

## Actividad 3

Leonardo Flores Torres

20 de septiembre de 2022

**Programar un algoritmo de su elección (no tan sencillo) y analizarlo de la siguiente forma:**

1. Graficar el tiempo de ejecución en función de  $N$ ,
2. sobre los mismos ejes graficar 2 cotas superiores y dos cotas inferiores,
3. repetir el punto 1 y 2 ejecutando el programa en otra computadora de distinto desempeño,
4. analizar los resultados y discutirlos. Escribir de la manera más completa las características de las 2 computadoras.

El algoritmo que se eligió fue computar un fractal, más específicamente, el ...

El extracto de código que se muestra a continuación es una representación del modo de trabajo que se lleva en `julia` usando el REPL<sup>1</sup>, asemeja un ambiente de trabajo y ejecución de comandos en la terminal.

```
julia> ns = 10:10:400           # valores que puede tomar N
julia> reps = 10                # numero de repeticiones
julia> timings = zeros(length(ns), 2) # arreglo bidimensional
julia> for rep in 1:reps
    for (index, n) in enumerate(ns)
        time = @elapsed mf.fractalCMap(n, n, maxiter=100)
        if rep == 1
            timings[index, 1] = n
        end
        timings[index, 2] += time
    end
end
julia> timings[:,2] = timings[:,2] / reps # promedio de tiempo
```

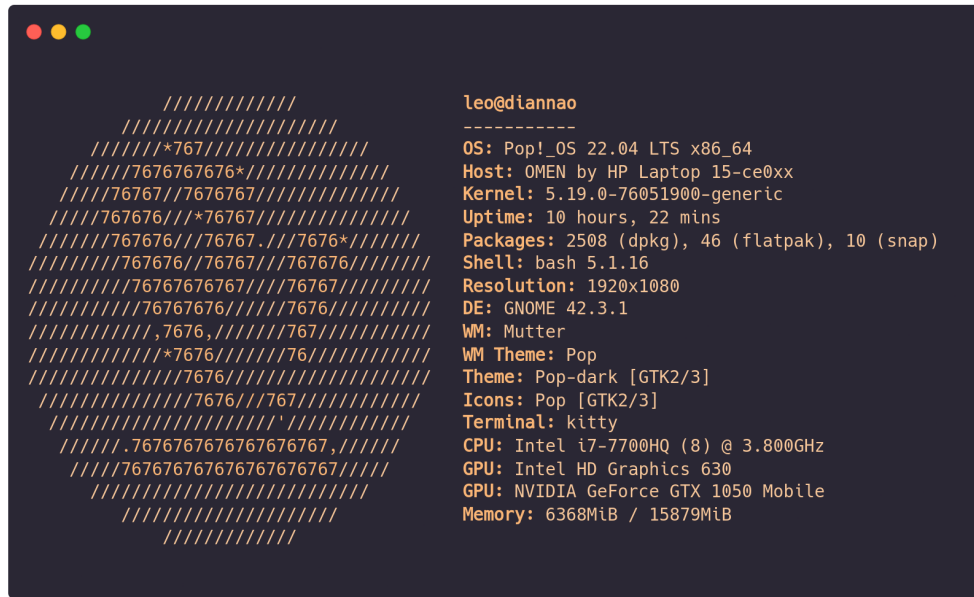
Para graficar el tiempo de ejecución se hicieron 10 repeticiones indicadas por `reps`, y se definió una variable `ns` para guardar el conjunto de valores que puede tomar  $N = 10, 20, 30, \dots, 400$ . La variable `timings` guarda en la primera columna el valor de  $N$ , mientras que en la segunda columna guarda el tiempo  $t(N)$  que le toma al algoritmo computar el fractal. Cada iteración del `loop` se suman los tiempos  $t(N)$  a sus respectivas entradas, y

---

<sup>1</sup>REPL es un acrónimo para Read-Eval-Print loop.

al final toda la columna de tiempos se divide entre la cantidad de repeticiones **reps** lo que resulta en tiempos promedio  $\bar{t}(N)$ .

Las características de las computadoras usadas, **diannao** y **hongdiannao**, se muestran en la figuras 1 y 2, respectivamente.

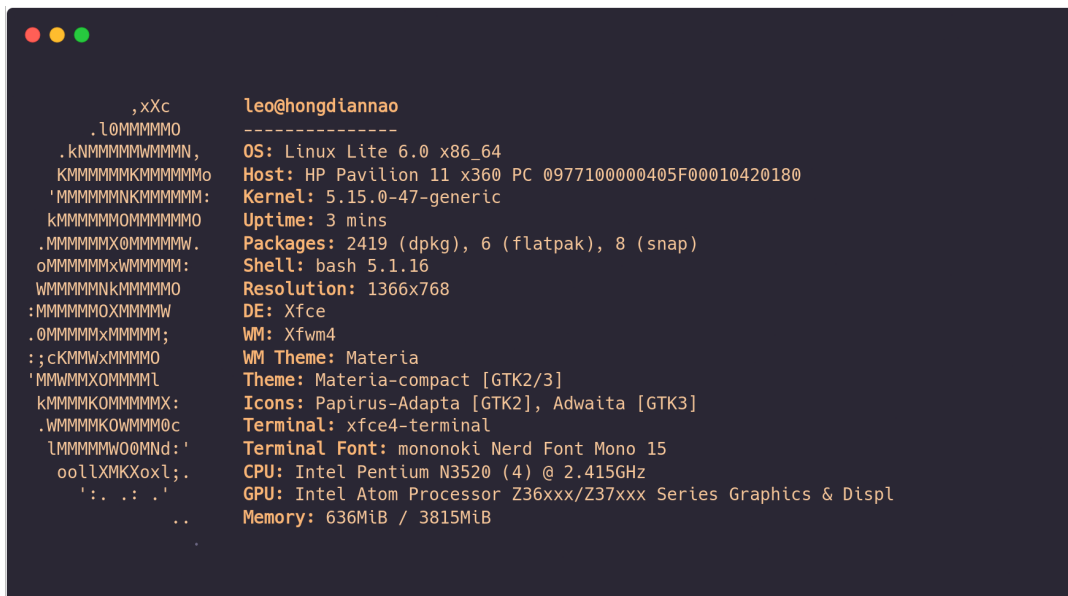


```

leopardiannao
-----
OS: Pop!_OS 22.04 LTS x86_64
Host: OMEN by HP Laptop 15-ce0xx
Kernel: 5.19.0-76051900-generic
Uptime: 10 hours, 22 mins
Packages: 2508 (dpkg), 46 (flatpak), 10 (snap)
Shell: bash 5.1.16
Resolution: 1920x1080
DE: GNOME 42.3.1
WM: Mutter
WM Theme: Pop
Theme: Pop-dark [GTK2/3]
Icons: Pop [GTK2/3]
Terminal: kitty
CPU: Intel i7-7700HQ (8) @ 3.800GHz
GPU: Intel HD Graphics 630
GPU: NVIDIA GeForce GTX 1050 Mobile
Memory: 6368MiB / 15879MiB

```

Figura 1: características del ordenador diannao.



```

leohongdiannao
-----
OS: Linux Lite 6.0 x86_64
Host: HP Pavilion 11 x360 PC 0977100000405F00010420180
Kernel: 5.15.0-47-generic
Uptime: 3 mins
Packages: 2419 (dpkg), 6 (flatpak), 8 (snap)
Shell: bash 5.1.16
Resolution: 1366x768
DE: Xfce
WM: Xfwm4
WM Theme: Materia
Theme: Materia-compact [GTK2/3]
Icons: Papyrus-Adapta [GTK2], Adwaita [GTK3]
Terminal: xfce4-terminal
Terminal Font: mononoki Nerd Font Mono 15
CPU: Intel Pentium N3520 (4) @ 2.415GHz
GPU: Intel Atom Processor Z36xxx/Z37xxx Series Graphics & Displ
Memory: 636MiB / 3815MiB

```

Figura 2: características del ordenador hongdiannao.

## Apéndice

```

14 module MandelbrotFractal
15
16 #####
17 Required libraries
18 #####
19
20 using Images
21 using Plots
22
23 #####
24 Image operations
25 #####
26
27 function imageSave(path, img)
28     save(path, img)
29 end
30
31 function visualize(rgbmap)
32     return plot(rgbmap, ticks=false)
33 end
34
35 #####
36 Fractal generation
37 #####
38
39 function mandelbrot(c, maxiter)
40     z = 0
41     n = 0
42     while abs(z) <= 2 && n < maxiter
43         z = z*z + c
44         n += 1
45     end
46     return n
47 end
48
49 function canvasRGB(height, width)
50     return zeros(RGB, height, width)
51 end
52
53 function canvasHSV(height, width)
54     return zeros(HSV, height, width)
55 end
56
57 xc(i, lx, nx) = lx * (2*i - 1) / nx
58
59 function fractalGrays(nx, ny; lx=2, ly=2, maxiter=80)
60     cv = canvasRGB(ny, nx)
61
62     for row in 1:ny
63         y = xc(row, ly, ny) - ly
64
65         for col in 1:nx

```

```

66         x = xc(col, lx, nx) - lx
67         c = x + y * 1im
68         m = mandelbrot(c, maxiter) / maxiter
69
70         pixel = 1 - m ▷ RGB
71         cv[row, col] = pixel
72     end
73 end
74 return cv
75 end
76
77 function fractalColors(nx, ny; lx=2, ly=2, maxiter=80)
78     cv = canvashsv(ny, nx)
79
80     for row in 1:ny
81         y = xc(row, ly, ny) - ly
82
83         for col in 1:nx
84             x = xc(col, lx, nx) - lx
85             c = x + y * 1im
86             m = mandelbrot(c, maxiter) / maxiter
87
88             hue = 360 * m           # H between 0 and 360 (color wheel)
89             sat = 0.85              # S between 0 and 1 (saturation)
90             val = m < 1 ? 1 : 0     # V between 0 and 1 (brightness)
91             cv[row, col] = HSV(hue, sat, val)
92         end
93     end
94     return cv
95 end
96
97 function fractalCMap(nx, ny; lx=2, ly=2, maxiter=80, cname="Oranges")
98     cv = canvasRGB(ny, nx)
99
100     cmap_divs = 200
101     cmap = colormap(cname, cmap_divs, logscale=true) ▷ reverse
102
103     for row in 1:ny
104         y = xc(row, ly, ny) - ly
105
106         for col in 1:nx
107             x = xc(col, lx, nx) - lx
108             c = x + y * 1im
109             m = mandelbrot(c, maxiter) / maxiter
110
111             # To select a color of the colormap
112             cv[row, col] = cmap[ ceil(m * cmap_divs) ▷ Int ]
113         end
114     end
115     return cv
116 end
117
118 end # module MandelbrotFractal

```