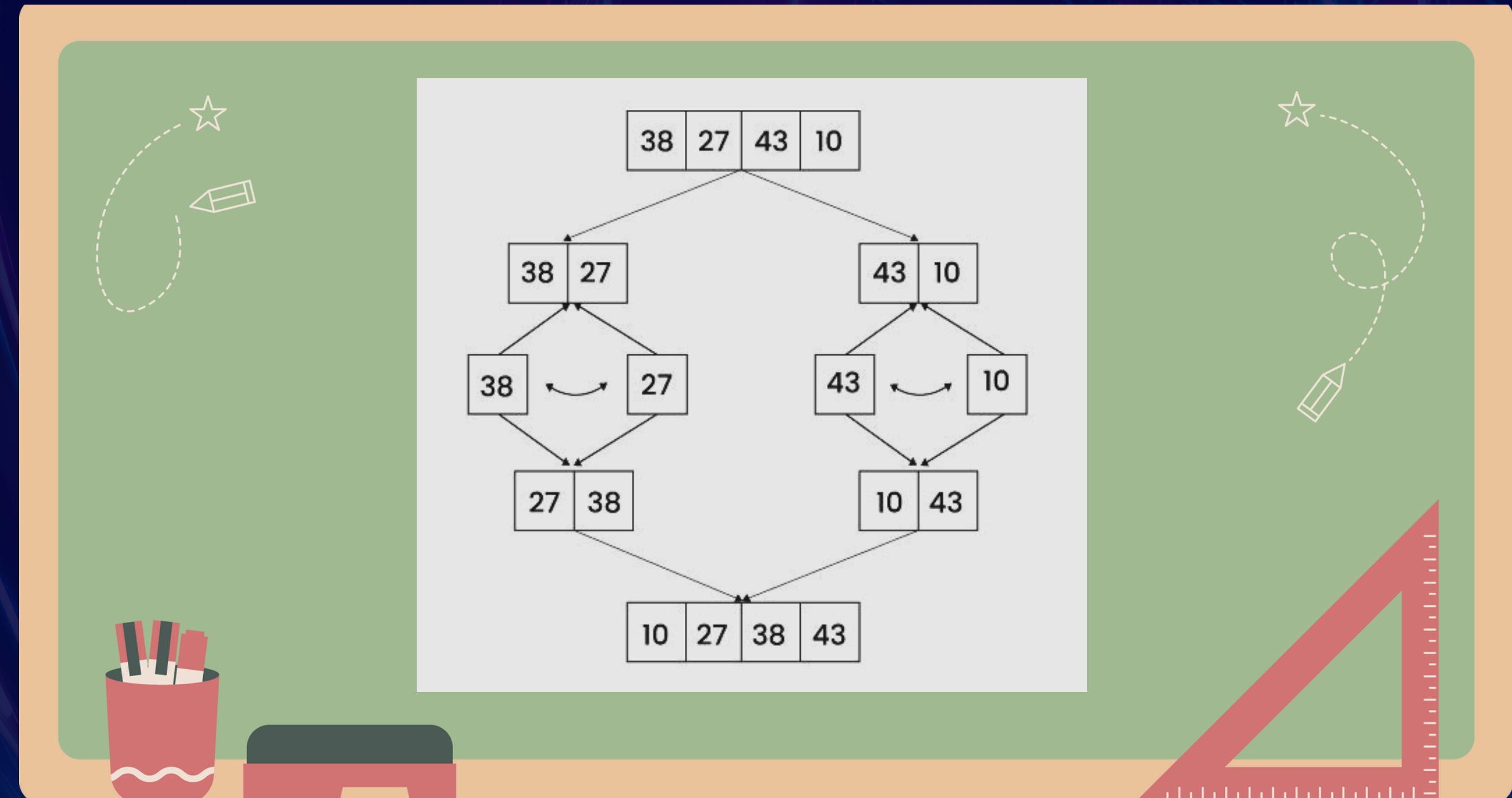
PRESENTADO POR:

RICARDO DOSE ARQUE CHUNGA

2024

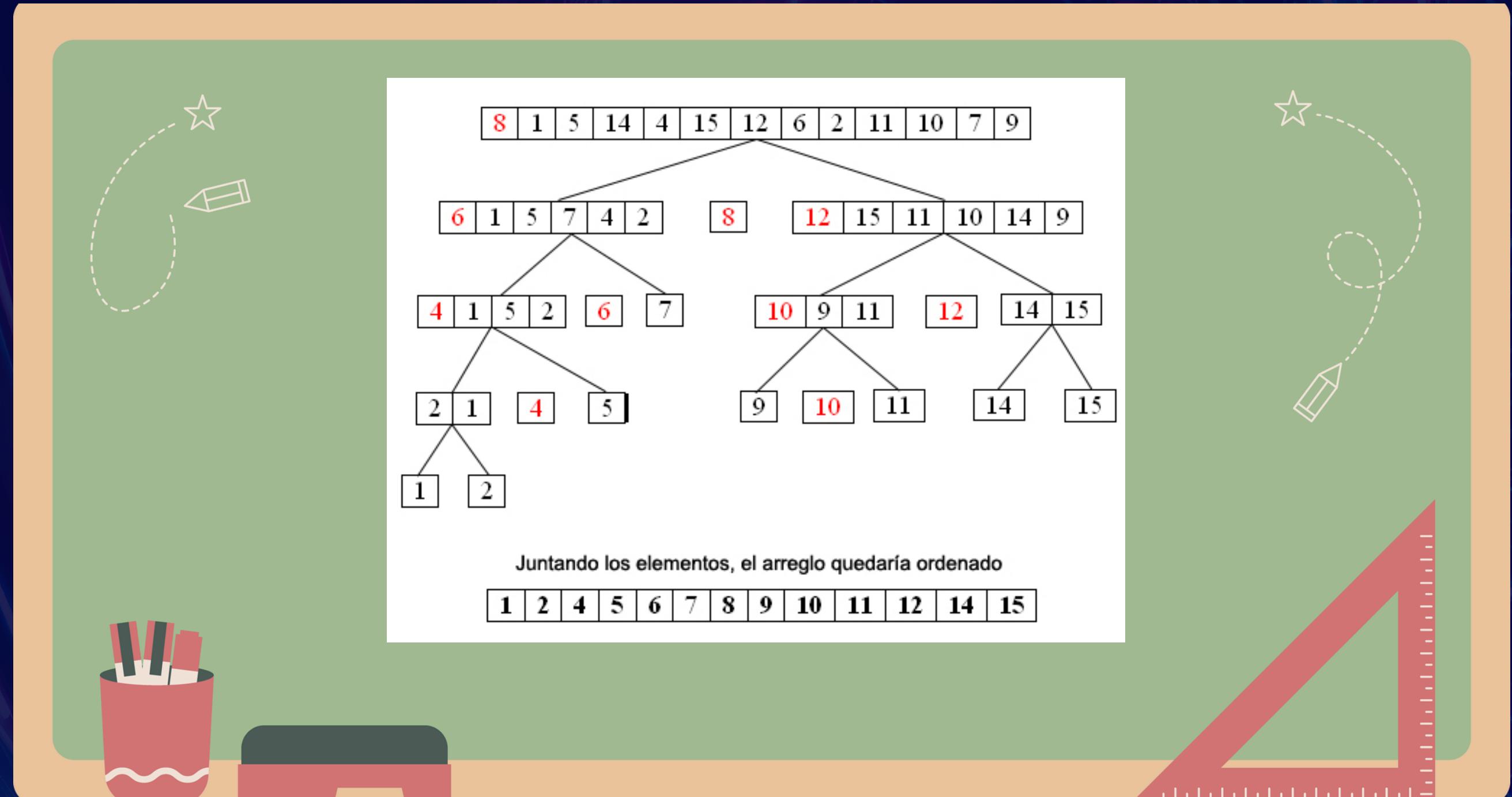
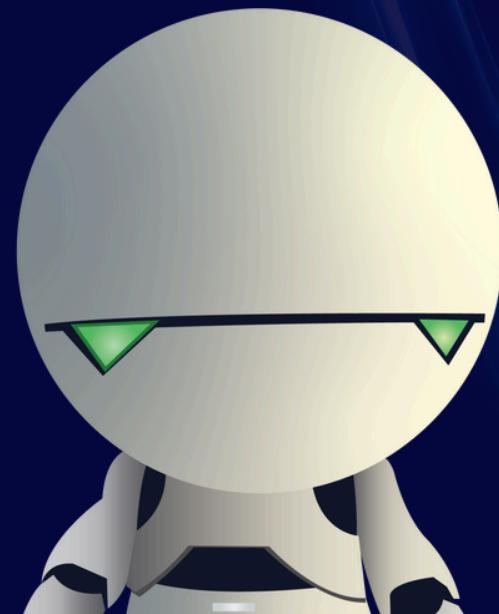
MERGE VS QUICK

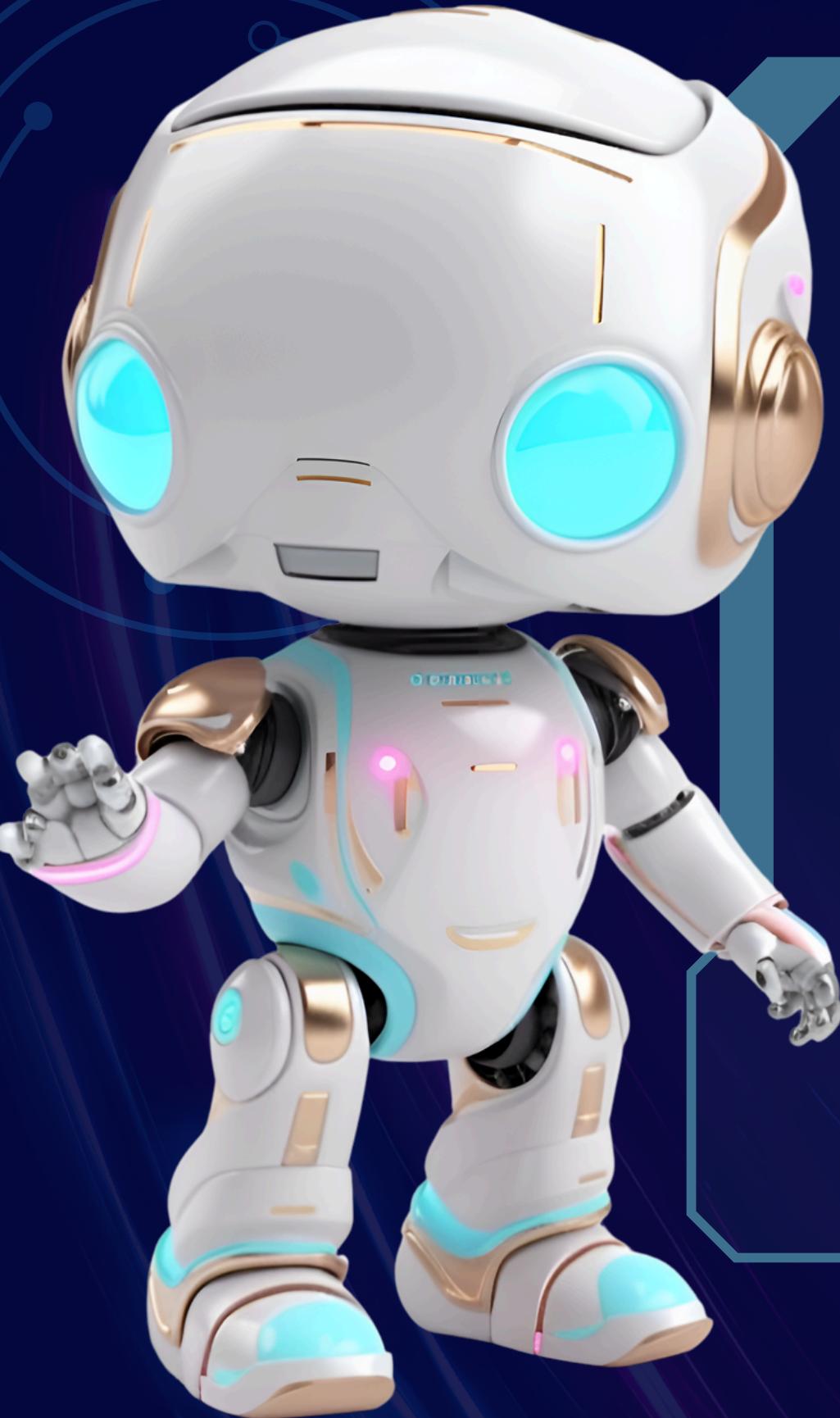
MERGE



03

QUICK





CODIFICACION EN DAVA

MERGE

```
public class MergeSort {  
  
    public void ordenarM(int[] array) {  
        mergeSort(array, 0, array.length - 1);  
    }  
  
    public void mergeSort(int[] numeros, int izq, int der){  
        if (izq < der){  
            int medio = (izq + der) / 2;  
            mergeSort(numeros, izq, medio);  
            mergeSort(numeros, medio + 1, der);  
            merge(numeros, izq, medio, der);  
        }  
    }  
  
    public void merge(int[] numeros, int izq, int medio, int der){  
        int n1 = medio - izq + 1;  
        int n2 = der - medio;  
  
        int[] izqArray = new int[n1];  
        int[] derArray = new int[n2];  
  
        for (int i = 0; i < n1; i++){  
            izqArray[i] = numeros[izq + i];  
        }  
        for (int j = 0; j < n2; j++){  
            derArray[j] = numeros[medio + 1 + j];  
        }  
  
        int i = 0, j = 0;  
        int k = izq;  
  
        while (i < n1 && j < n2){  
            if (izqArray[i] <= derArray[j]){  
                numeros[k] = izqArray[i];  
                i++;  
            }  
            else {  
                numeros[k] = derArray[j];  
                j++;  
            }  
            k++;  
        }  
    }  
}
```

```
} else {  
    numeros[k] = derArray[j];  
    j++;  
}  
k++;  
}  
  
while (i < n1){  
    numeros[k] = izqArray[i];  
    i++;  
    k++;  
}  
  
while (j < n2){  
    numeros[k] = derArray[j];  
    j++;  
    k++;  
}  
}  
  
Run main | Debug main  
public static void main(String[] args) {  
    MergeSort mergeSort = new MergeSort();  
    int[] array = {9, 4, 6, 2, 8, 5, 1, 3, 7};  
  
    mergeSort.ordenarM(array);  
  
    System.out.println("Array ordenado:");  
    for (int num : array){  
        System.out.print(num + " ");  
    }  
}
```

MERGE

```
public void ordenarM(int[] array) {  
    mergeSort(array, 0, array.length - 1);  
}
```

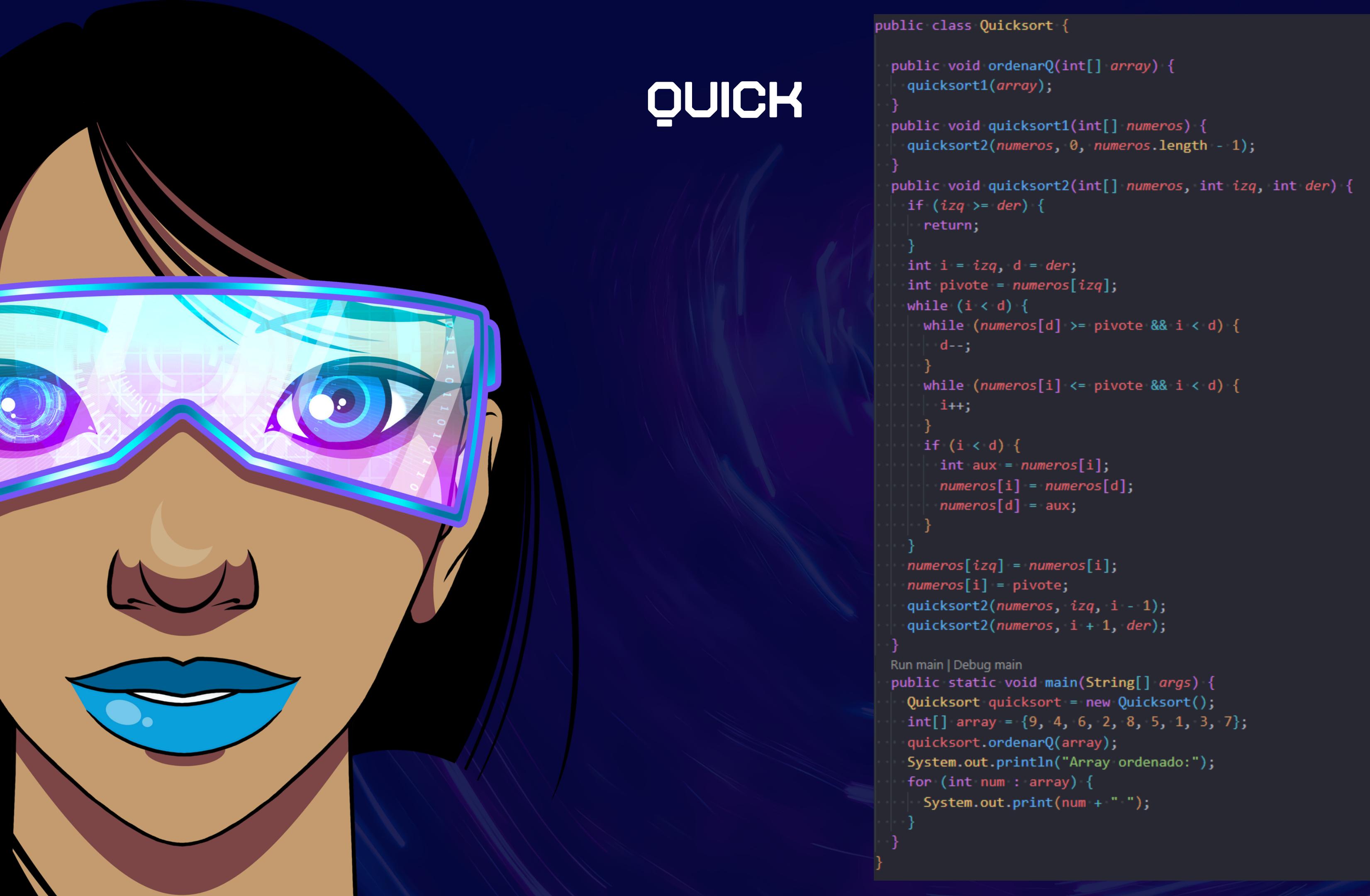
```
public void mergeSort(int[] numeros, int izq, int der) {  
    if (izq < der) {  
        int medio = (izq + der) / 2;  
        mergeSort(numeros, izq, medio);  
        mergeSort(numeros, medio + 1, der);  
        merge(numeros, izq, medio, der);  
    }  
}
```

MERGE

08

```
public void merge(int[] numeros, int izq, int medio, int der) {  
    int n1 = medio - izq + 1;  
    int n2 = der - medio;  
  
    int[] izqArray = new int[n1];  
    int[] derArray = new int[n2];  
  
    for (int i = 0; i < n1; i++) {  
        izqArray[i] = numeros[izq + i];  
    }  
    for (int j = 0; j < n2; j++) {  
        derArray[j] = numeros[medio + 1 + j];  
    }  
  
    int i = 0, j = 0;  
    int k = izq;  
  
    while (i < n1 && j < n2) {  
        if (izqArray[i] <= derArray[j]) {  
            numeros[k] = izqArray[i];  
            i++;  
        } else {  
            numeros[k] = derArray[j];  
            j++;  
        }  
        k++;  
    }  
  
    while (i < n1) {  
        numeros[k] = izqArray[i];  
        i++;  
        k++;  
    }  
  
    while (j < n2) {  
        numeros[k] = derArray[j];  
        j++;  
        k++;  
    }  
}
```

```
public static void main(String[] args) {  
    MergeSort mergeSort = new MergeSort();  
    int[] array = {9, 4, 6, 2, 8, 5, 1, 3, 7};  
  
    mergeSort.ordenarM(array);  
  
    System.out.println("Array ordenado:");  
    for (int num : array) {  
        System.out.print(num + " ");  
    }  
}
```



QUICK

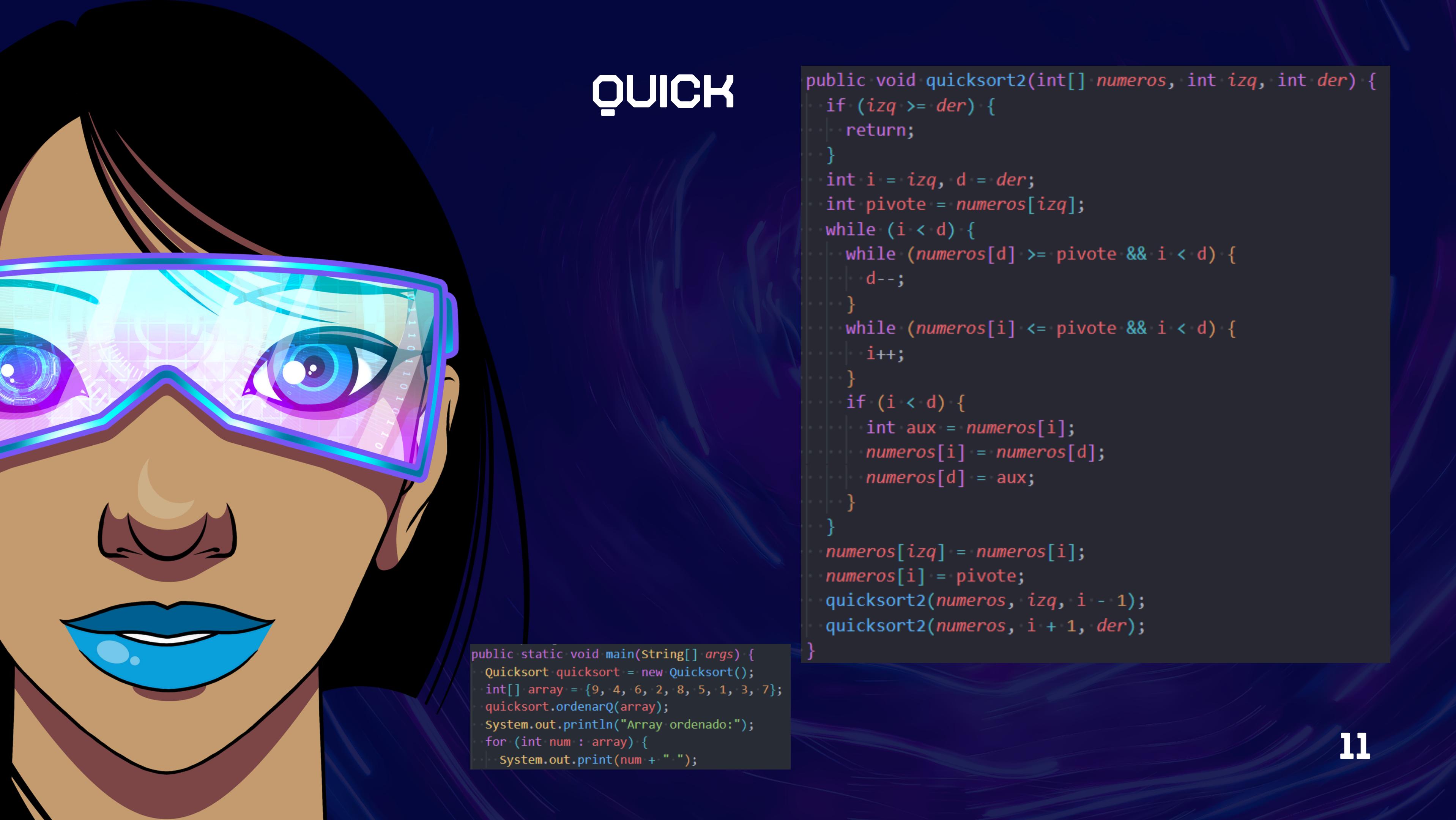
```
public class Quicksort {  
  
    public void ordenarQ(int[] array) {  
        quicksort1(array);  
    }  
    public void quicksort1(int[] numeros) {  
        quicksort2(numeros, 0, numeros.length - 1);  
    }  
    public void quicksort2(int[] numeros, int izq, int der) {  
        if (izq >= der) {  
            return;  
        }  
        int i = izq, d = der;  
        int pivot = numeros[izq];  
        while (i < d) {  
            while (numeros[d] >= pivot && i < d) {  
                d--;  
            }  
            while (numeros[i] <= pivot && i < d) {  
                i++;  
            }  
            if (i < d) {  
                int aux = numeros[i];  
                numeros[i] = numeros[d];  
                numeros[d] = aux;  
            }  
        }  
        numeros[izq] = numeros[i];  
        numeros[i] = pivot;  
        quicksort2(numeros, izq, i - 1);  
        quicksort2(numeros, i + 1, der);  
    }  
}  
Run main | Debug main  
public static void main(String[] args) {  
    Quicksort quicksort = new Quicksort();  
    int[] array = {9, 4, 6, 2, 8, 5, 1, 3, 7};  
    quicksort.ordenarQ(array);  
    System.out.println("Array ordenado:");  
    for (int num : array) {  
        System.out.print(num + " ");  
    }  
}
```

QUICK

```
public void ordenarQ(int[] array) {  
    quicksort1(array);  
}
```

```
public void quicksort1(int[] numeros) {  
    quicksort2(numeros, 0, numeros.length - 1);  
}
```

QUICK



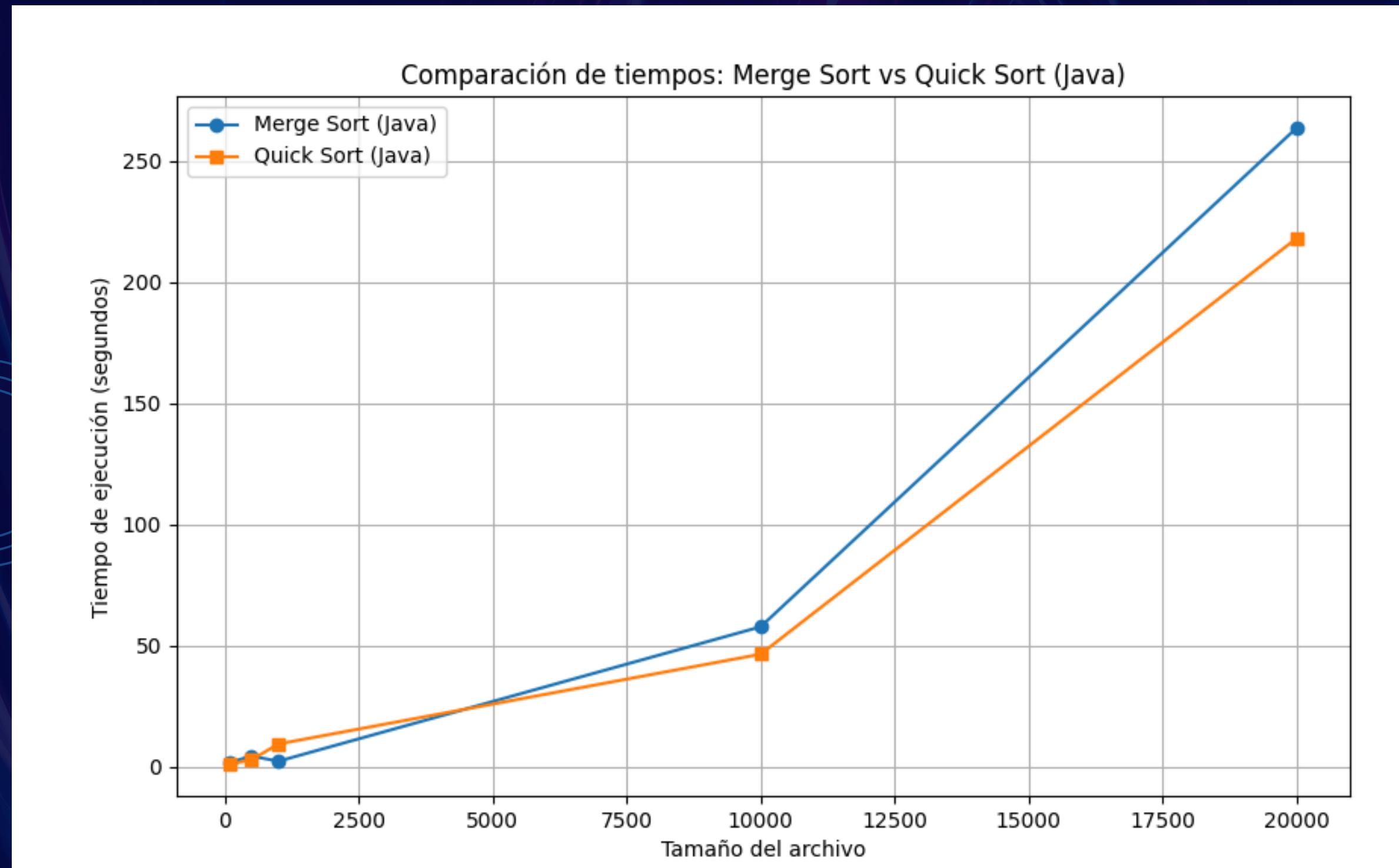
```
public static void main(String[] args) {  
    Quicksort quicksort = new Quicksort();  
    int[] array = {9, 4, 6, 2, 8, 5, 1, 3, 7};  
    quicksort.ordenarQ(array);  
    System.out.println("Array ordenado:");  
    for (int num : array) {  
        System.out.print(num + " ");  
    }  
}
```

```
public void quicksort2(int[] numeros, int izq, int der) {  
    if (izq >= der) {  
        return;  
    }  
    int i = izq, d = der;  
    int pivot = numeros[izq];  
    while (i < d) {  
        while (numeros[d] >= pivot && i < d) {  
            d--;  
        }  
        while (numeros[i] <= pivot && i < d) {  
            i++;  
        }  
        if (i < d) {  
            int aux = numeros[i];  
            numeros[i] = numeros[d];  
            numeros[d] = aux;  
        }  
    }  
    numeros[izq] = numeros[i];  
    numeros[i] = pivot;  
    quicksort2(numeros, izq, i - 1);  
    quicksort2(numeros, i + 1, der);  
}
```

COMPARACIONES

	Tamaño del archivo	Tiempo Merge Sort (s)	Tiempo Quick Sort (s)
1	100	1.696271131093611	1.0007141774596835
2	500	4.321845898880389	2.787508810377182
3	1000	2.1615275099325917	9.284704696995815
4	10000	57.67228993479342	46.356770251134385
5	20000	263.5509199325235	218.00025732221303

COMPARACIONES



VENTADAS

MERGE

VS

QUICK

- Estable: conserva el orden relativo.
- Complejidad garantizada: $O(n\log n)$ en cualquier caso.
- Manejo eficiente de grandes datos.
- Ideal para listas enlazadas.
- Comportamiento predecible y determinista.

- Promedio rápido: $O(n\log n)$ en la mayoría de los casos.
- Uso eficiente de memoria: es in-place.
- Fácil de implementar.
- Excelente rendimiento en sistemas modernos.
- Adaptable mediante estrategias de pivote.



DESVENTADAS

MERGE

V S

QUICK

- Mayor consumo de memoria: requiere espacio adicional.
- Más lento en promedio que Quicksort.
- Complejo de implementar in-place.
- Menor eficiencia en caché.
- No es el mejor para listas pequeñas.

- Peor caso lento: $O(n^2)$ si el pivote se elige mal.
- No es estable por defecto.
- Sensible a datos parcialmente ordenados.
- Puede consumir mucha pila en recursión.
- Difícil de parallelizar.



CONCLUSION

Quicksort es ideal para conjuntos de datos grandes y cuando se busca alta velocidad promedio con uso eficiente de memoria, especialmente en sistemas modernos. Sin embargo, puede ser ineficiente en su peor caso y no es estable por defecto.

Por otro lado, Merge Sort es más confiable para situaciones que requieren estabilidad, como datos sensibles al orden, y es adecuado para listas vinculadas o datos muy grandes. Su desventaja radica en su mayor consumo de memoria y menor eficiencia promedio comparado con Quicksort. La elección entre ambos depende del contexto y las necesidades específicas del problema.

GRACIAS