Paradigmas de Programación (JS en Node.js) **Práctica II – Básica (Revisar porque debería armar los enunciados)**

splice es un método de arrays de JavaScript que **modifica el array original** para **eliminar, insertar o reemplazar** elementos.

## Sintaxis

array.splice(start, deleteCount, ...itemsAInsertar)

* **start**: índice desde donde operar (admite negativos: -1 = último).
* **deleteCount**: cuántos elementos eliminar desde start. Si es 0, no borra.
* **...itemsAInsertar**: (opcional) elementos que se insertan desde start.

**Devuelve:** un **array con los elementos eliminados** (puede ser []).

Importante: **muta** el array original (a diferencia de slice, que no muta).

## Ejemplos rápidos

Partimos de:

const lista = [4, 1, 2, 4, 5, 8, 7, 6, 9, 10];

### 1) Eliminar

const copia1 = lista.slice(); // clon NO mutable con slice

const borrados = copia1.splice(3, 2); // desde índice 3, elimina 2 (el 4 y el 5)

console.log(copia1); // [4,1,2,8,7,6,9,10]

console.log(borrados); // [4,5]

### 2) Insertar (sin borrar)

const copia2 = lista.slice();

copia2.splice(2, 0, 99, 100); // en índice 2, borra 0 e inserta 99 y 100

console.log(copia2); // [4,1,99,100,2,4,5,8,7,6,9,10]

### 3) Reemplazar (borrar y poner otros)

const copia3 = lista.slice();

copia3.splice(5, 2, 777); // quita 2 (8,7) y pone 777 en su lugar

console.log(copia3); // [4,1,2,4,5,777,6,9,10]

### 4) Índices negativos

const copia4 = lista.slice();

copia4.splice(-3, 1); // -3 apunta al 6 (antepenúltimo). Elimina 1.

console.log(copia4); // [4,1,2,4,5,8,7,9,10]

### 5) Truncar o vaciar

const copia5 = lista.slice();

copia5.splice(4); // desde índice 4, elimina TODO lo que sigue

console.log(copia5); // [4,1,2,4]

const copia6 = lista.slice();

copia6.splice(0, copia6.length); // vacía el array

console.log(copia6); // []

## Patrón útil: eliminar mientras iterás

Si vas a borrar varias posiciones, **recorré de derecha a izquierda** o ajustá el índice:

const arr = [1,2,3,2,4,2];

for (let i = arr.length - 1; i >= 0; i--) {

if (arr[i] === 2) arr.splice(i, 1);

}

console.log(arr); // [1,3,4]

## splice vs slice

* **splice**: muta el array, sirve para borrar/insertar/reemplazar. Devuelve lo borrado.
* **slice**: **no** muta; copia una porción (slice(inicio, finExcl)) y devuelve un nuevo array.

## Alternativa inmutable para “eliminar”

Si **no** querés mutar, usá filter:

const sinMayoresA6 = lista.filter(n => n <= 6);

// lista queda igual; sinMayoresA6 es un nuevo array

¡De una! Acá tenés una **tablita de recetas** súper práctica para splice con ejemplos claros.  
Partimos de:

const lista = [4, 1, 2, 4, 5, 8, 7, 6, 9, 10];

# Recetas con splice

## Insertar

**Al inicio**

const a = lista.slice();

a.splice(0, 0, 99); // inserta 99 en índice 0

// [99, 4, 1, 2, 4, 5, 8, 7, 6, 9, 10]

**Al medio**

const b = lista.slice();

const mid = Math.floor(b.length / 2);

b.splice(mid, 0, 77, 88); // inserta 77 y 88 en el medio

**Al final**

const c = lista.slice();

c.splice(c.length, 0, 42); // inserta 42 al final

**Después de un valor X (primera aparición)**

const d = lista.slice();

const i = d.indexOf(2); // localizar 2

if (i !== -1) d.splice(i + 1, 0, 3); // insertar 3 después del 2

## Eliminar

**Por índice**

const e = lista.slice();

e.splice(3, 1); // elimina 1 elemento en índice 3

**Un rango**

const f = lista.slice();

f.splice(2, 4); // elimina 4 elementos desde índice 2

**Desde un índice hasta el final**

const g = lista.slice();

g.splice(5); // elimina todo desde índice 5

**Todos los que cumplan una condición (in-place)**

Recorriendo de derecha a izquierda para no saltarte elementos.

const h = lista.slice();

// eliminar todos los > 6

for (let j = h.length - 1; j >= 0; j--) {

if (h[j] > 6) h.splice(j, 1);

}

**Eliminar la “primera aparición” de un valor**

const i2 = lista.slice();

const j = i2.indexOf(4);

if (j !== -1) i2.splice(j, 1);

## Reemplazar

**Reemplazar 1 elemento**

const k = lista.slice();

k.splice(5, 1, 777); // quita 1 en índice 5 y pone 777

**Reemplazar un rango por varios elementos**

const l = lista.slice();

l.splice(2, 3, 100, 101, 102); // quita 3 desde 2 y pone 3 nuevos

## Mover elementos (in-place)

**Mover un elemento de from a to**

const m = lista.slice();

const from = 1; // valor 1

const to = 4;

const [moved] = m.splice(from, 1);

m.splice(to, 0, moved);

// ahora el 1 quedó en índice 4

**Mover un bloque (rango)**

const n = lista.slice();

const [a1, a2] = n.splice(2, 2); // saca [2,4]

n.splice(7, 0, a1, a2); // inserta ese bloque en índice 7

## Índices negativos

**Contar desde el final**

const o = lista.slice();

o.splice(-3, 1); // elimina el antepenúltimo

## Truncar y vaciar

**Truncar a N elementos**

const p = lista.slice();

p.splice(4); // deja [4,1,2,4]

**Vaciar array**

const q = lista.slice();

q.splice(0, q.length); // []

## Recuperar lo eliminado

**splice devuelve lo borrado**

const r = lista.slice();

const borrados = r.splice(3, 2); // quita 2 elementos

// borrados -> [4, 5]

**Este ejercicio esta bueno y es con SOME**

¡Vamos paso a paso! El código:

const r5 = lista.some((v, i) =>

lista.some((w, j) => w === v && j !== i)

);

## ¿Qué hace some?

* Array.prototype.some(cb) recorre el array y **devuelve true si al menos un elemento hace que el callback cb devuelva true**.
* **Corta temprano**: en cuanto encuentra un true, deja de recorrer.

## Desglose del código

Hay **dos** some, uno “afuera” y otro “adentro”.

1. **some exterior**:  
   Recorre cada elemento de lista como par (v, i):
   * v = valor actual.
   * i = índice del valor actual.

Su misión: responder **“¿existe algún v que tenga un duplicado en otra posición?”**  
Para eso llama al…

1. **some interior**:  
   Para ese v fijo, vuelve a recorrer **toda** la lista como (w, j) y evalúa:
2. w === v && j !== i
   * w === v → ¿hay otro valor igual?
   * j !== i → ¿está en **otra** posición? (evitar contar al mismo elemento)

Si encuentra **al menos un** w en otro índice j que sea igual a v, este some interior devuelve true.  
Y como el exterior sólo necesita un true interno para decir “hay duplicados”, **también devuelve true y se detiene**.

### En castellano lógico

“¿Existe un elemento v (en posición i) tal que **existe** otro elemento w (en posición j distinta) con w === v?”  
Si la respuesta es sí para algún v, r5 es true.

## Traza rápida con tu lista

lista = [4, 1, 2, 4, 5, 8, 7, 6, 9, 10]

* Exterior: i = 0, v = 4  
  Interior recorre:
  + j = 0 → w=4, pero j !== i es falso (mismo elemento) → no cuenta
  + j = 1 → w=1, 1 === 4 → falso
  + j = 2 → w=2, 2 === 4 → falso
  + j = 3 → w=4, 4 === 4 y 3 !== 0 → **verdadero** ✅  
    → some interior devuelve true  
    → some exterior **corta** y devuelve true.

Resultado final: r5 === true (porque el 4 se repite).

## ¿Por qué está j !== i?

Si no estuviera, **cada elemento coincidiría consigo mismo** (w === v cuando j === i) y el resultado sería true incluso sin duplicados. Esa comparación evita el falso positivo.

## Complejidad

* **Tiempo:** O(n²) en el peor caso (par de bucles anidados).
* **Espacio:** O(1) (no crea estructuras auxiliares).

## Sutilezas de igualdad

Se usa === (igualdad estricta):

* Números como 4 funcionan perfecto.
* NaN === NaN es **falso** → dos NaN no se considerarían “duplicados” con esta condición.
* Para **objetos/arrays**, compara **referencias**: dos objetos con el mismo contenido pero distintas referencias **no** son iguales.

## Variantes (con las mismas restricciones)

Si querés que además cuente NaN como duplicado, podés extender la condición:

const r5\_NaNOK = lista.some((v, i) =>

lista.some((w, j) =>

j !== i && (w === v || (Number.isNaN(w) && Number.isNaN(v)))

)

);

O, si quisieras **detectar cuál(es) valor(es) duplicado(s)** (no sólo booleano), aún con some/filter:

const duplicados = lista.filter((v, i, arr) =>

arr.some((w, j) => w === v && j < i) // apareció antes

);

// duplicados -> [4] en tu ejemplo

Esto te devuelve **cada repetido a partir de su segunda aparición** (útil para listar cuáles se repiten).

Si te sirve, te muestro cómo convertirlo en un test autocontenible que imprime paso a paso dónde corta cada some.