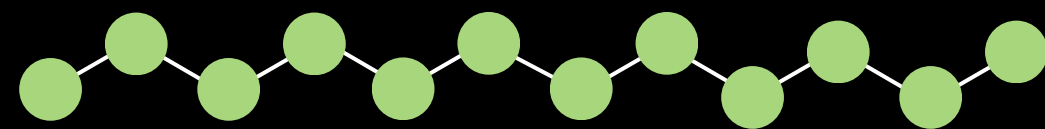


TAREA 3

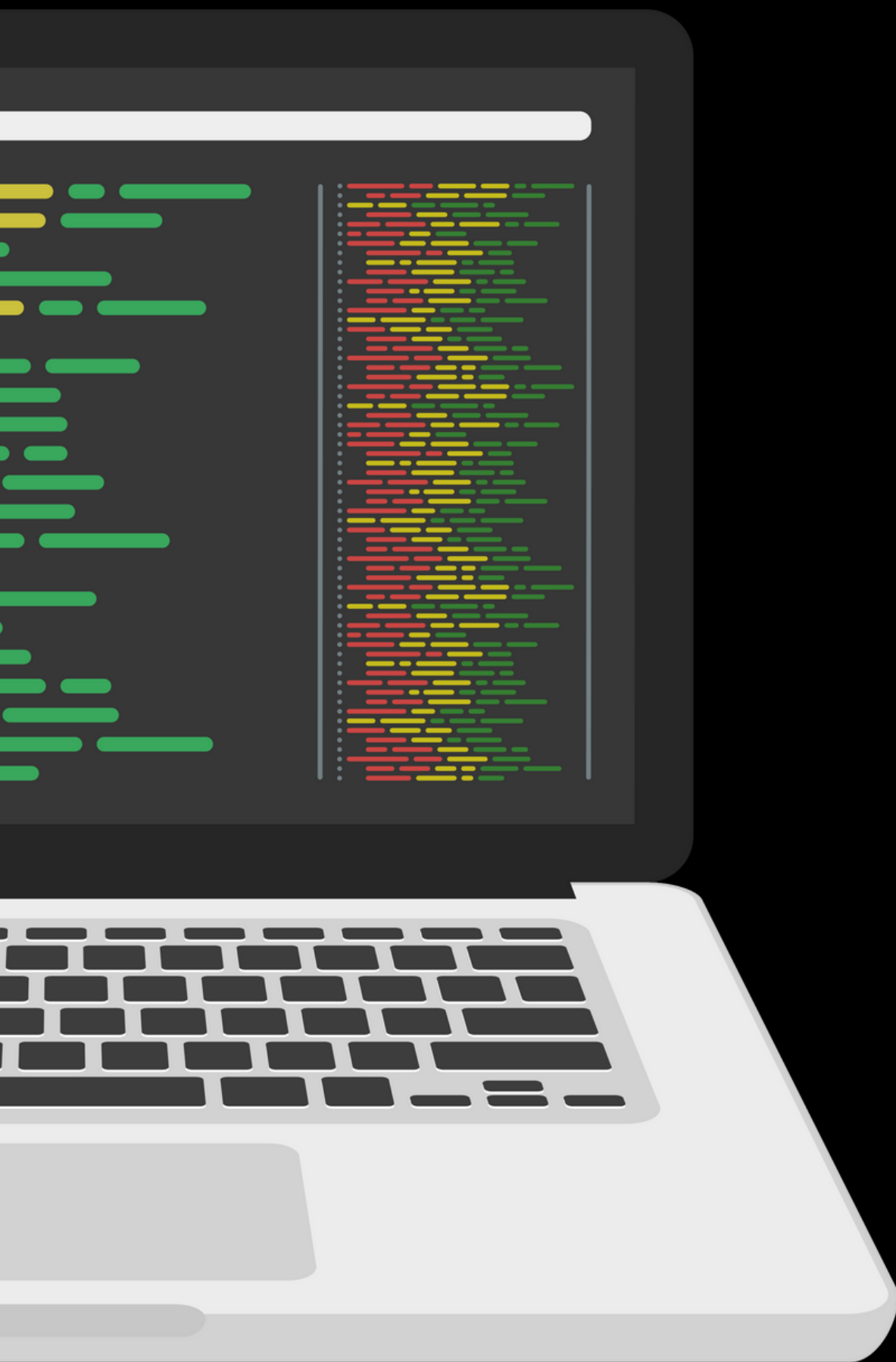
Diseño avanzado de aplicaciones web



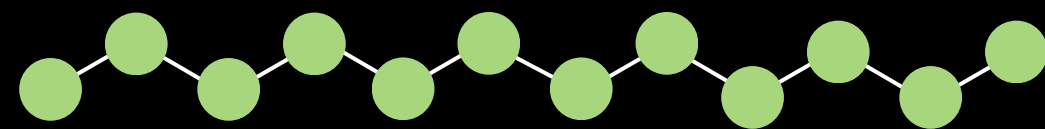
Grupo 3

TAREA A REALIZAR

- Conjunto de N trabajos donde cada uno posee un tiempo de ejecución.
- Conjunto de M clusters para asignar estos N trabajos.
- Desarrollar cada código en JS y en C.



DEMO



Grupo 3

ALGORITMO

- Ordenar los N trabajos de mayor a menor.
- Se utilizó `mergeSort` de complejidad $O(n \times \log n)$ para ordenar.
- Se utilizó `ListScheduling`, un algoritmo codicioso que sigue la heurística para lograr óptimos locales con la esperanza de lograr un óptimo global.
- Complejidad de `ListScheduling` es de $O(n)$.
- Input: cantidad tasks ➡ tiempo aleatorio entre 0 y 20.000

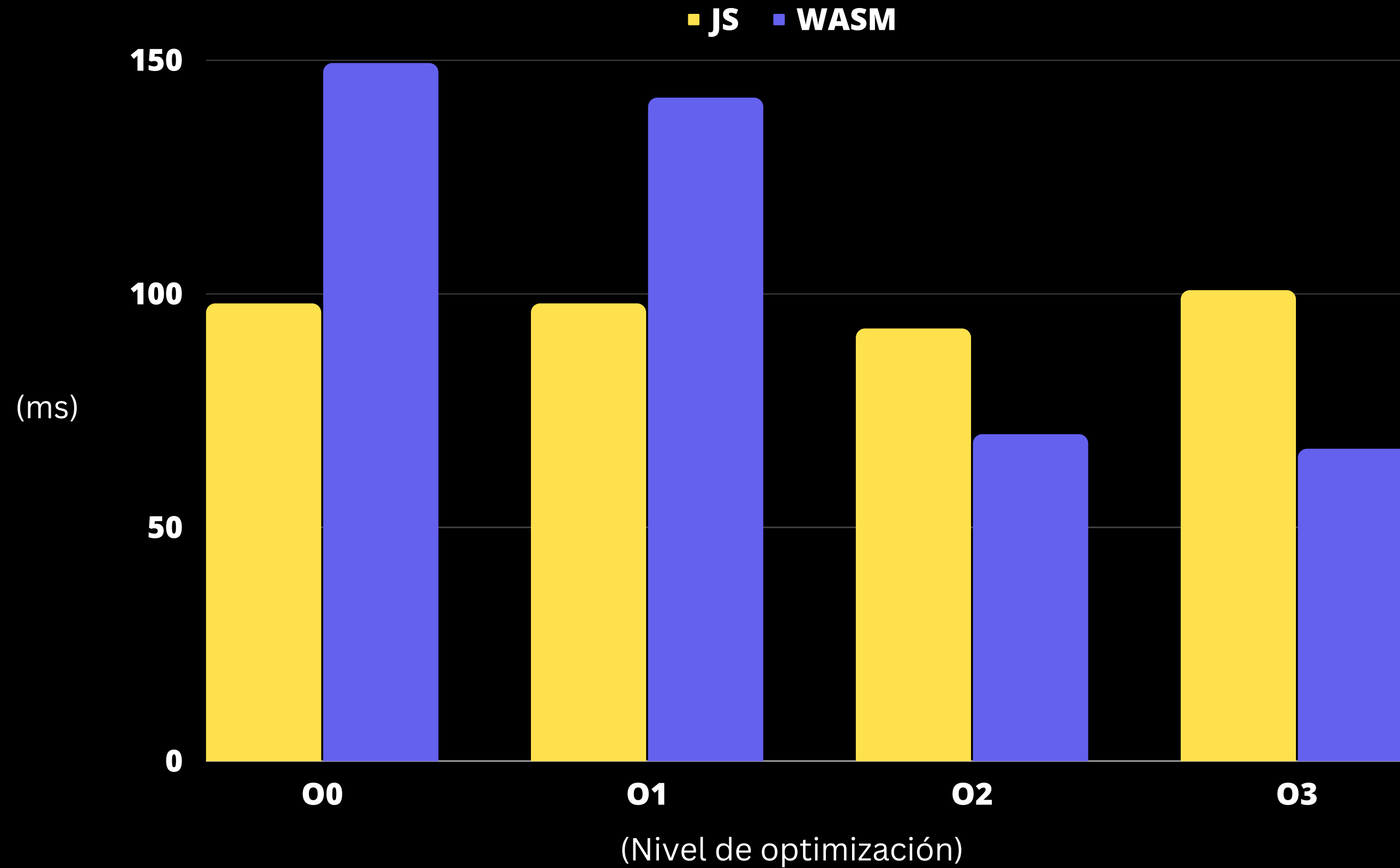
COMPILAR CON EMCC

```
emcc -sEXPORTED_FUNCTIONS=_listScheduling,_mergeSort,_malloc,_free -  
sEXPORTED_RUNTIME_METHODS=ccall -O3 -o main.js list_scheduling.c  
mergesort.c
```

- Funciones exportadas desde C.
- *Runtime methods* nativos de Emscripten para soportar funcionalidades de C/C++.
- Archivos desde donde se extraen las funciones de C.
- Nivel de optimización de código.

COMPARACIÓN OPTIMIZACIONES

Nivel medio



MERGESORT EN C

```
1 void mergeSort(int arr[], int l, int r)
2 {
3     if (l < r) {
4         // Same as (l+r)/2, but avoids overflow for
5         // large l and h
6         int m = l + (r - l) / 2;
7
8         // Sort first and second halves
9         mergeSort(arr, l, m);
10        mergeSort(arr, m + 1, r);
11
12        merge(arr, l, m, r);
13    }
14 }
```

```
1 void merge(int arr[], int l, int m, int r)
2 {
3     int i, j, k;
4     int n1 = m - l + 1;
5     int n2 = r - m;
6
7     /* create temp arrays */
8     int L[n1], R[n2];
9
10    /* Copy data to temp arrays L[] and R[] */
11    for (i = 0; i < n1; i++)
12        L[i] = arr[l + i];
13    for (j = 0; j < n2; j++)
14        R[j] = arr[m + 1 + j];
15
16    /* Merge the temp arrays back into arr[l..r]*/
17    i = 0; // Initial index of first subarray
18    j = 0; // Initial index of second subarray
19    k = l; // Initial index of merged subarray
20    while (i < n1 && j < n2) {
21        if (L[i] <= R[j]) {
22            arr[k] = L[i];
23            i++;
24        }
25        else {
26            arr[k] = R[j];
27            j++;
28        }
29        k++;
30    }
31
32    /* Copy the remaining elements of L[], if there
33    are any */
34    while (i < n1) {
35        arr[k] = L[i];
36        i++;
37        k++;
38    }
39
40    /* Copy the remaining elements of R[], if there
41    are any */
42    while (j < n2) {
43        arr[k] = R[j];
44        j++;
45        k++;
46    }
47 }
```

LISTCHEDULING EN C




```
1  int** listScheduling(int* tasks, int n_workers, int n_tasks)
2  {
3      int* current_time_workers = calloc(n_workers, sizeof(int));
4      int** workers = calloc(n_workers, sizeof(int*));
5      for (int i = 0; i < n_workers; i++)
6      {
7          workers[i] = calloc(n_tasks, sizeof(int));
8      }
9
10     for (int i = n_tasks - 1; i >= 0; i--)
11     {
12         int minimunIndex = findMinimunIndex(current_time_workers, n_workers);
13         current_time_workers[minimunIndex] += tasks[i];
14         int taskIndex = findMinimunIndex(workers[minimunIndex], n_tasks);
15         workers[minimunIndex][taskIndex] = tasks[i];
16     }
17     free(current_time_workers);
18
19     return workers;
20 }
```


MERGESORT EN JAVASCRIPT

```
1 function mergeSort(arr, l, r){
2     // const start = performance.now();
3     if(l >= r){
4         return; // returns recursively
5     }
6     let m = l + parseInt((r-l)/2);
7     mergeSort(arr, l, m);
8     mergeSort(arr, m+1, r);
9     merge(arr, l, m, r);
10 }
```

```
1 function merge(arr, l, m, r)
2 {
3     let n1 = m - l + 1;
4     let n2 = r - m;
5
6     // Create temp arrays
7     let L = new Array(n1);
8     let R = new Array(n2);
9
10    // Copy data to temp arrays L[] and R[]
11    for (let i = 0; i < n1; i++)
12        L[i] = arr[l + i];
13    for (let j = 0; j < n2; j++)
14        R[j] = arr[m + 1 + j];
15
16    // Merge the temp arrays back into arr[l..r]
17
18    // Initial index of first subarray
19    let i = 0;
20
21    // Initial index of second subarray
22    let j = 0;
23
24    // Initial index of merged subarray
25    let k = l;
26
27    while (i < n1 && j < n2) {
28        if (L[i] <= R[j]) {
29            arr[k] = L[i];
30            i++;
31        }
32        else {
33            arr[k] = R[j];
34            j++;
35        }
36        k++;
37    }
38
39    // Copy the remaining elements of
40    // L[], if there are any
41    while (i < n1) {
42        arr[k] = L[i];
43        i++;
44        k++;
45    }
46
47    // Copy the remaining elements of
48    // R[], if there are any
49    while (j < n2) {
50        arr[k] = R[j];
51        j++;
52        k++;
53    }
54 }
```

LISTCHEDULING EN JAVASCRIPT



```
1 function listScheduling(tasks, n_workers, n_tasks) {  
2   var workers = Array(n_workers).fill([]);  
3   var current_time_workers = Array(n_workers).fill(0);  
4  
5   for (let i = 0; i < n_workers; i++) {  
6     workers[i] = Array(n_tasks).fill(0);  
7   }  
8  
9   for (var i = n_tasks - 1; i >= 0; i--) {  
10    var minimunIndex = findMinimunIndex(current_time_workers);  
11    current_time_workers[minimunIndex] += tasks[i];  
12    const tasksIndex = findMinimunIndex(workers[minimunIndex]);  
13    workers[minimunIndex][tasksIndex] = tasks[i];  
14  }  
15  
16  return workers;  
17 }
```

INTERACCIÓN DE WASM



```
1  // WASM Module //
2
3  Module.onRuntimeInitialized = () => {
4    document.getElementById('run-wasm').onclick = () => {
5      wasmTime.style.display = "none";
6      const { tasks, n_workers, n_tasks } = buildInitialWorkersTasks();
7
8      const tasksArray = Uint32Array.from(tasks);
9
10     const taskArrayPtr = Module._malloc(tasksArray.byteLength);
11     Module.HEAPU32.set(tasksArray, taskArrayPtr>>2);
12
13     let workersPtr;
14     wasmTime.style.display = "none";
15     wasmLoading.style.display = "flex";
16     setTimeout( () => {
17       const startTime = performance.now();
18       workersPtr = Module.ccall(
19         'listScheduling',
20         'number',
21         ['number', 'number', 'number'],
22         [taskArrayPtr, n_workers, n_tasks]);
23
24       const endTime = performance.now();
25
26       wasmTime.children[0].children[0].innerHTML = `${Math.round((endTime - startTime) * 100) / 100} ms`;
27       wasmLoading.style.display = "none";
28       wasmTime.style.display = "flex";
29
30     }, 0)
31
32     const workersAssigned = new Uint32Array(Module.HEAPU32.buffer, workersPtr, offsetResult(n_workers, n_tasks));
33     const workersAssignedFiltered = filterWorkersAssigned(workersAssigned, n_workers, n_tasks);
34     const resultWorkersAssigned = extractResult(workersAssignedFiltered, n_workers, n_tasks);
35     console.log({ resultWorkersAssigned })
36
37     Module._free(taskArrayPtr);
38   }
39 }
```

CASOS DE USO



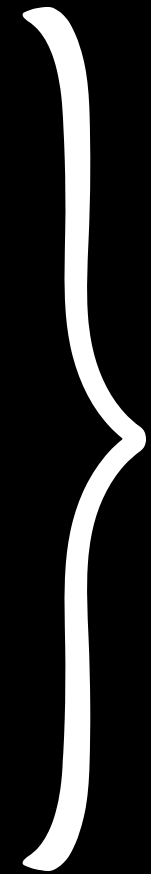
Reducción de tiempo de carga al inicializar la app.

Renderizado del diseño.

Descarga de los archivos.

OTROS CASOS DE USO

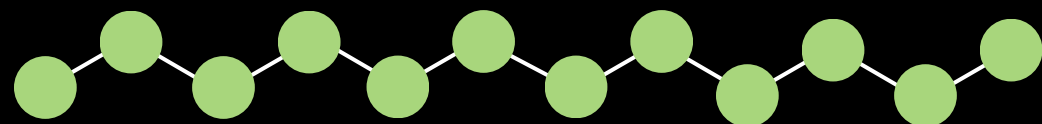
- Edición de audio y video
- Videojuegos
- Simulaciones
- Encriptación



PERFORMANCE

APRENDIZAJES

- No olvidar compilar de la forma más óptima para producción .
JS no se queda atrás.
- Usar WASM no es difícil y es una opción que se **tiene** que tener en cuenta cuando se busque velocidad.
- Hay que tener cuidado con los valores de retorno desde C al querer usarlos en JS.





Grupo 3