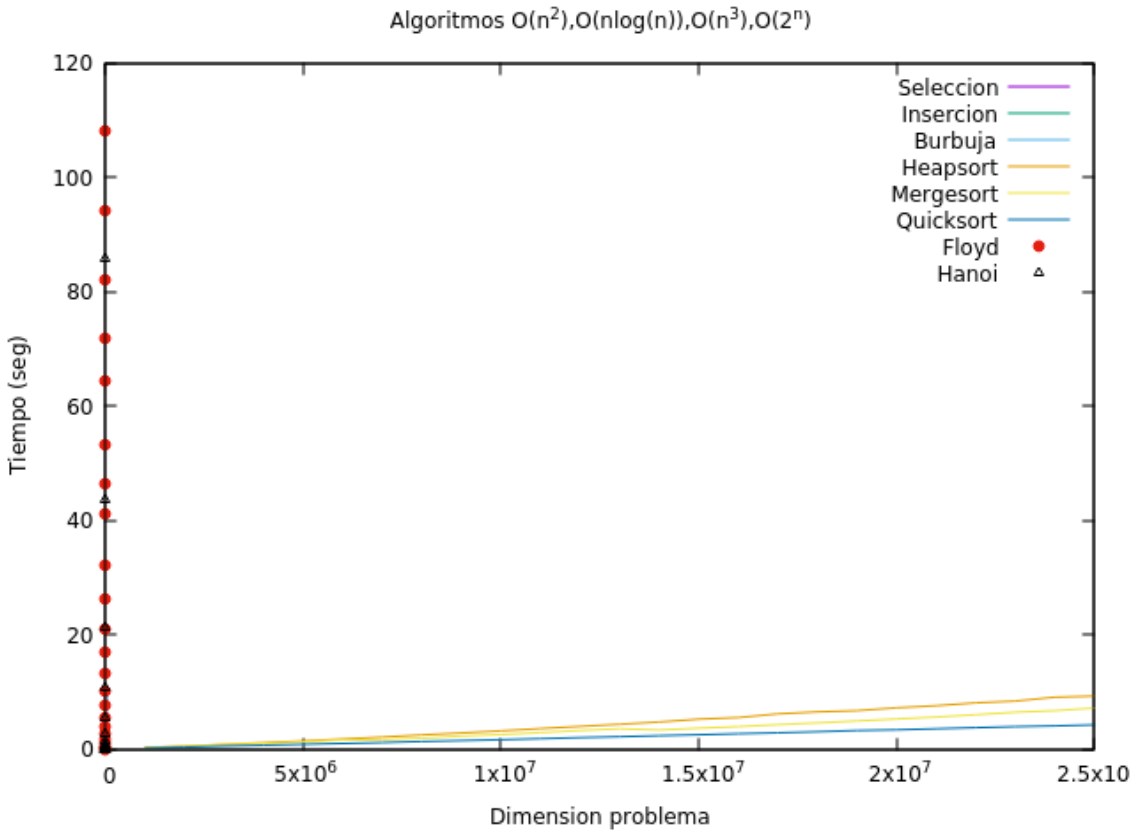
ALGORÍTMICA UGR (2019-2020)

Práctica 1:

Análisis de Eficiencia de Algoritmos



Trabajo realizado por: Jose Luis Pedraza Román

Contenidos:

1) ***Introducción***
2) ***Algoritmos de Ordenación O(n^2)***
3) ***Algoritmos de Ordenación O(nlog(n))***
4) ***Comparativa Algoritmos de Ordenación***
5) ***Comparativa Algoritmos de Ordenación con y sin Optimización***
6) ***Algoritmo de Floyd***
7) ***Algoritmo de las Torres de Hanoi***

1) Introducción:

En esta práctica se analizan diferentes algoritmos, estos son algunos algoritmos de ordenación, el algoritmo de Floyd y el algoritmo que resuelve el juego de las torres de Hanoi.

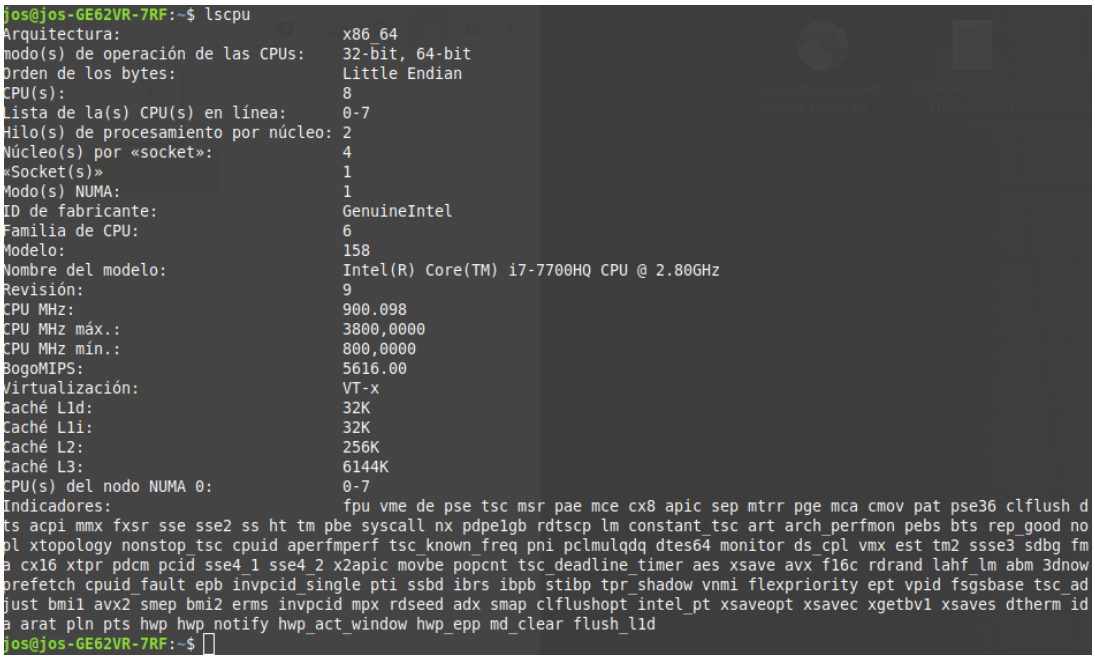
Para llevar acabo dicho estudio se han realizado los análisis empíricos e híbridos de cada uno de ellos.

En el análisis empírico se han hecho varias ejecuciones y calculado la media de los tiempos de ejecución (posteriormente explicados) para graficar los resultados y poder sacar conclusiones comparando algoritmos del mismo tipo y probando con diferentes opciones de compilación, como son –O2 y –O3 las cuales aplican un factor de optimización al código. También decir que se ha usado la librería “chrono” (incluir la opción de compilación –std=gnu++0x) para medir los tiempos de ejecución de cada algoritmo, incluyéndola dentro de los códigos y modificándolos sustancialmente para calcular la media de los tiempos de 3 ejecuciones para cada entrada del programa y para poder ejecutarlos a través de un script.sh y así automatizar las entradas del programa como se muestra a continuación:

```
#!/bin/bash

#./script.sh ejecutable liminf limsup intervalo
#./script.sh burbuja 1000 50000 2000
#el script genera un fichero .dat con los resultados y con
nombre salida(loquesea).dat para que este identificado


#                    $2          $3          $4
#   burbuja:        1000 a     63500 de     2500 en 2500
#   insercion:      1000 a     63500 de     2500 en 2500
#  seleccion:     1000 a     63500 de     2500 en 2500

#   mergesort:     1000000 a  26000000 de  1000000 en 1000000
#   quicksort:     1000000 a  26000000 de  1000000 en 1000000
#   heapsort:      1000000 a  26000000 de  1000000 en 1000000

#   floyd:          120 a      3000 de     120 en 120
#   hanoi:          15   a      40 de        1 en 1

echo "" >> salida$1.dat
let i=$2
while (($i < $3))
do
    ./$1 $i >> salida$1.dat
    let i+=$4
done
```

En el análisis híbrido se ha ajustado cada una de las gráficas que representan los tiempos de ejecución de los algoritmos para diferentes tamaños de problema con la función correspondiente a su orden de eficiencia.

También cabe destacar el equipo en donde se han realizado las pruebas:



```
jos@jos-GE62VR-7RF:~$ lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              8
Lista de la(s) CPU(s) en línea:      0-7
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:              4
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              158
Nombre del modelo:                   Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Revisión:                            9
CPU MHz:                             900.098
CPU MHz máx.:                        3800,0000
CPU MHz mín.:                        800,0000
BogoMIPS:                            5616.00
Virtualización:                      VT-x
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            6144K
CPU(s) del nodo NUMA 0:              0-7
Indicadores:                         fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush d
ts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good no
pl xtopology nonstop_tsc cpuid aperfmperf tsc_known_freq pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg fm
a cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnow
prefetch cpuid_fault epb invpcid_single pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_ad
just bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushopt intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm id
a arat pln pts hwp hwp_notify hwp_act_window hwp_epp md_clear flush_l1d
jos@jos-GE62VR-7RF:~$
```

Y que para las mismas no se ha estado realizando ninguna tarea a la vez, por lo que se ha aprovechado prácticamente en su totalidad la potencia del equipo para obtener unos resultados correctos.

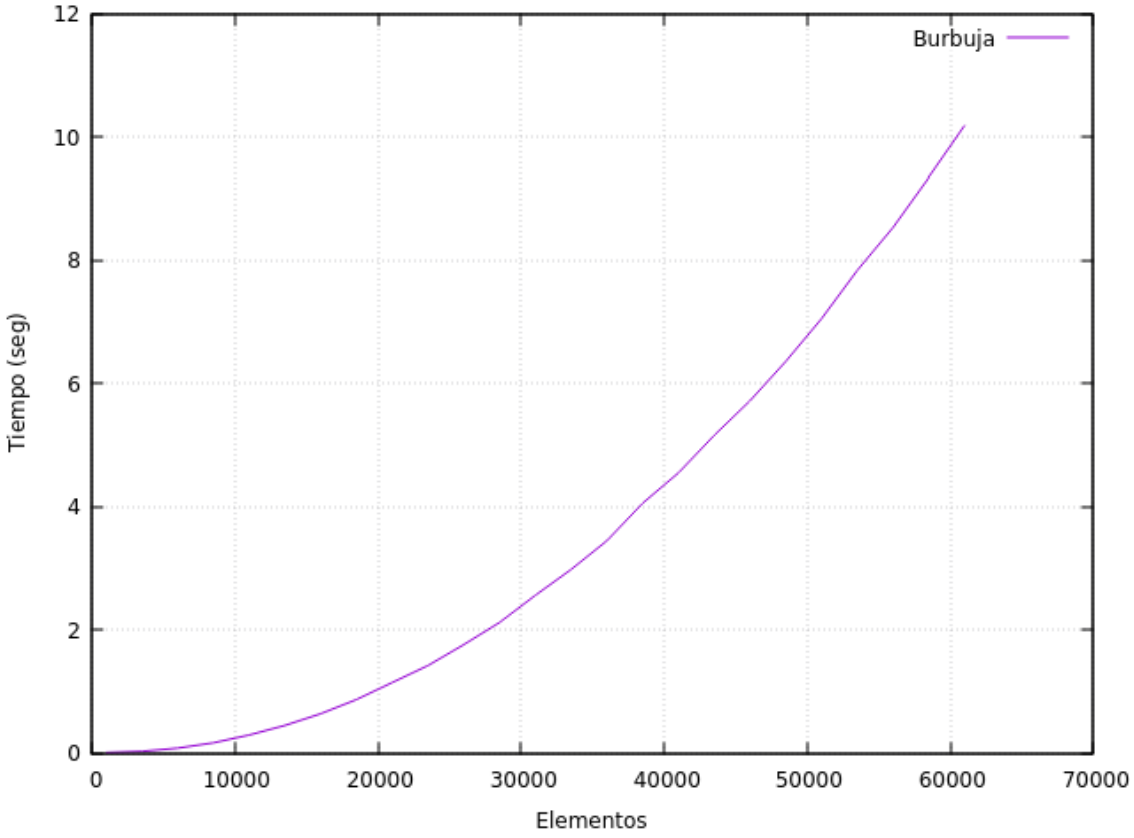
2) *Algoritmos de Ordenación de Orden O(n^2)*

Para este tipo de algoritmos con una eficiencia teórica de orden O(n^2) se han obtenido los siguientes tiempos de ejecución en función del tamaño de la entrada del problema, en este caso, el número de componentes del vector a ordenar comenzando en 1000, de 2500 en 2500, hasta 61000.

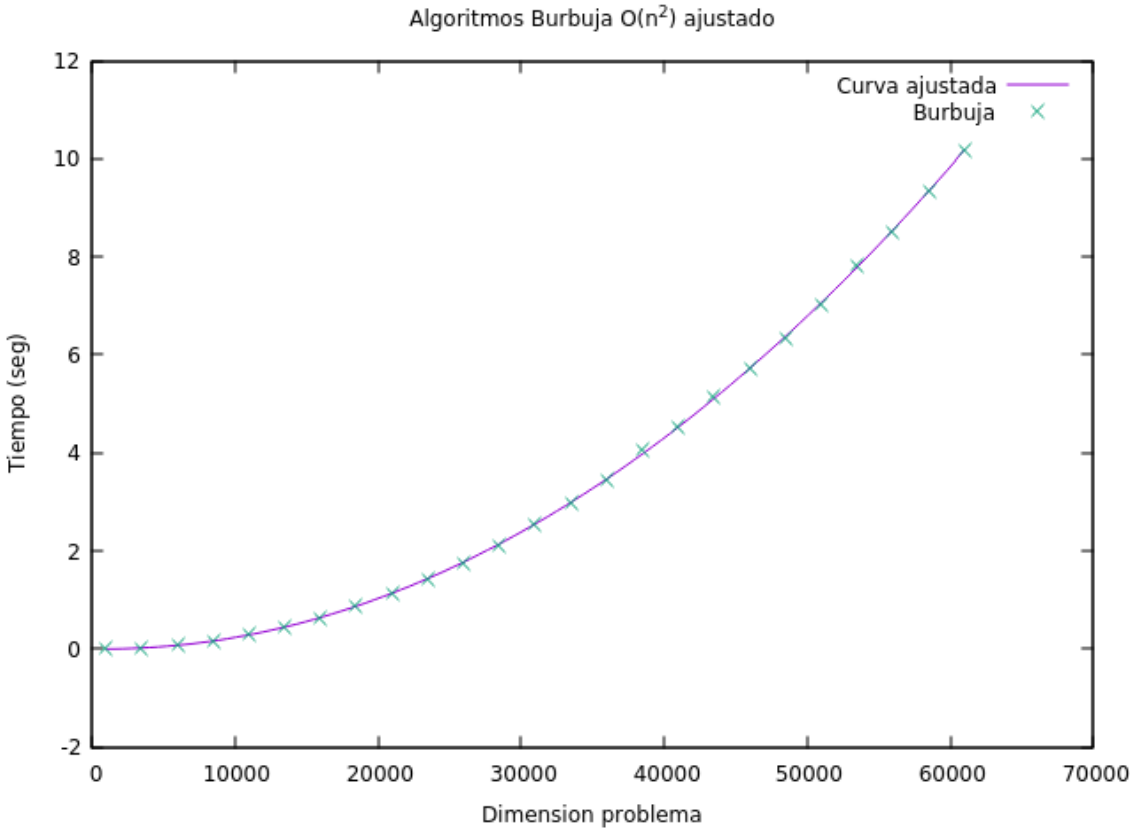
Orden de eficiencia O(n^2)			
Dimensión (elementos)	Burbuja	Inserción	Selección
1000	0.00233345	0.00161735	1.11161e-09
3500	0.0220215	0.0107103	0.0133117
6000	0.0741775	0.0304569	0.0387415
8500	0.162025	0.0609614	0.0774742
11000	0.288338	0.10288	0.129626
13500	0.441998	0.154063	0.195214
16000	0.632698	0.22659	0.273711
18500	0.864927	0.295406	0.36757
21000	1.14052	0.375318	0.471659
23500	1.4169	0.468104	0.590739
26000	1.75495	0.57592	0.723723
28500	2.11671	0.693095	0.867589
31000	2.55638	0.834557	1.02573
33500	2.97524	0.956152	1.19957
36000	3.44262	1.10781	1.38369
38500	4.05224	1.26848	1.55658
41000	4.54637	1.49898	1.79299
43500	5.15496	1.60774	2.0181
46000	5.71493	1.79192	2.25633
48500	6.35276	1.9982	2.46992
51000	7.0463	2.34461	2.83972
53500	7.83095	2.48081	3.08651
56000	8.52961	2.67549	3.34637
58500	9.33606	2.9197	3.61743
61000	10.1715	3.3904	3.96781

A continuación, se muestran las gráficas obtenidas con gnuplot y su ajuste, donde se puede apreciar claramente la función característica de los algoritmos cuadráticos.

i. Algoritmo Burbuja



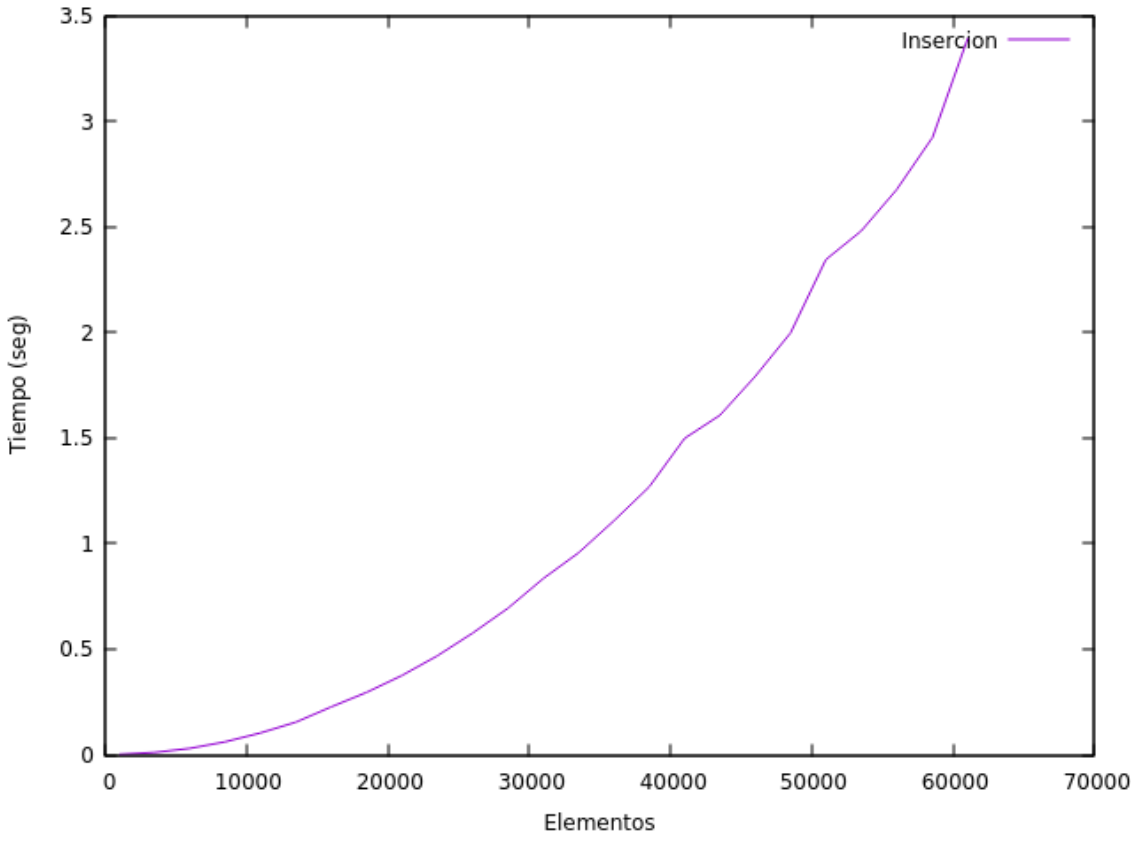
Enfoque híbrido: Burbuja ajustado a $T(n) = a_0*n^2 + a_1*n + a_2$



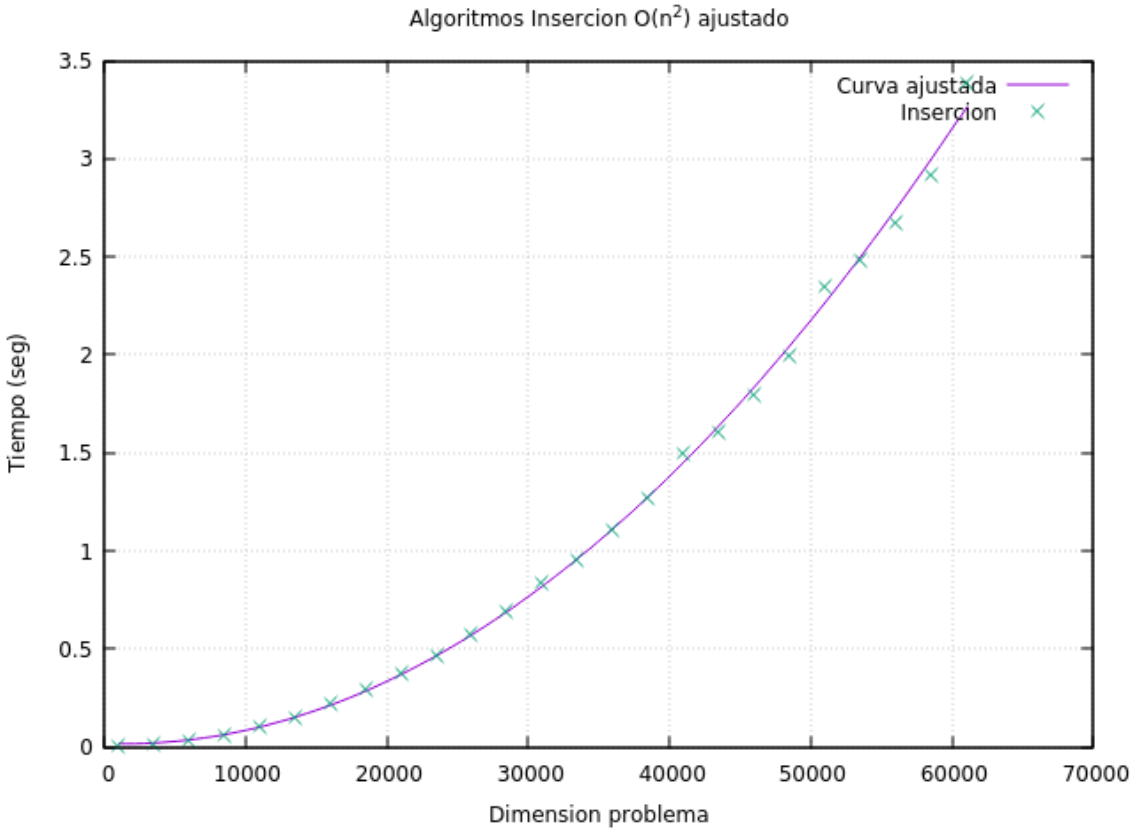
Valor de las constantes tras el ajuste:

```
gnuplot> f(x) = a0*x*x+a1*x+a2
gnuplot> fit f(x) 'salidaburbuja-media.dat' via a0,a1,a2
iter      chisq       delta/lim  lambda   a0             a1             a2
   0 7.4682744241e+19   0.00e+00  9.98e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 1.2929839997e+16  -5.77e+08  9.98e+07    1.313807e-02   9.999802e-01   1.000000e+00
   2 2.2352772593e+09  -5.78e+11  9.98e+06   -1.833424e-05   9.999797e-01   1.000000e+00
   3 2.0053939845e+09  -1.15e+04  9.98e+05   -2.008802e-05   9.999596e-01   1.000000e+00
   4 1.9973410861e+09  -4.03e+02  9.98e+04   -2.004765e-05   9.979497e-01   9.999999e-01
   5 1.3838400220e+09  -4.43e+04  9.98e+03   -1.668646e-05   8.306525e-01   9.999891e-01
   6 3.0968982431e+06  -4.46e+07  9.98e+02   -7.858702e-07   3.922900e-02   9.999381e-01
   7 3.6282348256e+00  -8.54e+10  9.98e+01    3.289140e-09  -4.999681e-05   9.999327e-01
   8 2.8637689759e+00  -2.67e+04  9.98e+00    3.680752e-09  -6.948246e-05   9.996464e-01
   9 2.7072725027e+00  -5.78e+03  9.98e-01    3.656743e-09  -6.769100e-05   9.718122e-01
  10 1.9398970842e-01  -1.30e+06  9.98e-02    3.035296e-09  -2.132363e-05   2.514343e-01
  11 1.3559675609e-02  -1.33e+06  9.98e-03    2.819054e-09  -5.189317e-06   7.666122e-04
  12 1.3557490918e-02  -1.61e+01  9.98e-04    2.818298e-09  -5.132981e-06  -1.086434e-04
  13 1.3557490918e-02  -2.06e-08  9.98e-05    2.818298e-09  -5.132979e-06  -1.086739e-04
iter      chisq       delta/lim  lambda   a0             a1             a2

After 13 iterations the fit converged.
final sum of squares of residuals : 0.0135575
rel. change during last iteration : -2.05877e-13

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0248244
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00061625

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 2.8183e-09       +/- 1.712e-11    (0.6075%)
a1              = -5.13298e-06     +/- 1.097e-06    (21.36%)
a2              = -0.000108674     +/- 0.0147       (1.353e+04%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.968  1.000
a2              0.741 -0.863  1.000
gnuplot>
```

ii. Algoritmo de Inserción



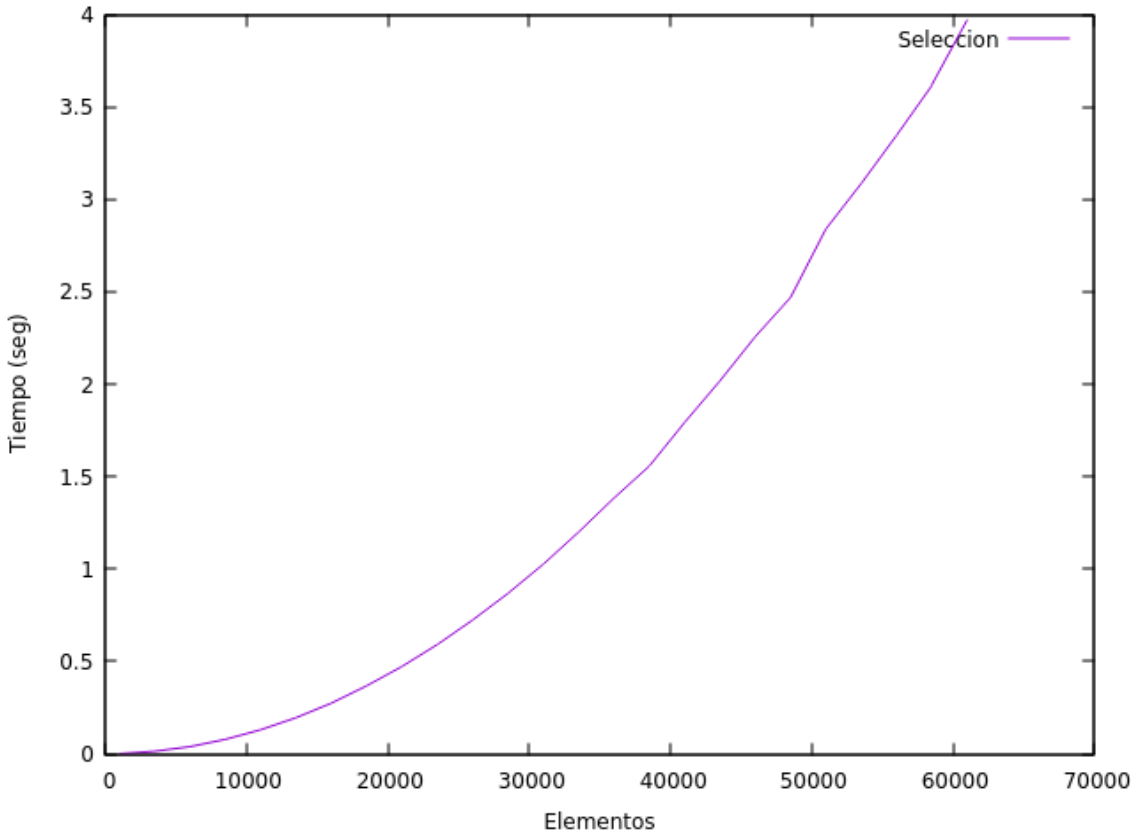
Enfoque híbrido: Inserción ajustado a $T(n) = a_0*n^2 + a_1*n + a_2$



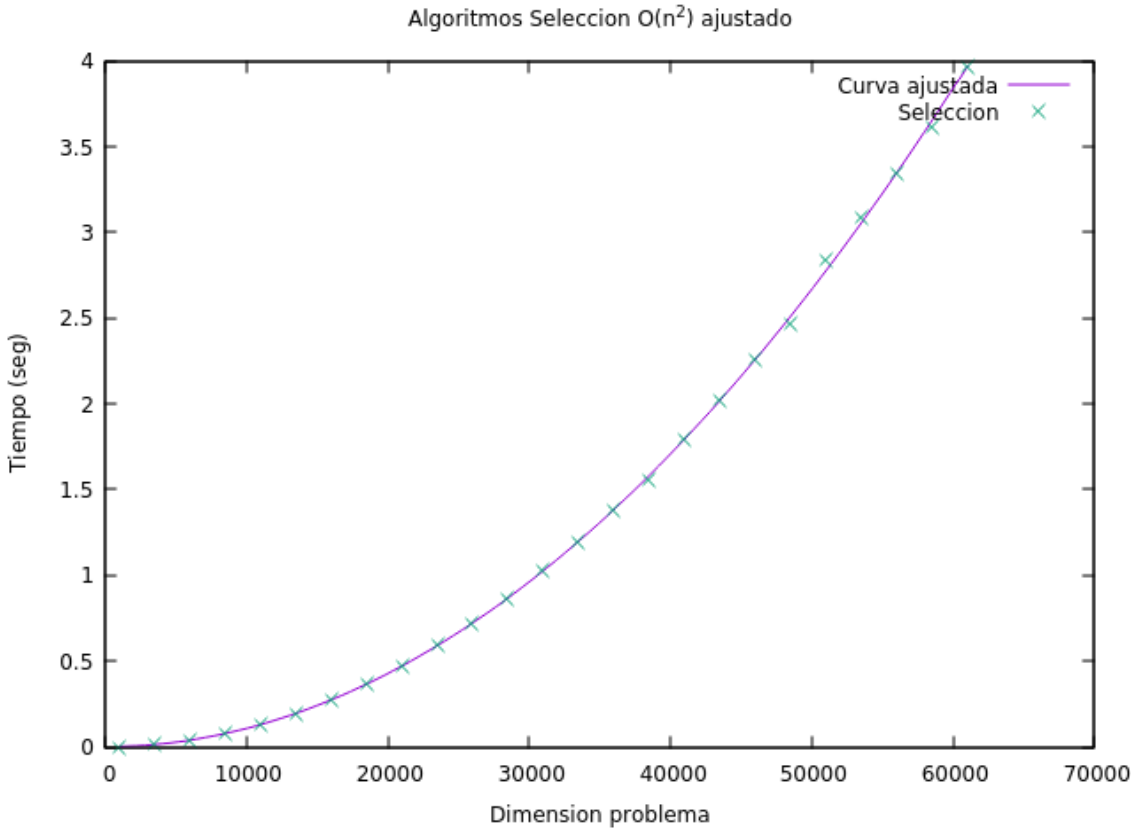
Valor de las constantes tras el ajuste:

```
gnuplot> fit f(x) 'salidainsercion-media.dat' via a0,a1,a2
iter      chisq       delta/lim  lambda   a0             a1             a2
   0 7.4682744516e+19   0.00e+00  9.98e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 1.2929840045e+16  -5.77e+08  9.98e+07    1.313807e-02   9.999802e-01   1.000000e+00
   2 2.2352619374e+09  -5.78e+11  9.98e+06   -1.833609e-05   9.999797e-01   1.000000e+00
   3 2.0053786628e+09  -1.15e+04  9.98e+05   -2.008987e-05   9.999596e-01   1.000000e+00
   4 1.9973258259e+09  -4.03e+02  9.98e+04   -2.004949e-05   9.979497e-01   9.999999e-01
   5 1.3838294491e+09  -4.43e+04  9.98e+03   -1.668832e-05   8.306531e-01   9.999891e-01
   6 3.0968745155e+06  -4.46e+07  9.98e+02   -7.877883e-07   3.923267e-02   9.999381e-01
   7 3.5615755714e+00  -8.70e+10  9.98e+01    1.367962e-09  -4.617659e-05   9.999328e-01
   8 2.7971696591e+00  -2.73e+04  9.98e+00    1.759577e-09  -6.566247e-05   9.996513e-01
   9 2.6458594941e+00  -5.72e+03  9.98e-01    1.735968e-09  -6.390094e-05   9.722821e-01
  10 2.1586713489e-01  -1.13e+06  9.98e-02    1.124906e-09  -1.830836e-05   2.639415e-01
  11 4.1416570820e-02  -4.21e+05  9.98e-03    9.122765e-10  -2.443645e-06   1.746238e-02
  12 4.1414458531e-02  -5.10e+00  9.98e-04    9.115341e-10  -2.388251e-06   1.660175e-02
  13 4.1414458531e-02  -6.03e-09  9.98e-05    9.115341e-10  -2.388249e-06   1.660172e-02
iter      chisq       delta/lim  lambda   a0             a1             a2

After 13 iterations the fit converged.
final sum of squares of residuals : 0.0414145
rel. change during last iteration : -6.03171e-14

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0433875
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.00188248

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 9.11534e-10      +/- 2.992e-11    (3.283%)
a1              = -2.38825e-06     +/- 1.917e-06    (80.25%)
a2              = 0.0166017        +/- 0.02569      (154.8%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.968  1.000
a2              0.741 -0.863  1.000
gnuplot>
```

iii. Algoritmo de Selección



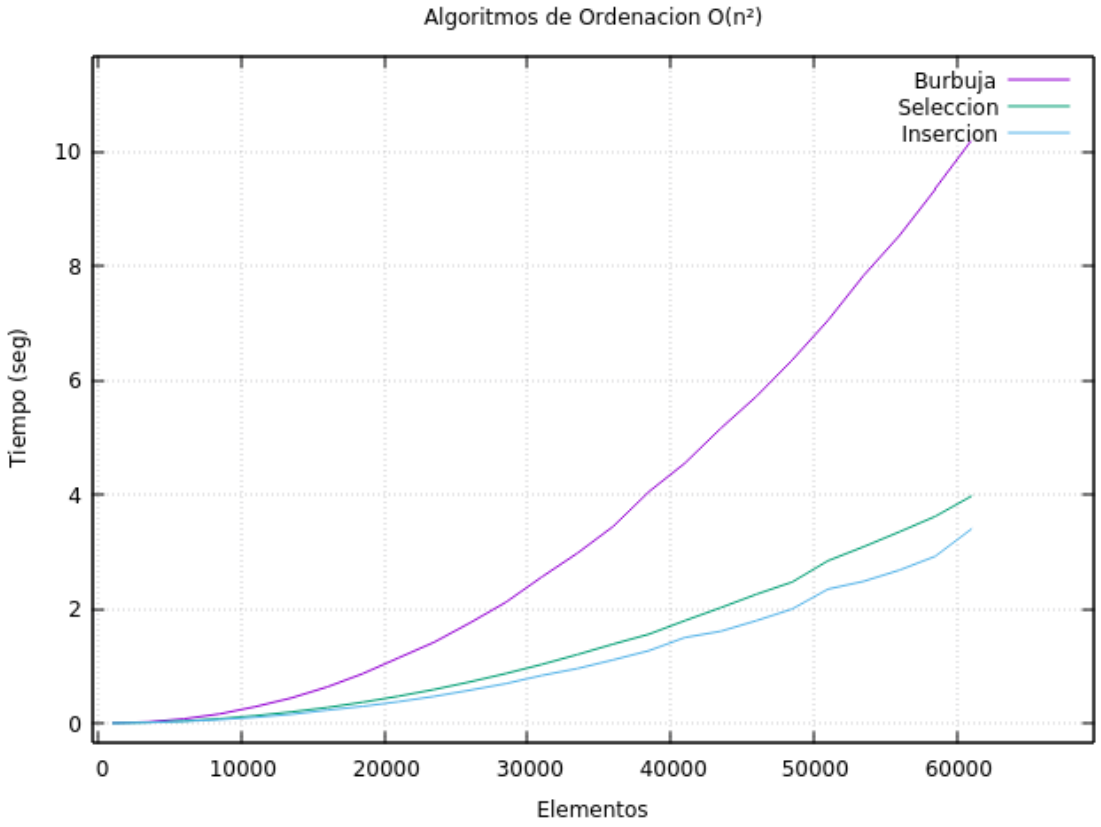
Enfoque híbrido: Selección ajustado a $T(n) = a_0*n^2 + a_1*n + a_2$



Valor de las constantes tras el ajuste:

```
gnuplot> f(x) = a0*x*x+a1*x+a2
gnuplot> fit f(x) 'salidaseleccion-media.dat' via a0,a1,a2
iter      chisq       delta/lim  lambda   a0             a1             a2
   0 7.4682744487e+19   0.00e+00  9.98e+08    1.000000e+00   1.000000e+00   1.000000e+00
   1 1.2929840040e+16  -5.77e+08  9.98e+07    1.313807e-02   9.999802e-01   1.000000e+00
   2 2.2352565480e+09  -5.78e+11  9.98e+06   -1.833589e-05   9.999797e-01   1.000000e+00
   3 2.0053732738e+09  -1.15e+04  9.98e+05   -2.008967e-05   9.999596e-01   1.000000e+00
   4 1.9973204585e+09  -4.03e+02  9.98e+04   -2.004929e-05   9.979497e-01   9.999999e-01
   5 1.3838257303e+09  -4.43e+04  9.98e+03   -1.668812e-05   8.306534e-01   9.999891e-01
   6 3.0968662547e+06  -4.46e+07  9.98e+02   -7.876179e-07   3.923397e-02   9.999381e-01
   7 3.6230635671e+00  -8.55e+10  9.98e+01    1.537365e-09  -4.483288e-05   9.999327e-01
   8 2.8586057268e+00  -2.67e+04  9.98e+00    1.928975e-09  -6.431843e-05   9.996464e-01
   9 2.7021218389e+00  -5.79e+03  9.98e-01    1.904966e-09  -6.252704e-05   9.718133e-01
  10 1.8904115858e-01  -1.33e+06  9.98e-02    1.283545e-09  -1.616153e-05   2.514644e-01
  11 8.6256356527e-03  -2.09e+06  9.98e-03    1.067311e-09  -2.787234e-08   8.067890e-04
  12 8.6234511381e-03  -2.53e+01  9.98e-04    1.066556e-09   2.846151e-08  -6.843125e-05
  13 8.6234511381e-03  -3.08e-08  9.98e-05    1.066556e-09   2.846348e-08  -6.846185e-05
iter      chisq       delta/lim  lambda   a0             a1             a2

After 13 iterations the fit converged.
final sum of squares of residuals : 0.00862345
rel. change during last iteration : -3.07579e-13

degrees of freedom    (FIT_NDF)                        : 22
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0197984
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000391975

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 1.06656e-09      +/- 1.365e-11    (1.28%)
a1              = 2.84635e-08      +/- 8.746e-07    (3073%)
a2              = -6.84619e-05     +/- 0.01172      (1.713e+04%)

correlation matrix of the fit parameters:
                a0     a1     a2
a0              1.000
a1             -0.968  1.000
a2              0.741 -0.863  1.000
gnuplot>
```

Una vez hecho el análisis empírico e híbrido de cada algoritmo de ordenación de orden $O(n^2)$ por separado es el momento de compararlos en una misma gráfica:



Como podemos ver, cuanto mayor es la dimensión del problema (número de elementos del vector a ordenar), más se aprecia la diferencia de tiempo entre los algoritmos. El mejor tiempo se obtiene para el algoritmo de ordenación por Selección. Por el contrario, el algoritmo de ordenación Burbuja obtiene los peores tiempos, con gran diferencia respecto a Inserción y Selección, que para tamaños de problema grandes tienen una diferencia de menos de medio segundo aproximadamente.

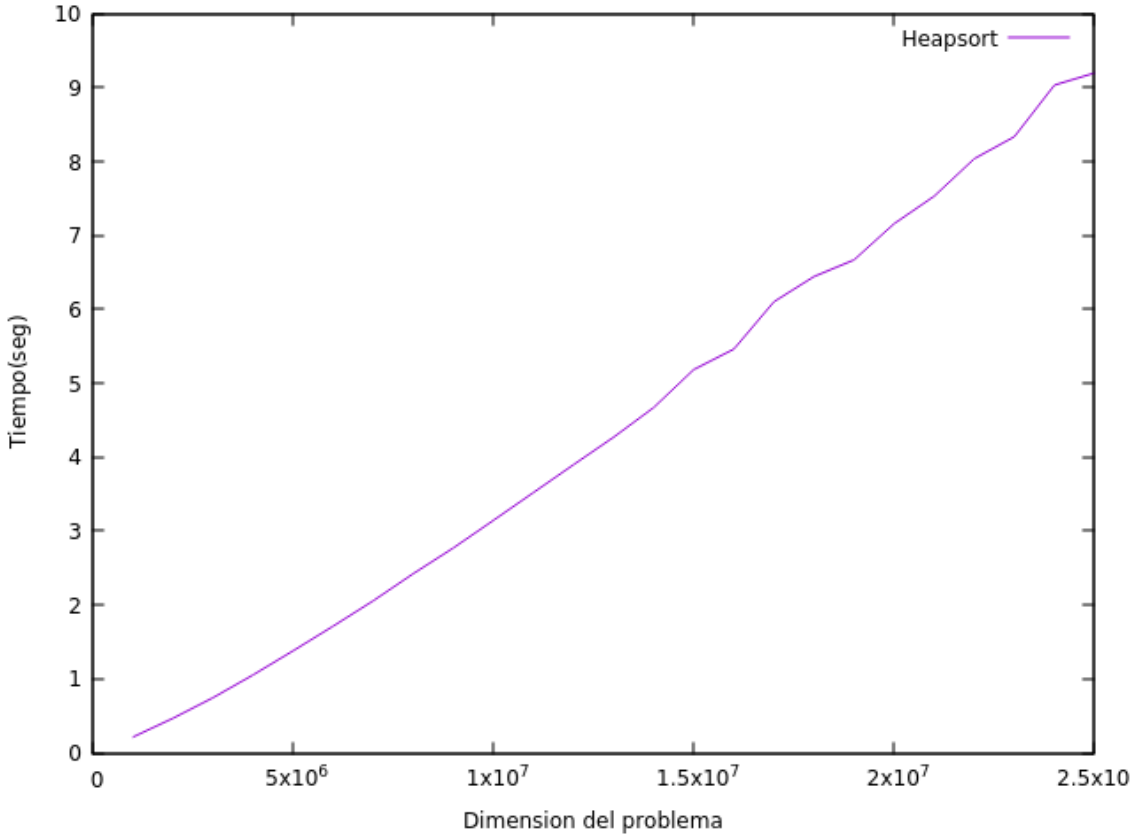
3) Algoritmos de Ordenación de Orden O(nlogn)

Para este tipo de algoritmos con una eficiencia teórica de orden $O(n^2)$ se han obtenido los siguientes tiempos de ejecución en función del tamaño de la entrada del problema, en este caso, el número de componentes del vector a ordenar comenzando en 1000000, de 1000000 en 1000000, hasta 25000000.

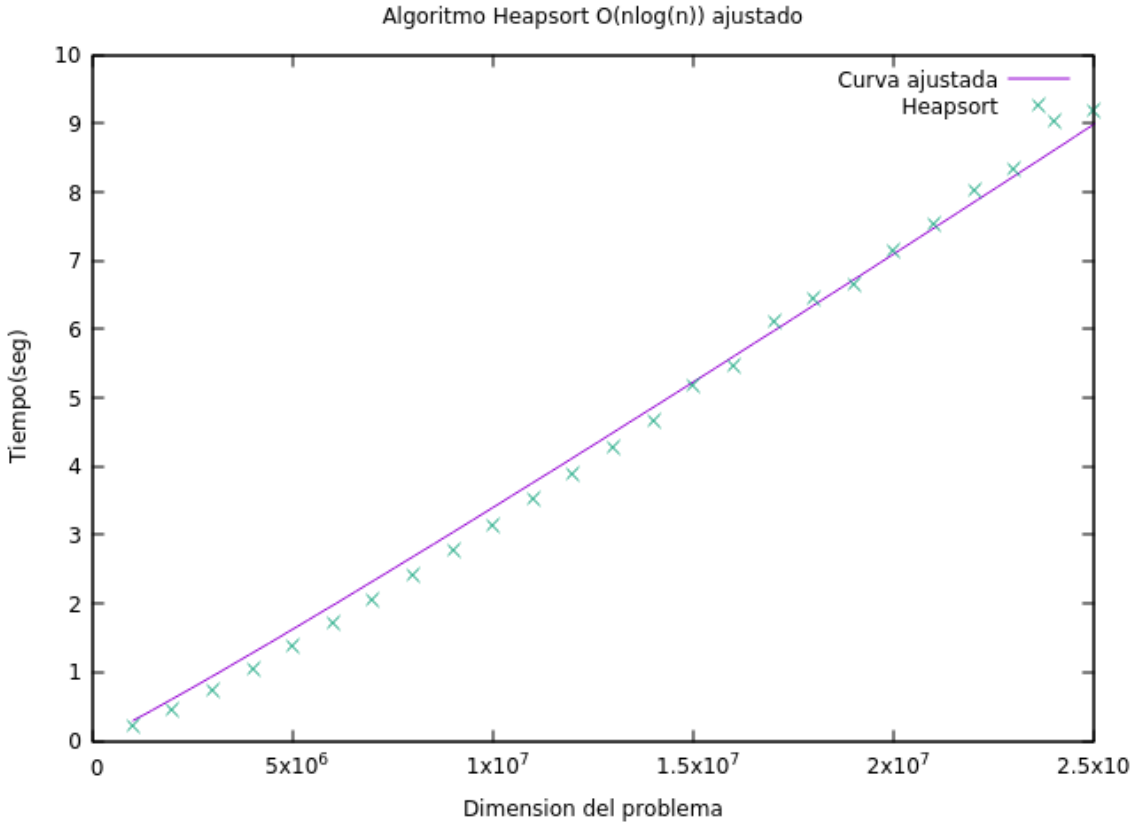
Orden de eficiencia O(nlog(n))			
Dimensión (elementos)	Heapsort	Mergesort	Quicksort
$1*10^6$	0.21187	0.208997	0.138882
$2*10^6$	0.464664	0.43057	0.291777
$3*10^6$	0.743614	0.743943	0.449823
$4*10^6$	1.05295	0.897515	0.609289
$5*10^6$	1.37959	1.19975	0.762986
$6*10^6$	1.71161	1.5662	0.933892
$7*10^6$	2.05634	1.57882	1.09681
$8*10^6$	2.42214	1.8639	1.27618
$9*10^6$	2.77046	2.24335	1.43591
$10*10^6$	3.14536	2.49551	1.59502
$11*10^6$	3.52204	2.83289	1.77258
$12*10^6$	3.89851	3.18597	1.94187
$13*10^6$	4.26904	3.49516	2.11638
$14*10^6$	4.67064	3.28462	2.28479
$15*10^6$	5.18346	3.58936	2.46421
$16*10^6$	5.46264	3.88189	2.64738
$17*10^6$	6.10263	4.22679	2.80091
$18*10^6$	6.44357	4.52828	2.96646
$19*10^6$	6.66715	4.86093	3.17993
$20*10^6$	7.15683	5.20619	3.33331
$21*10^6$	7.53007	5.56328	3.49112
$22*10^6$	8.03744	5.95751	3.67838
$23*10^6$	8.33518	6.3802	3.88384
$24*10^6$	9.03367	6.65836	4.0132
$25*10^6$	9.1974	7.12851	4.20191

A continuación, se muestran las gráficas obtenidas de los algoritmos de orden logarítmico con gnuplot y su ajuste.

i. Algoritmo Heapsort



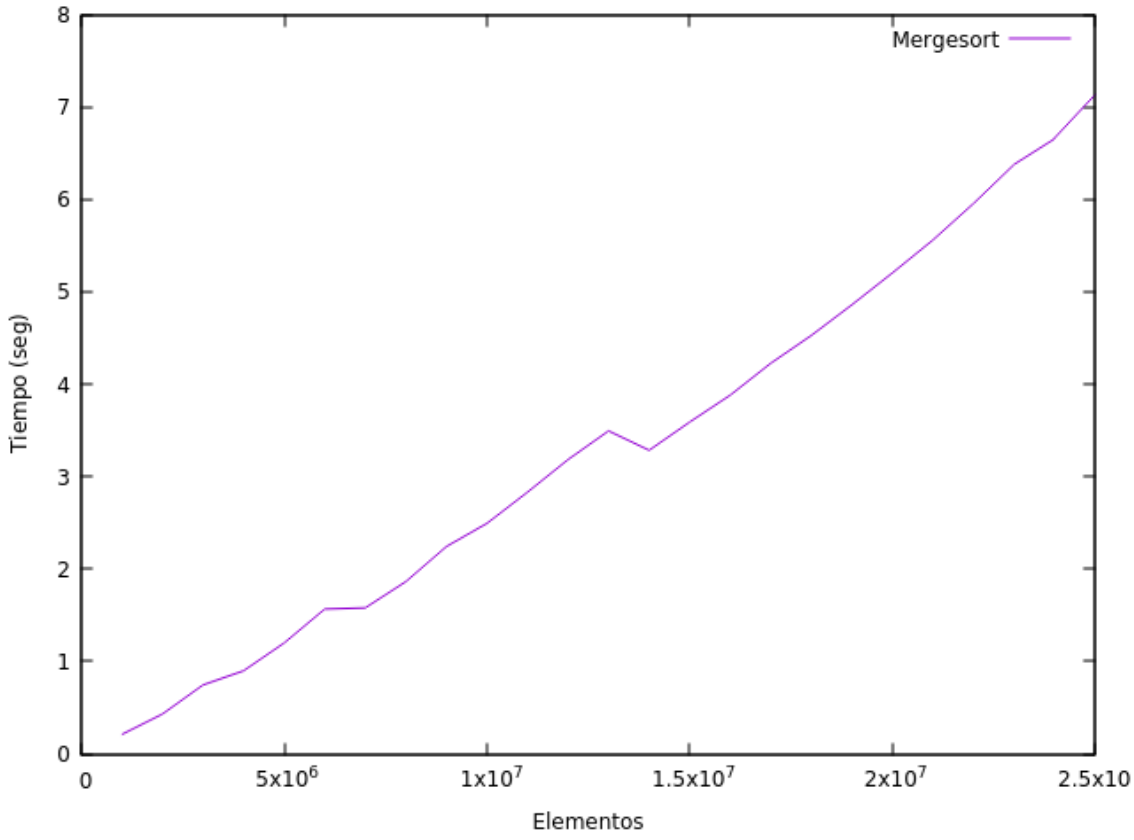
Enfoque híbrido: Heapsort ajustado a $T(n) = a_0*n*\log(n)$



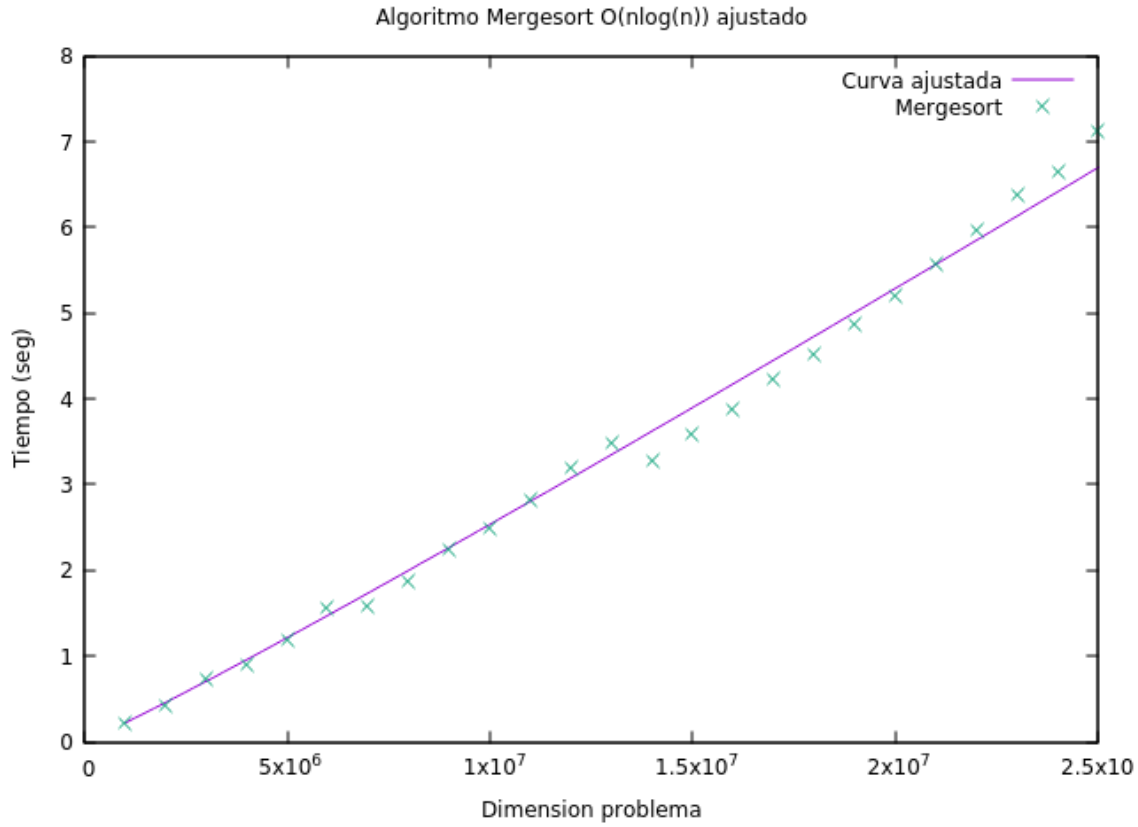
Valor de la constante tras el ajuste:

```
gnuplot> f(x) =  a0*x*log(x)
gnuplot> fit f(x) 'salidaheapsort-media.dat' via a0
iter      chisq       delta/lim  lambda   a0
   0 1.5452729302e+18   0.00e+00  2.49e+08    1.000000e+00
   1 2.2859067014e+15  -6.75e+07  2.49e+07    3.846156e-02
   2 3.6545265245e+08  -6.25e+11  2.49e+06    1.539958e-05
   3 1.0687621978e+00  -3.42e+13  2.49e+05    2.117909e-08
   4 1.0629150022e+00  -5.50e+02  2.49e+04    2.111758e-08
   5 1.0629150022e+00  -2.09e-11  2.49e+03    2.111758e-08
iter      chisq       delta/lim  lambda   a0

After 5 iterations the fit converged.
final sum of squares of residuals : 1.06292
rel. change during last iteration : -2.08902e-16

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.210447
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0442881

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 2.11176e-08      +/- 1.693e-10    (0.8017%)
gnuplot>
```

ii. Algoritmo Mergesort



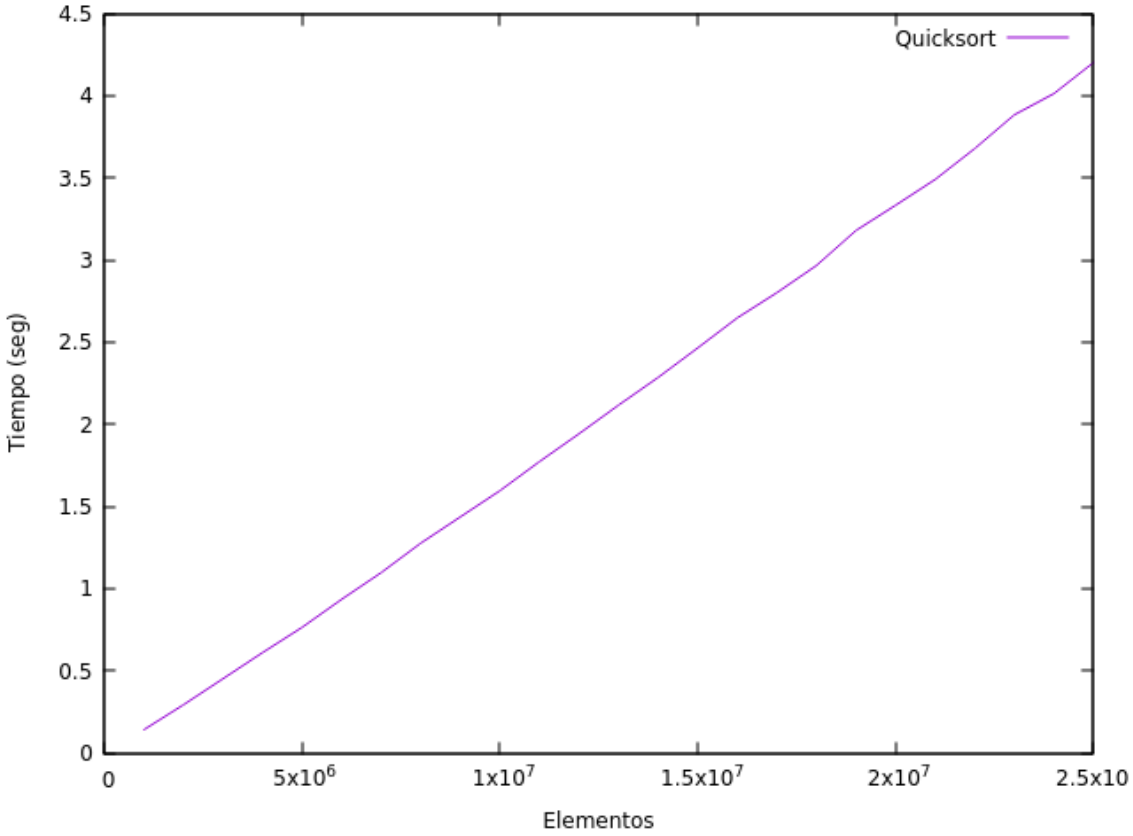
Enfoque híbrido: Mergesort ajustado a T(n) = a_0*n*log(n)



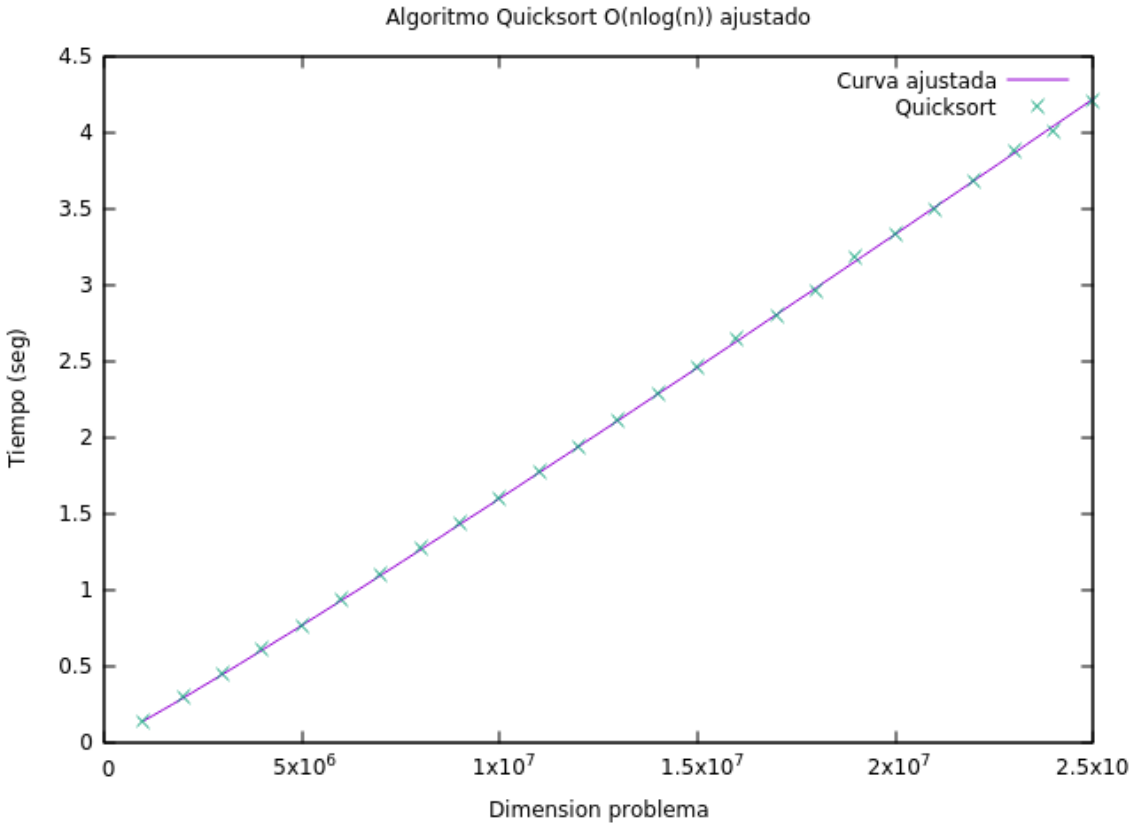
Valor de la constante tras el ajuste:

```
gnuplot> f(x) =  a0*x*log(x)
gnuplot> fit f(x) 'salidamergesort-media.dat' via a0
iter      chisq       delta/lim  lambda   a0
   0 1.5452729468e+18   0.00e+00  2.49e+08    1.000000e+00
   1 2.2859067260e+15  -6.75e+07  2.49e+07    3.846155e-02
   2 3.6545265630e+08  -6.25e+11  2.49e+06    1.539420e-05
   3 8.3276990418e-01  -4.39e+13  2.49e+05    1.579285e-08
   4 8.2692270880e-01  -7.07e+02  2.49e+04    1.573133e-08
   * 8.2692270880e-01   6.71e-11  2.49e+05    1.573133e-08
   * 8.2692270880e-01   6.71e-11  2.49e+06    1.573133e-08
   * 8.2692270880e-01   1.34e-11  2.49e+07    1.573133e-08
   * 8.2692270880e-01   2.69e-11  2.49e+08    1.573133e-08
   * 8.2692270880e-01   5.37e-11  2.49e+09    1.573133e-08
   * 8.2692270880e-01   2.69e-11  2.49e+10    1.573133e-08
   5 8.2692270880e-01  -4.03e-11  2.49e+09    1.573133e-08
iter      chisq       delta/lim  lambda   a0

After 5 iterations the fit converged.
final sum of squares of residuals : 0.826923
rel. change during last iteration : -4.02779e-16

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.185621
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.0344551

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 1.57313e-08      +/- 1.493e-10    (0.9492%)
gnuplot>
```

iii. Algoritmo Quicksort



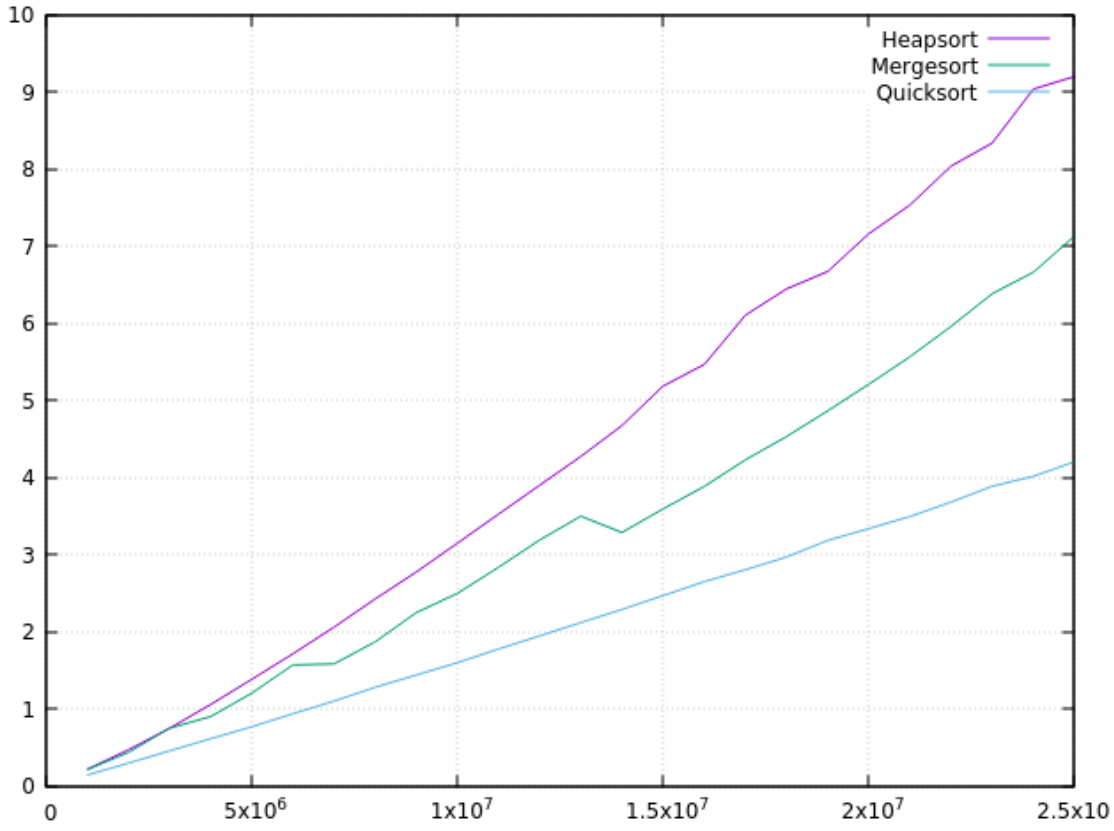
Enfoque híbrido: Mergesort ajustado a $T(n) = a_0*n*\log(n)$



Valor de la constante tras el ajuste:

```
gnuplot> f(x) =  a0*x*log(x)
gnuplot> fit f(x) 'salidaquicksort-media.dat' via a0
iter      chisq       delta/lim  lambda   a0
   0 1.5452729648e+18   0.00e+00  2.49e+08    1.000000e+00
   1 2.2859067526e+15  -6.75e+07  2.49e+07    3.846155e-02
   2 3.6545265965e+08  -6.26e+11  2.49e+06    1.538837e-05
   3 9.4690762593e-03  -3.86e+15  2.49e+05    9.969339e-09
   4 3.6218810140e-03  -1.61e+05  2.49e+04    9.907826e-09
   5 3.6218810140e-03  -2.99e-10  2.49e+03    9.907826e-09
iter      chisq       delta/lim  lambda   a0

After 5 iterations the fit converged.
final sum of squares of residuals : 0.00362188
rel. change during last iteration : -2.99348e-15

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.0122846
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.000150912

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 9.90783e-09      +/- 9.882e-12    (0.09974%)
gnuplot>
```

Una vez hecho el análisis empírico e híbrido de cada algoritmo de ordenación de orden O(nlog(n)) por separado es el momento de compararlos en una misma gráfica:

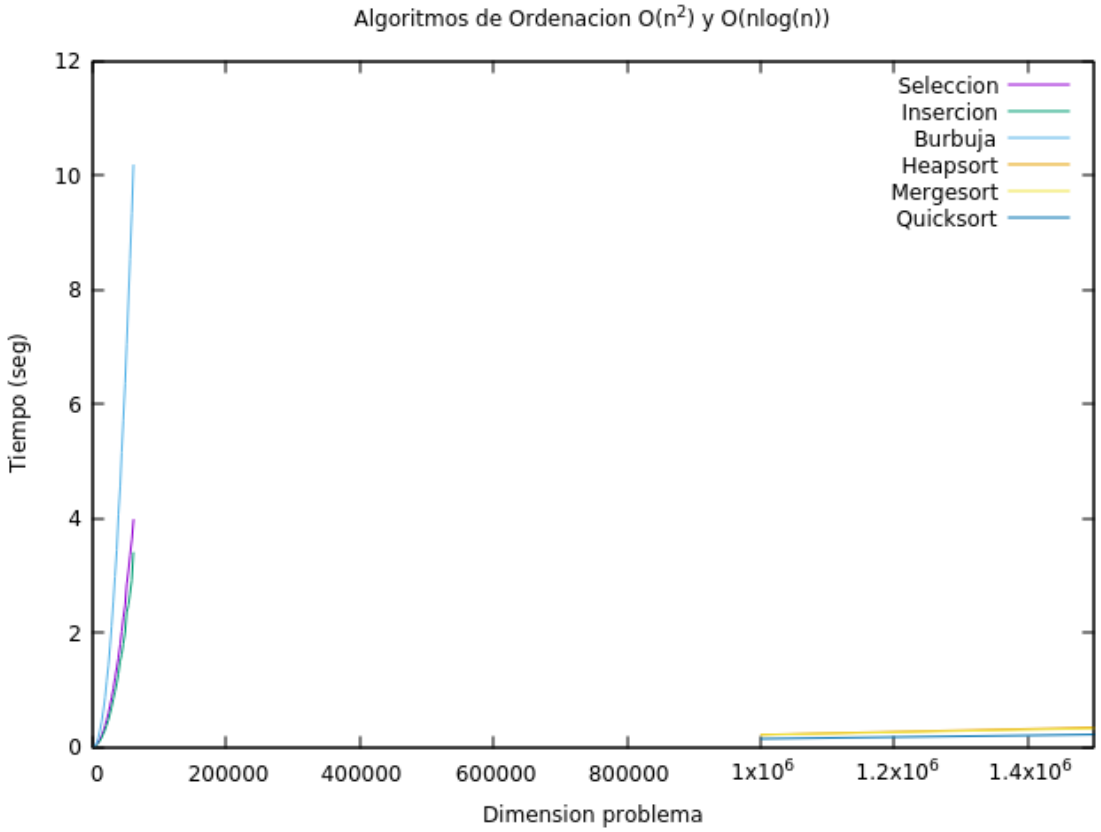


Como podemos ver, cuanto mayor es la dimensión del problema (número de elementos del vector a ordenar), más se aprecia la diferencia de tiempo entre los

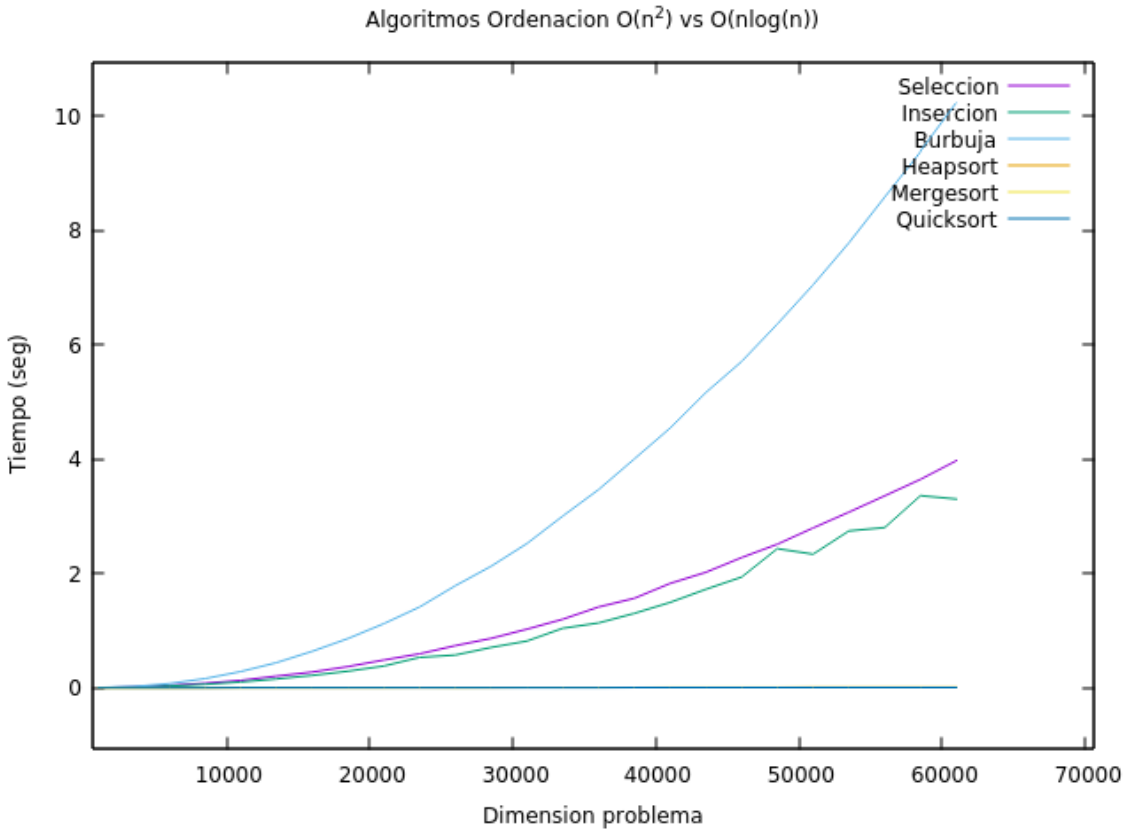
algoritmos. El mejor tiempo se obtiene para el algoritmo de ordenación por Quicksort. Por el contrario, el algoritmo de ordenación Heapsort obtiene los peores tiempos, con una diferencia de 2 segundos respecto a Mergesort y 5 segundos respecto al Quicksort, para 25000000 de elementos.

4) Comparativa Algoritmos de Ordenación de orden O(n^2) y O(nlog(n))

Para tamaños de problema grandes es claro que es necesario usar los algoritmos de ordenación del orden de O(nlog(n)) dado que los cuadráticos les sería prácticamente imposible alcanzar el objetivo de ordenar un vector de 1000000 de elementos.



Para tamaños de problema pequeños los algoritmos de orden O(nlog(n)) son los más recomendables dado que tardan todos menos de medio segundo para el mayor tamaño probado para los algoritmos cuadráticos (60000 elementos). La diferencia en los algoritmos de orden logarítmico para tamaños pequeños es prácticamente despreciable aunque sigue siendo Quicksort el mejor y Heapsort el peor.

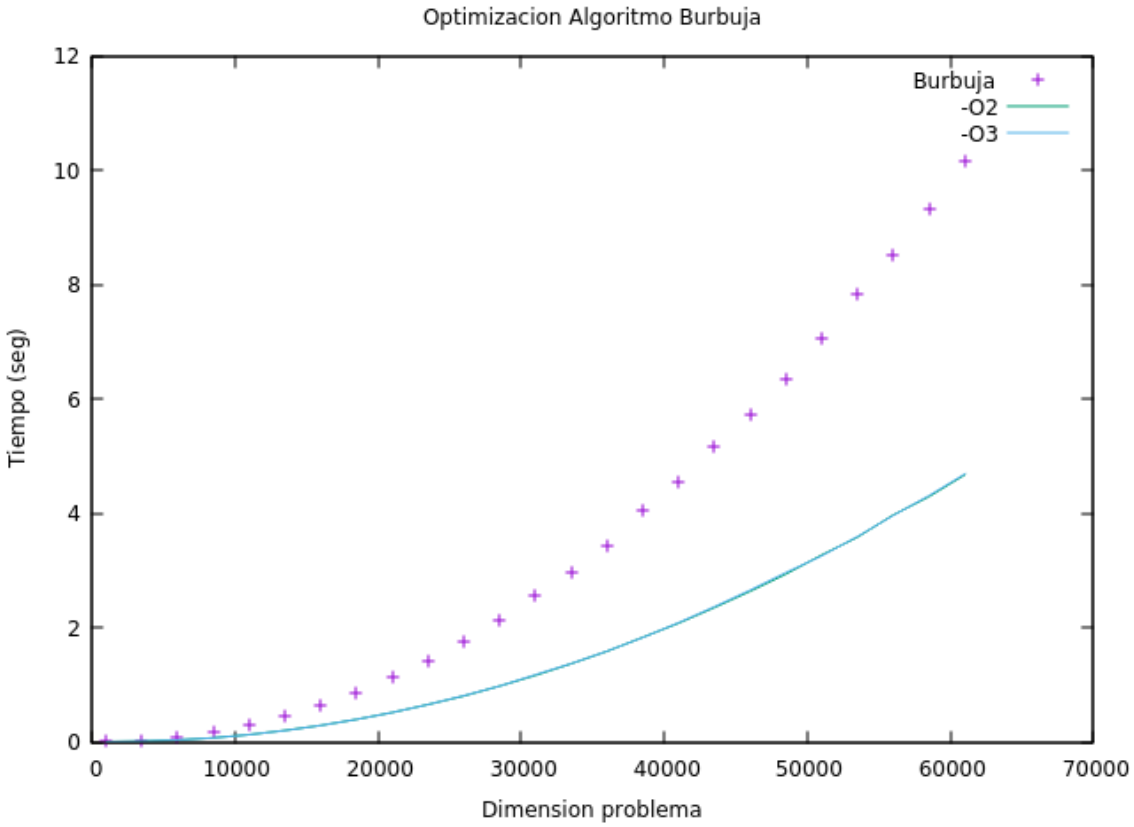


5) *Comparativa Algoritmos de Ordenación con Optimización*

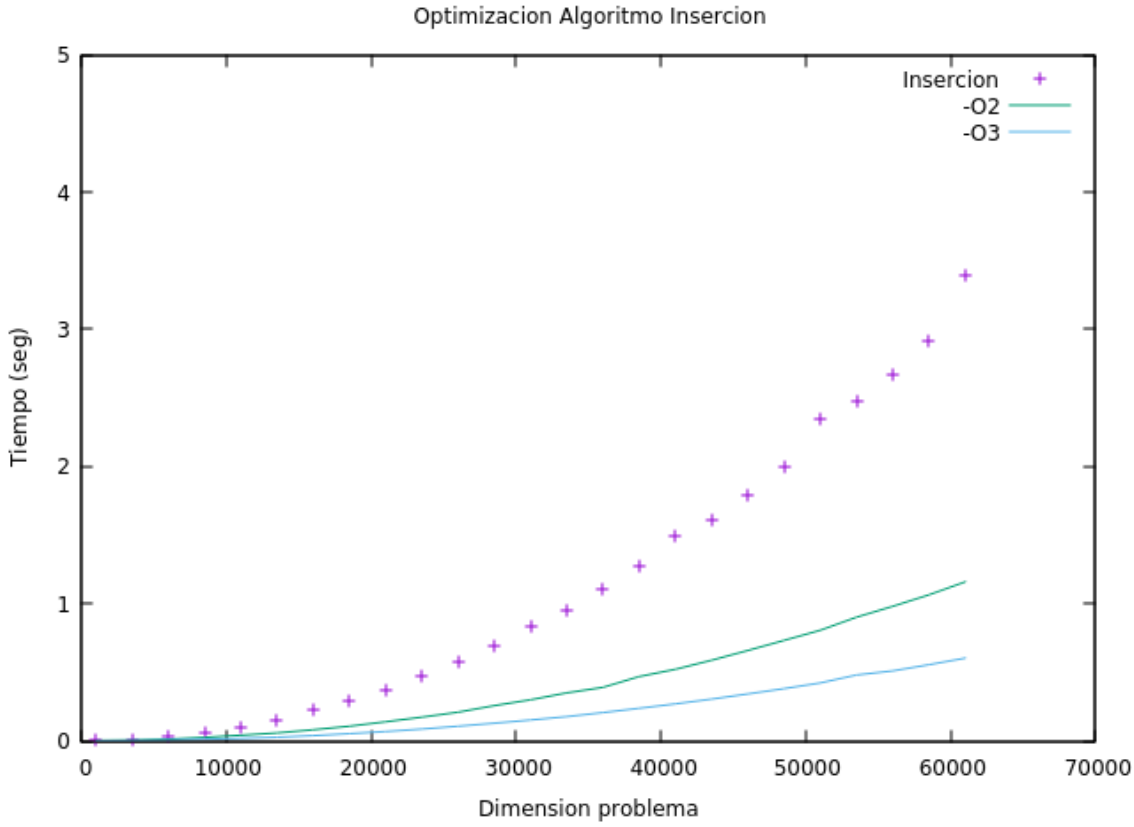
Se ha realizado el mismo análisis anterior para los algoritmos con diferentes opciones de optimización (-O2 y –O3) en la compilación del programa de prueba y se han comparado los resultados con y sin optimización. Estas opciones “simplifican” en número y forma a nivel de ensamblador el código generado en el compilado.

i) Algoritmos O(n^2)

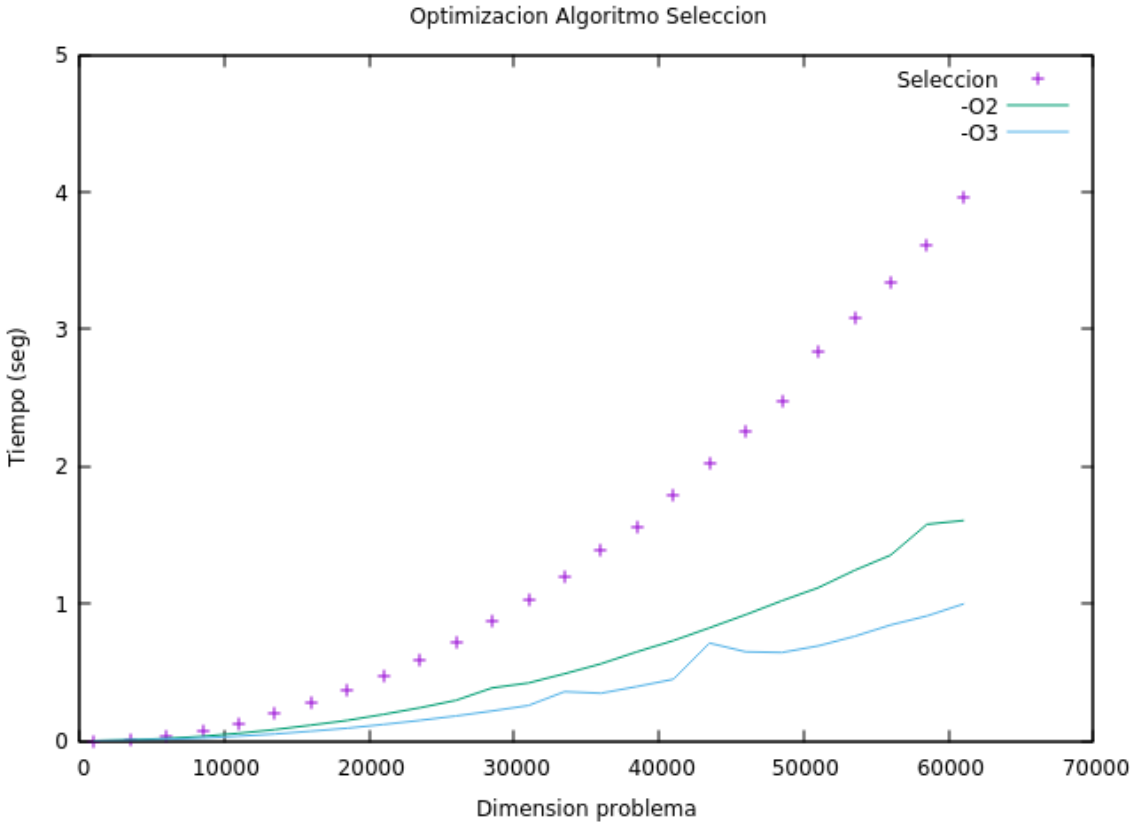
a. Burbuja:



b. Inserción:

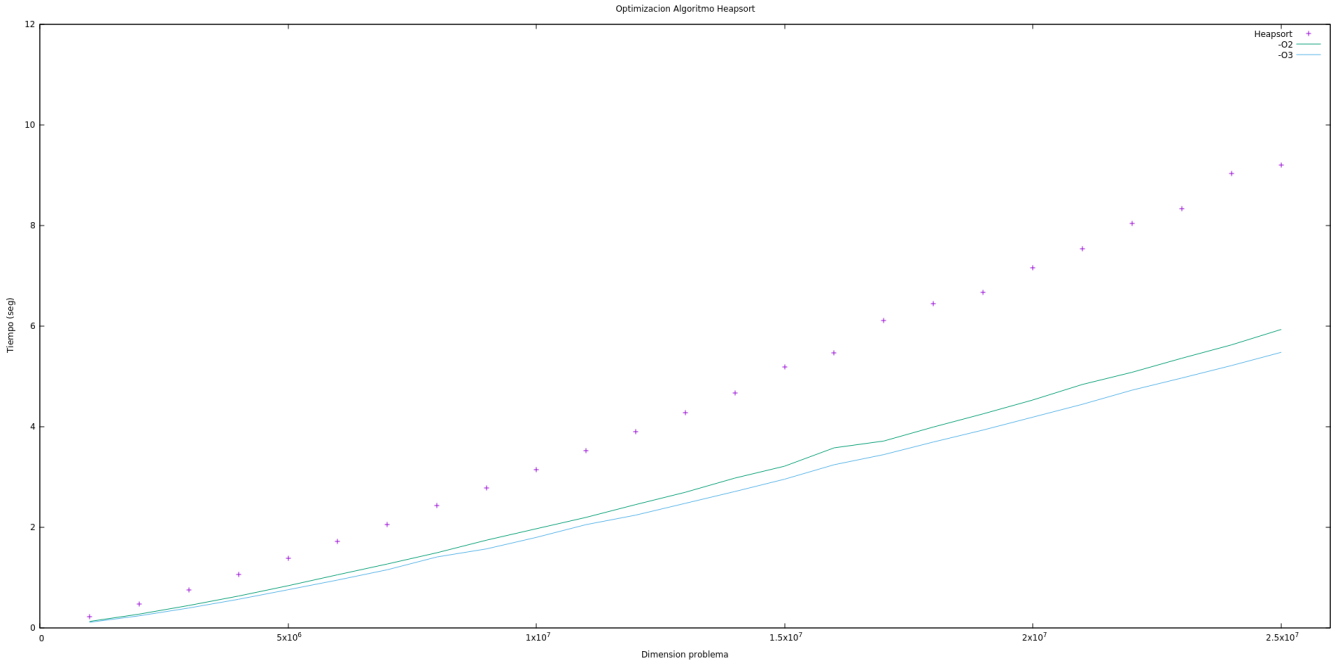


c. Selección:

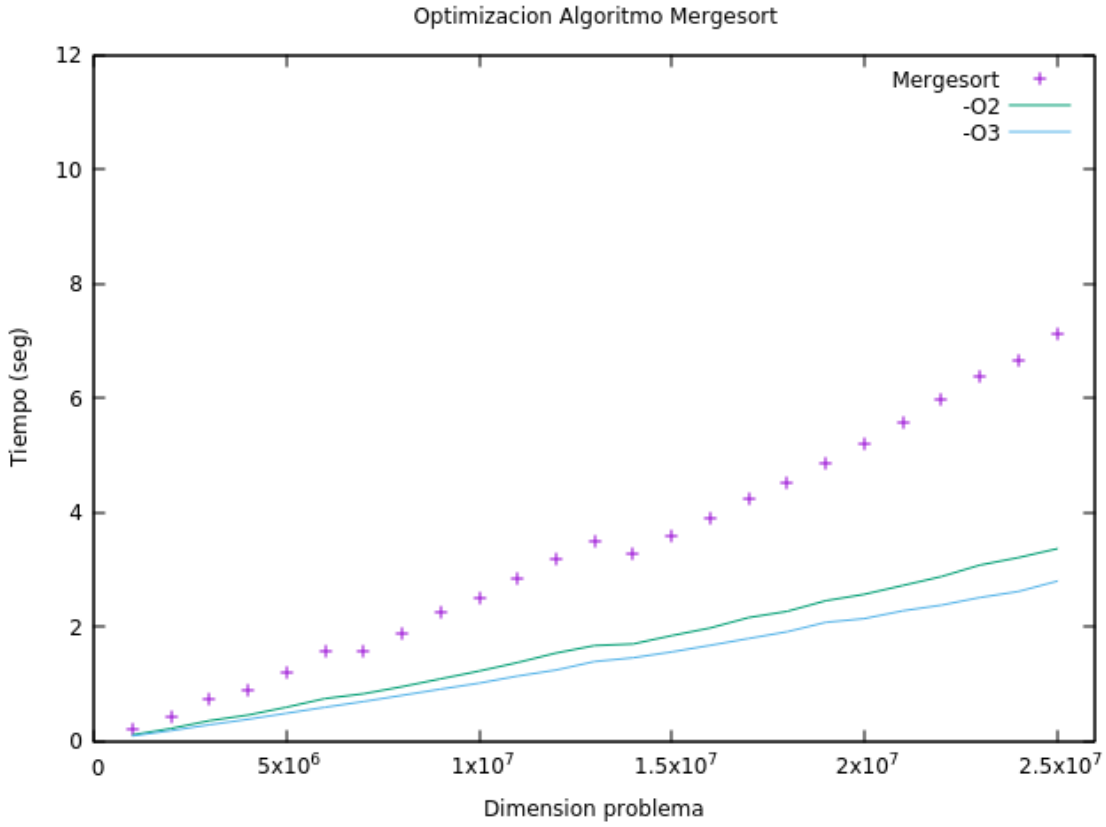


ii) Algoritmos O(nlog(n))

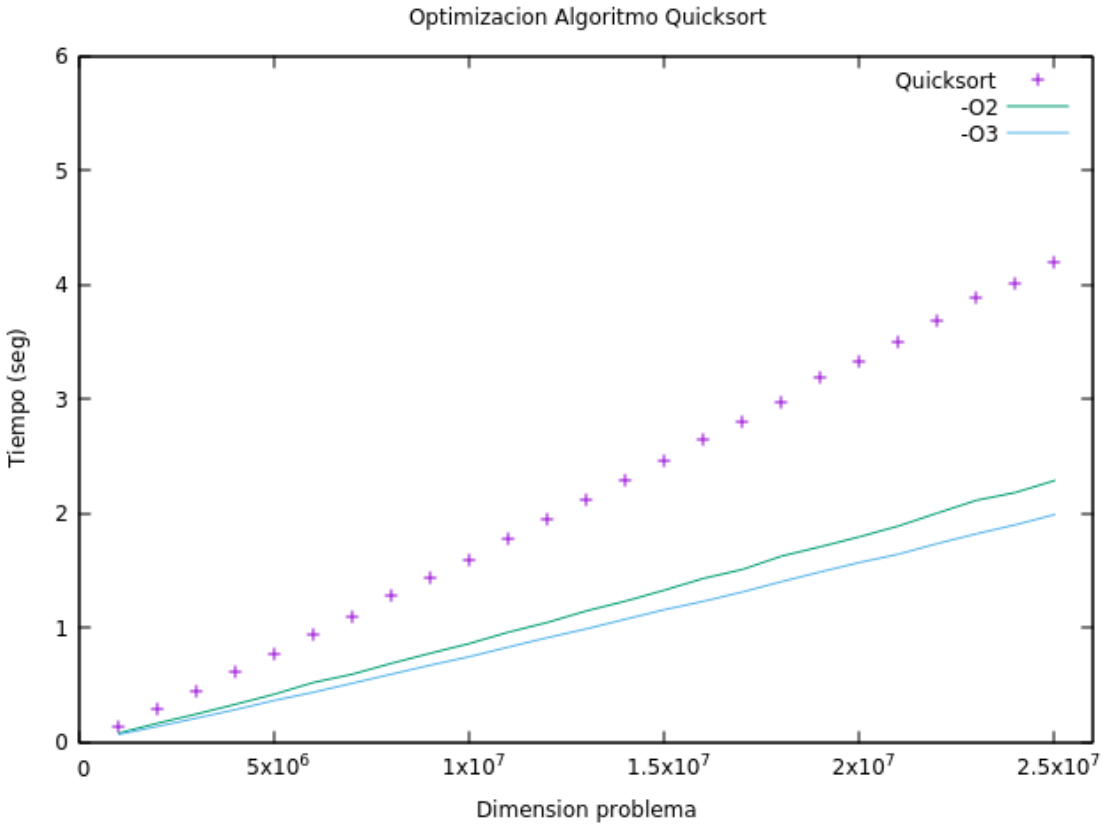
a. Heapsort:



b. Mergesort:



c. *Quicksort:*



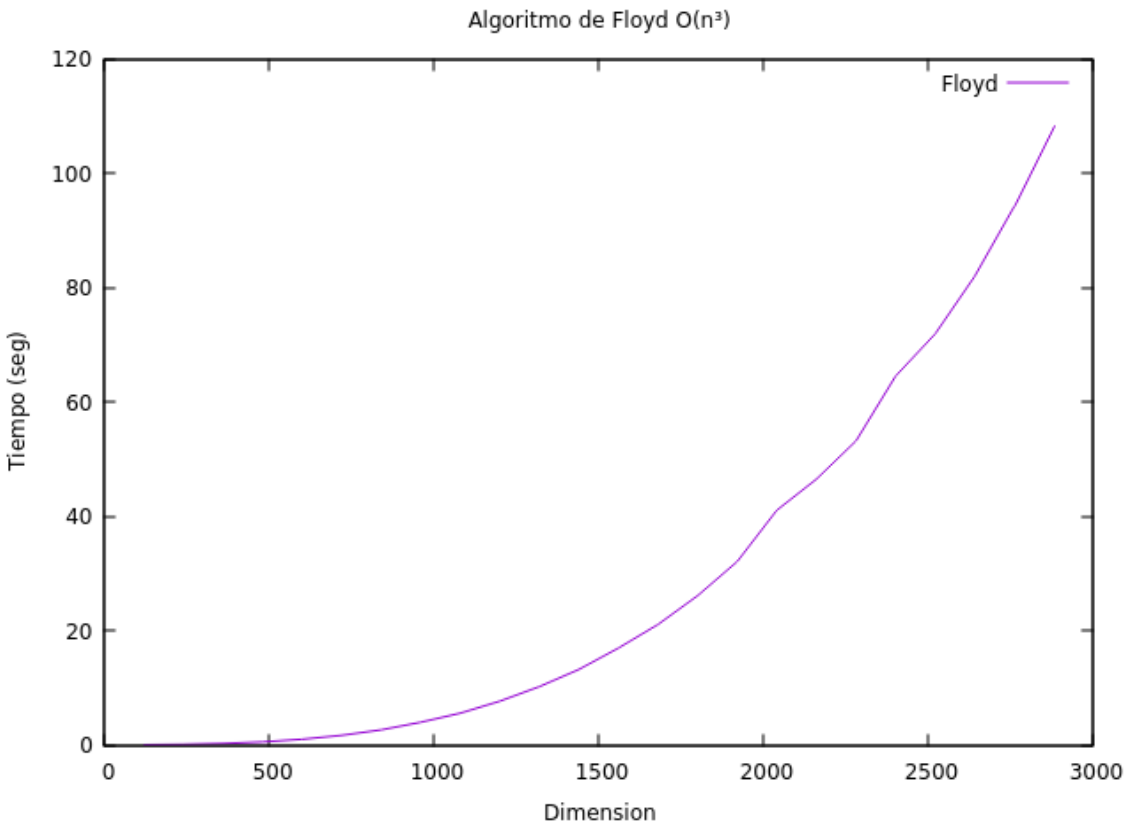
6) *Algoritmo de Floyd O(n^3)*

Para este tipo de algoritmos con una eficiencia teórica de orden O(n^3) se han obtenido los siguientes tiempos de ejecución en función del tamaño de la entrada del problema, en este caso, la dimensión de la matriz que representa los pares de nodos de un grafo dirigido y del cual se quiere calcular el costo del camino mínimo entre cada par de nodos. comenzando en 120, de 120 en 120, hasta 2880.

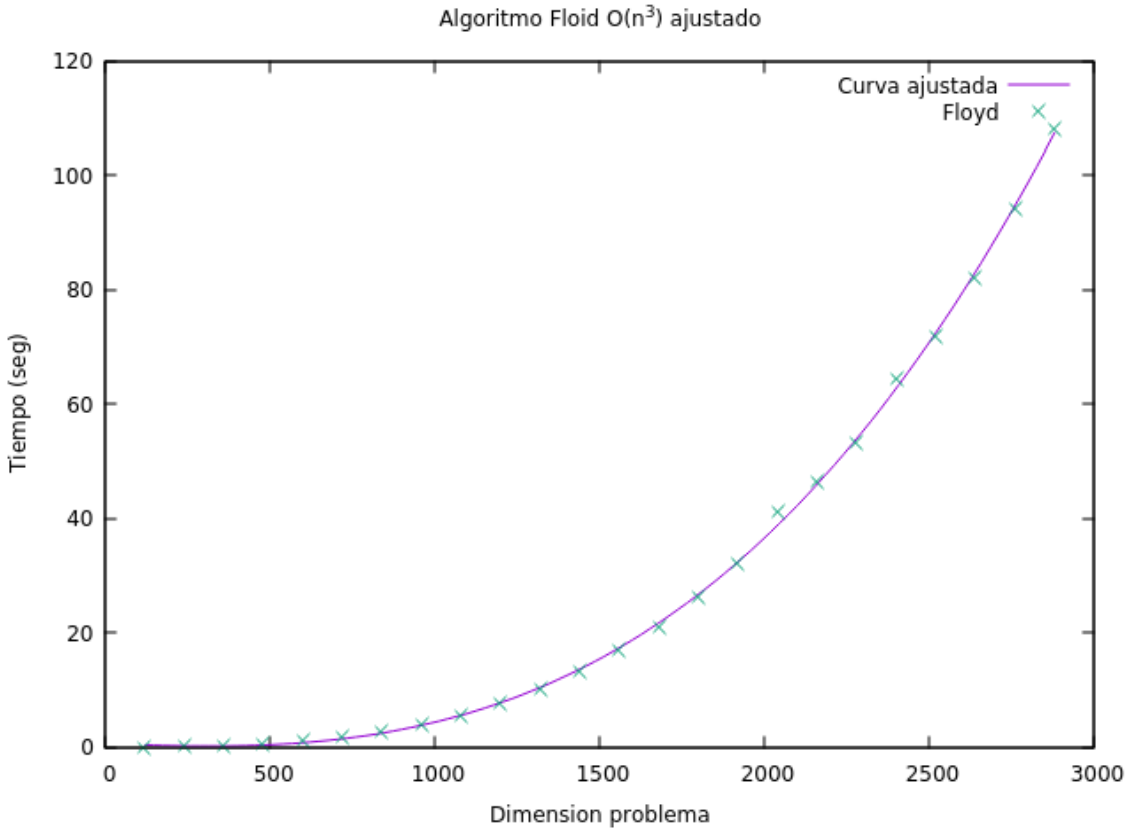
Orden de eficiencia O(n^3)	
Dimensión	Foyd
120	0.00878197
240	0.0625277
360	0.207162
480	0.504981
600	0.947648
720	1.64487
840	2.60446
960	3.9263

1080	5.5313
1200	7.62077
1320	10.1595
1440	13.1828
1560	16.9363
1680	21.0648
1800	26.0691
1920	32.0736
2040	41.019
2160	46.4523
2280	53.1511
2400	64.4914
2520	71.893
2640	81.9666
2760	94.2081
2880	108.139

A continuación, se muestran las gráficas obtenidas con gnuplot y su ajuste, donde se puede apreciar claramente la función característica de los algoritmos cúbicos.



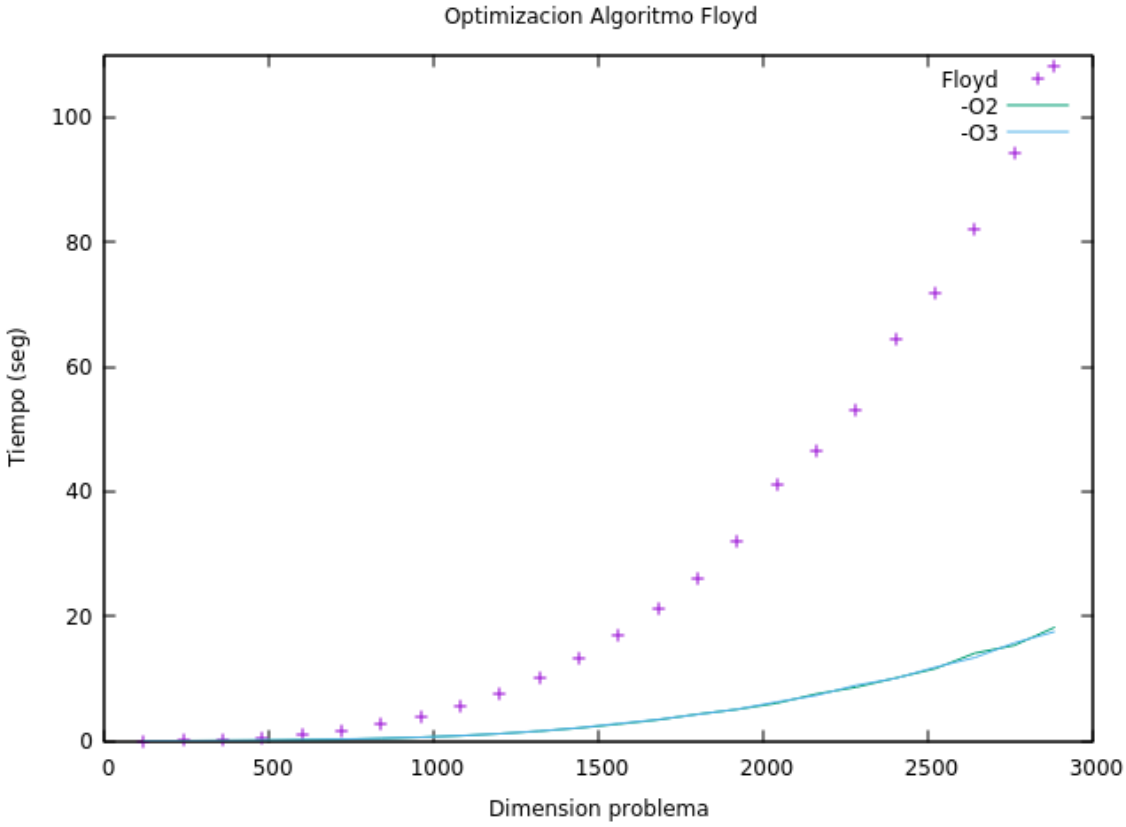
Enfoque híbrido: Floyd ajustado a T(n) = a0*x*x*x+a1*x*x+a2*x+a3



Valor de las constantes tras el ajuste:

```
gnuplot> f(x) = a0*x*x*x+a1*x*x+a2*x+a3
gnuplot> fit f(x) 'salidafloyd-media.dat' via a0,a1,a2,a3
iter      chisq       delta/lim  lambda   a0             a1             a2             a3
   0 2.2554348287e+21   0.00e+00  4.85e+09    1.000000e+00   1.000000e+00   1.000000e+00   1.000000e+00
   1 2.3972048478e+17  -9.41e+08  4.85e+08    9.916217e-03   9.996064e-01   9.999998e-01   1.000000e+00
   2 1.0119987475e+13  -2.37e+09  4.85e+07   -3.960648e-04   9.995592e-01   9.999998e-01   1.000000e+00
   3 1.0030840300e+13  -8.89e+02  4.85e+06   -3.954356e-04   9.952704e-01   9.999954e-01   1.000000e+00
   4 4.9020292940e+12  -1.05e+05  4.85e+05   -2.763616e-04   6.954540e-01   9.996890e-01   9.999997e-01
   5 2.5272703068e+09  -1.94e+08  4.85e+04   -6.031659e-06   1.479030e-02   9.989904e-01   9.999991e-01
   6 7.0555727082e+05  -3.58e+08  4.85e+03    2.465950e-07  -1.017593e-03   9.986744e-01   9.999984e-01
   7 6.6504990474e+05  -6.09e+03  4.85e+02    2.409612e-07  -9.915195e-04   9.695680e-01   9.999244e-01
   8 4.1775337443e+04  -1.49e+06  4.85e+01    6.323411e-08  -2.461046e-04   2.401820e-01   9.980685e-01
   9 1.3614683704e+01  -3.07e+08  4.85e+00    4.002387e-09   2.322316e-06  -2.902810e-03   9.972670e-01
  10 1.3136020132e+01  -3.64e+03  4.85e-01    3.810347e-09   3.122670e-06  -3.671005e-03   9.797081e-01
  11 1.2983595720e+01  -1.17e+03  4.85e-02    3.917880e-09   2.580700e-06  -2.873521e-03   6.659570e-01
  12 1.2978567382e+01  -3.87e+01  4.85e-03    3.941228e-09   2.463023e-06  -2.700367e-03   5.978361e-01
  13 1.2978567359e+01  -1.83e-04  4.85e-04    3.941279e-09   2.462767e-06  -2.699991e-03   5.976879e-01
iter      chisq       delta/lim  lambda   a0             a1             a2             a3

After 13 iterations the fit converged.
final sum of squares of residuals : 12.9786
rel. change during last iteration : -1.82638e-09

degrees of freedom    (FIT_NDF)                        : 20
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.805561
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.648928

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 3.94128e-09      +/- 3.687e-10    (9.356%)
a1              = 2.46277e-06      +/- 1.681e-06    (68.24%)
a2              = -0.00269999      +/- 0.002193     (81.23%)
a3              = 0.597688         +/- 0.7755       (129.8%)

correlation matrix of the fit parameters:
                a0     a1     a2     a3
a0              1.000
a1             -0.987  1.000
a2              0.926 -0.973  1.000
a3             -0.721  0.797 -0.899  1.000
gnuplot>
```

i) Comparativa Algoritmo de Floyd con Optimización –O2 y –O3



7) Algoritmo para Torres de Hanoi O(2^n)

Las Torres de Hanoi es un rompecabezas inventado en 1883 por el matemático francés Édouard Lucas.
Consiste en tres columnas y un número indeterminado de discos que definen la complejidad de la solución, en tanto su cantidad aumente.
Los discos tienen diferente tamaño y están ordenados de forma ascendente.
El objetivo del juego es traspasar todos los discos de la primera columna hacia la tercera manteniendo el orden ascendente (el disco más grande abajo), tal como estaban al inicio.
Las reglas del juego son:

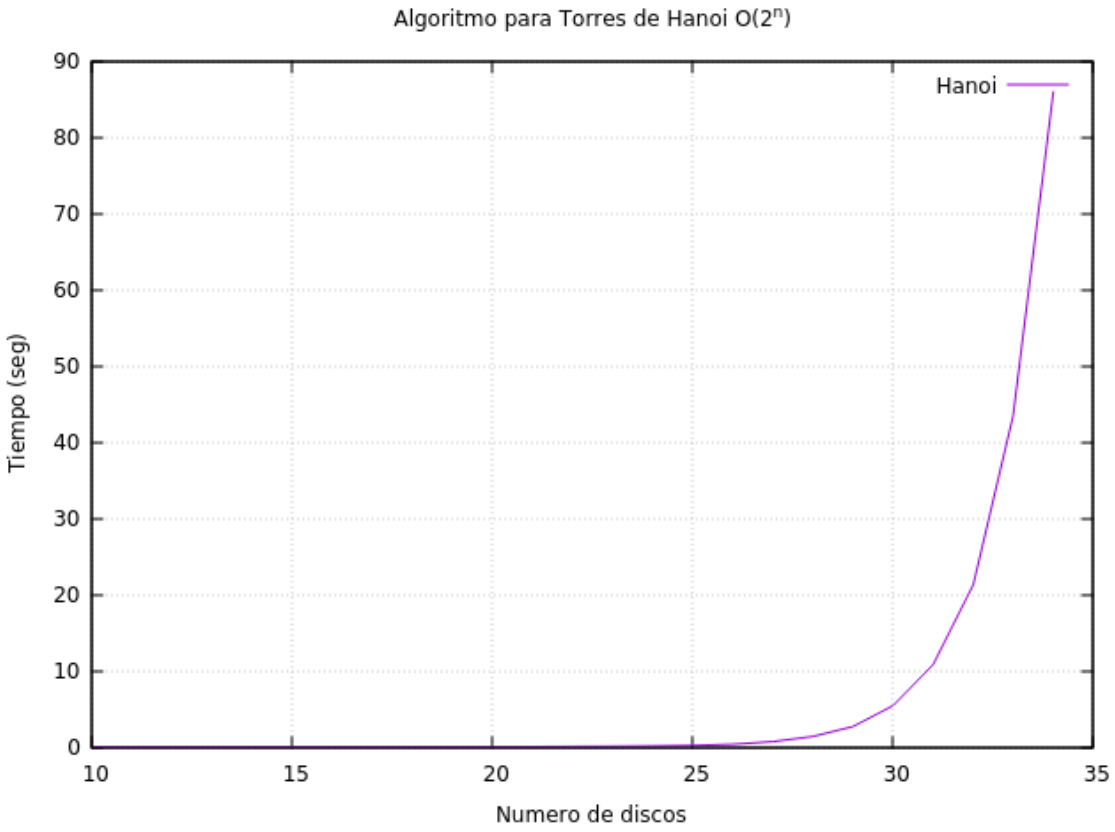
- Solo un disco se puede mover a la vez.
- Un disco de mayor tamaño no puede ponerse sobre uno más pequeño que él.
- Solo se puede desplazar el disco en la parte superior de una columna.

Para mover n discos son necesarios 2^n-1 pasos.
En nuestro caso se tiene una implementación recursiva del algoritmo, dado que es una función que se llama a sí misma hasta que se cumple la condición, aunque también se podría resolver de manera iterativa.

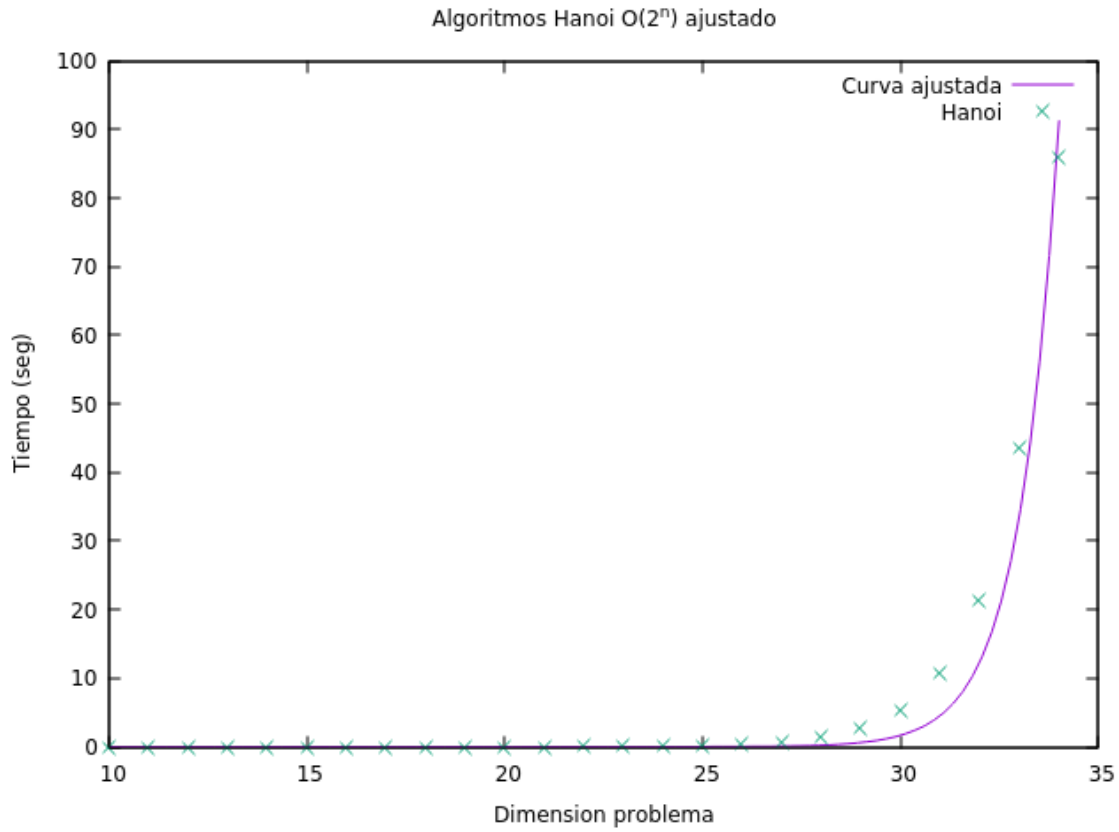
Para este tipo de algoritmos con una eficiencia teórica de orden $O(2^n)$ se han obtenido los siguientes tiempos de ejecución en función del tamaño de la entrada del problema, en este caso, el número de discos con los que se quiere resolver el rompecabezas. comenzando en 120, de 120 en 120, hasta 2880.

Orden de eficiencia $O(n^n)$	
Numero de Discos	Hanoi
10	2.0342e-05
11	3.98473e-05
12	7.76167e-05
13	0.000182285
14	0.000306611
15	0.000207873
16	0.000456533
17	0.000779292
18	0.0014682
19	0.00288642
20	0.00569848
21	0.0106663
22	0.0211244
23	0.0431034
24	0.0852143
25	0.168702
26	0.333921
27	0.663048
28	1.33978
29	2.65751
30	5.40084
31	10.7781
32	21.2854
33	43.5722
34	85.9018

A continuación, se muestran las gráficas obtenidas con gnuplot y su ajuste, donde se puede apreciar claramente la función característica de los algoritmos exponenciales.



Enfoque híbrido: Hanoi ajustado a T(n) = a0*exp(x)



Valor de la constante tras el ajuste:

```
gnuplot> f(x) = a0*exp(x)
gnuplot> fit f(x) 'salidahanoi-media.dat' via a0
iter      chisq       delta/lim  lambda   a0
   0 3.9371053125e+29   0.00e+00  1.25e+14    1.000000e+00
   1 5.8241202848e+26  -6.75e+07  1.25e+13    3.846154e-02
   2 9.3111420470e+19  -6.26e+11  1.25e+12    1.537846e-05
   3 1.4897709492e+09  -6.25e+15  1.25e+11    6.166991e-11
   4 2.6717646350e+02  -5.58e+11  1.25e+10    1.563033e-13
   5 2.6717646112e+02  -8.92e-04  1.25e+09    1.563009e-13
iter      chisq       delta/lim  lambda   a0

After 5 iterations the fit converged.
final sum of squares of residuals : 267.176
rel. change during last iteration : -8.92159e-09

degrees of freedom    (FIT_NDF)                        : 24
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 3.33652
variance of residuals (reduced chisquare) = WSSR/ndf   : 11.1324

Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a0              = 1.56301e-13      +/- 5.317e-15    (3.402%)
gnuplot>
```

i) Comparativa Algoritmo Hanoi con Optimización –O2 y –O3