Algoritmo Merge Sort

Maestría en Ciencia de Datos

Edgar Ruiz Tovar
Rodrigo Alvarado Sánchez
Pablo Francisco Cantú Villanueva

Universidad Autónoma de Querétato

5 de marzo de 2025

La persona detrás del algoritmo

- John Von Neumann.

La persona detrás del algoritmo

- John Von Neumann.
- Donald Knuth lo considera el creador del algoritmo **merge sort** (1945).

La persona detrás del algoritmo

- John Von Neumann.
- Donald Knuth lo considera el creador del algoritmo **merge sort** (1945).
- Contribuciones: método Montecarlo, números pseudoaleatorios (cuadrados medios), autómatas celulares, proyecto Manhattan.

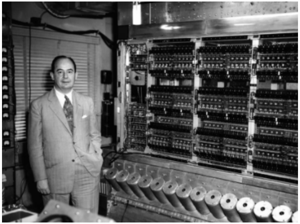

Figura: John Von Neumann (1903-1957)

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)
- Se compone de:

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)
- Se compone de:
    - Dividir: el problema general en sub-problemas.

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)
- Se compone de:
  - Dividir: el problema general en sub-problemas.
  - Vencer: resolver los sub-problemas de forma recursiva.

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)
- Se compone de:
  - Dividir: el problema general en sub-problemas.
  - Vencer: resolver los sub-problemas de forma recursiva.
  - Combinar: las soluciones.

Divide et impera

- Algoritmos *Divide and conquer* (divide y vencerás)
- Se compone de:
  - Dividir: el problema general en sub-problemas.
  - Vencer: resolver los sub-problemas de forma recursiva.
  - Combinar: las soluciones.
- Batalla de Gaugamela: Macedonia vs Persia.

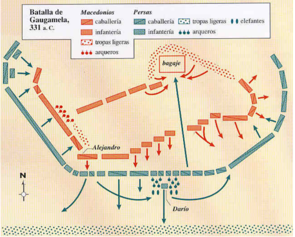

Figura: Disposición de tropas: batalla de Gaugamela.

Merge sort: idea

- Siguiendo la estrategia de dividir y vencer:

Merge sort: idea

- Siguiendo la estrategia de dividir y vencer:
- Separar el arreglo en mitades.

Merge sort: idea

- Siguiendo la estrategia de dividir y vencer:
- Separar el arreglo en mitades.
- Ordenar de manera recursiva cada mitad

Merge sort: idea

- Siguiendo la estrategia de dividir y vencer:
- Separar el arreglo en mitades.
- Ordenar de manera recursiva cada mitad
- Combinar las mitades ordenadas.

Merge sort: idea

- Siguiendo la estrategia de dividir y vencer:
- Separar el arreglo en mitades.
- Ordenar de manera recursiva cada mitad
- Combinar las mitades ordenadas.

-
```
[45, 41, 15, 29, 90, 92, 7, 57, 17, 16]
[45, 41, 15, 29, 90]   [92, 7, 57, 17, 16]
[45, 41, 15]   [29, 90]
[45, 41]   [15]
[45]   [41]
[41, 45]
[15, 41, 45]
[29]   [90]
[29, 90]
[15, 29, 41, 45, 90]
[92, 7, 57]   [17, 16]
[92, 7]   [57]
[92]   [7]
[7, 92]
[7, 57, 92]
[17]   [16]
[16, 17]
[7, 16, 17, 57, 92]
[7, 15, 16, 17, 29, 41, 45, 57, 90, 92]
```

Pseudocódigo

- Función merge:

```
MergeSort(lista)

    Si la lista tiene 1 elemento o está vacía
        retornar lista

    mitad1 = primera mitad de la lista
    mitad2 = segunda mitad de la lista

    mitad1 = MergeSort(mitad1)
    mitad2 = MergeSort(mitad2)

    listaOrdenada = UnirListas(mitad1, mitad2)
    retornar listaOrdenada
```

Pseudocódigo

- Función merge:

```
MergeSort(lista)

    Si la lista tiene 1 elemento o está vacía
        retornar lista

    mitad1 = primera mitad de la lista
    mitad2 = segunda mitad de la lista

    mitad1 = MergeSort(mitad1)
    mitad2 = MergeSort(mitad2)

    listaOrdenada = UnirListas(mitad1, mitad2)
    retornar listaOrdenada
```

- Función para unir:

```
UnirListas(lista1, lista2)
    resultado = lista vacía
    i = 0
    j = 0

    mientras i sea menor al tamaño de lista1 y j sea menor a tamaño de lista2
        si lista1[i] es menor o igual que lista2[j]
            agregar lista1[i] a resultado
            incrementar i
        si no
            agregar lista2[j] a resultado
            incrementar j
    mientras i sea menor al tamaño de lista1
        agregar lista1[i]  a resultado
        incrementar i

    mientras j sea menor al tamaño de lista2
        agregar lista2[j]  a resultado
        incrementar j
    retornar resultado
```

Implementación del código

- A continuación se presenta el código implementado en el lenguaje C#.

Complejidad computacional

- **Dividir**: en mitades $O(1)$

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva de la función *merge_sort* trabaja con uno de $\frac{k}{2}$.

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva de la función *merge_sort* trabaja con uno de $\frac{k}{2}$.
- **Combinar**: Cada combinación de arreglos tiene complejidad $O(n)$.

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva de la función *merge_sort* trabaja con uno de $\frac{k}{2}$.
- **Combinar**: Cada combinación de arreglos tiene complejidad $O(n)$.
- Para cada nivel n de recursión se tienen que combinar n elmeentos.

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva de la función *merge_sort* trabaja con uno de $\frac{k}{2}$.
- **Combinar**: Cada combinación de arreglos tiene complejidad $O(n)$.
- Para cada nivel n de recursión se tienen que combinar n elmeentos.
- Luego, la expresión viene dada por: $T(n) = 2T(\frac{n}{2}) + O(n)$

Complejidad computacional

- **Dividir**: en mitades $O(1)$
- **Vencer**: ordenar de manera recursiva cada mitad.
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva queda como $\frac{k}{2}$
- Si el tamaño del arreglo es k, entonces con cada llamada recursiva de la función *merge_sort* trabaja con uno de $\frac{k}{2}$.
- **Combinar**: Cada combinación de arreglos tiene complejidad $O(n)$.
- Para cada nivel n de recursión se tienen que combinar n elmeentos.
- Luego, la expresión viene dada por: $T(n) = 2T(\frac{n}{2}) + O(n)$
- Usando el teorema maestro $O(n \log n)$