

Estudio hecho en la sesión anterior:

a) Generar Imagen Gris

```

O(1)
for i=0; i < anchura; i++
    for j=0; j < altura; j++
        O(1)
        {
            O(x)+O(y) {
                O(x) arg = getGrayScale(input, getRGB(i, j)) ← suponiendo O(1) coste
                {
                    for k=0; k < 4; k++
                        O(1)
                } se reduce a O(1), siempre se hace un
                    número de veces independiente de la
                    entrada
                O(y) input.setRGB(i, j, gray) ← suponiendo coste y
            }
            O(z)+O(a) {
                O(z) File imagen2 = new File("Imagen2"); ← suponiendo coste z
                O(a) ImageIO.write(input, "png", imagen2); ← suponiendo coste a
            }
        }
    
```

coste total = $O(\text{altura} \times \text{anchura} \times (O(x) + O(y)) + O(z) + O(a))$

Si suponemos $O(x) = O(y) = O(z) = O(a) = O(1)$, el coste sería:

$O(\text{altura} \times \text{anchura})$

b) Histograma Imagen

```

O(1)
for i=0; i < 255; i++
    O(1)
    {
        file imagen = new File("ruta"); O(a)
        BufferedImage input = ImageIO.read(imagen); O(b)
        O(1)
        {
            for i=0; i < anchura; i++
                for j=0; j < altura; j++
                    {
                        Histograma [getGrayScale(input, getRGB(i, j))]++; O(c) * O(d)
                    }
                O(1)
            }
        }
    
```

la que mas se ejecuta
(suponiendo $\text{anchura} \times \text{altura} > 255$)

coste total = $O(a) + O(b) + \text{anchura} \times \text{altura} \times (O(c) + O(d))$

suponiendo $O(a) = O(b) = O(c) = O(d) = O(1)$, el coste sería

$O(\text{anchura} \times \text{altura})$

c) Imprime Histograma

```

for i=0; i < longitud; i++
    print(O(1))
    printLn()
    
```

suponiendo coste print = $O(1)$

coste total = $O(\text{longitud})$

d) Generar Imagen Ordenada Columnas

$O(1)$ las mismas suposiciones que antes

```

for i=0; i < anchura; i++
    {
        {
            for j=0; j < altura; j++
                O(1)
            } O(altura)
        }
        {
            if (O(1))
                burbuja ← O(altura^2) O(altura^2)
            else
                quickSort ← O(altura^2)
        }
    }
    
```

$O(\text{anchura} \times \text{altura}^2)$

```

for i=0; i < anchura; i++
    for j=0; j < altura; j++
        O(1) ← reducción (for k 4 iter) a O(1)
    
```

$O(1)$ ← suposiciones

total = $O(\text{anchura} \times \text{altura}^2)$

Se espera que los apartados a y b se comporten de manera muy parecida.
 Por otra parte, el apartado c debería tardar menos. La longitud del vector es constante para todas las fotos 256, aunque se consideró como variable. Esto significa que para todas las fotos debe tardar prácticamente lo mismo y debe ser muy poco.
 Respecto al apartado d, ya sea con el método de la burbuja o quicksort va a ser el que más tarde. La diferencia entre burbuja y quicksort es que burbuja siempre tiene la misma cantidad de operaciones, pero las de quicksort pueden variar dependiendo de si está ordenado o no y cuánto lo esté, por lo que debería terminar antes.

Ordenador 1

Nanosegundos	a	b	c	d.0	d.1
320x214	11193116383508	11193214214648	11193233718529	11193434602934	1.11935E+13
640x360	11193812601580	11194004088819	11194018328865	11194291756550	1.11945E+13
640x427	11194746931973	11194906908436	11194916027238	11195473464743	1.11957E+13
1024x1024	11196485874695	11197024632466	11197028378938	11199772630903	1.12007E+13
1536x1536	11202280445728	11203399492224	11203401705369	11211626096046	1.12135E+13

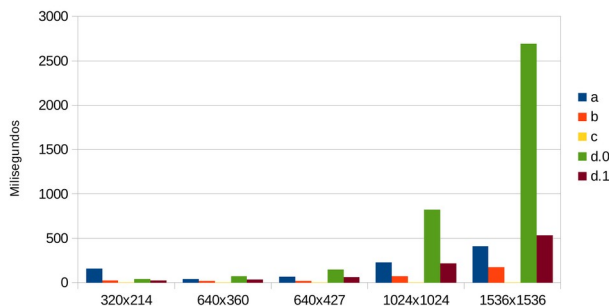
Milisegundos	a	b	c	d.0	d.1	
320x214		564	59	27	281	143
640x360		289	87	17	290	207
640x427		280	84	9	551	230
1024x1024		829	321	4	2855	863
1536x1536		1757	738	2	8556	1876

Ordenador 2

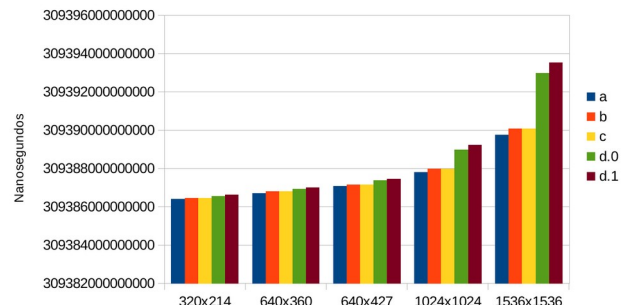
Nanosegundos	a	b	c	d.0	d.1
320x214	3.09386411E+14	3.09386455E+14	3.09386466E+14	3.09386571E+14	3.09387E+14
640x360	3.09386721E+14	3.09386817E+14	3.0938682E+14	3.09386938E+14	3.09387E+14
640x427	3.093871E+14	3.09387161E+14	3.09387164E+14	3.09387382E+14	3.09387E+14
1024x1024	3.09387807E+14	3.09387987E+14	3.09387988E+14	3.09388991E+14	3.09389E+14
1536x1536	3.09389764E+14	3.0939009E+14	3.09390091E+14	3.09392982E+14	3.09394E+14

Milisegundos	a	b	c	d.0	d.1	
320x214		156	22	2	41	24
640x360		38	19	2	73	37
640x427		65	20	3	146	60
1024x1024		229	70	1	824	219
1536x1536		412	174	1	2693	532

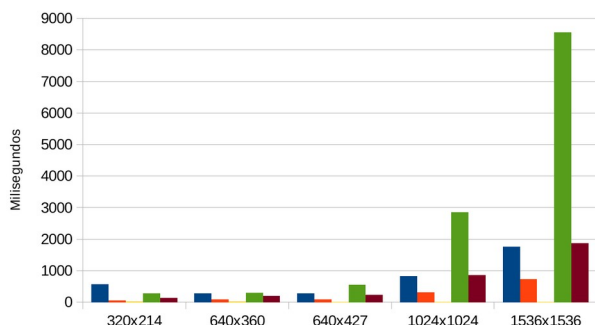
Ordenador 2



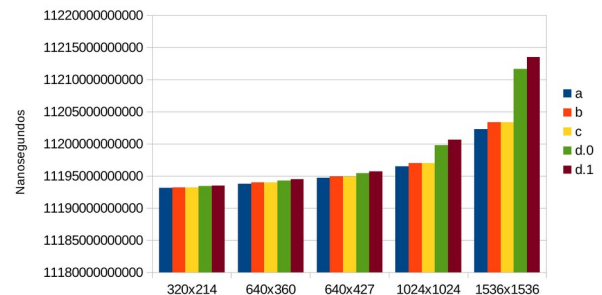
Ordenador 2



Ordenador 1



Ordenador 1



Los resultados obtenidos son los esperados al medir en milisegundos, aunque se observa una diferencia considerable entre los apartados a y b. Esta puede deberse al bucle de cuatro iteraciones dentro de los otros bucles. Es posible que las operaciones de guardar la imagen sumen un tiempo que no se consideraba previamente.

Como se había previsto, los dos últimos apartados, son los más intensos cuando las magnitudes crecen. También se observa muy claramente una diferencia entre usar un método de ordenado u otro llegando a hacer comparables el apartado a y el uso de quicksort aunque se esperasen que la complejidad del apartado a fuese menor.

Respecto al tiempo usando los milisegundos, aunque se ve una tendencia creciente atendiendo al tamaño de la entrada, los valores no son los esperados.

El tiempo de ejecución del apartado c crece, aunque se dijo que esto no ocurriría y se observa muy poca diferencia entre todos los apartados.

Podemos concluir que ha habido algún problema al programar el cálculo del tiempo en milisegundos.

No somos capaces de determinar este.