Web Assembly

Equipo 09

Para compilar:

emscripten

Compilación de C a JS/WASM

```
emcc lib/main.c -s WASM=1 -o public/main.js -s INITIAL_MEMORY=1GB
-s EXPORTED_FUNCTIONS="['_solution', '_malloc', '_free']" -s
EXPORTED_RUNTIME_METHODS=ccall
```

Llamada de la función de C en JavaScript

```
const bytesPerElement = 4; // assuming each element is a 32-bit integer
const tasksPtr = _malloc(tasks.length * bytesPerElement);
Module.HEAP32.set(tasks, tasksPtr / bytesPerElement);

const t0 = performance.now();
const resultC = UTF8ToString(
  Module.ccall(
    'solution',
    'number',
    ['number', 'number', 'number'],
    [tasks.length, tasksPtr, clusters]
  )
);
const t1 = performance.now();
```

¿Cuál es el problema?

Fuerza bruta

- Para 5 trabajos y 3 clusters:
  - Cada trabajo posee 3 opciones donde ejecutarse
  - Permutaciones = 3*3*3*3*3 = 5**3 = 125

- Para 1000 trabajos y 8 clusters:
  - Permutaciones = 1000**8 = 10**24 = Mucho

Algoritmo en C

Flujo de la solución

```
char *solution(int n, int times[], int l)
{
  srand(time(NULL)); // Inicializar generador de números aleatorios

  State assignments[n];
  generateInitialSolution(n, l, times, assignments);
  localSearch(n, l, times, assignments, generateNeighbourExchangeTask);

  printTimesEachCluster(l, assignments, n, times);
  printTheoricTimes(n, l, times);

  // return a string with the assignments variable
  char *assignmentsString = (char *)malloc(n * sizeof(char));
  for (int i = 0; i < n; i++)
  {
    assignmentsString[i] = assignments[i].cluster + '0';
  }
  return assignmentsString;
}
```

State

```
typedef struct
{
  int task;     // Índice del task
  int cluster; // Índice de la cluster asignada
} State;
```

Inicialización

```
void generateInitialSolution(int n, int l, int times[], State assignments[])
{
  for (int i = 0; i < n; i++)
  {
    assignments[i].task = i;
    assignments[i].cluster = rand() % l;
  }
}
```

Búsqueda local

```
void localSearch(int n, int l, int times[], State assignments[], State (*generateNeighbour)(int, int, int[], State[])
{
  int iterations = 0;
  int NUMBER_ITERATIONS = 10000;
  while (iterations < NUMBER_ITERATIONS)
  {
    State bestNeighbour = findBestNeighbour(n, l, times, assignments, generateNeighbour);
    assignments[bestNeighbour.task].cluster = bestNeighbour.cluster;
    iterations++;
  }
}
```

Generar vecino

```
State generateNeighbourExchangeTask(int indexTask, int l, int times[], State assignments[])
{
  State neighbour;
  neighbour.task = indexTask;
  neighbour.cluster = rand() % l; // Seleccionar una cluster al azar
  return neighbour;
}
```

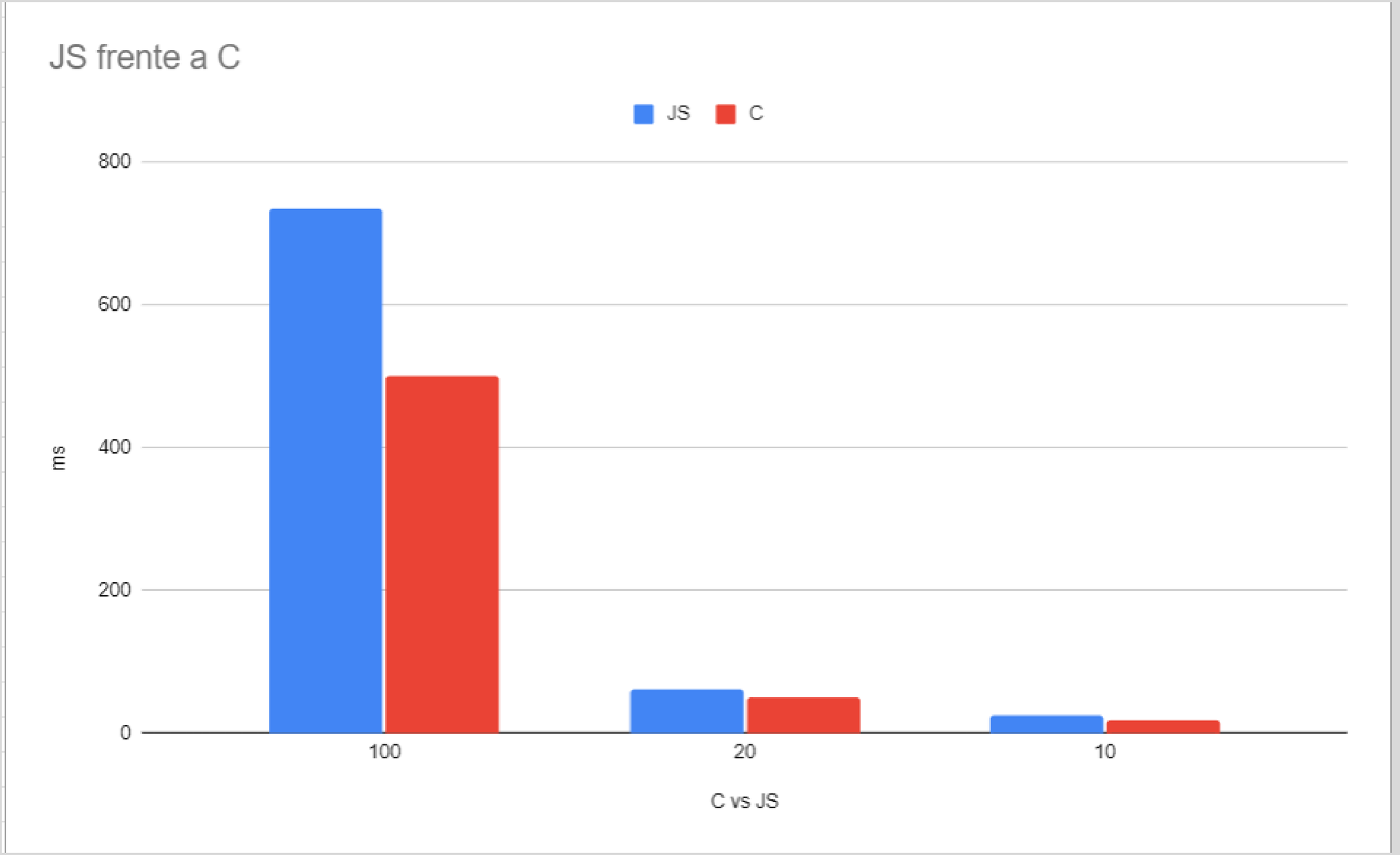
Mejor vecino

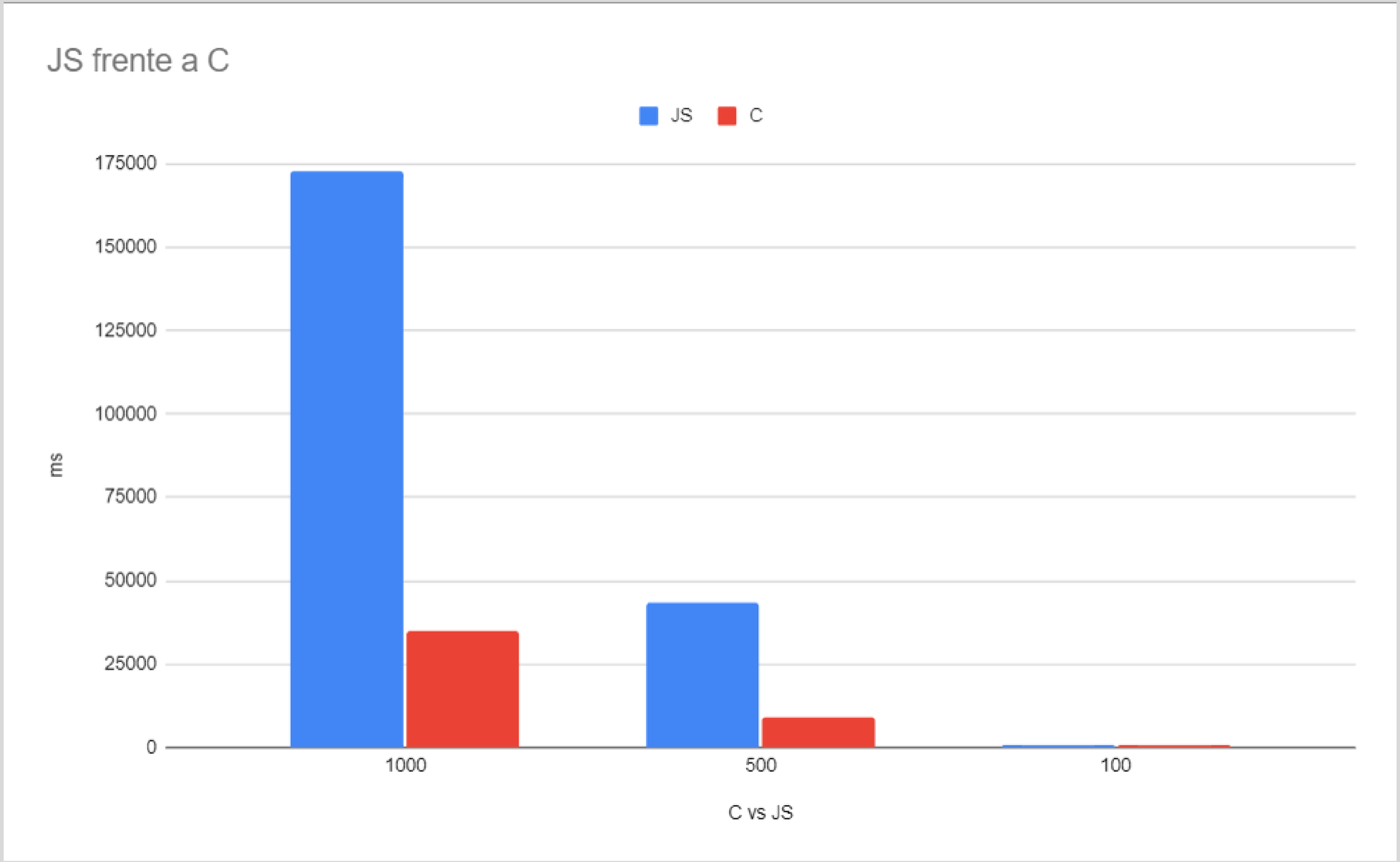
```
// Función para encontrar la solución vecina con menor time de cluster
State findBestNeighbour(int n, int l, int times[], State assignments[], State (*generateNeighbour)(int, int, int[], State[]))
{
  State bestNeighbour = assignments[0];
  int bestTime = getMaxTimesFromCluster(l, assignments, n, times);

  for (int i = 0; i < n; i++)
  {
    // asignacion "item - cluster" aleatoria
    State neighbour = generateNeighbour(i, l, times, assignments);
    int timeNeighbour = calculateTimeMaxTemporalAssignment(l, assignments, n, times, neighbour);
    if (timeNeighbour < bestTime)
    {
      bestNeighbour = neighbour;
      bestTime = timeNeighbour;
    }
  }
  return bestNeighbour;
}
```

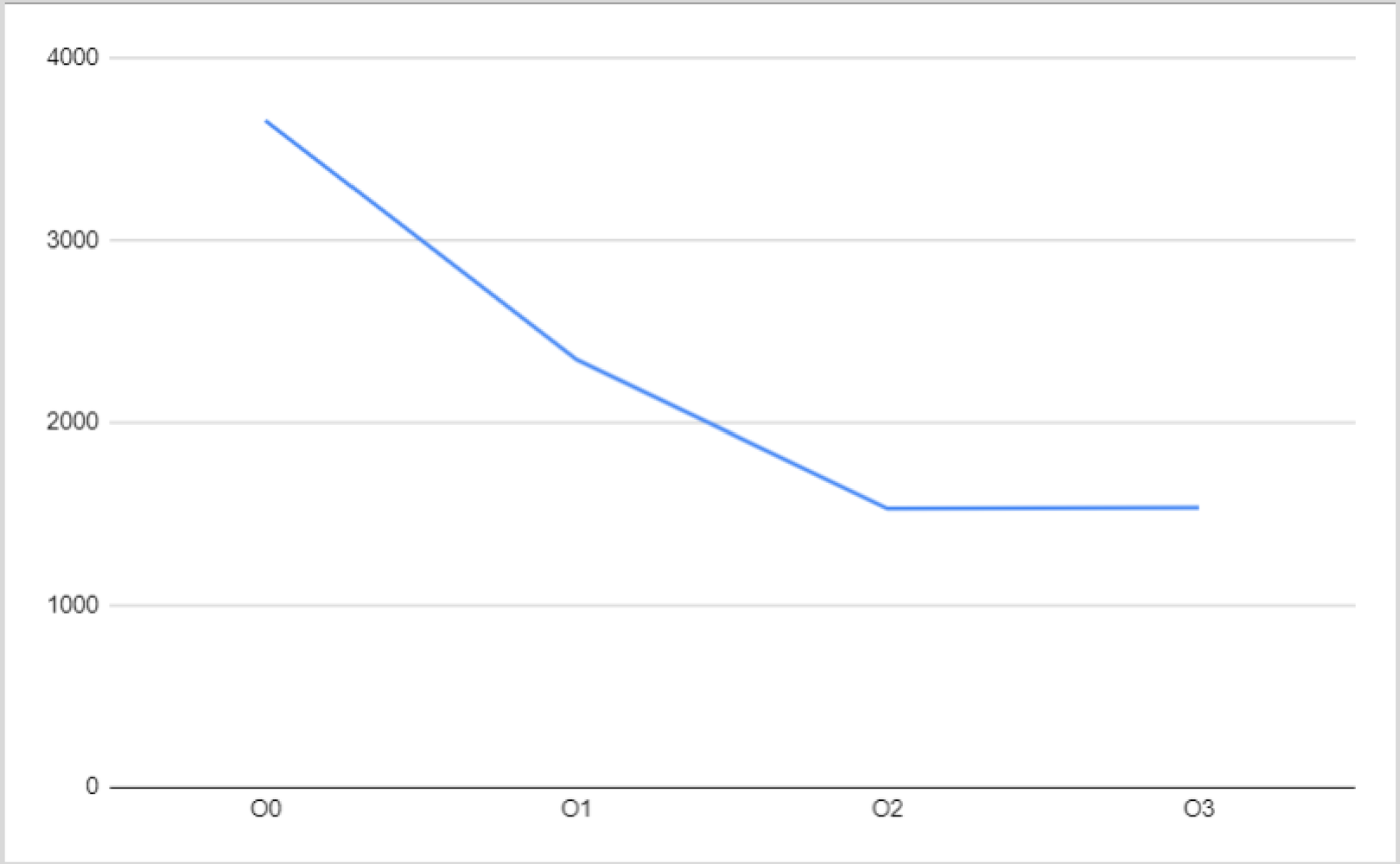
Calcular Tiempos

```
int *getTimesEachCluster(int l, State assignments[], int n, int times[])
{
  int *timesClusters = (int *)calloc(l, sizeof(int));
  for (int i = 0; i < n; i++)
  {
    timesClusters[assignments[i].cluster] += times[i];
  }
  return timesClusters;
}
```


JS frente a C
JS
C
800
600
400
200
0
ms
100
20
10
C vs JS


JS frente a C
JS
C
175000
150000
125000
100000
75000
50000
25000
0
ms
1000
500
100
C vs JS

Desempeño C con distintas optimizaciones (flag -O)



WASM permite darle eficiencia a nuestras aplicaciones web y reutilizar código existente

Web Assembly

Equipo 09