13/10/23

[346, 72, 13, 187, 29, 1250, 950]

Merge Sort

346, 72, 13, 187, 29, 1250, 950

346, 72, 13, 187 | 29, 1250, 950

346, 72 | 13, 187 | 29, 1250 | 950

346 | 72 | 13 | 187 | 29 | 1250 | 950

72 346 | 13 187 | 29 1250 | 950

13 72 187 346 | 29 950 1250

13, 29, 72, 187, 346, 950, 1250

Radix sort

0346	1250	0013	0013
0072	0950	0029	0029
0013	0072	0346	0072
0187	0013	1250	0187
0029	0346	0950	1250
1250	0187	0072	0346
0950	0029	0187	0950

0	0	1	3
0	0	2	9
0	0	7	2
0	1	8	7
0	3	4	6
0	9	5	0
1	2	5	0

Bubble Sort [346, 72, 13, 187, 29, 1250, 950]

Iteración 1 posición 0

346 72 13 187 29 1250 950
72 346 13 187 29 1250 950
13 346 72 187 29 1250 950 Sin cambios
13 346 72 187 29 1250 950 Sin cambios
13 346 72 187 29 1250 950 Sin cambios
13 346 72 187 29 1250 950 Sin cambios

Iteración 2 posición 1

13 346 72 187 29 1250 950
13 72 346 187 29 1250 950 = Sin cambios
13 72 346 187 29 1250 950
13 29 346 187 72 1250 950 = Sin cambios
13 29 346 187 72 1250 950 = Sin cambios

Iteración 3 pos 2

13 29 346 187 1250 950
13 29 187 346 1250 950
13 29 72 346 187 1250 950 = sin cambios
13 29 72 346 187 1250 950 = sin cambios

Iteración 4 pos 3

13 29 72 346 187 1250 950
13 29 72 187 346 1250 950 sin cambios
13 29 72 187 346 1250 950 sin cambios

Iteración 5 pos 4

13 29 72 187 346 1250 950 = sin cambios
13 29 72 187 346 1250 950 = sin cambios

Iteración 6 pos 5

13 29 72 187 346 1250 950
13 29 72 187 346 950 1250 ✓

Quick Sort arreglo [346, 72, 13, 187, 29, 1250, 950]

Pivote = 187
primero = 0
ultimo = 6
i = 0
j = 6

i <= Pivote
346 <= 187 x
i = 0

j > Pivote
950 > 187 ✓
1250 > 187 ✓
29 > 187 x
j = 4

i ≤ j
0 ≤ 4 ✓ swap

i j
[29, 72, 13, 187, 346, 1250, 950]

pivote = 187
primero = 0
ultimo = 6
i = 1
j = 3

i < Pivote
72 < 187 ✓
13 < 187 ✓
187 < 187 x
i = 3

j > Pivote
187 > 187 x
j = 3

i ≤ j
3 ≤ 3 ✓ swap

j
[29 72 13 187 346 1250 950]

pivote = 187
primero = 0
ultimo = 6
i = 4
j = 2

i < Pivote
346 < 187 x
i = 4

j > Pivote
13 > 187 x
j = 2

i ≤ j
4 ≤ 2

0 < primero < j 0 < 2 ✓
i < ultimo 4 < 6 ✓

29, 72, 13, 187, 346, 1250, 950

primero = 0
ultimo = 2
i = 0
j = 2
pivote = 72

i < Pivote
29 < 72 ✓
72 < 72 ✗
i = 1

j > Pivote
13 > 72 ✗
j = 2

i ≤ j
1 ≤ 2 ✓ swap

29, 13, 72, 187, 346, 1250, 950

i = 2
j = 1

i < pivote
72 < 72 ✗

j > pivote
3 > 72 ✗

primero < j
ultimo
0 < 1 ✓
2 < 2 ✗

i ≤ j
2 ≤ 1 ✗

29, 13, 72, 187, 346, 1250, 950

primero = 0
ultimo = 1
i = 0
j = 1
pivote = 29

i < Pivote
29 < 29 ✗
i = 0

j > Pivote
13 > 29 ✗
j > 1

i ≤ j
0 ≤ 1 ✓ Swap

13 29 72 187 346 1250 950

i = 1
j = 0

i < Pivote
29 < 29 ✗

j > pivote
13 > 29 ✗

i ≤ j
1 ≤ 0 ✗

primero < j
1 < ultimo
0 < 0 ✗
1 < 1 ✗

[13, 29, 72, 187, 346, 1250, 750]

primero = 4

ultimo = 7

i = 4

j = 6

pivote = 1250

i <= pivote

346 < 1250

1250 < 1250

i = 5

i <= j

5 <= 6 V swap

[13, 29, 72, 187, 346, 950, 1250]

i = 6

j = 4

i <= Pivote

1250 < 1250 X

i = 6

j > Pivote

346 > 1250 X

j = 4

Primero < j 4 < 5 V

i < ultimo 6 < 6 X

13, 29, 72, 187, 346, 950, 1250

primero = 4

ultimo = 5

i = 4

j = 5

pivote = 346

i < Pivote

346 < 346 X

i = 4

j > Pivote

950 > 346 V

346 > 346 X

j = 4

i <= j

4 <= 4 V swap

13, 29, 72, 187, 346, 950, 1250

i = 5

j = 3

i < Pivote

950 < 346 X

i = 5

j > Pivote

187 > 346 X

j = 3

i <= j

5 <= 3 X

primero < j 4 < 3 X

i < ultimo 5 < 5 X

Dificultades

- Radixsort:
  * Es más eficiente si trabaja con números enteros ya que divide por partes a los números y si fuera otro tipo de dato le es más difícil trabajar
  * No es tan eficiente para conjuntos pequeños

- Mergesort:
  * Usa una gran cantidad de memoria para guardar la información de la particiones que hace del arreglo
  * Si es muy largo puede tener mucho tiempo de ejecución

- Bubble sort:
  * Su eficiencia si se trata del tamaño del arreglo

- Quick sort:
  * La elección del pivote (aunque puede ser utilizando alguna herramienta que lo aleatoricce)
  * Es complejo de implementar
  * Para datos pequeños puede ser ineficiente

Gil Ortega Luis Alberto 20110/2023

La variable orden se usa para darle seguimiento a la cantidad de elementos en el array. Los fors uno se utiliza para contar cuantos elementos en el arreglo arr tienen el mismo dígito en la posición exponente y el otro es para calcular las posiciones finales de los elementos en el arreglo salida

función de método merge

Su función es combinar dos subarreglos ordenados en un solo arreglo ordenado. primero copia a los elementos del arreglo arr a un arreglo temporal. de ahí entra en juego 3 indices i, j y k que sirven para rastrear la posición actual en el arreglo original, luego en cada iteración, compara el elemento guardado en temp(i) con el elemento en temp(j), entonces si temp(i) es ≤ a temp(j) copia temp[i] al arreglo original en la posición k, aumenta i y k, después uno de los subarreglos puede que ya no tenga elementos pero el otro si. asi que el método merge los une.