

# PRÁCTICA AC – FASE II

Ejercicios en Ensamblador, SPEC y Benchmark reducido

Realizada por **Pavel Razgovorov** ([pr18@alu.ua.es](mailto:pr18@alu.ua.es)) con N.I.E. Y0888160-Y del **grupo 1** de prácticas (lunes 13:00-15:00)



## Índice

Parte 1: Ejercicios de Ensamblador x86 con Visual Studio .....	3
Objetivo del ejercicio .....	3
Algoritmo de la burbuja (bubble-sort). Implementación .....	3
Análisis de tiempos de ejecución y conclusiones.....	5
Parte 2: Proyecto de Evaluación de Rendimiento con SPEC.....	6
Objetivo del ejercicio .....	6
Preparación y búsqueda de referencias .....	6
Comparación y análisis de resultados.....	8
Conclusiones obtenidas a través de los resultados.....	9
Ranking de velocidad en función de la cantidad de núcleos.....	10
Evaluación de los miembros del equipo .....	11

## Parte 1: Ejercicios de Ensamblador x86 con Visual Studio

### Objetivo del ejercicio

Necesitamos familiarizarnos tanto con el Entorno de desarrollo Visual Studio, así como la herramienta de código Ensamblador (ASM x86) *inline* que viene incluida.

En el ejercicio 1 se nos plantea un ejemplo de implementación para un primer contacto, en el ejercicio 2 se nos plantea otro ejemplo de cómo podemos fácilmente integrar el código en una interfaz con Windows Forms y en el último se nos plantea el algoritmo de la burbuja y nos piden que hagamos una implementación de éste, pero hecho totalmente en ASM.

### Algoritmo de la burbuja (bubble-sort). Implementación

Partiendo del algoritmo –en pseudocódigo– que nos dan en el enunciado (una vez corregido), podemos hacer una primera implementación en C (que usaremos para una comparativa):

```
for (int i = 0; i < tam-1; i++) {  
    for (int j = tam-1; j > i; j--) {  
        if (v2[j] > v2[j-1]) { // menor o mayor dependiendo del sentido  
            int intercambio = v2[j-1];  
            v2[j-1] = v2[j];  
            v2[j] = intercambio;  
        }  
    }  
}
```

Una vez interiorizado el funcionamiento, tratamos de implementar el algoritmo en ASM teniendo en cuenta los siguientes consejos:

- Acceder a memoria es caro. Cuanto más uso hagamos de los registros, mejor.
- Hacer saltos es caro. Cuantos menos tengamos que hacer, mejor.
- Repetir instrucciones y el código redundante es CARÍSIMO. Hay que pensar bien.

Posiblemente, siguiendo estos consejos, nuestro algoritmo en ASM se parecerá muy poco la idea inicial planteada con C, pero el objetivo no es acercarse al pensamiento humano, si no al del procesador.

La implementación en ASM del algoritmo es la siguiente:

```
__asm {
    mov eax, w; //eax será el puntero al vector
    mov edx, tam; //edx, registro multiuso, cargamos el tamaño del array
    dec edx; //tam-1

    //mov ebx, 0; //ebx será i
    xor ebx, ebx; //hacer un xor a un registro es más eficiente que ponerle un 0
bucle1:
    cmp ebx, edx; //si son iguales, salir (i < tam-1)
    je fin;

    mov ecx, edx; //ecx será j = tam-1
bucle2:
    cmp ecx, ebx; //si son iguales, salir (j > i)
    je fin_bucle1;

    //FORMULA PARA ACCEDER A LA MEMORIA: Puntero inicial + posición del array *
    tamaño (en bytes) del dato
    mov esi, [eax+ecx*4]; //esi será v[j]
    dec ecx; //decrementamos j para tener el numero j-1
    mov edi, [eax+ecx*4]; //edi será v[j-1]
    //inc ecx; //deshacemos el incremento
    cmp esi, edi; //si uno es menor que otro, intercambiamos (v[j] < v[j-1])
    jl intercambio; //jl si ascendente; jg si descendente
    jmp bucle2; //terminamos bucle. j-- (ya hemos decrementado antes) y volvemos
    a empezar

intercambio:
    mov [eax+ecx*4], esi; //ponemos en v[j-1] esi, que era v[j]
    inc ecx; //deshacemos el incremento
    mov [eax+ecx*4], edi; //ponemos en v[j] edi, que era v[j-1]
    dec ecx; //terminamos bucle. j-- y volvemos a empezar
    jmp bucle2;

fin_bucle1:
    inc ebx; //terminamos bucle. i++ y volvemos a empezar
    jmp bucle1;

fin:
}
```

Aquí hay varias cosas que aclarar:

1. eax y edx son siempre valores constantes en el programa. Ambos muy consultados.
2. Respecto a que xor ebx, ebx es más eficiente que mov ebx, 0... [\[link\]](#)
3. En la fórmula para acceder al array uso el 4 como tamaño del dato porque sé que el array es de enteros y al tener un tamaño de 32 bits,  $\frac{32 \text{ bits}}{8 \text{ bits/byte}} = 4 \text{ bytes}$
4. Curiosamente, aunque definamos el acceso a memoria con operaciones de suma y multiplicación ~tal cual~, la única traducción que sufre es :  
`mov esi, [eax+ecx*4]      →      mov esi, DWORD PTR [eax+ecx*4]`
5. No existe ningún bloque de código “fin\_bucle2” debido a que hacemos los decrementos a mitad de comparación de los valores, así como en el intercambio (aprovechamos esa instrucción que necesitamos hacer para tener el valor j-1). La instrucción de salto al bucle2 la repetimos en ambos sitios y nos ahorramos saltos.

## Análisis de tiempos de ejecución y conclusiones

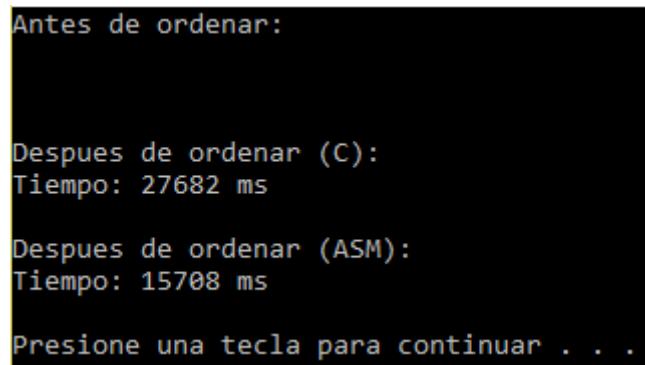
Quiero hacer una comparativa del tiempo de ejecución del algoritmo en C y en ASM para ver cuál es el incremento de velocidad que sufre al ser implementado al nivel más bajo posible. Para ello, hemos utilizado las funciones QueryPerformanceFrequency y QueryPerformanceCounter, ambas disponibles en el API de Windows, que recogemos en esta pequeña función.

```
long long milliseconds_now() {
    static LARGE_INTEGER s_frequency;
    static BOOL s_use_qpc = QueryPerformanceFrequency(&s_frequency);
    if (s_use_qpc) {
        LARGE_INTEGER now;
        QueryPerformanceCounter(&now);
        return (1000LL * now.QuadPart) / s_frequency.QuadPart;
    }
    else return GetTickCount();
}
```

Su funcionamiento es sencillo: hacemos una llamada antes de lo que queramos medir, y otra cuando haya terminado. El resultado, en milisegundos, será la diferencia entre ambas medidas.

Para obtener resultados más fiables necesitaremos una talla del problema considerable, así que lo mejor es aumentar el tamaño del vector a ordenar a 100K elementos (a cuanto mayor sea la talla, mayor diferencia habrá entre uno y otro; las comparativas se hacen en función de las necesidades de cada problema, así que este resultado será solamente orientativo).

Ejecutamos el programa y el resultado obtenido es el siguiente:



```
Antes de ordenar:

Despues de ordenar (C):
Tiempo: 27682 ms

Despues de ordenar (ASM):
Tiempo: 15708 ms

Presione una tecla para continuar . . .
```

$Incremento = 27682 - 15708 = 11974 \text{ ms} \approx 12 \text{ segundos}$

$Aceleración_{\%} = \frac{27682 - 15708}{15708} \times 100 = 76\%$

Conclusión: Si bien programar en ensamblador no es nada productivo, a la hora de implementar un programa podríamos cederle algunas tareas a este lenguaje, siempre y cuando éstas supongan una gran ralentización en el equipo que se ejecuta siendo el tiempo de respuesta un elemento crítico y, en la medida de lo posible, que las tareas sean fáciles de implementar. Como hemos podido observar, el incremento es simplemente BESTIAL.

## Parte 2: Proyecto de Evaluación de Rendimiento con SPEC

### Objetivo del ejercicio

Se nos presenta una suite de benchmarking para ordenadores, de nombre SPEC CPU2000, la cual analiza el rendimiento del procesador, la memoria y la potencia del compilador a nivel de componente.

Debemos de, primero, buscar en su página oficial de resultados publicados una configuración de computador que se parezca a la de los ordenadores de la EPS y analizar los resultados obtenidos. Después, ejecutar los test en un ordenador del laboratorio y comparar los resultados.

Como último ejercicio se nos pide elaborar un ranking siguiendo los test SPEC CPU2000INT de cuatro configuraciones de un mismo fabricante y con frecuencia de reloj parecida, pero que tengan 2, 4, 8 y 16 núcleos para comprobar qué tanta mejora sigue habiendo cada vez que duplicamos la cantidad de núcleos disponibles.

### Preparación y búsqueda de referencias

Antes de comenzar, tenemos que puntuar que el programa que se nos facilita está obsoleto desde el año 2007, ya que fue sustituido por una nueva versión, el SPEC CPU2006. Por esto, los últimos resultados publicados son de aquella época y, por tanto, los componentes analizados son también lo son. Sin embargo, los ordenadores que tenemos que analizar utilizan componentes de a partir del año 2011, por lo que hay un desfase de casi un lustro. Con todo, elegiremos una configuración de características lo más parecidas posible, así como un procesador de la misma familia/gama y aprovechar para comprobar qué tanta mejora hubo durante ese intervalo de tiempo.

Para comenzar, se deben de conocer las especificaciones que tomaremos como referencia, que son, a grandes rasgos (solo lo que el programa analiza), las siguientes:

- CPU: Intel Pentium G840 Dual Core @ 2.8GHz, 65W
- RAM: 2x4GB @ 1333MHz DDR3
- Compilador: Visual Studio 2010 Compiler

Buscando en los resultados del SPEC [\[link\]](#), intentando encontrar un CPU Pentium con una frecuencia parecida, encontramos el siguiente [\[link\]](#):

- CPU: Intel Pentium D 920 Dual Core @ 2.8GHz, 95W
- RAM: 2x1GB @ 800MHz DDR2
- Compilador: Intel C++ Compiler 9.1

A grandes rasgos, ambos CPU's no parecen distinguirse mucho, pero, si miramos un poco por Internet [\[link\]](#), observamos estas principales diferencias:

	<b>G840 (EPS)</b>	<b>D 920 (SPEC)</b>
<b>Fabricación</b>	32 nm	65 nm
<b>Socket</b>	1155	755
<b>Caché</b>	0'5 MB/núcleo	2 MB/núcleo
<b>ISA's</b>	SSE (todas) y MMX	SSE (1, 2, 3) y MMX
<b>T. Virtualización</b>	Sí	No
<b>GPU Integrada</b>	Sí	No

Lo que podemos concluir antes de realizar los test al ordenador y compararlos con los datos de la web, es que el G840 ha sido fabricado en 32 nm, por lo que consumirá menos energía y no le hará falta tanta refrigeración. También tiene muchos más juegos de instrucciones; si el compilador es capaz de detectar esto y sacarle provecho, el incremento será muy considerable. Aunque esta última ventaja posiblemente no la podamos apreciar, tiene tecnologías de virtualización, lo cual hace que los programas que utilizan una máquina virtual para ejecutarse (Java, C#...) se optimicen mejor. Como contrapartida tenemos que el D 920 tiene 4 veces más memoria caché que el otro, haciendo que en programas de tráfico de datos quede muy a la altura.

Respecto a la memoria RAM, tenemos muchas diferencias: mayor capacidad, mayor frecuencia de reloj y tecnología de fabricación más nueva.

Y, respecto al compilador... Supuestamente el de Intel es, con diferencia, mucho mejor que el de Visual Studio; sobre todo cuando utilizas las opciones de optimización [\[link\]](#).

Ahora, pasamos a ejecutar los test, con el comando `runspec --reportable --config=win32-x86-vc7.cfg -T base int`, del cual explicamos sus argumentos:

- `--reportable`: notificará los resultados exportándolos a un archivo
- `config=win32-x86-vc7.cfg`: elige el fichero de configuración para compilar y ejecutar los test. Tenemos archivos de configuración para casi cada sistema operativo y para algunas otras arquitecturas en la carpeta config.
- `-T base int`: configura qué tipos de pruebas se realizarán y de qué manera. Base realiza las pruebas haciendo una media aritmética (también está la opción peak, picos, y all, ambas) e int realiza las pruebas con los programas de números enteros (también está la opción fp, punto flotante).

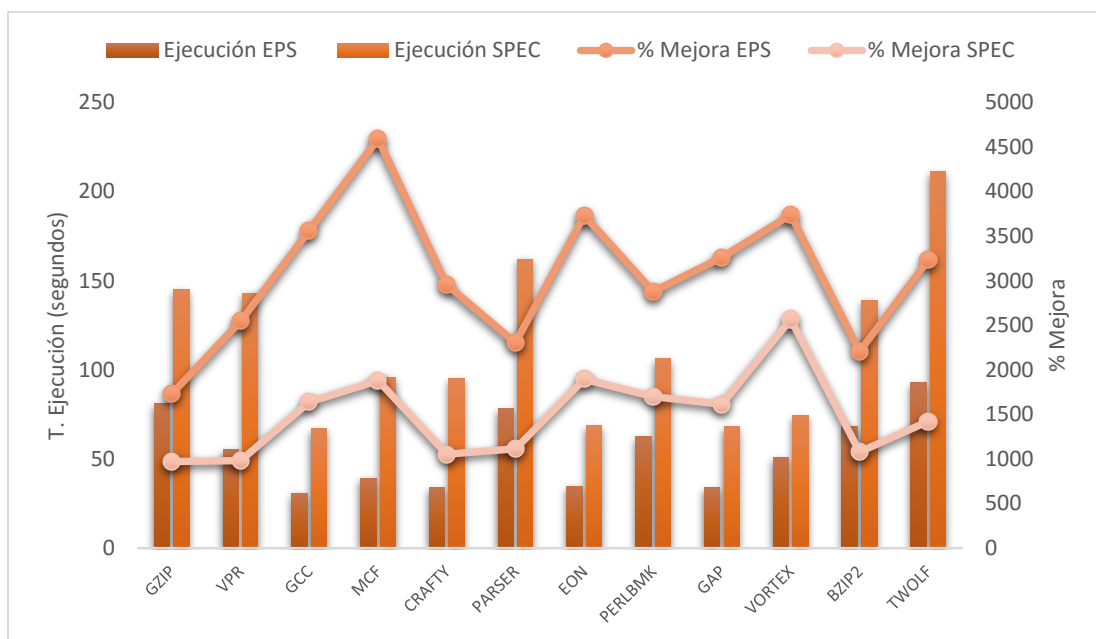
Una vez terminado, podremos visualizar el resultado en la carpeta result.

## Comparación y análisis de resultados

Ponemos los resultados encontrados y obtenidos en una tabla:

Test	Referencia	Ejecución SPEC	Ejecución EPS	% Mejora SPEC	% Mejora EPS
GZIP	1400	145	81	969	1729
VPR	1400	143	55	980	2543
GCC	1100	67,1	30,9	1639	3564
MCF	1800	95,8	39,2	1878	4590
CRAFTY	1000	95,1	33,9	1051	2954
PARSER	1800	162	78,2	1112	2301
EON	1300	68,6	34,8	1894	3731
PERLBMK	1800	106	62,6	1698	2875
GAP	1100	68,2	33,7	1612	3260
VORTEX	1900	74,1	50,8	2565	3739
BZIP2	1500	139	68	1078	2207
TWOLF	3000	211	92,8	1421	3232
MEDIA G.	1521	107	51	1424	2962

Para poder visualizar mejor la información, nos construimos un gráfico:



Y también calculamos el porcentaje de mejora respecto uno y otro siguiendo la fórmula

$$\frac{Ejecución_{EPS} - Ejecución_{SPEC}}{Ejecución_{SPEC}} \times 100:$$

Test	GZIP	VPR	GCC	MCF	CRAFTY	PARSER	EON	PERLBMK	GAP	VORTEX	BZIP2	TWOLF	MEDIA
SPEC	145	143	67,1	95,8	95,1	162	68,6	106	68,2	74,1	139	211	107
EPS	81	55	30,9	39,2	33,9	78,2	34,8	62,6	33,7	50,8	68	92,8	51
%	79	160	117	145	180	107	97	70	102	45	104	128	109



## Conclusiones obtenidas a través de los resultados

Las diferencias observadas son variadas; en algunos casos hay una mejora de más del 100% (teniendo como pico la prueba Crafty, siendo 2'8 veces más rápida) y en otros queda por debajo (siendo la prueba Vortex la que menos mejora, tan sólo 0'45 veces). Esto es debido a las características de los procesadores (tal y como hemos predicho anteriormente), ya que en las pruebas que más despunta están relacionadas con operaciones aritméticas (compresión, cálculo de costes, ajedrez, simulaciones) debido a la diferencia en el set de instrucciones y, en las que menos, tienen que ver con carga de datos (procesamiento de lenguaje, intérprete de lenguaje, bases de datos) debido a la diferencia en las memorias caché.

De media (geométrica, ya que es más fiable si no hay ninguna medida) obtenemos un 109% de mejora de rendimiento. Teniendo en cuenta que hay una diferencia de 4 años entre ambos procesadores, podemos calcular el incremento de mejora por año con la fórmula

$$\left( \sqrt[4]{Ej_{SPEC}/Ej_{EPS}} - 1 \right) \times 100 = \left( \sqrt[4]{107/51} - 1 \right) \times 100 \approx 20'35\%$$

La prueba VPR es un caso especial, ya que se testea la memoria RAM y su arquitectura. En este caso, había mucha diferencia entre la máquina del laboratorio y la que hemos elegido como referencia, pues es 2'6 veces más rápida.

También podemos calcular la mejora anual de las memorias RAM, que nos da un

$$\left( \sqrt[4]{143/55} - 1 \right) \times 100 \approx 27\%$$

Queda en duda el aspecto del compilador y qué tan buena mejora ofrece. Pero para ello hubiéramos necesitado tener ambas configuraciones idénticas para poder así evaluar la diferencia. Sin embargo, el usar máquinas diferentes y compiladores también diferentes, pone en duda la fiabilidad de los datos obtenidos, ya que no se trata simplemente de versiones diferentes o con distintos argumentos, sino de dos productos totalmente distintos. El resultado hubiera sido distinto, a favor del ordenador de la EPS, si tuviéramos disponible el compilador de Intel.

## Ranking de velocidad en función de la cantidad de núcleos

Para realizar este ranking, escogeremos procesadores cuya frecuencia de reloj por núcleo sea parecida, así como su fabricante. Recogeremos como dato comparativo el porcentaje de incremento de las pruebas base y pico. Los procesadores elegidos son los siguientes:

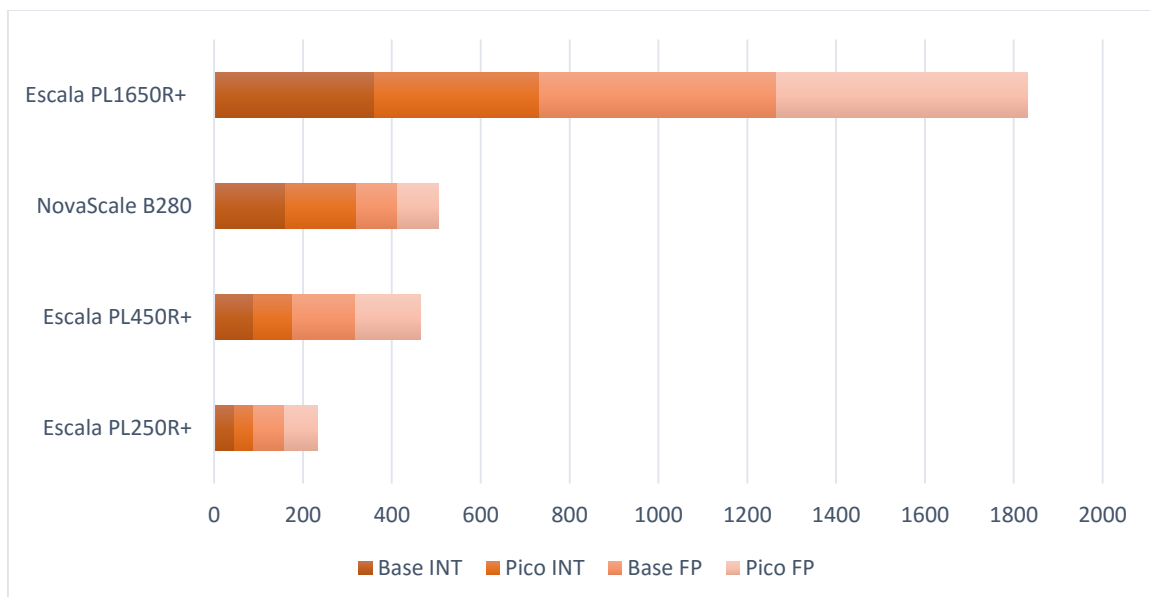
Configuración	Cores	RAM	Base INT	Pico INT	Base FP	Pico FP
Escala PL250R+ (POWER5+, 2.1GHz)	2	32 GB	44,8	44,5	69,4	73,1
Escala PL450R+ (POWER5+, 2.1GHz)	4	32 GB	87,4	89,9	140	148
NovaScale B280 (Xeon E5335, 2.0GHz)	8	8 GB	160	161	91,1	92,4
Escala PL1650R+ (POWER5+, 2.2GHz)	16	128 GB	360	372	535	564

Antes de nada, es necesario comentar que el CPU elegido para el caso de los 8 núcleos tiene poco que ver con el resto (haciendo que el resultado pueda ser extraño en algunos casos), por los siguientes motivos:

1. Usa una arquitectura x86, cuando los demás usan PowerPC
2. Tiene mucha menos memoria RAM que el resto
3. A nivel de diseño, el Xeon tiene mucha menos memoria caché, al igual que no presenta HyperThreading (el POWER5+ tiene el SMT, un equivalente), entre otras muchas mejoras [\[link\]](#) y [\[link\]](#).
4. En pocas palabras, el uso del Xeon de Intel está orientado a Estaciones de Trabajo y Servidores y el POWER5+ de IBM es un CPU para Mainframes.

Aun así, hemos escogido ese porque no había ningún CPU de POWER5+ con 8 núcleos y el Xeon de Intel era el que más se le acercaba.

Para visualizar bien el ranking nos apoyamos de un gráfico:



Podemos observar, salvo por el caso de los 8 núcleos, cómo el mismo procesador (el POWER5+ de IBM), al ir incrementando en cantidad de núcleos, su rendimiento se va casi duplicando, pero el cambio es un pelín menos significativo cuando se trata de operaciones en punto flotante. Tomando como referencia la prueba de enteros, el procesador de 16 cores, que tiene 8 veces más que el de 2, obtiene una puntuación 8 veces mayor; para el caso de punto flotante, solo llega a 7'7 veces más. Sin embargo, en valor absoluto, el incremento de rendimiento para punto flotante es abismal (para enteros hay una diferencia de un 315%, pero para la de punto flotante es de 491%).

Es importante hacer el apunte de que, teóricamente, a cuantos más núcleos tenga el CPU, más rendimiento le vas a sacar; sin embargo, esto no es cierto al 100%, ya que debemos de preocuparnos de aumentar y mejorar otras características como puede ser el ancho de banda [\[link\]](#) o también de que los programas estén bien optimizados, pues la mayoría no están preparados para ser ejecutados paralelamente (por eso la mayoría de PC's del mercado no tienen más de 4 núcleos; porque no hace falta). Hoy en día, sobre todo en el sector móvil, se apuesta por una arquitectura de CPU multi-proceso heterogénea, HMP (como la big.LITTLE [\[link\]](#)), la cual quiere decir que se dispone de muchos cores, pero agrupados según sus características en de bajo consumo, de alto rendimiento o incluso un punto medio. Nunca trabajan todos al mismo tiempo ya que esa no es la intención, sino reducir el consumo al máximo sin sacrificar el rendimiento.

## Evaluación de los miembros del equipo

Para esta práctica fui elegido como director del equipo y desde el primer día he intentado mantener un orden sobre mis compañeros. Consideré la parte grupal como un ejercicio que constaba de tres partes: recogida de datos y realización de la memoria, implementación del Benchmark reducido y presentación en clase. Estas tareas se las he asignado a María Rico, Eddie Rodríguez conmigo y César Sarrión con Miguel Sánchez, respectivamente. Todos han cumplido con mis instrucciones y han hecho las cosas tal y como yo les he pedido que las hicieran, y cuando no creían que estaba acertando en mis decisiones me lo han comentado.

Es cierto que César nos estuvo atrasando un poco hasta el penúltimo día antes de la entrega, pero ha cumplido con el plazo y eso es lo que cuenta. Así que, como estoy contento por el trabajo que han realizado, por mi parte todos tienen un 10.