

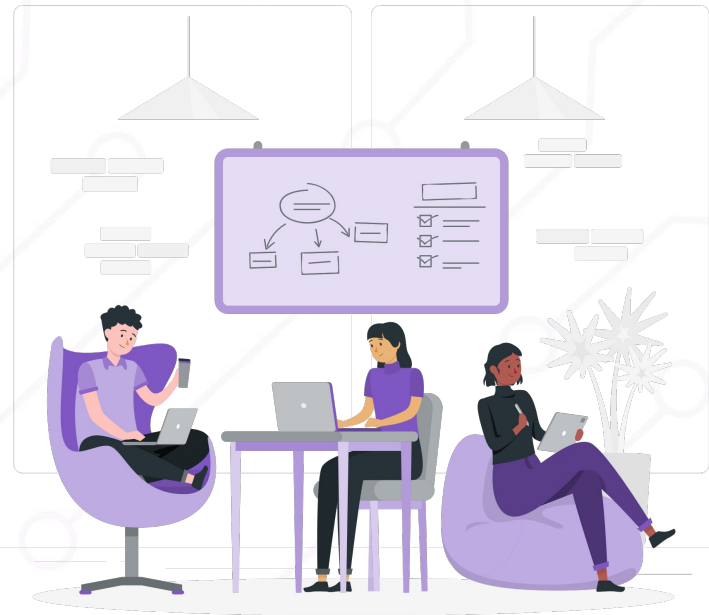
ALGORITMOS DE ORDENAMIENTO

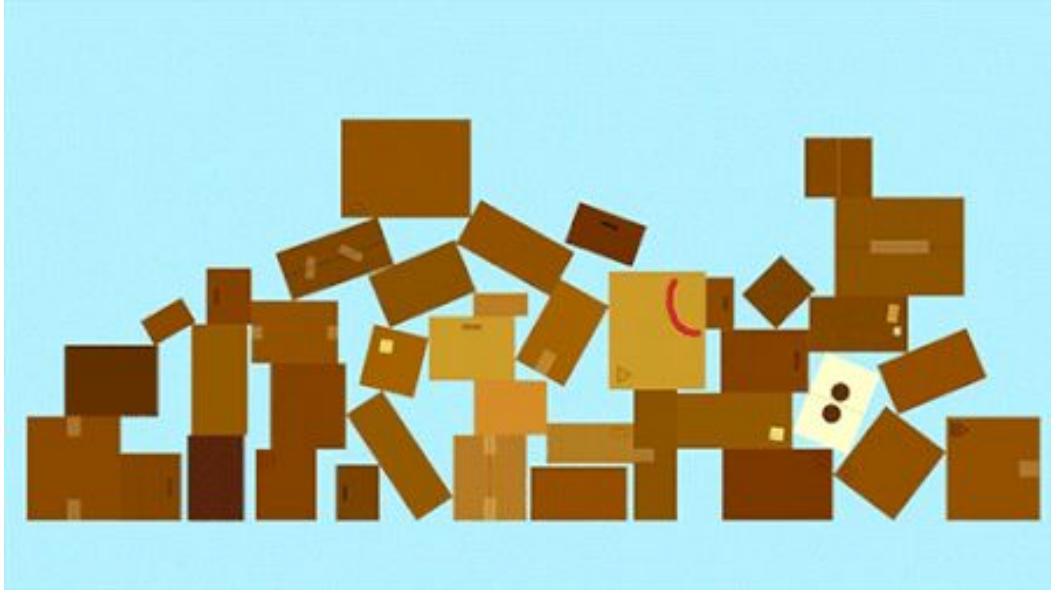
DEV.FX
DESARROLLAMOS(PERSONAS);

dev

OBJETIVOS DE LA SESIÓN

1. EL ALCANCE DE LA SESIÓN DE HOY SERÁ EL TEMA ALGORITMOS DE ORDENAMIENTO EN JAVASCRIPT
2. ENTENDER QUE SON LOS ALGORITMOS DE ORDENAMIENTO Y CÓMO FUNCIONAN.
3. VEREMOS SU FUNCIONAMIENTO EN CÓDIGO PARA ENTENDER MEJOR EL TEMA.
4. SEGUIREMOS CON LA DINÁMICA DE LOS EJERCICIOS PARA SEGUIR PRACTICANDO Y REFORZANDO EL TEMA VISTO DURANTE LA SESIÓN.





Algoritmos de ordenamiento

Un algoritmo de ordenamiento, es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, **el resultado de salida es un reordenamiento de la entrada de datos.**

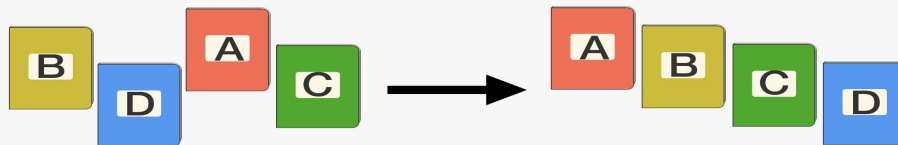
8 5 3 1 4 7 9

¿Qué podemos ordenar?

Cualquier estructura de datos con elementos que sean ordenables

- Podemos **ordenar números**, porque unos son mayores que otros.
- Podemos **ordenar meses**, porque unos vienen antes que otros
- Podemos **ordenar palabras**, por el orden en el alfabeto.

Sorting Algorithms



Algoritmos de ordenación

Algunos ejemplos:

- **Ordenamiento por inserción**
- **Ordenamiento de burbuja**
- **Ordenamiento por selección**

Receta algoritmos de ordenación

En la cocina del código primero:

1. Recorremos el array buscando el elemento mínimo.
2. Intercambiamos ese elemento con el que esté en la primera posición.
3. Buscamos el siguiente mínimo que no hayamos ordenado.
4. Lo cambiamos por el segundo.
5. Repetir hasta terminar.



Código para ordenar

¿Qué necesitamos conocer?

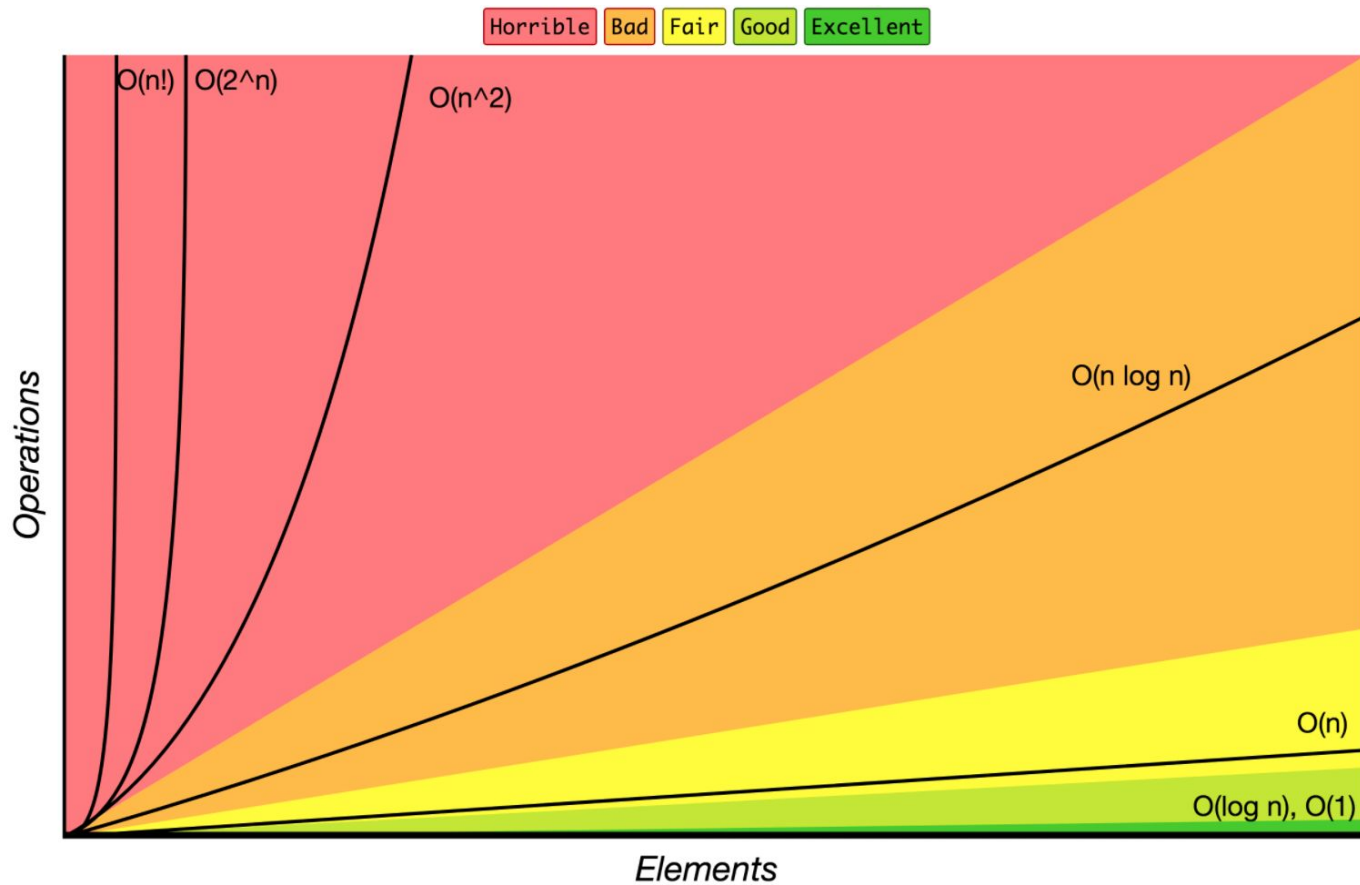
Cuál es la posición en la que va a ir el elemento ordenado: 0,1,2,3 ... ,N

Cuál es el elemento a partir del cual vamos a buscar el mínimo.

El mínimo en cada repetición.

Algunos algoritmos de ordenamiento agrupados según estabilidad tomando en cuenta la [complejidad computacional](#).

Estables				
Nombre traducido	Nombre original	Complejidad	Memoria	Método
Ordenamiento de burbuja	Bubblesort	$O(n^2)$	$O(1)$	Intercambio
Ordenamiento de burbuja bidireccional	Cocktail sort	$O(n^2)$	$O(1)$	Intercambio
Ordenamiento por inserción	Insertion sort	$O(n^2)$ ("en el peor de los casos")	$O(1)$	Inserción
Ordenamiento por casilleros	Bucket sort	$O(n)$	$O(n)$	No comparativo
Ordenamiento por cuentas	Counting sort	$O(n+k)$	$O(n+k)$	No comparativo
Ordenamiento por mezcla	Merge sort	$O(n \log n)$	$O(n)$	Mezcla
Ordenamiento con árbol binario	Binary tree sort	$O(n \log n)$	$O(n)$	Inserción



(peor caso, caso promedio y mejor caso)

Los órdenes usados con más frecuencia son los **órdenes numéricos** y **órdenes lexicográficos**.

0000	abcd	—	abcd	⌋	abc	⌋	ab	⌋	a	⌋	∅
0001	abc		abc	⌋	abcd	⌋	abc	⌋	ab	⌋	a
0010	abd		abd	—	abd	⌋	abcd	⌋	abc	⌋	ab
0011	ab		ab	⌋	ab	⌋	abd	⌋	abcd	⌋	abc
0100	acd		acd	—	acd	⌋	ac	⌋	abd	⌋	abcd
0101	ac		ac	⌋	ac	⌋	acd	⌋	ac	⌋	abd
0110	ad		ad	—	ad	⌋	ad	⌋	acd	⌋	ac
0111	a		a	⌋	a	⌋	a	⌋	ad	⌋	acd
1000	bcd		bcd	⌋	bcd	⌋	bc	⌋	b	⌋	ad
1001	bc		bc	⌋	bc	⌋	bcd	⌋	bc	⌋	b
1010	bd		bd	—	bd	⌋	bd	⌋	bcd	⌋	bc
1011	b		b	⌋	b	⌋	b	⌋	bd	⌋	bcd
1100	cd		cd	—	cd	⌋	cd	⌋	c	⌋	bd
1101	c		c	⌋	c	⌋	c	⌋	cd	⌋	c
1110	d		d	⌋	d	—	d	⌋	d	⌋	cd
1111	∅		∅	⌋	∅	⌋	∅	⌋	∅	⌋	d

Ordenamiento de la burbuja (bubble sort)

Funciona **revisando cada elemento de la lista que va a ser ordenada con el siguiente**, intercambiándolos de posición si están en el orden equivocado.

Es necesario **revisar varias veces toda la lista hasta que no se necesiten más intercambios**, lo cual significa que la lista está ordenada

6 5 3 1 8 7 2 4

Ordenamiento por inserción

Es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria.

6 5 3 1 8 7 2 4

Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay k elementos ordenados de menor a mayor, se toma el elemento $k+1$ y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor o cuando ya no se encuentran elementos

Insertion Sort

Iteration 0 (unsorted array): [5,3,1,4,6]

Iteration 1, key is 3 (was at index 1): [5,3,1,4,6] → [3,5,1,4,6]

Iteration 2, key is 1 (was at index 2): [3,5,1,4,6] → [1,3,5,4,6]

Iteration 3, key is 4 (was at index 3,): [1,3,5,4,6] → [1,3,4,5,6]

HORA DEL CÓDIGO

