WebAssembly

Grupo 3

Martín Ocqueteaou

Nicolás Fraga

Agustín Ríos

1

Demostración

2

Unir WASM a la aplicación

De JavaScript a C++

```
function makePtrArrayC(array, mymod) {
  const newPtrArray = mymod._calloc(array.length, 4)
  for (let i = 0; i < array.length; i++) {
    mymod.setValue(newPtrArray + i * 4, array[i], "i32");
  }
  return newPtrArray;
}

function caller(array) {
  return Module().then(async (mymod) => {
    let newArray = makePtrArrayC(array, mymod);
    return await mymod._caller(newArray, array.length);
  });
}
```

Algunos inconvenientes

- Instalar y hacer funcionar emscripten
- Información adicional agregada a la compilación que no teníamos idea de cómo usar (modularización, exports)

Makefile

```
all: src/algorithm.cpp
	emcc -O3 -s WASM=1 -o src/c_implementation.js \
	-s EXTRA_EXPORTED_RUNTIME_METHODS='["getValue", "setValue"]' \
	-s EXPORTED_FUNCTIONS="['_calloc', '_caller']" \
	-s EXPORT_ES6=1 -s MODULARIZE=1 src/algorithm.cpp
```

Algunos inconvenientes

- Incompatibilidad de tipos entre C y C++
- Refactoring de implementaciones que pensábamos que estaban listas

```
extern "C" {
bool caller(int* arr, int n) {
  return equiSumUtil(arr, n);
}
```

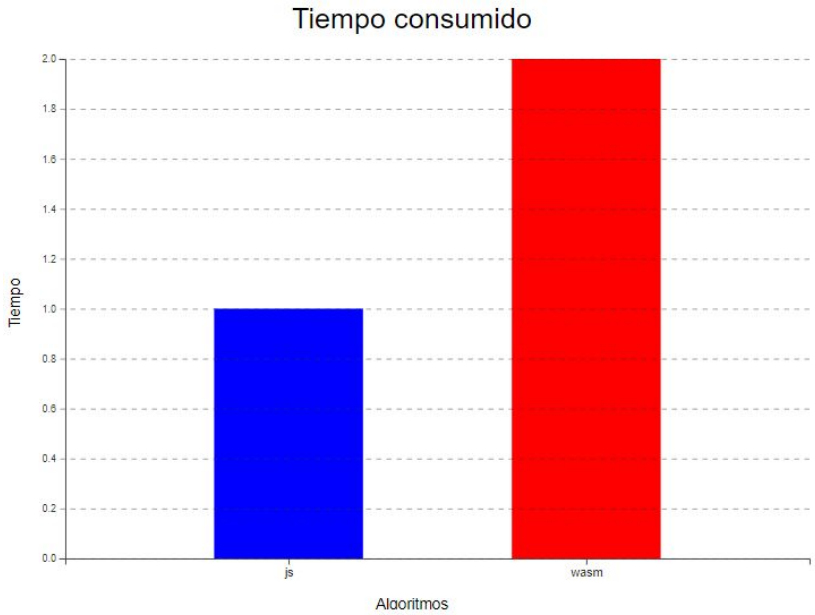
Algunos otros inconvenientes

- Enfoque de promesas causó confusión al intentar resolver una función compilada desde C.

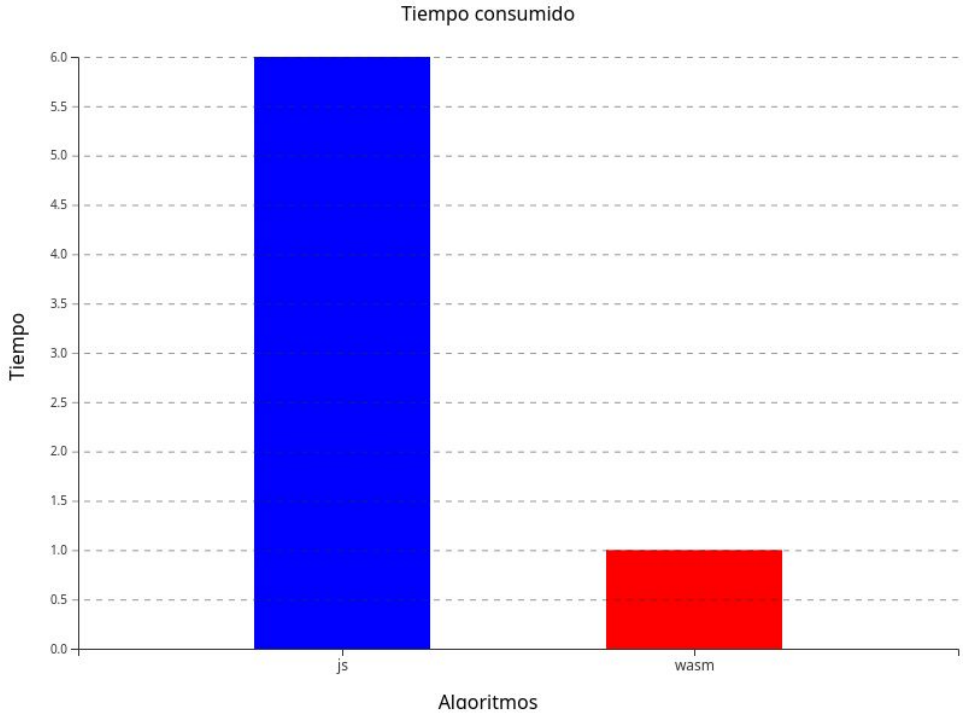
3

Performance: Visualización con D3

Mediciones de rendimiento



Mediciones de rendimiento



BIG CONCEPT

Bring the attention of your audience over a key concept using icons or illustrations