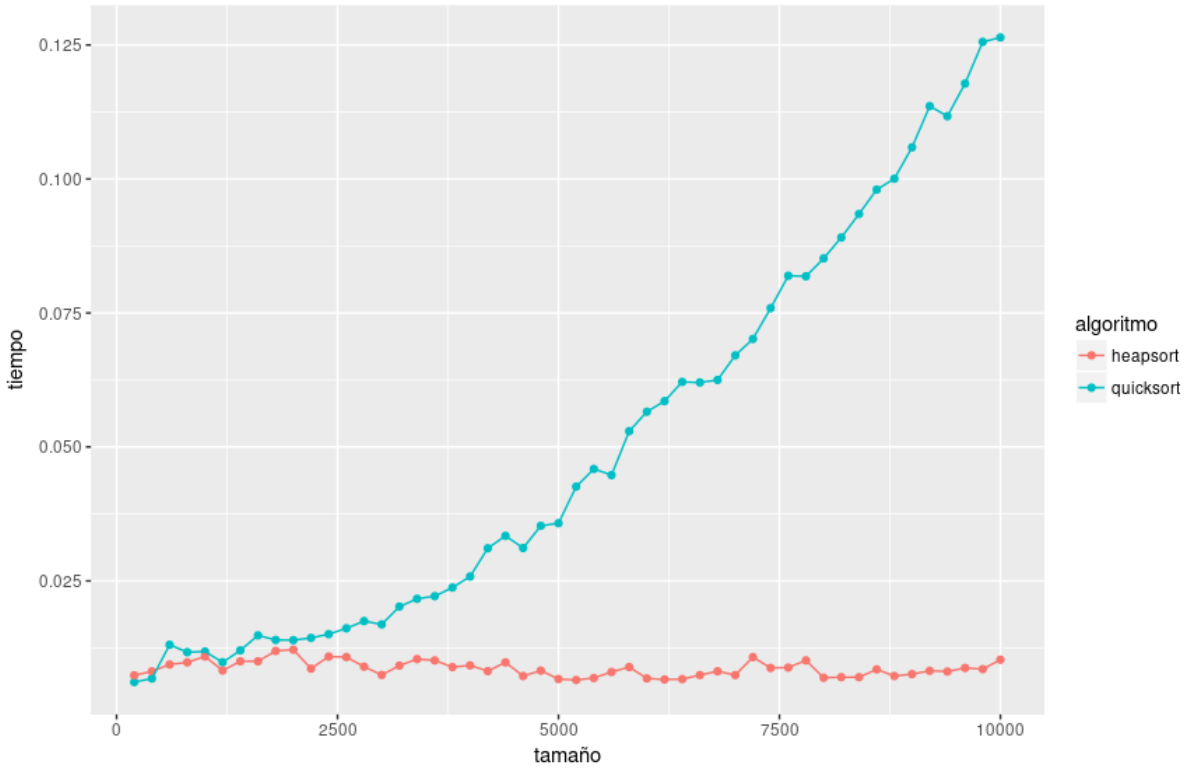
ALGORÍTMICA

PRÁCTICA 1: EFICIENCIA



AUTORES:
Pablo Moreno Megías, Diego Lerena García, Manuel Vallejo Felipe, Ángel Díaz de la Torre, Francisco Navarro Morales, Marcel Kemp Muñoz y David Redondo Correa

Algorítmica [PRACTICAS]
Segundo curso del Grado de Ingeniería Informática.
Universidad de Granada.
curso 2016-2017.

Índice

1 Eficiencia empírica

1.1. Algoritmos de orden $nlogn$.

En primer lugar realizamos el análisis empírico de los algoritmos de ordenación de orden $nlogn$. Obtuvimos, con un script, para cada algoritmo los tiempos de ejecución para distintos tamaños y reflejamos los datos obtenidos en gráficas.

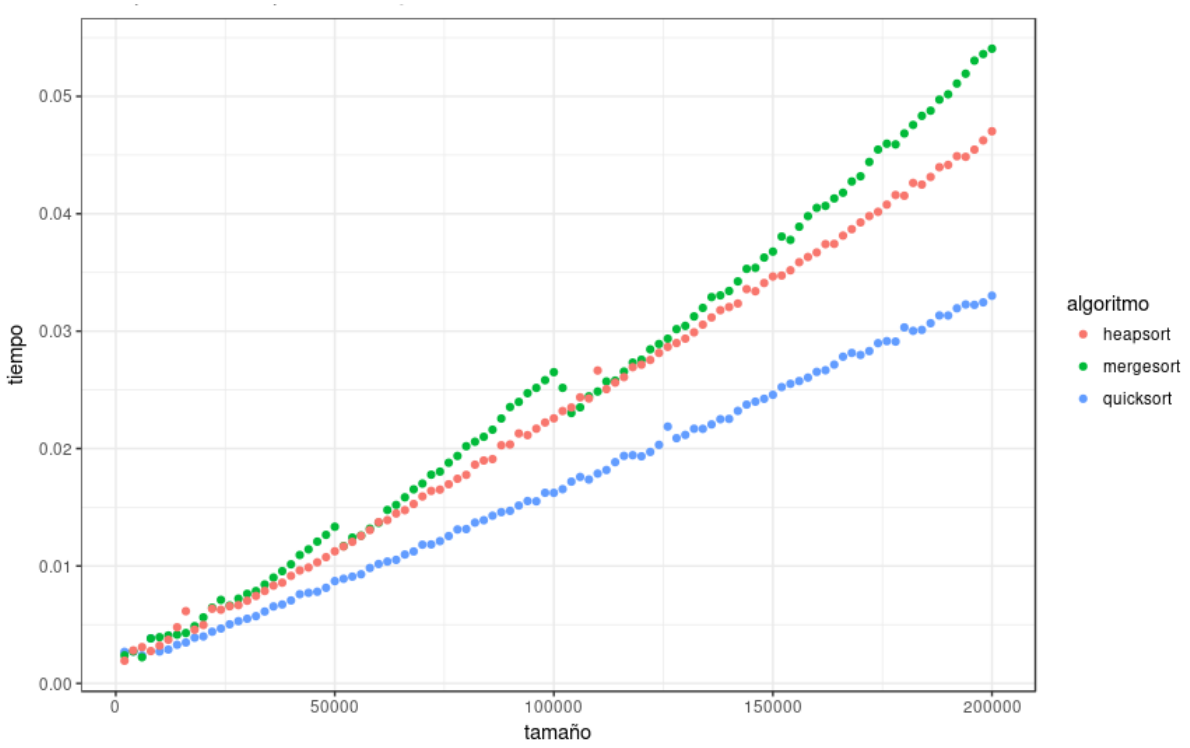


Figura 1: Eficiencia empírica de los algoritmos de ordenación de orden $nlogn$.

Podemos ver como aunque los 3 algoritmos tengan el mismo análisis O, el algoritmo de inserción es más eficiente que los demás y el burbuja el que menos, aunque esto a una gran cantidad de datos se hace insignificante.

1.2. Algoritmos de orden n^2.

Del mismo modo, obtuvimos los tiempos empíricos para los algoritmos de ordenación de orden n^2 y obtuvimos mejores resultados para el algoritmo de selección.

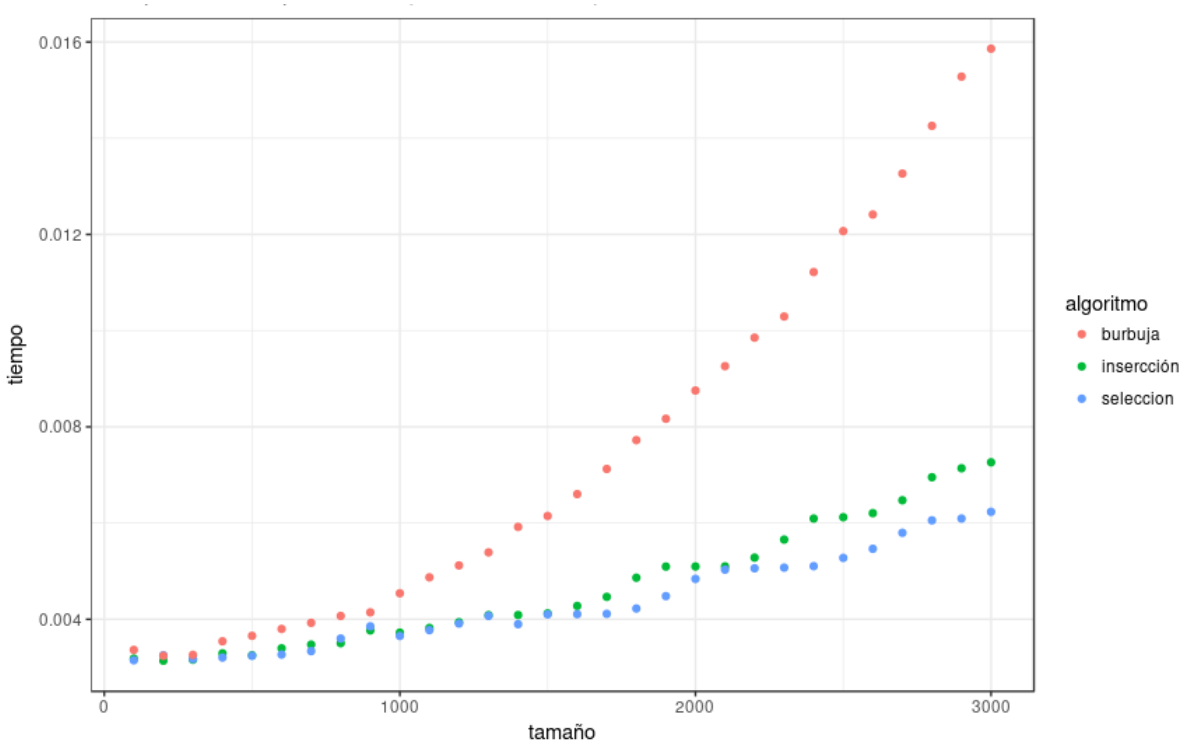


Figura 2: Eficiencia empírica de los algoritmos de ordenación de orden n^2.

1.3. Algoritmo de orden n^3.

El único algoritmo de este orden era el de Floyd. Los tiempos obtenidos se reflejan en la siguiente gráfica:

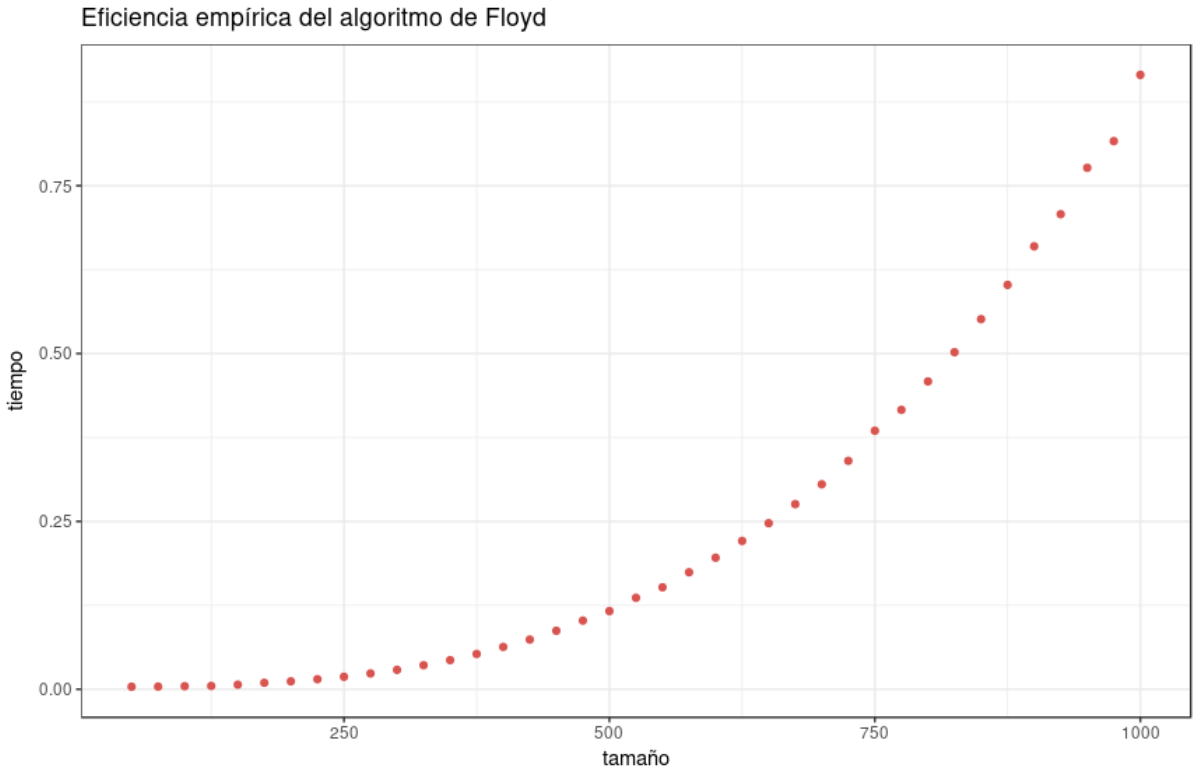


Figura 3: Eficiencia empírica del algoritmo de Floyd.

Para tomar estos tiempos tuvimos que reducir un poco los tamaños en comparación con los algoritmos de búsqueda.

1.4. Algoritmo de orden 2^n.

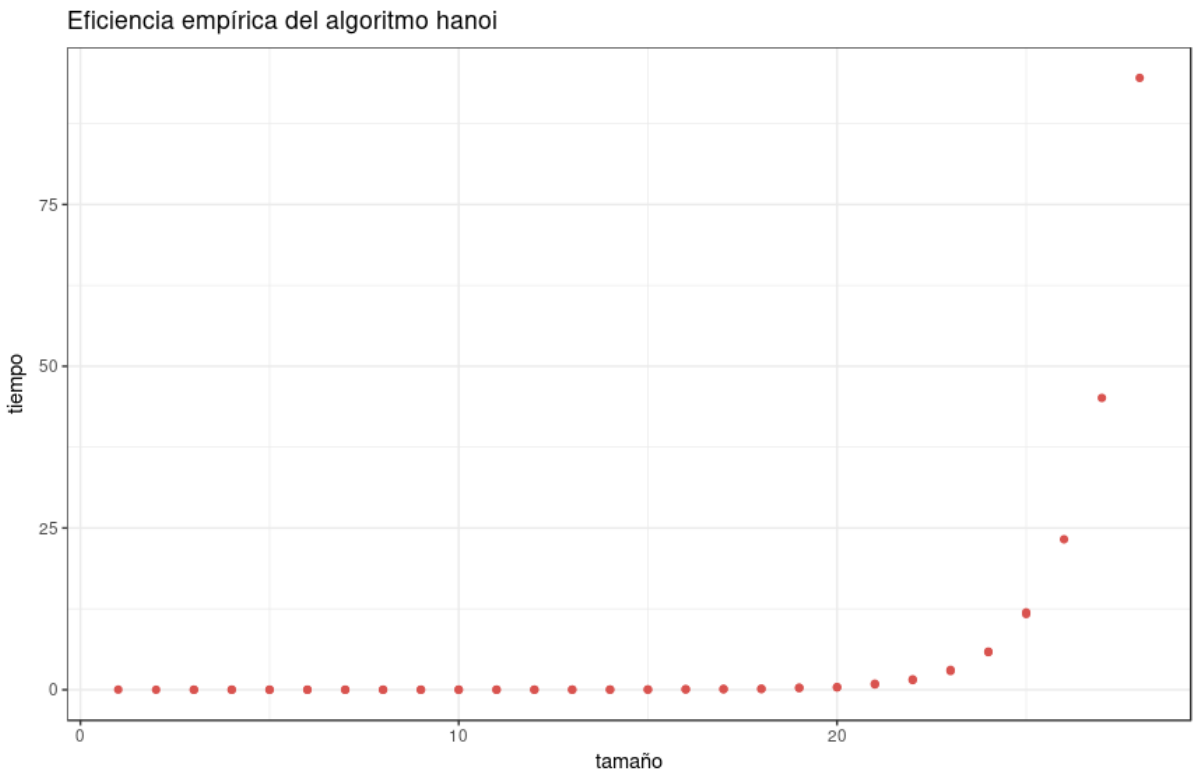


Figura 4: Eficiencia empírica del algoritmo de Hanoi.

EL principal problema que encontramos con este algoritmo fue que, por su orden de eficiencia, era imposible ejecutarlo para tamaños superiores a 50 elementos en tiempos razonables, por eso tomamos tiempos hasta 25 elementos. Para llegar a esta conclusión fuimos intentando ejecutar tamaños de 100, 90, 80... elementos hasta que vimos que con 20 se ejecutaba (tras unos minutos).

2 Eficiencia híbrida.

Hemos realizados los distintos ajustes de los tiempos anteriormente comentados con gnuplot tal y como se indica en el guión y hemos guardado algunas capturas de pantalla de los resultados obtenidos. Las gráficas obtenidas son las siguientes:

2.1. Algoritmos de orden $nlogn$.

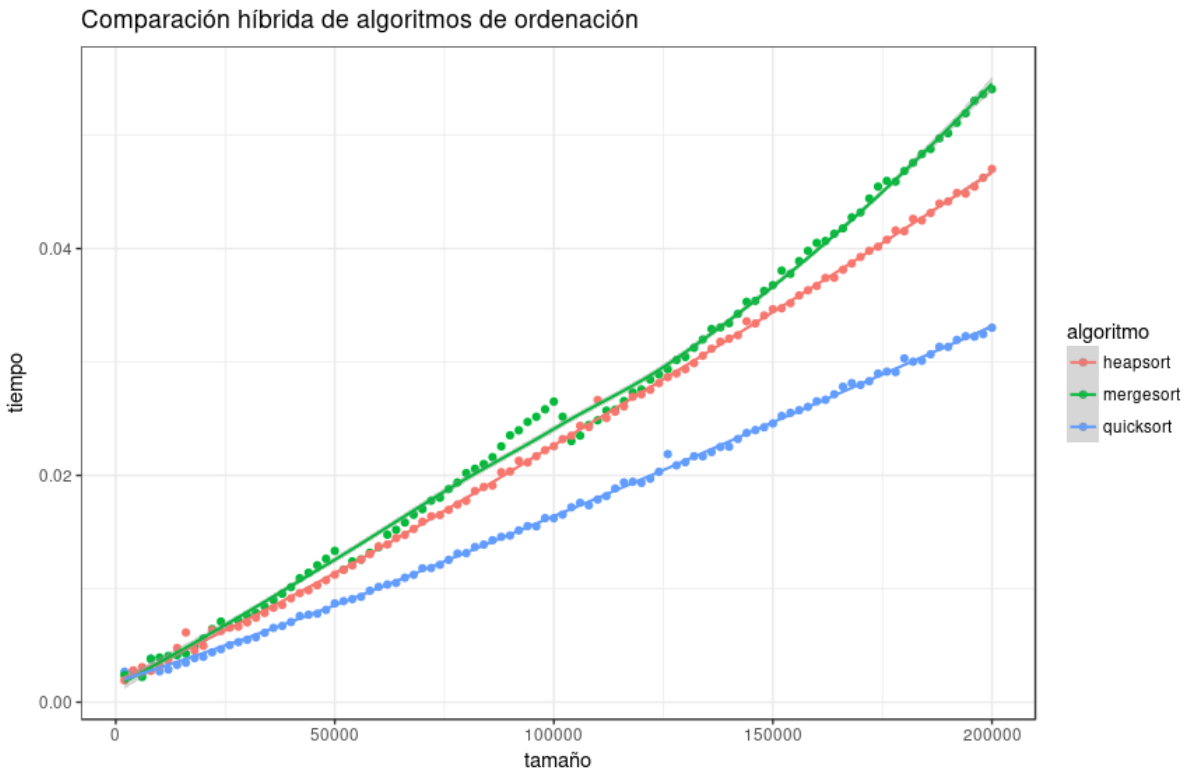


Figura 5: Eficiencia híbrida de los algoritmos de ordenación de orden $nlogn$.

Aquí podemos ver como aunque los algoritmos tengan el mismo análisis O, no se ajustan a la misma gráfica, sino que hay algoritmos más eficientes que otros. Hemos ajustado los tiempos a la función $f(n) = a * nlog(b * n) + c$. Los datos del ajuste de los 3 algoritmos son los siguientes:

data.txt

```
###############MERGESORT####################
Final set of parameters                 Asymptotic Standard Error
=======================                 ==========================

a                       = 4.34068e-05           +/- 2.013e-05        46.38%
b                       = 0.901865              +/- 3.606            399.8%
c                       = 0.00511096            +/- 0.01734          339.2%


correlation matrix of the fit parameters:

                       a          b          c
a                       1.000
```

```
b                  -0.999  1.000
c                   0.882 -0.896  1.000

##############QUICKSORT####################
Final set of parameters               Asymptotic Standard Error
=======================               ==========================

a                   = 3.02225e-09          +/- 1.214e-07        4018%
b                   = 0.901855             +/- 312.4            3.464e+04%
c                   = 0.00171908           +/- 0.0001046        6.083%


correlation matrix of the fit parameters:

                    a          b          c
a                   1.000
b                  -0.999  1.000
c                   0.882 -0.896  1.000

##############HEAPSORT####################
Final set of parameters               Asymptotic Standard Error
=======================               ==========================

a                   = 3.02225e-09          +/- 1.214e-07        4018%
b                   = 0.901855             +/- 312.4            3.464e+04%
c                   = 0.00171908           +/- 0.0001046        6.083%


correlation matrix of the fit parameters:

                    a          b          c
a                   1.000
b                  -0.999  1.000
c                   0.882 -0.896  1.000
```

Luego las funciones que se ajustan a los tiempos de los distintos algoritmos son:

- Heapsort: $f(n) = 4{,}34068 * 10^5 * nlog(0{,}901865 * n) + 0{,}00511096$
- Quicksort: $f(n) = 3{,}02225 * 10^9 * nlog(0{,}901855 * n) + 0{,}00171908$
- Mergesort: $f(n) = 3{,}02225 * 10^9 * nlog(0{,}901855 * n) + 0{,}00171908$

2.2. Algoritmos de orden n^2.

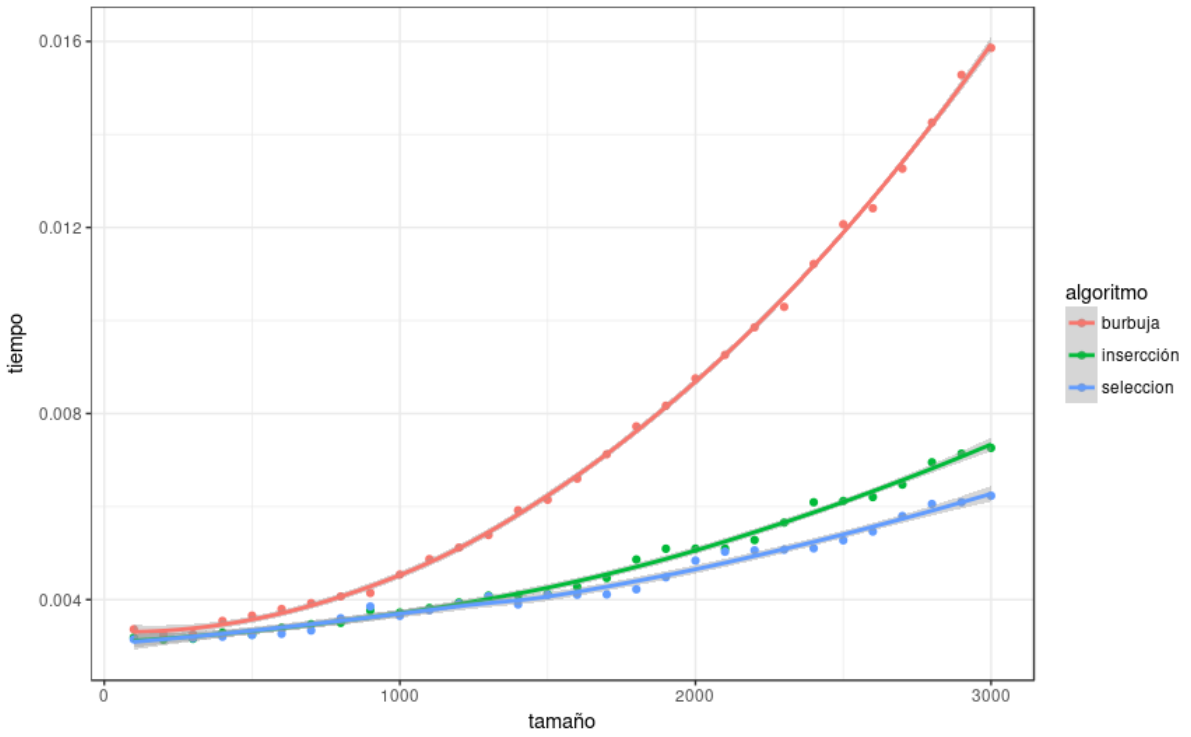


Figura 6: Eficiencia híbrida de los algoritmos de ordenación de orden n^2.

```
kemp@kemp-Satellite-Pro-C660: ~/Escritorio                    (26 %) 14:53

gnuplot> f(x)=a*x**2+b*x+c
gnuplot> fit f(x) 'burbuja.dat' via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 5.9591181566e-04   0.00e+00  8.08e-02    2.993484e-09  -4.207628e-06   1.190902e-02
   1 5.9591181566e-04   0.00e+00  8.08e-03    2.993484e-09  -4.207628e-06   1.190902e-02
iter      chisq       delta/lim  lambda   a             b             c

After 1 iterations the fit converged.
final sum of squares of residuals : 0.000595912
rel. change during last iteration : 0

degrees of freedom    (FIT_NDF)                      : 47
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00356075
variance of residuals (reduced chisquare) = WSSR/ndf   : 1.2679e-05

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 2.99348e-09      +/- 6.763e-11    (2.259%)
b               = -4.20763e-06     +/- 7.115e-07    (16.91%)
c               = 0.011909         +/- 0.001573     (13.21%)

correlation matrix of the fit parameters:
                a      b      c
a               1.000
b              -0.969  1.000
c               0.760 -0.876  1.000
gnuplot> fit f(x) 'seleccion.dat' via a,b,c
iter      chisq       delta/lim  lambda   a             b             c
   0 9.2814813504e-01   0.00e+00  8.08e-02    2.993484e-09  -4.207628e-06   1.190902e-02
   1 6.7719462087e-01  -3.71e+04  8.08e-03    1.004685e-09  -9.887614e-07   4.145080e-02
   2 5.2458438304e-01  -2.91e+04  8.08e-04    3.808691e-09  -4.223345e-05   1.814104e-01
   3 5.2129781530e-01  -6.30e+02  8.08e-05    4.784008e-09  -5.333305e-05   2.058119e-01
   4 5.2129779874e-01  -3.18e-03  8.08e-06    4.786255e-09  -5.335832e-05   2.058659e-01
iter      chisq       delta/lim  lambda   a             b             c

After 4 iterations the fit converged.
final sum of squares of residuals : 0.521298
rel. change during last iteration : -3.17816e-08

degrees of freedom    (FIT_NDF)                      : 47
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.105316
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0110914
```

Figura 7: Ajuste burbuja

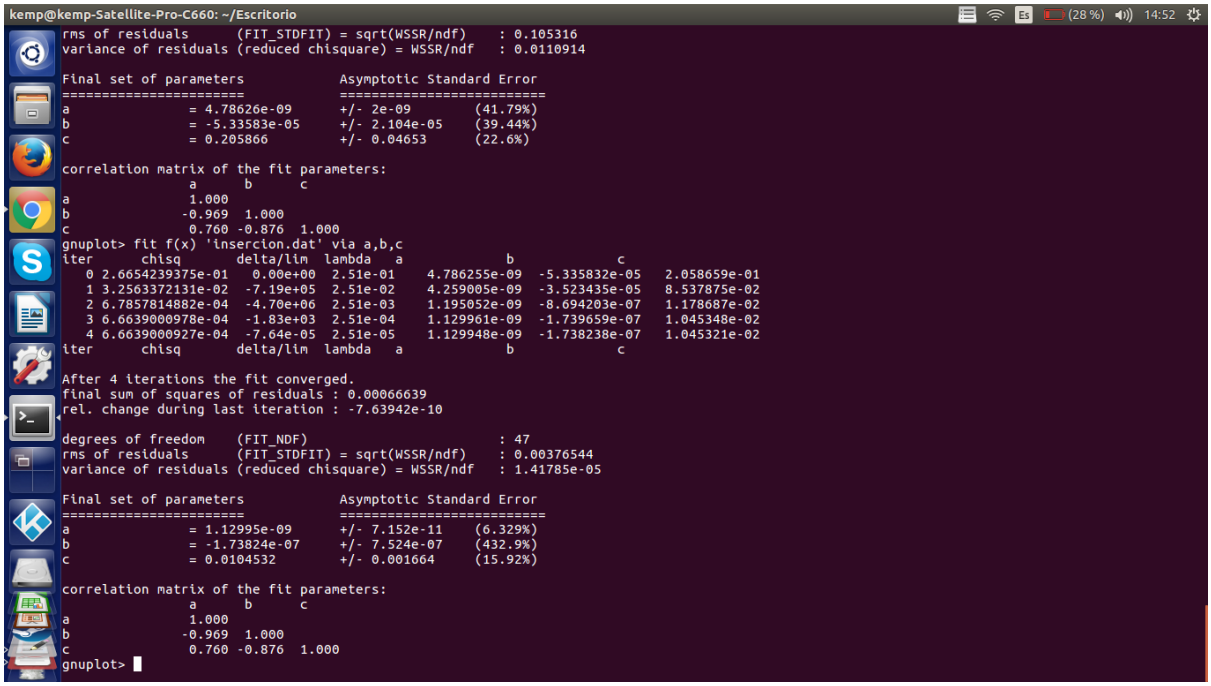


Figura 8: Ajuste inserción

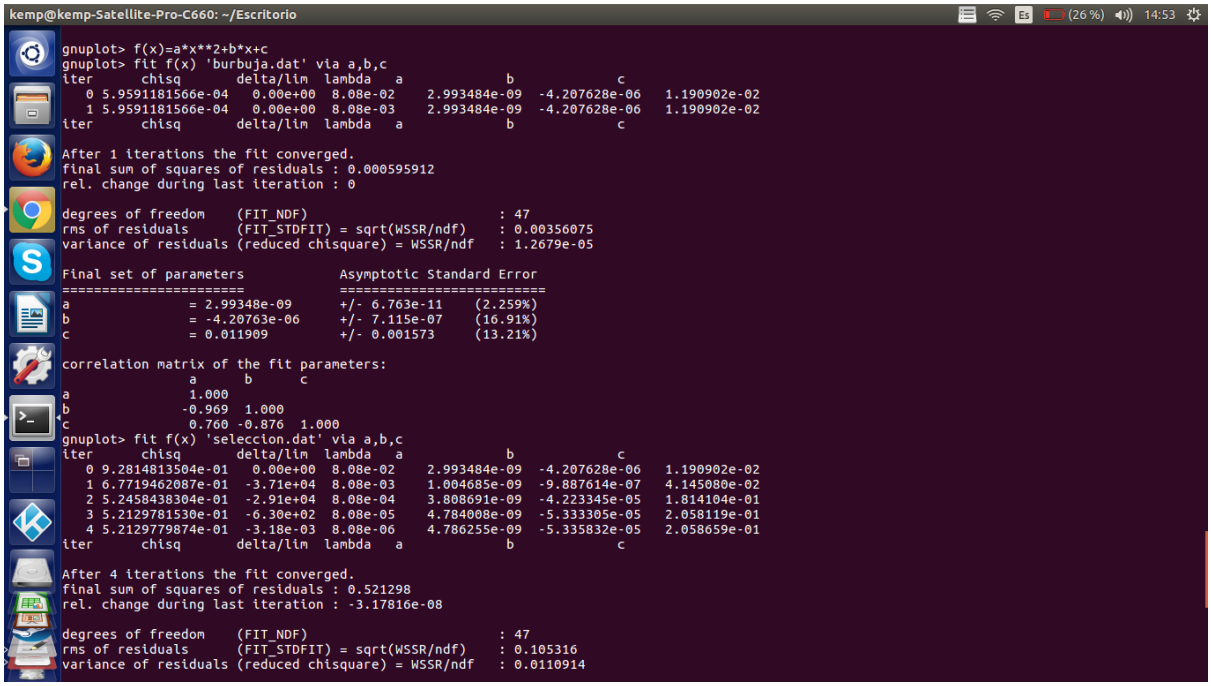


Figura 9: Ajuste selección

2.3. Algoritmo de orden n^3.

Aquí podemos ver perfectamente una parábola, que cuantos más datos más se parecerá a una parábola n^3. Los datos del ajuste están en la misma imagen:

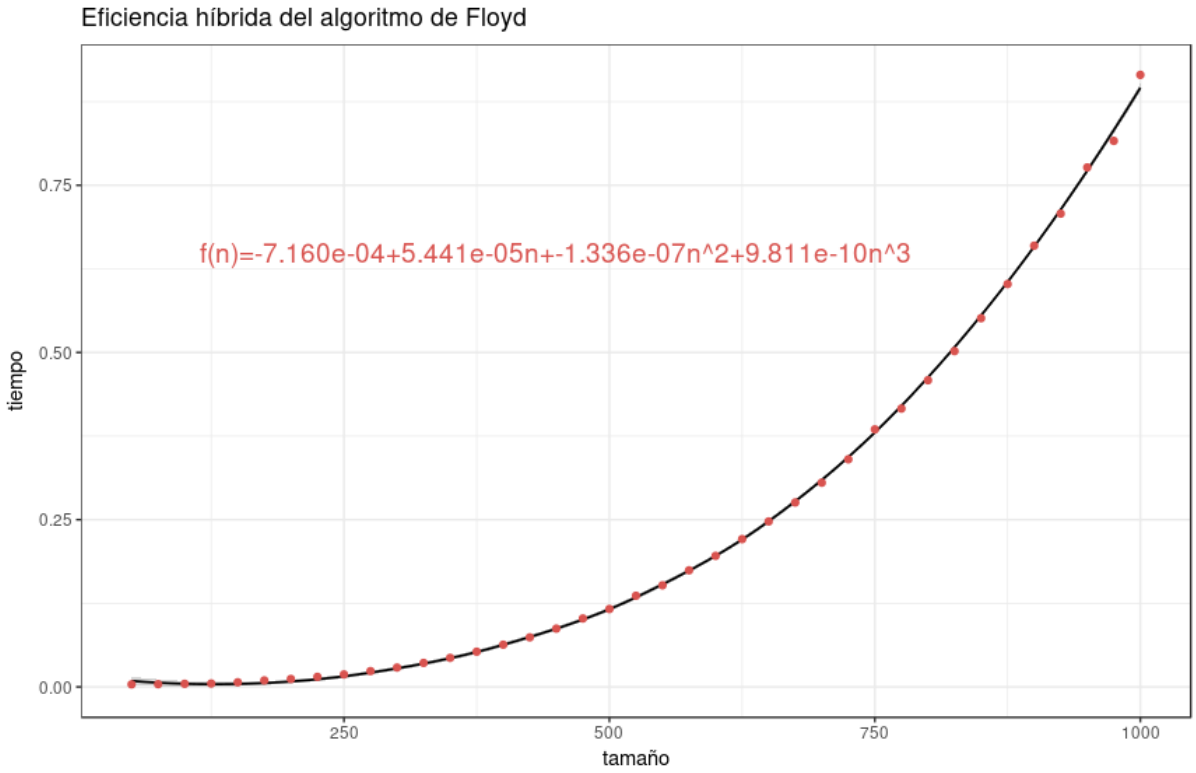


Figura 10: Eficiencia híbrida del algoritmo de Floyd.

2.4. Algoritmo de orden 2^n.

Podemos ver como sigue la filosofía de un algoritmo exponencial, a partir de un número de datos, en este caso 28, el algoritmo se hace prácticamente inútil, ya que el tiempo que tarda es demasiado. Los datos de la regresión son los siguientes:

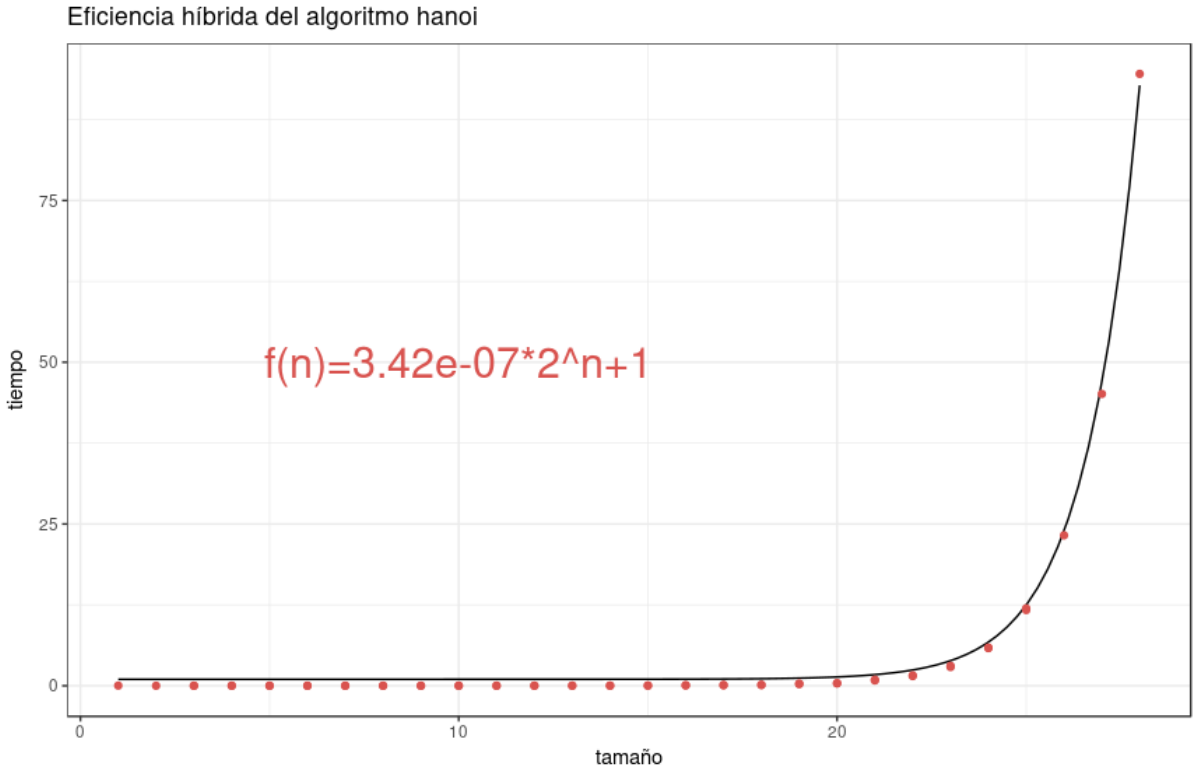


Figura 11: Eficiencia híbrida del algoritmo de las torres de Hanoihhanoi

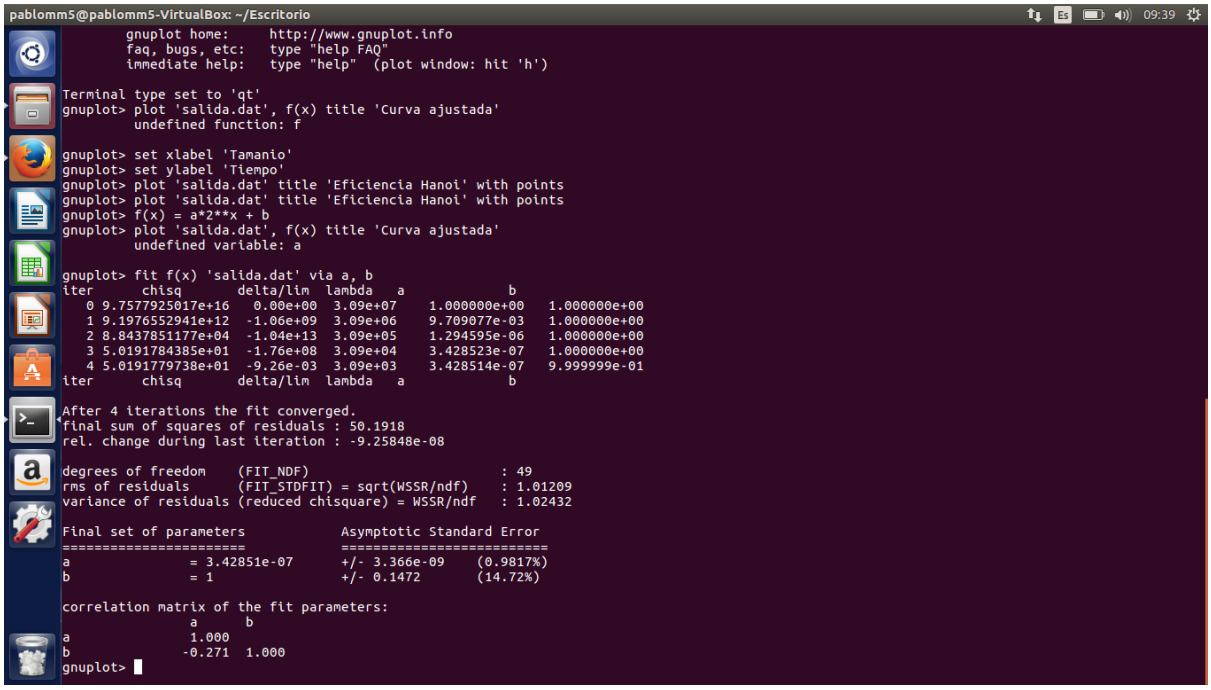


Figura 12: Ajuste Hanoi

3 Otros análisis

3.1. Burbuja con distintas optimizaciones.

En la siguiente gráfica se ve claramente como con diferentes optimizaciones del compilador la eficiencia cambia. Sin optimización, lógicamente, el algoritmo es menos eficiente. Con optimización 1, el algoritmo ya va tardando menos, sobre todo cuando el número de datos aumenta. Y finalmente, con optimización 2, la eficiencia mejora considerablemente, de manera que se empieza a notar aunque el número de datos no sea tan significativo.

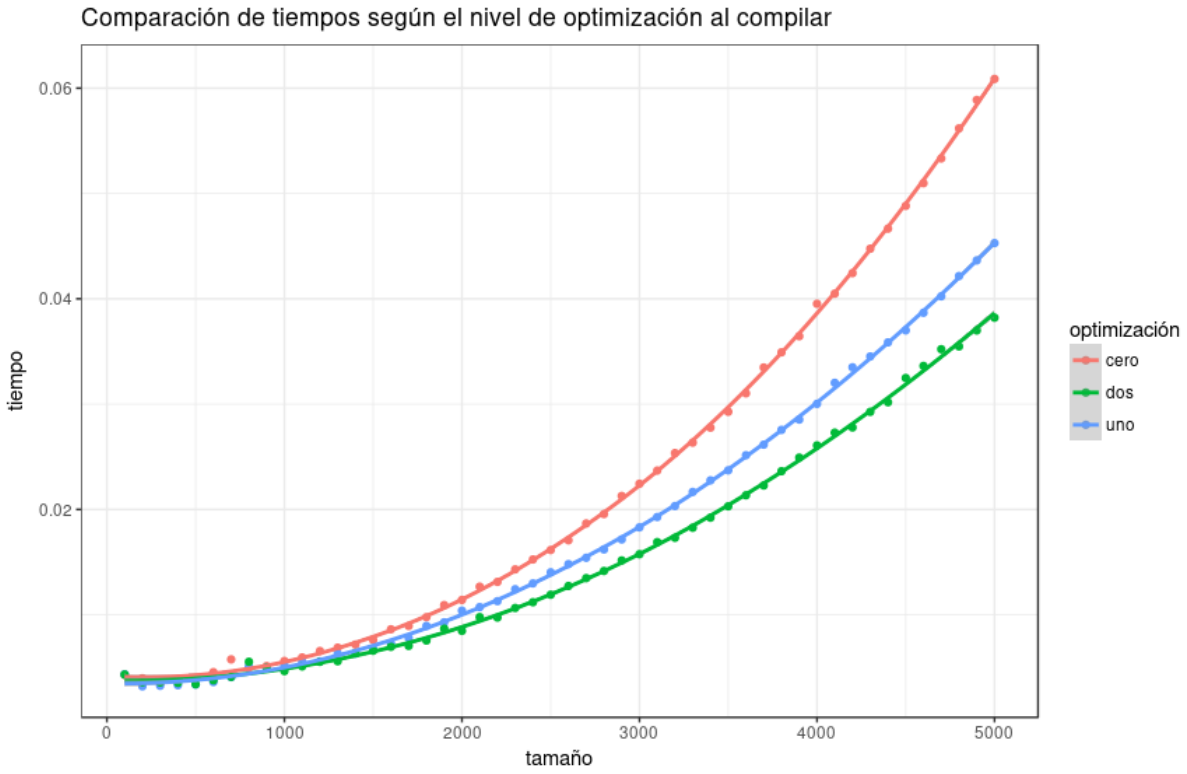


Figura 13: Comparación de ejecuciones del algoritmo burbuja según la optimización.

3.2. Comparación del Heapsort en distintos ordenadores.

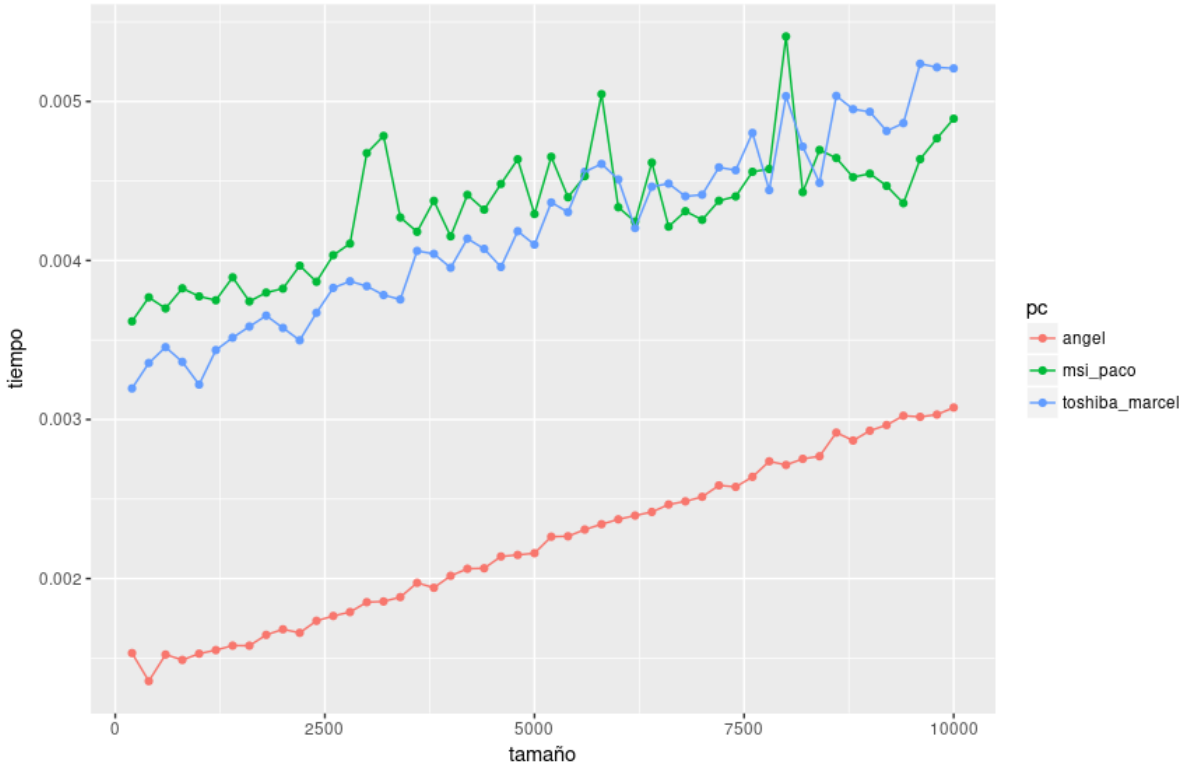


Figura 14: Comparación de ejecuciones del algoritmo heapsort con distintos ordenadores

Hemos probado a ejecutar el algoritmo heapsort en tres ordenadores distintos. Mientras que en el msi y el toshiba los tiempos eran muy similares, el tercer ordenador, un Lenovo i7, ha obtenido tiempos mucho más rápidos (ordenador etiquetado como ángel)

3.3. Mal ajuste de la función de Hanoi

Hemos comprobado qué ocurriría si hacemos un ajuste a los tiempos de un algoritmo con una función que no se corresponde a su eficiencia teórica, concretamente hemos ajustado el algoritmo de las torres de Hanoi con $f(x) = x + x^2$. El resultado:

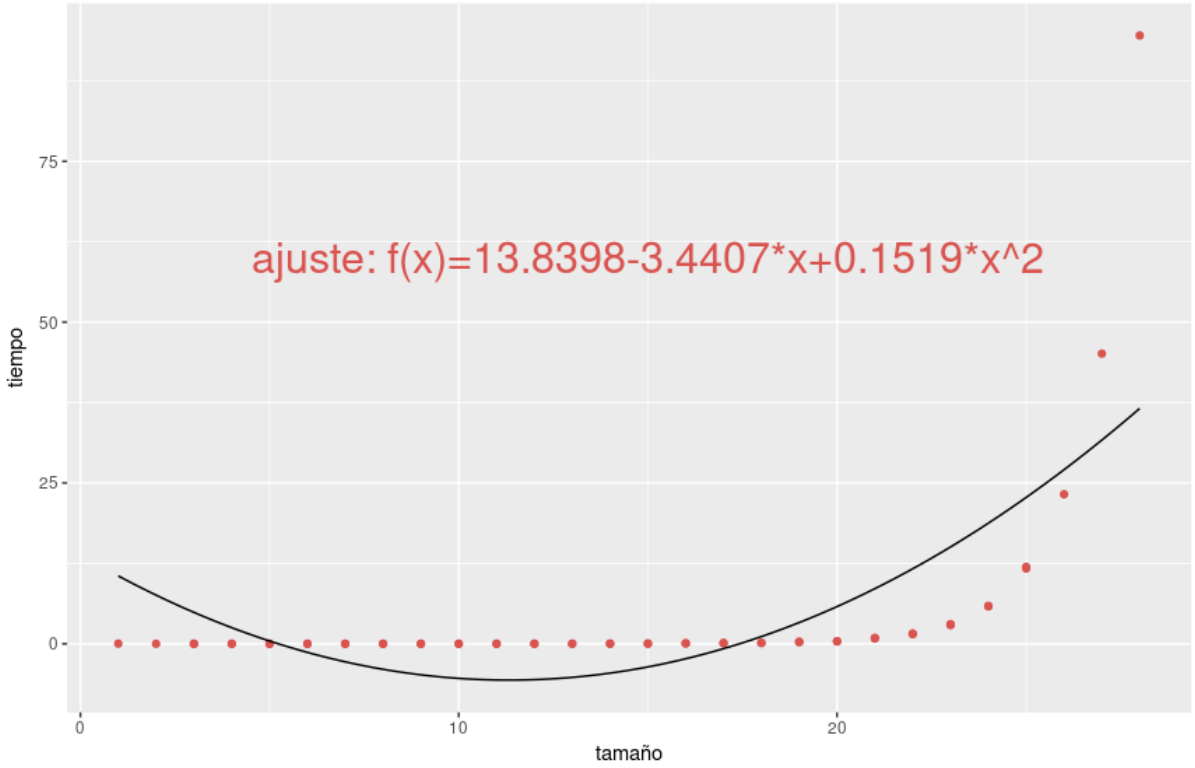


Figura 15: Mal ajuste de Hanoi

3.4. Comparación de Heapsort y Quicksort en el peor de los casos del Quicksort

Nos ha parecido interesante hacer una comparación especial de los algoritmos de ordenación quicksort y heapsort ya que, aunque el primero da mejores tiempos por lo general, hay casos (cuando el vector a ordenar está parcialmente ordenado) en los que los tiempos empeoran hasta el punto de llegar a obtener tiempos de orden n^2. No así el heapsort que, aunque da resultados generalmente peores que el quicksort, se mantiene en $nlogn$ sea cual sea el estado del vector.

Figura 16: Comparación del quicksort y el heapsort con vectores aleatoriamente ordenados

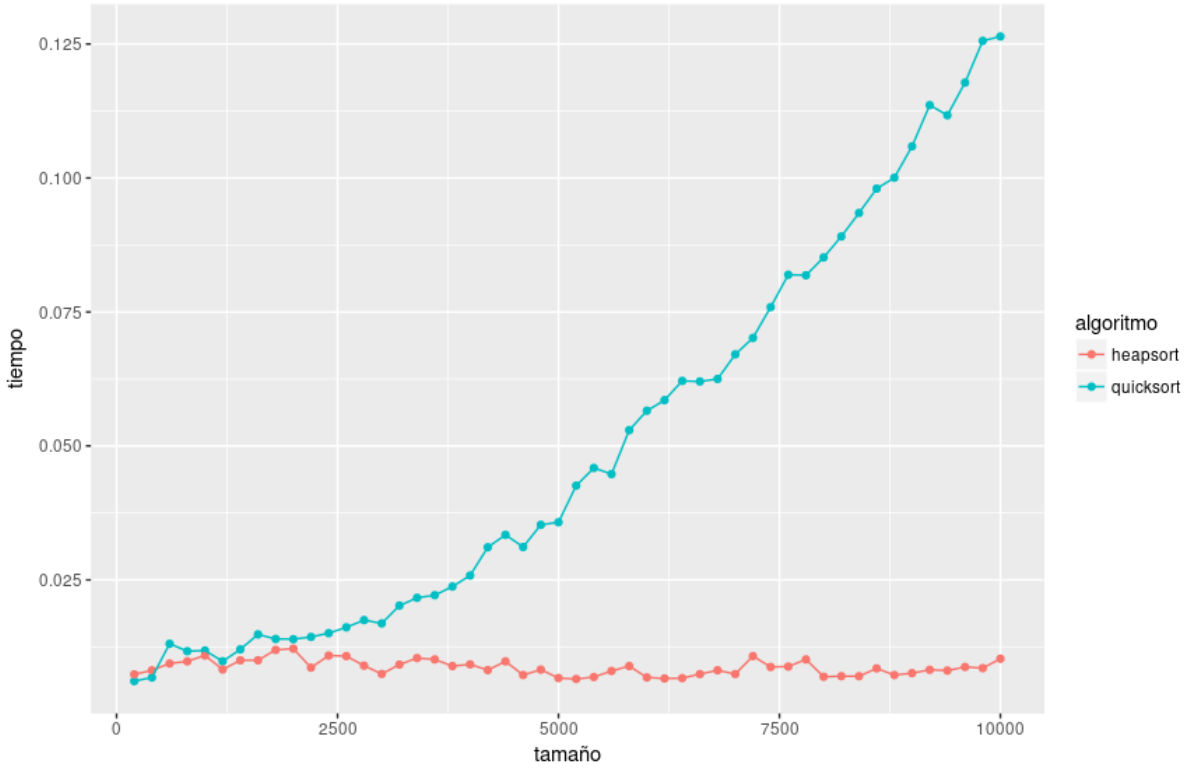


Figura 17: Comparación de quicksort y el heapsort con el vector parcialmente ordenado (u ordenado)

4 Ordenador usado para medir los tiempos

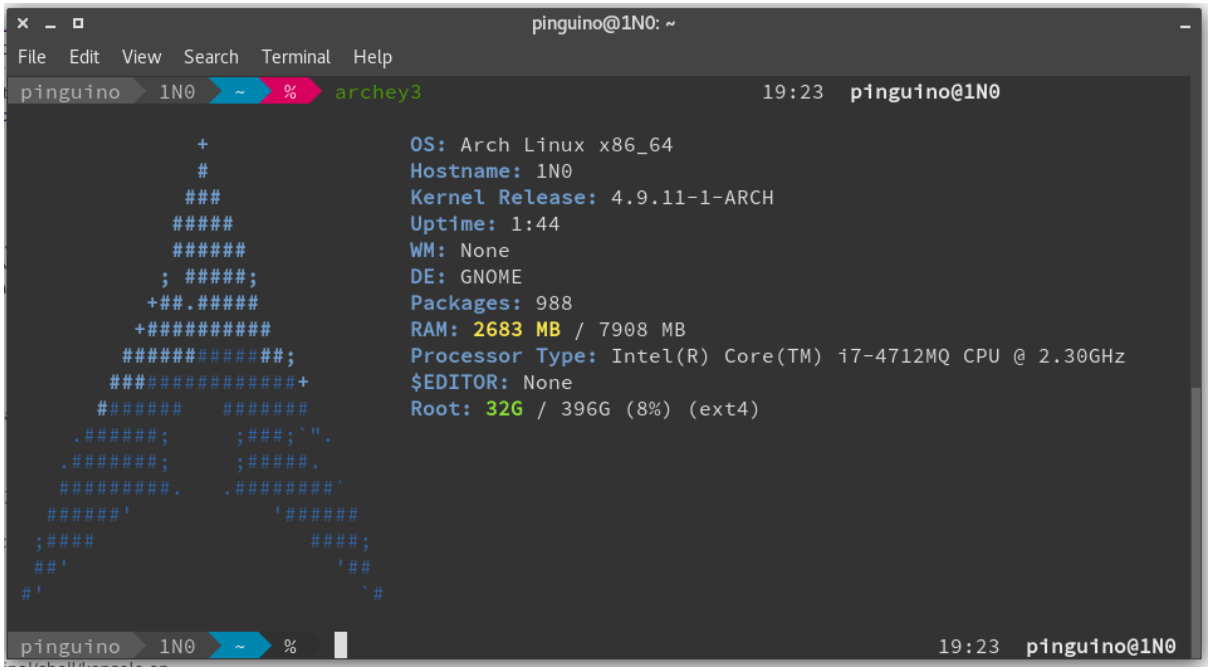


Figura 18: ARCH LINUX