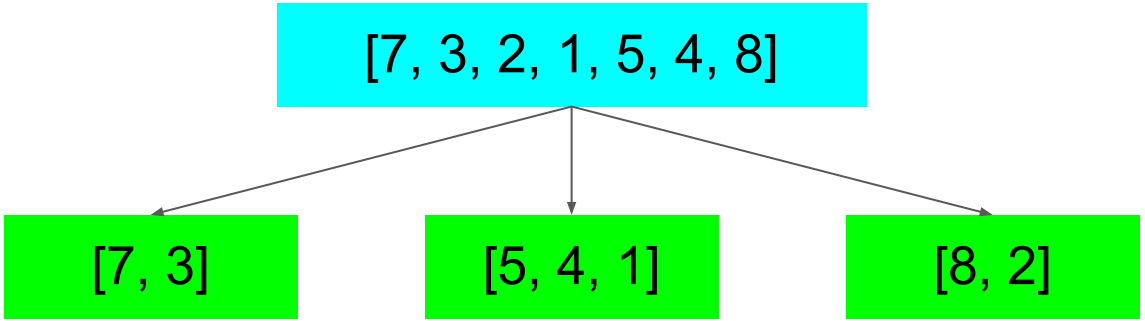
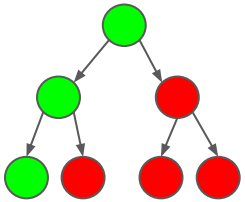
3-partition problem



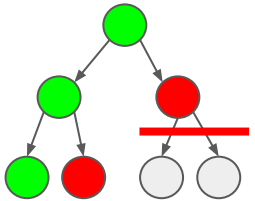
Algoritmos usados

Se probaron 2 versiones del algoritmo

Algoritmo sin podas
(ineficiente)



Algoritmo con podas
(eficiente)



WEBASSEMBLY

01

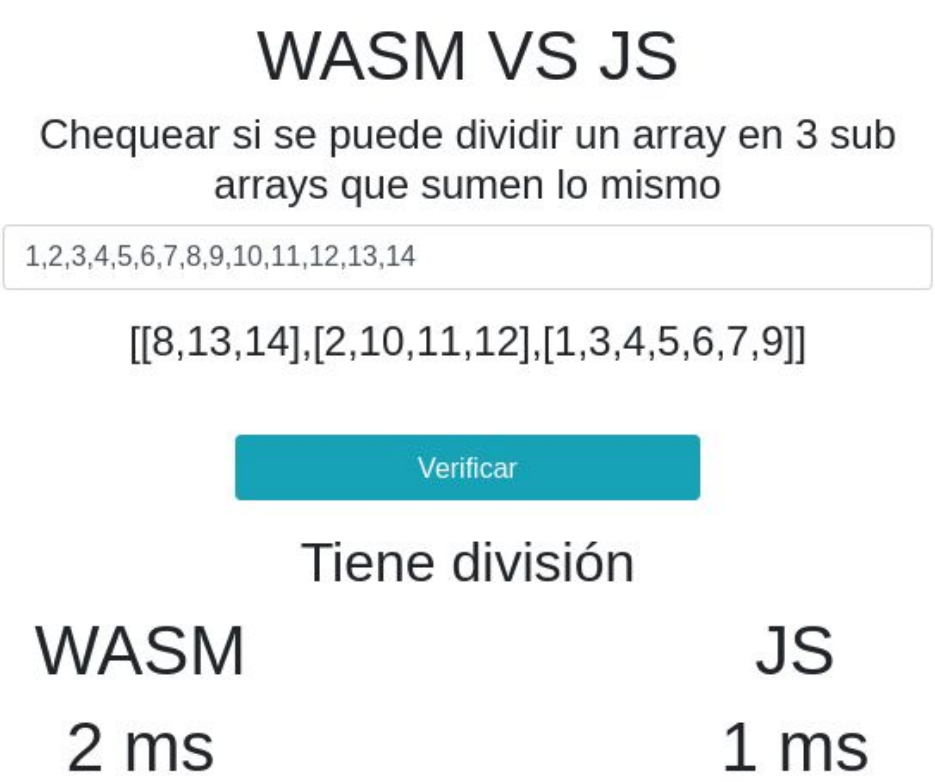
Demo algoritmo ineficiente

02

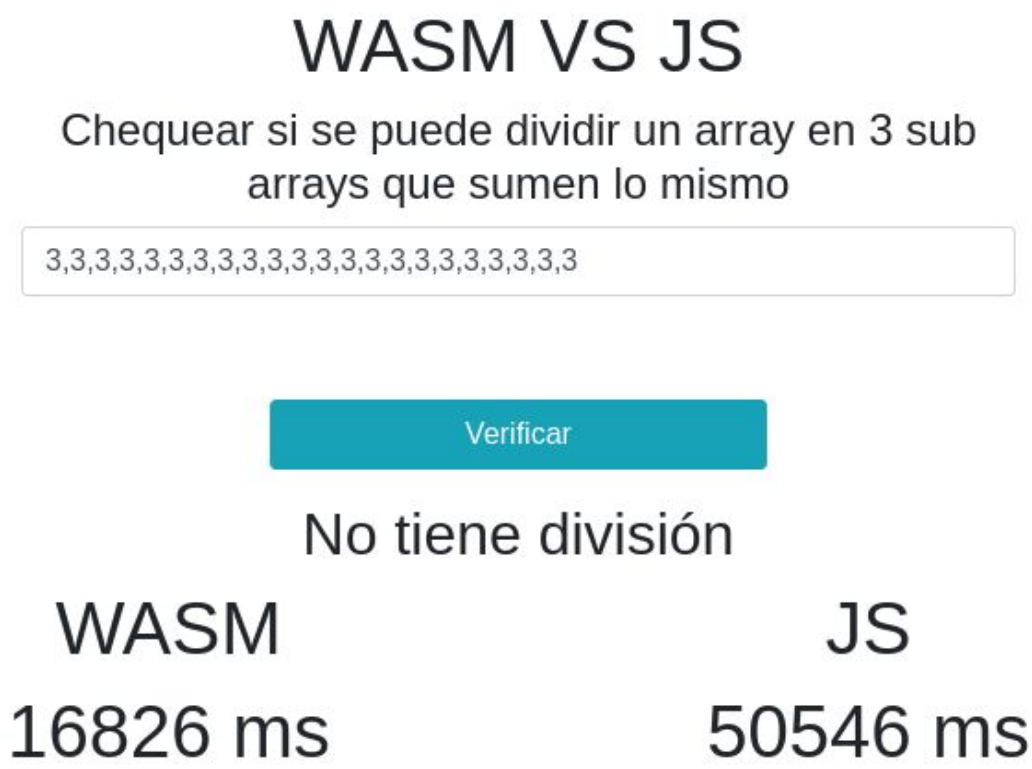
Demo algoritmo eficiente

03
Análisis de los resultados

Comparación de las ejecuciones: Inputs fáciles



Comparación de las ejecuciones: Inputs difíciles



WEBASSEMBLY

04

Análisis del código

Código en C

Código en bajo nivel

```
// Function to check array can be
// partition to 3 subsequences of
// equal sum or not
extern "C" {
  int checkEqualSum(int arr[], int N)
  {
    // Initialise 3 sums to 0
    int sum1, sum2, sum3;

    sum1 = sum2 = sum3 = 0;

    // Function Call
    if (checkEqualSumUtil(arr, N, sum1,
                          sum2, sum3, 0)== 1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
  }
}
```

WEBASSEMBLY

Código en bajo nivel

```
// Utility function to check array can
// be partition to 3 subsequences of
// equal sum or not
int checkEqualSumUtil(int arr[], int N,
                      int sm1, int sm2,
                      int sm3, int j)
{
  // Base Case
  if (j == N)
  {
    if (sm1 == sm2 && sm2 == sm3)
      return 1;
    else
      return 0;
  }
```

Código en C

Código en bajo nivel

```
else
{
  // When element at index
  // j is added to sm1
  int l = checkEqualSumUtil(arr, N,
                            sm1 + arr[j],
                            sm2, sm3, j + 1);

  // When element at index
  // j is added to sm2
  int m = checkEqualSumUtil(arr, N, sm1,
                            sm2 + arr[j],
                            sm3, j + 1);

  // When element at index
  // j is added to sm3
  int r = checkEqualSumUtil(arr, N, sm1, sm2,
                            sm3 + arr[j], j + 1);

  // Return maximum value among
  // all above 3 recursive call
  return max(max(l, m), r);
}
```

WEBASSEMBLY

Compilación de C a WASM

Exportar las funciones necesarias

```
EMCC=emcc

all: subset.cpp
  $(EMCC) -O3 -s WASM=1 -o wasmSubset.js \
  -s EXPORTED_RUNTIME_METHODS='["getValue", "setValue"]' \
  -s EXPORTED_FUNCTIONS="['_checkEqualSum', '_calloc']" \
  -s EXPORT_ES6=1 \
  -s MODULARIZE=1 subset.cpp
```

Paso de datos de JS a WASM

Transformar la lista generada en javascript a un array para ser utilizado por el algoritmo en WASM.

```
const generateArrayC = (list, mod) => {
  const arrayC = mod._calloc(list.length, 4)
  for (let i = 0; i < list.length; i++) {
    mod.setValue(arrayC + i * 4, list[i], "i32");
  }
  return arrayC;
}
```

Paso de datos de JS a WASM

Utilización del algoritmo WASM en nuestro index.js

```
const runWasm = (mod, arrayC, list) => {
  let startDateC = window.performance.now();
  const resultC = mod._checkEqualSum(arrayC, list.length);
  let endDateC = window.performance.now();
  return {startDateC, resultC, endDateC}
}
```

Manejo de memoria

Liberación de memoria cuando no es necesaria

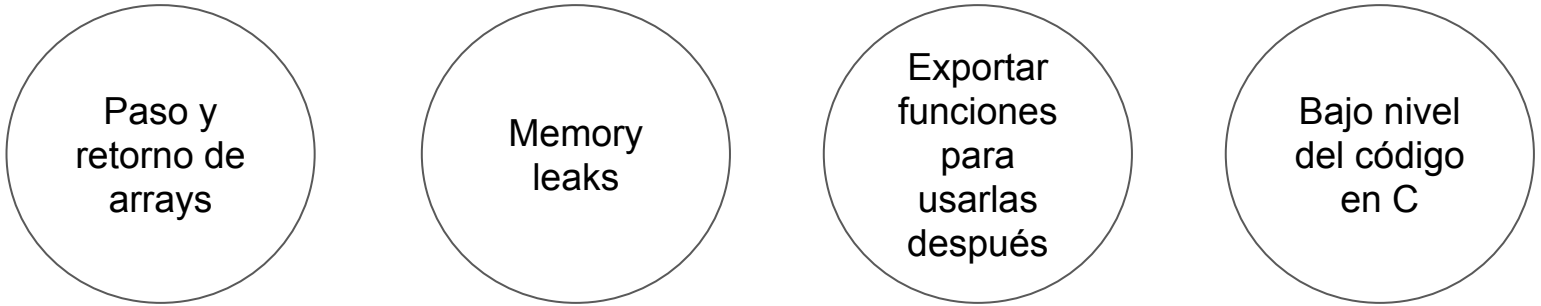
```
const runner = (mod) => {
  const list = document.querySelector('#input').value.split(',');
  if(list.length) {
    document.querySelector('#loading').innerHTML = 'Cargando ...';
    //document.querySelector('#list').innerHTML = JSON.stringify(list);
    const arrayC = generateArrayC(list, mod);
    const {startDateC, endDateC} = runWasm(mod, arrayC, list);
    mod._free(arrayC);
```

WEBASSEMBLY

05

Principales problemas

Principales problemas



06

Aprendizajes

Aprendizajes

- Pasar y retornar datos de js a wasm puede ser muy simple o increíblemente complejo, dependiendo del tipo de dato que sea.
- WASM no agrega una mejora en algoritmos simples, sólo para algoritmos de gran complejidad.
- En general, sólo conviene usar wasm al tener un problema de alta complejidad que deba resolverse en el frontend.