Arboles (ABB y APO)

Calculo de eficiencia ABB O(n log n)

Eficiencia Empirica

En el caso de un arbol ABB recorrido en inorden. Estudiamos dos partes:

1. ¿Cuánto tarda en insertar? O(n*log(n))
2. ¿Cuánto tarda en listar? O(n)

Pero, al tener O(nlog(n)) + O(n) obtenemos como resultado O(nlog(n)).

Pasos a seguir para medir la eficiencia:

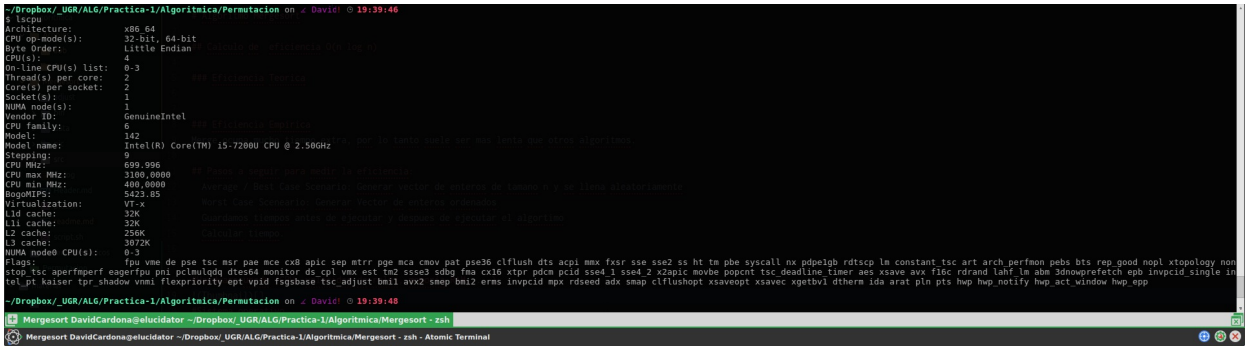
Average / Best Case Scenario: En estos casos, los cuales coinciden, se dan cuando el vector este poblado y repartido de valores.

Worst Case Sceneario: En el caso del peor caso, se da cuando el vector que nos dan está ordenado.

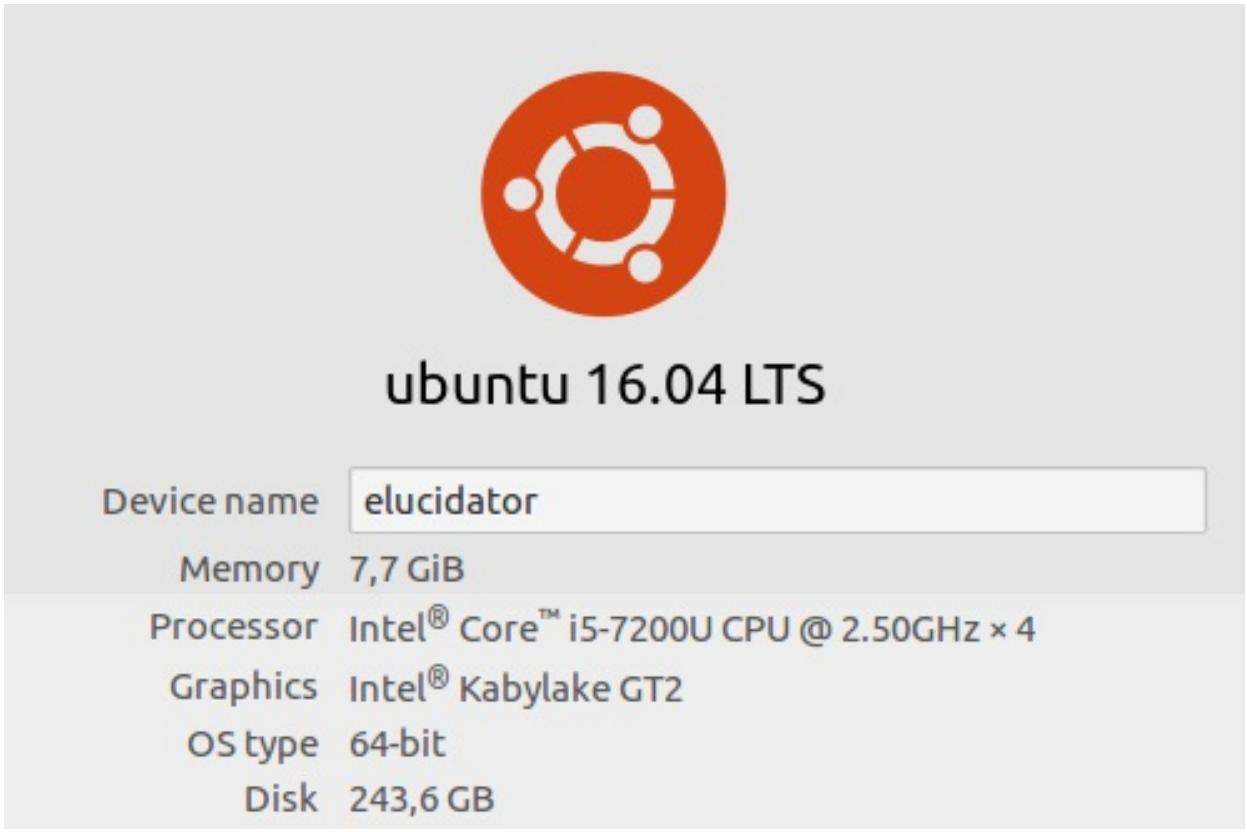
Guardamos tiempos antes de ejecutar y despues de ejecutar el algortimo Calcular tiempo.

Información

Hardware usado (CPU, velocidad de reloj, memoria RAM, ...)



Sistema operativo



Compilador utilizado y opciones de compilación

```
gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper
Target: x86_64-linux-gnu
gcc version 5.4.0 20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.9)

g++ usoabb.cpp -o usoabb
```

Desarrollo completo del cálculo de la eficiencia teórica y gráfica.

Empírica

La eficiencia empírica calculada utilizando 2 script que facilitan la obtencion de los .dat y mostrar las gráficas, mostrados a continuación:

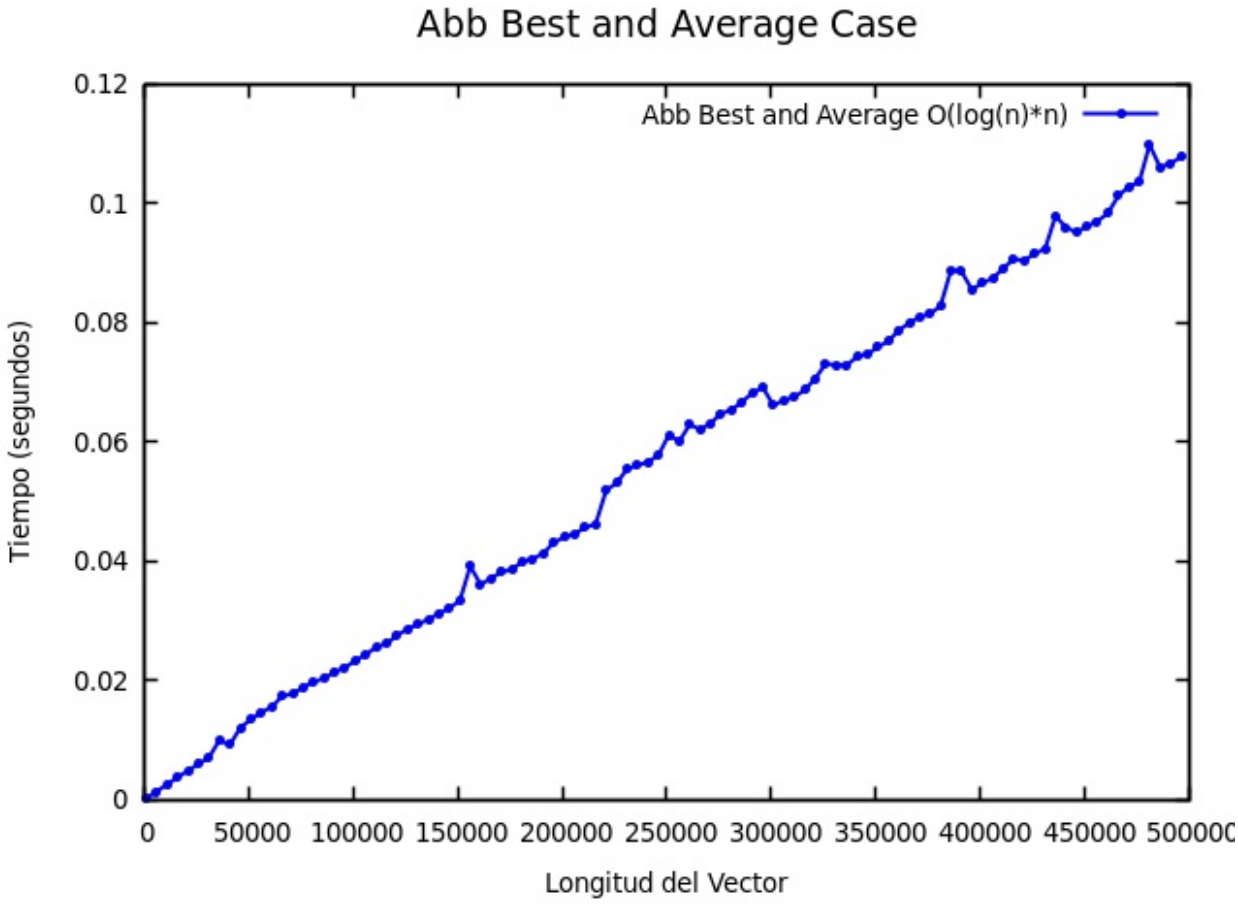
```
#!/bin/bash

# Script de bash para crear una imagen .png con GNU-PLOT

#Variables:
OUTPUT=./data/grafica_tiempo_best_case_abb.png
TITULO="Abb Best and Average Case"
XLABEL="Longitud del Vector"
YLABEL="Tiempo (segundos)"
LEYENDA="Abb Best and Average O(log(n)*n)"
FICHERO_DATOS="./data/tiempo_best_case_usoabb.dat"
COLOR=blue

gnuplot<<FIN
# Terminal para png:
set terminal pngcairo enhanced font 'Verdana,10'
set border linewidth 1.5

# Estilo de línea y color:
set style line 1 lc rgb '$COLOR' lt 1 lw 2 pt 7 pi 0 ps 0.5
set pointintervalbox 0

# Nombre de la imagen resultante:
set output '$OUTPUT'

# Título y ejes:
set title "$TITULO" enhanced font 'Verdana,14'
set xlabel "$XLABEL"
set ylabel "$YLABEL"

set autoscale

plot "$FICHERO_DATOS" title '$LEYENDA' with linespoints ls 1
FIN
```

```
#!/bin/bash

#########################################################################
# Algoritmica, Practica 1
# Medir el tiempo del algoritmo de un algoritmo de ordenación.
#########################################################################

# Script de bash que obtiene los datos para el algoritmo de burbuja

# Variables:
PROGRAMA=$1
SALIDA=./data/tiempo_best_case_$1.dat
MENSAJE_INICIO="Se inicia la ejecución del algoritmo $1:"
MENSAJE_FINAL="Fin de la ejecución. Se ha creado un fichero con los resultados."
```

```
# Se genera el ejecutable con el algoritmo de ordenación:
g++ -std=c++11 ./src/$PROGRAMA.cpp -o ./bin/$PROGRAMA
echo "$MENSAJE_INICIO"

# Variables:
INICIO=1000
FIN=500000
INCREMENTO=5000

i=$INICIO
echo > $SALIDA
while [ $i -le $FIN ]
do
    echo Vector size = $i
    echo "`./bin/$PROGRAMA $i 10000`" >> $SALIDA
    i=$((i+$INCREMENTO))
done

rm -fr ./bin/$PROGRAMA

echo "$MENSAJE_FINAL"
```

En el Best y Average Case:

```
INICIO=1000
FIN=500000
INCREMENTO=5000
```



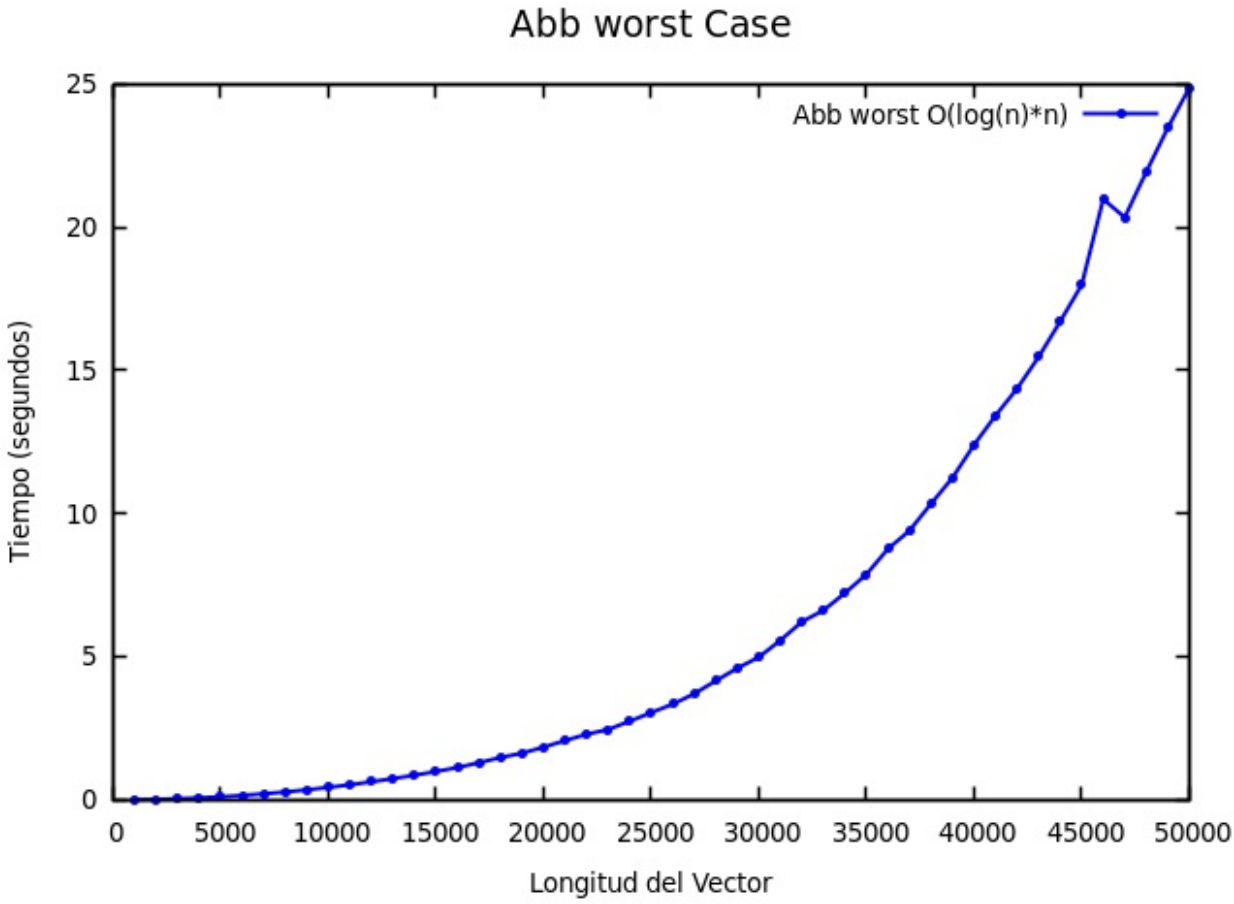
Datos obtenidos:

N Elementos	Tiempo
1000	0.000265
6000	0.001402
11000	0.002604
16000	0.003749
21000	0.004905
26000	0.006001
31000	0.007109
36000	0.009983
41000	0.00923

46000	0.012107
51000	0.013481
56000	0.014664
61000	0.015556
66000	0.017403
71000	0.017748
76000	0.018868
81000	0.019777
86000	0.020335
91000	0.021422
...	...

En el Worst Case:

```
INICIO=1000
FIN=30000
INCREMENTO=1000
```



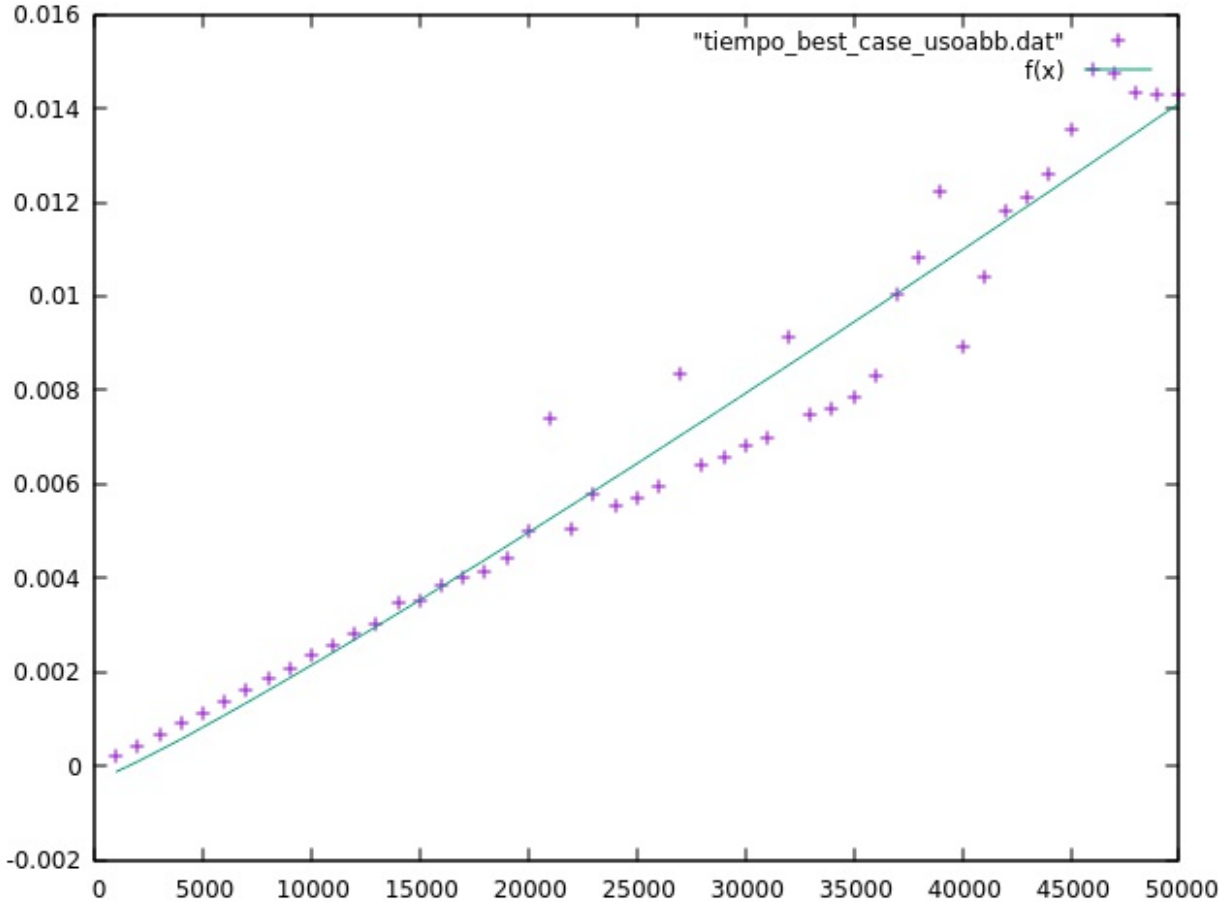
Datos obtenidos:

N Elementos	Tiempo
1000	0.003679
2000	0.017552
3000	0.038251
4000	0.067119
5000	0.106073
6000	0.152441
7000	0.209037
8000	0.2736
9000	0.349507

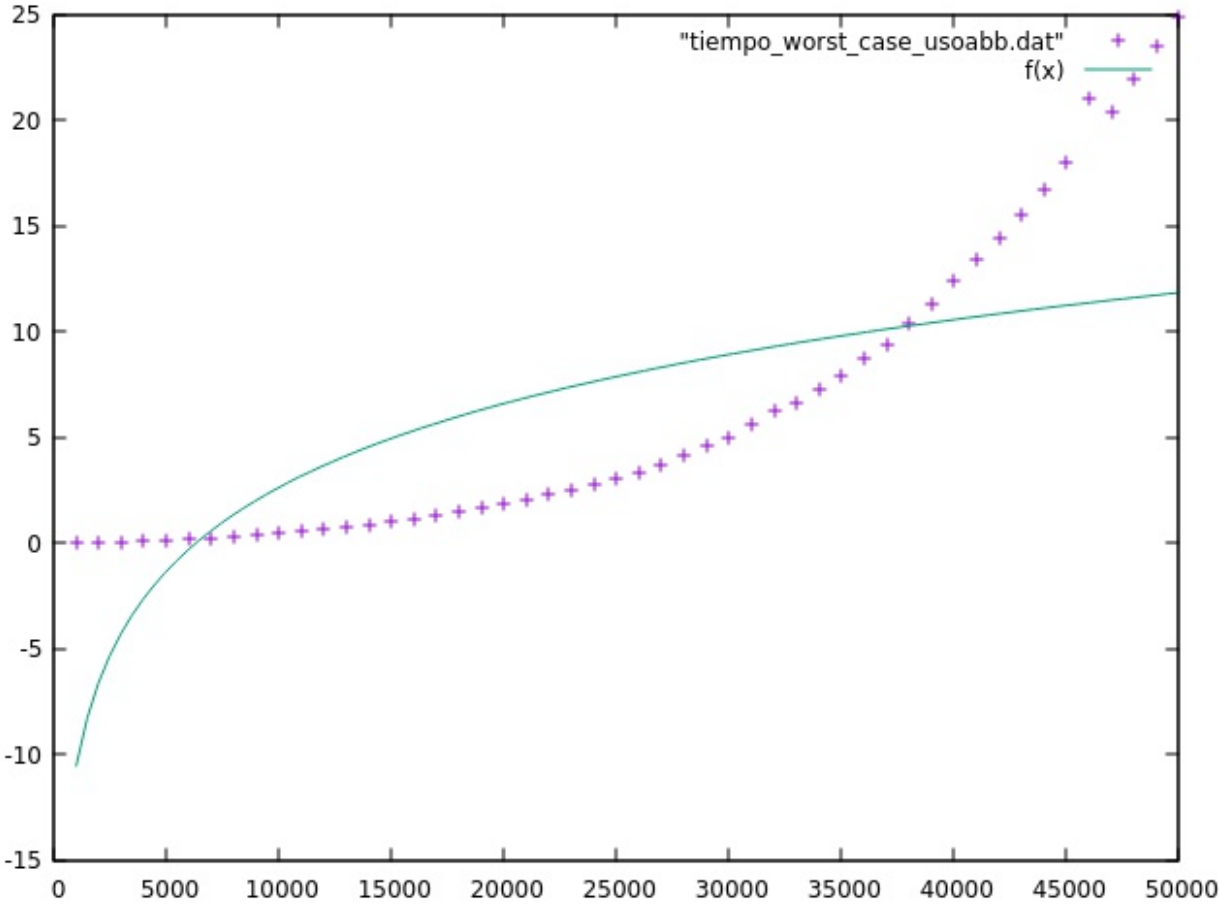
10000	0.433026
11000	0.526738
12000	0.630105
13000	0.742552
14000	0.855664
15000	0.989432
16000	1.12852
17000	1.29927
18000	1.47868
19000	1.63101
...	...

Ajuste de la curva teórica a la empírica: mostrar resultados del ajuste y gráfica.

Best and Average case


"tiempo_best_case_usoabb.dat"
f(x)
0.016
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
-0.002
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000

Worst case



Calculo de eficiencia APO O(n log n)

Eficiencia Empirica

En el caso de un arbol APO. Estudiamos una eficiencia de:

1. ¿Cuánto tarda en insertar? O(n*log(n))
2. ¿Cuánto tarda en listar? O(n*log(n))

Pero, al tener O(nlog(n)) + O(nlog(n)) obtenemos como resultado O(nlog(n)).

Pasos a seguir para medir la eficiencia:

En este caso Average, Best y Worst coinciden en eficiencia por tanto siguiendo los mismos pasos que en el ABB utilizaremos los script.

Desarrollo completo del cálculo de la

eficiencia teórica y gráfica.

Empírica

La eficiencia empírica calculada utilizando 2 script que facilitan la obtencion de los .dat y mostrar las gráficas, mostrados a continuación:

```
#!/bin/bash

# Script de bash para crear una imagen .png con GNU-PLOT

#Variables:
OUTPUT=./data/grafica_tiempo_best_case_abb.png
TITULO="Abb Best and Average Case"
XLABEL="Longitud del Vector"
YLABEL="Tiempo (segundos)"
LEYENDA="Abb Best and Average O(log(n)*n)"
FICHERO_DATOS="./data/tiempo_best_case_usoabb.dat"
COLOR=blue

gnuplot<<FIN
# Terminal para png:
set terminal pngcairo enhanced font 'Verdana,10'
set border linewidth 1.5

# Estilo de línea y color:
set style line 1 lc rgb '$COLOR' lt 1 lw 2 pt 7 pi 0 ps 0.5
set pointintervalbox 0

# Nombre de la imagen resultante:
set output '$OUTPUT'

# Título y ejes:
set title "$TITULO" enhanced font 'Verdana,14'
set xlabel "$XLABEL"
set ylabel "$YLABEL"

set autoscale

plot "$FICHERO_DATOS" title '$LEYENDA' with linespoints ls 1
FIN
```

```
#!/bin/bash

#####################################################################
# Algoritmica, Practica 1
# Medir el tiempo del algoritmo de un algoritmo de ordenación.
#####################################################################
```

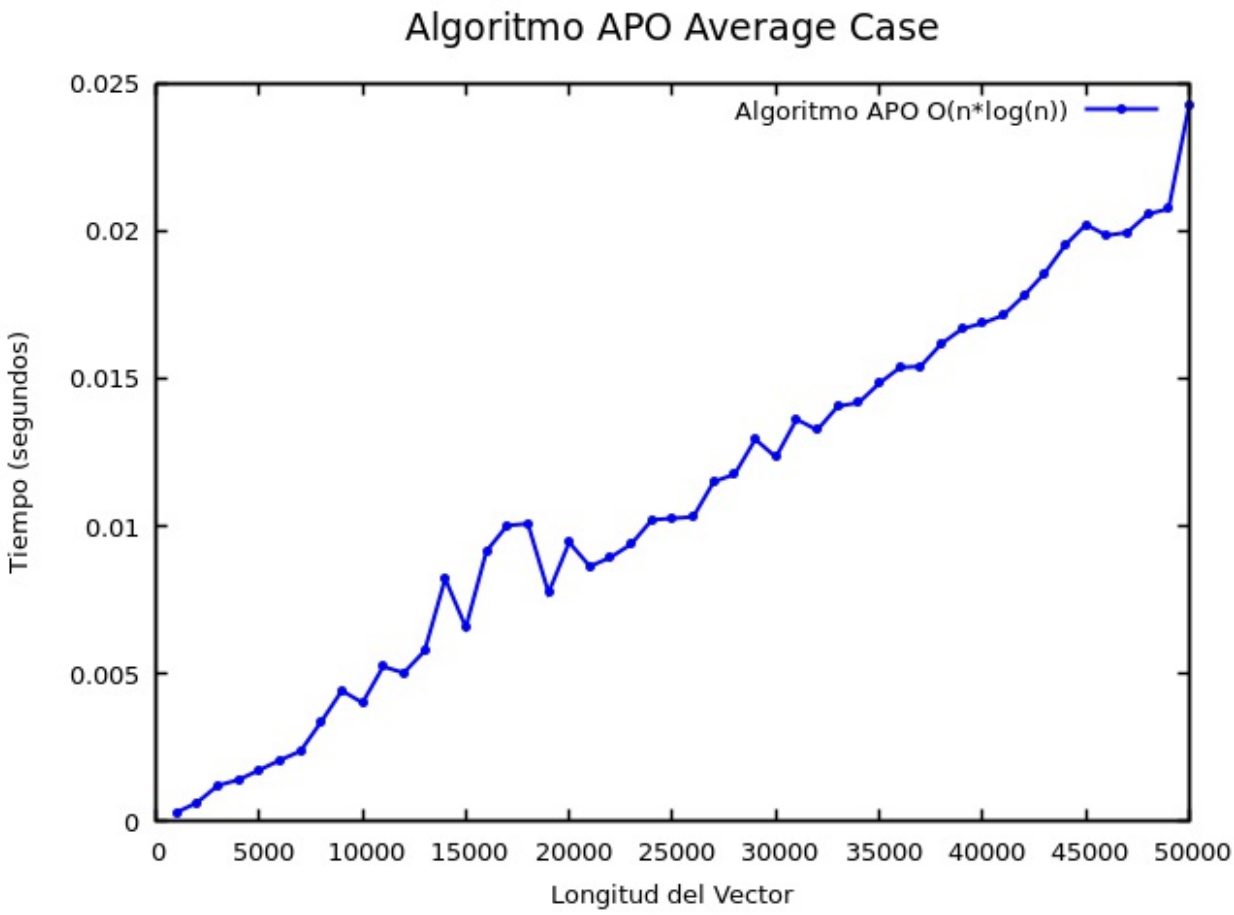
```
# Script de bash que obtiene los datos para el algoritmo de burbuja

# Variables:
PROGRAMA=$1
SALIDA=./data/tiempo_best_case_$1.dat
MENSAJE_INICIO="Se inicia la ejecución del algoritmo $1:"
MENSAJE_FINAL="Fin de la ejecución. Se ha creado un fichero con los resultados."

# Se genera el ejecutable con el algoritmo de ordenación:
g++ -std=c++11 ./src/$PROGRAMA.cpp -o ./bin/$PROGRAMA
echo "$MENSAJE_INICIO"

# Variables:
INICIO=1000
FIN=30000
INCREMENTO=1000

i=$INICIO
echo > $SALIDA
while [ $i -le $FIN ]
do
    echo Vector size = $i
    echo "`./bin/$PROGRAMA $i 10000`" >> $SALIDA
    i=$((i+$INCREMENTO))
done

rm -fr ./bin/$PROGRAMA

echo "$MENSAJE_FINAL"
```

Datos obtenidos:

N Elementos	Tiempo
1000	0.000299514
2000	0.000635297
3000	0.00121193
4000	0.00140474
5000	0.00173683
6000	0.00206811
7000	0.00237509
8000	0.00337264

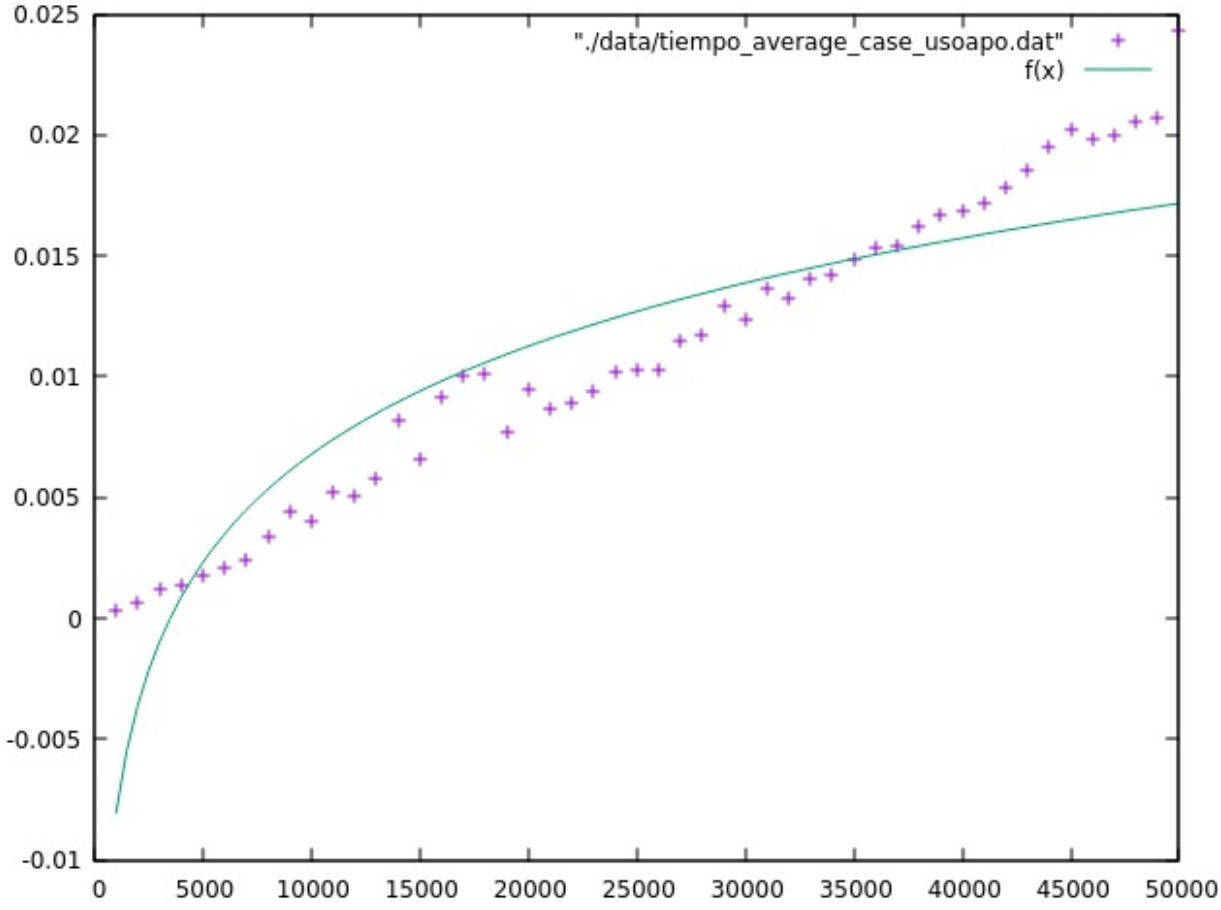
9000	0.00442333
10000	0.00401548
11000	0.00524227
12000	0.00502815
13000	0.00577124
14000	0.0082218
16000	0.00915717
17000	0.0100221
15000	0.00656923
18000	0.0100829
19000	0.00774256
20000	0.00945171
...	...

En el cualquiera de los 3 casos:



Ajuste de la curva teórica a la empírica: mostrar resultados del ajuste y gráfica.

Ajuste gráfico:



Archivo fit:

```
*******************************************************************************
Sat Mar 17 20:55:11 2018


FIT:    data read from "./data/tiempo_average_case_usoapo.dat"
        format = z
        #datapoints = 50
        residuals are weighted equally (unit weight)

function used for fitting: f(x)
    f(x) = (a*(log(x)/log(2))+b)
fitted parameters initialized with current variable values

iter      chisq       delta/lim  lambda   a             b
   0 1.1691052659e+04   0.00e+00  1.01e+01    1.000000e+00   1.000000e+00
   5 3.4672146434e-04  -2.54e-03  1.01e-04    4.476003e-03  -5.269472e-02

After 5 iterations the fit converged.
final sum of squares of residuals : 0.000346721
rel. change during last iteration : -2.54315e-08

degrees of freedom    (FIT_NDF)                        : 48
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.00268763
variance of residuals (reduced chisquare) = WSSR/ndf   : 7.22336e-06
```

```
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = 0.004476         +/- 0.0002993    (6.687%)
b               = -0.0526947       +/- 0.004282     (8.126%)

correlation matrix of the fit parameters:
                a      b
a               1.000
b              -0.996  1.000
```