CLASE 07 | ALGORTIMOS I

POR ALEJO BENGOECHEA

ALGORITMOS I

▼ **ALGORITMO**

Un ***algoritmo*** es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. O sea, una serie de pasos a seguir para completar una tarea.

Los ***algoritmos*** deben tener una serie de características:

Resuelve un problema: este es el objetivo principal del algoritmo, fue diseñado para eso. Si no cumple el objetivo, no sirve para nada.

Debe ser comprensible: el mejor algoritmo del mundo no te va a servir si es demasiado complicado de implementar.

Hacerlo eficientemente: no sólo queremos tener la respuesta perfecta (o la más cercana), si no que también queremos que lo haga usando la menor cantidad de recursos posibles.

▼ **¿CÓMO MEDIMOS LA EFICIENCIA?**

Tiempo: no depende solamente de que el algoritmo sea eficiente. También depende de la computadora que usemos, el procesador, la consola, etc.

Espacio: espacio de memoria que requiere.

Otros Recursos: red, gráficos, hardware, etc.

▼ **COMPLEJIDAD**

¿Por qué es importante medir la complejidad de un algoritmo?

Primero, porque nos permite ***predecir su comportamiento***. Si estamos en frente de un algoritmo muy complejo, puede que ocupe todo el procesador de nuestra PC y esta se rompa.

Segundo, porque nos permite ***compararlo***. Si podemos saber objetivamente qué tan complejo es, cuando lo comparemos con otros podremos ver cuál es

más eficiente.

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
                       ^^^

E1 O(N)        E2 LOG(N)
1              5 //es + o - que 5? -----------  +
2              8 //es + o - que 8? -----------  +
3              9 :D
4
5
6
7
8
9:D
```

Este es un ejemplo de dos algoritmos (uno eficiente y otro ineficiente) que buscan un número entre muchos.

```
var array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

function sumArray(array, n) {
  for(let i = 0; i < array.length; i++) {
    for(let j = 0; j < array.length; j++) {
      if(array[i] + array[j] === n) {return true;}    // 100 operaciones
    }
  }
console.log("No existe");                             // 1 operación
return false;                                         // 1 operación
}
```

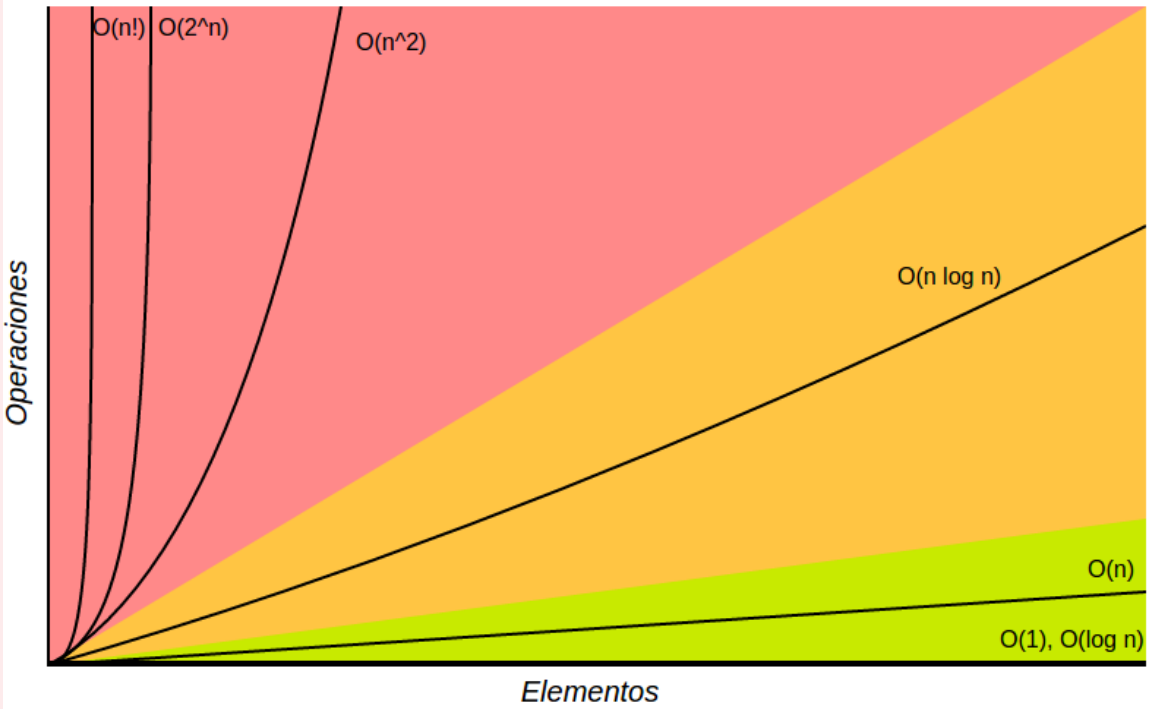
En este ejemplo vemos que se hace un *ciclo for* anidado a partir de un arreglo de 10 elementos. Si vamos al peor de los casos (en el que ninguna suma de dos elementos sea igual a **n**), el algoritmo tendrá que hacer 100 operaciones. Y, además, tendrá que hacer dos más, por las otras ejecuciones.

Cuando medimos la eficiencia tenemos que tener en cuenta siempre el peor de los casos (en el que el algoritmo haga la mayor cantidad de operaciones. En este caso, el algoritmo haría **102 operaciones**, es decir, **O(n^2) + 2**. Pero cuando hacemos mediciones de eficiencia, no tendremos en cuenta los valores constantes, ya que a la hora del resultado, **100 operaciones** o **102** no será una diferencia significativa. Por lo que quedaría **O(n^2)**.

```
function(arr) {
  console.log("Hola");   // 1 operación
  console.log("Chau");   // 1 operación
```

```
  let acu = 2 + 3;       // 1 operación
  return array[0];       // 1 operación
}
```

En este caso, indistintamente de la longitud del arreglo que pasemos por parámetro, este algoritmo siempre hará 4 operaciones. Ni más ni menos. A esto se lo conoce como **O(1)**, que es el mejor de los casos en el rendimiento de un algoritmo.



▼ ALGORITMOS DE ORDENAMIENTO I

▼ BUBBLE SORT

Este algoritmo se encarga de ordenar los elementos de una lista.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a078980c-ae23-4c4b-9e5c-b1eb74ebe82b/1.mp4

▼ INSERTION SORT

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1591d161-b75c-42ff-a893-610b88ef6335/2.mp4

▼ **SELECTION SORT**

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/de556d5d-93b9-4417-a9c1-e943e9081d7e/3.mp4