Práctica 1

Pablo Baeyens Fernández
Antonio Checa Molina
Iñaki Madinabeitia Cabrera
José Manuel Muñoz Fuentes
Darío Sierra Martínez

Algorítmica

Índice

1. Ejercicio 1

En el ejercicio 1 se nos pide hallar la eficiencia empírica de los algoritmos presentados en la sesión. Para ello hemos modificado los códigos de los programas para que tengan como salida el tamaño que se les pasó como argumento junto al tiempo que han tardado en realizar la tarea del respectivo algoritmo. Usando la biblioteca *chrono* y la estructura que se menciona en la sesión se consiguen medir los tiempos de los programas de forma precisa.

Una vez hecho esto, se elaboró un script de bash para automatizar la ejecución de cada algoritmo con varios tamaños y recoger todas las salidas en un archivo para luego poder usar esos datos. El script es el siguiente:

```
# !/bin/bash
# Uso: nombredelejecutable inicial salto final
# Ejemplo: fibonacci 10 5 80 ejecuta fibonacci 10, fibonacci 15, ..., fibonacci 80
# Salida: nombredelejecutable.dat

> $(basename $1 .exe).dat
for i in $(seq $2 $3 $4); do
  ./$1 $i >> $(basename $1 .exe).dat
done
```

Las tablas generadas con este proceso se presentan a continuación, habiendo una tabla para cada orden de eficiencia y una última table que reúne los tiempos de todos los algoritmos de ordenación.

Todos los tiempos que se muestran en las tablas están en segundos, mientras que *Tamaño* menciona el tamaño total de la muestra con la que se obtuvo cada tiempo de ejecución: en los algoritmos de ordenación hace referencia a la capacidad del vector que se ordenaba, mientras que en Fibonacci o Hanói hace referencia a las iteraciones totales del programa.

Debido al tamaño de las tablas (ya que cada una debía tener al menos 25 entradas), cada tabla se muestra en una página.

Cabe destacar que la diferencia entre algoritmos de ordenación de un orden y de otro resulta tan pronunciada que los algoritmos de orden $n \cdot \log(n)$ se han ejecutado una segunda vez con un rango de valores en el que obtienen tiempos de ejecución minúsculos, muchos órdenes de magnitud por debajo del obtenido por los otros algoritmos de ordenación. En la tabla en la que se presentan se usan tamaños mucho mayores, mientras que en la tabla comparativa de algoritmos de ordenación toman estos mismos valores.

1.1. Tabla de los algoritmos cuadráticos

Tamaño	Tiempo de Burbuja	Tiempo de Inserción	Tiempo de Selección
2000	0,0141	0,005	0,0057
4000	0,0516	0,0211	0,023
6000	0,1138	0,0423	0,0485
8000	0,203	0,0749	0,0853
10000	0,3178	0,1194	0,1335
12000	0,4675	0,1719	0,1922
14000	0,6378	0,2302	0,2608
16000	0,8319	0,3068	0,3446
18000	1,0515	0,3948	0,4365
20000	1,2976	0,4715	0,5323
22000	1,5667	0,5756	0,6479
24000	1,8566	0,6691	0,7654
26000	2,1866	0,8164	0,9028
28000	2,5267	0,9151	1,0435
30000	2,9087	1,0581	1,2003
32000	3,3202	1,1995	1,3639
34000	3,7281	1,3595	1,5385
36000	4,1982	1,5143	1,723
38000	4,6725	1,684	1,9174
40000	5,1504	1,8823	2,1246
42000	5,6988	2,0988	2,3405
44000	6,2511	2,2693	2,5674
46000	6,8294	2,4624	2,8047
48000	7,4305	2,6661	3,054
50000	8,0857	2,9284	3,3125

1.2. Tabla de los algoritmos $n \cdot log(n)$

Tamaño	Tiempo de Heapsort	Tiempo de Mergesort	Tiempo de Quicksort
1000000	0,158862	0,25279	0,245436
2000000	0,333955	0,55784	0,508929
3000000	0,512390	0,92310	0,883956
4000000	0,689444	1,26719	1,068840
5000000	0,875720	1,66226	1,422010
6000000	1,063160	2,06275	1,804380
7000000	1,254210	2,46109	1,880770
8000000	1,448250	2,89135	2,214050
9000000	1,636350	3,30651	2,563270
10000000	1,802540	3,69426	2,943020
11000000	2,013440	4,14343	3,320140
12000000	2,211360	4,60380	3,736370
13000000	2,399370	5,06756	4,035230
14000000	2,589330	5,51359	3,893640
15000000	2,786440	5,96415	4,248670
16000000	2,984850	6,40422	4,579140
17000000	3,174740	6,93151	4,953660
18000000	3,358080	7,39067	5,305020
19000000	3,563030	7,85371	5,679610
20000000	3,803150	8,35113	6,070570
21000000	3,950770	8,85375	6,459050
22000000	4,171410	9,26481	6,862640
23000000	4,371020	9,70679	7,275080
24000000	4,558990	10,26190	7,763610
25000000	4,774830	10,71420	8,117880

1.3. Tabla del algoritmo cúbico (Floyd)

Tamaño	Tiempo de Floyd
30	0,000280424
60	0,001601650
90	0,005779860
120	0,011250300
150	0,021107200
180	0,035468500
210	0,055724200
240	0,082467900
270	0,119058000
300	0,160319000
330	0,213182000
360	0,281825000
390	0,358154000
420	0,437591000
450	0,543534000
480	0,652500000
510	0,790683000
540	0,929593000
570	1,097870000
600	1,279820000
630	1,479150000
660	1,703250000
690	1,941560000
720	2,199710000
750	2,485840000

1.4. Tabla del algoritmo de Fibonacci ($O(\frac{1+\sqrt{5}}{2})^n$)

Tamaño	Tiempo de Fibonacci
1	1.79e-07
2	1.88e-07
3	2.05e-07
4	2.89e-07
5	2.61e-07
6	2.87e-07
7	4.65e-07
8	7.53e-07
9	1.131e-06
10	1.365e-06
11	1.796e-06
12	1.671e-06
13	3.612e-06
14	7.294e-06
15	7.834e-06
16	1.2102e-05
17	1.7977e-05
18	2.9478e-05
19	5.3614e-05
20	7.2343e-05
21	0.000132759
22	0.000189094
23	0.000315282
24	0.000499445
25	0.000675111

1.5. Tabla del algoritmo de Hanoi ($O(2^n)$)

Tamaño	Tiempo de Hanoi
1	1.85e-07
2	2.52e-07
3	3.63e-07
4	5.65e-07
5	5.96e-07
6	1.206e-06
7	1.909e-06
8	3.047e-06
9	5.639e-06
10	1.1064e-05
11	2.1002e-05
12	3.7705e-05
13	7.8102e-05
14	0.000165379
15	0.000334135
16	0.000589484
17	0.000988963
18	0.0017404
19	0.00368885
20	0.00684205
21	0.0136331
22	0.0273772
23	0.05418
24	0.107202
25	0.214262

1.6. Tabla de los algoritmos de ordenación

Tamaño	Burbuja	Inserción	Selección	Heapsort	Mergesort	Quicksort
2000	0,014089400	0,00501	0,005698	0,000235	0,000272	0,000255
4000	0,051553600	0,02113	0,022951	0,000509	0,000580	0,000381
6000	0,113751000	0,04231	0,048492	0,000883	0,001106	0,000627
8000	0,202951000	0,07491	0,085303	0,001128	0,001366	0,000811
10000	0,317834000	0,11939	0,133457	0,001417	0,001760	0,001152
12000	0,467509000	0,17190	0,192178	0,001813	0,002363	0,001354
14000	0,637843000	0,23016	0,260748	0,002180	0,002288	0,001517
16000	0,831875000	0,30682	0,344565	0,002435	0,002648	0,001768
18000	1,051490000	0,39484	0,436515	0,002733	0,003157	0,001986
20000	1,297600000	0,47146	0,532265	0,003284	0,003823	0,002305
22000	1,566720000	0,57562	0,647896	0,003487	0,004252	0,002436
24000	1,856610000	0,66907	0,765415	0,003741	0,004980	0,002720
26000	2,186600000	0,81644	0,902801	0,004212	0,004381	0,002916
28000	2,526680000	0,91512	1,043460	0,004471	0,004894	0,003335
30000	2,908670000	1,05814	1,200250	0,004781	0,005159	0,003443
32000	3,320190000	1,19949	1,363910	0,005306	0,005530	0,003685
34000	3,728090000	1,35952	1,538500	0,005509	0,006219	0,003967
36000	4,198150000	1,51431	1,723020	0,005901	0,006601	0,004178
38000	4,672490000	1,68403	1,917410	0,006102	0,007015	0,004356
40000	5,150410000	1,88228	2,124580	0,006620	0,007535	0,004715
42000	5,698770000	2,09879	2,340510	0,006936	0,008117	0,004858
44000	6,251140000	2,26926	2,567410	0,007331	0,009466	0,005189
46000	6,829350000	2,46238	2,804650	0,007502	0,009306	0,005421
48000	7,430450000	2,66613	3,053950	0,007841	0,009883	0,005697
50000	8,085720000	2,92835	3,312500	0,008407	0,010657	0,005931

2. Ejercicio 2

En el ejercicio 2 se pide realizar las gráficas de los algoritmos, que hemos realizado metiendo en *gnuplot* los datos mostrados en las tablas anteriores, con los siguientes resultados:

2.1. Gráfica de los algoritmos cuadráticos

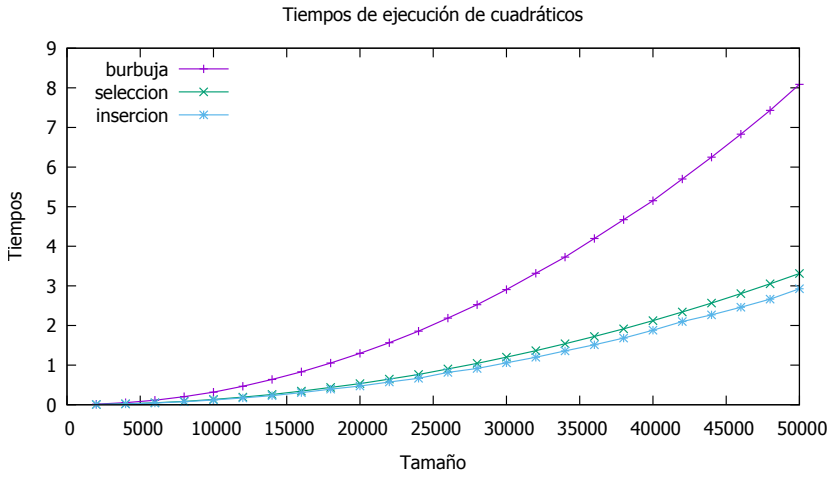

Figura 1: Algoritmos cuadráticos

El algoritmo de la burbuja requiere más del doble de tiempo que cualquiera de los otros dos, que se encuentran casi igualados, teniendo un menor tiempo el de inserción. Probablemente este menor tiempo sea el motivo de que en *quicksort* y *mergesort* se use el algoritmo de inserción para subvectores pequeños.

2.2. Gráfica de los algoritmos $n \cdot log(n)$

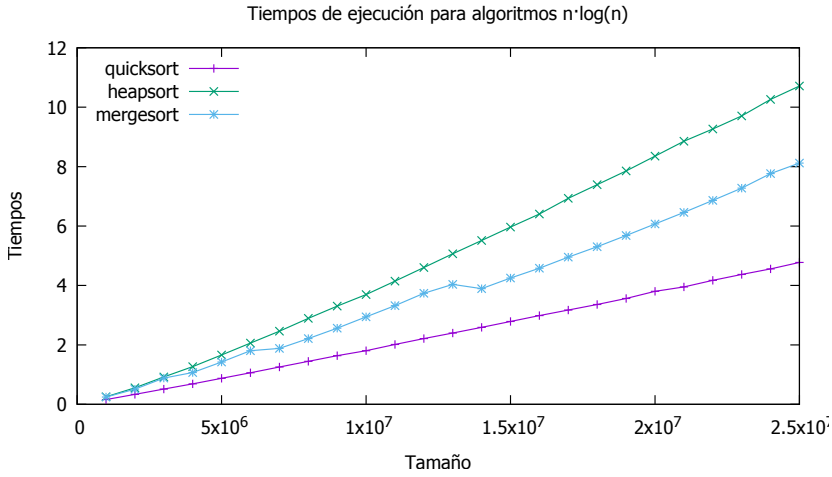

Figura 2: Algoritmos $n \cdot log(n)$

En el algoritmo *mergesort* se aprecian pequeños escalones. Probablemente en cada escalón se incrementa el número de llamadas recursivas, de forma que el tamaño de los subvectores para los que se llama a la función de ordenado por selección (que actúa para tamaños de entrada pequeños) es menor.

El algoritmo *quicksort* también usa el algoritmo de inserción por debajo de cierto umbral, pero usa un umbral más pequeño. Esto puede ser la causa de que no presente los escalones.

2.3. Gráfica del algoritmo de Floyd (cúbico)

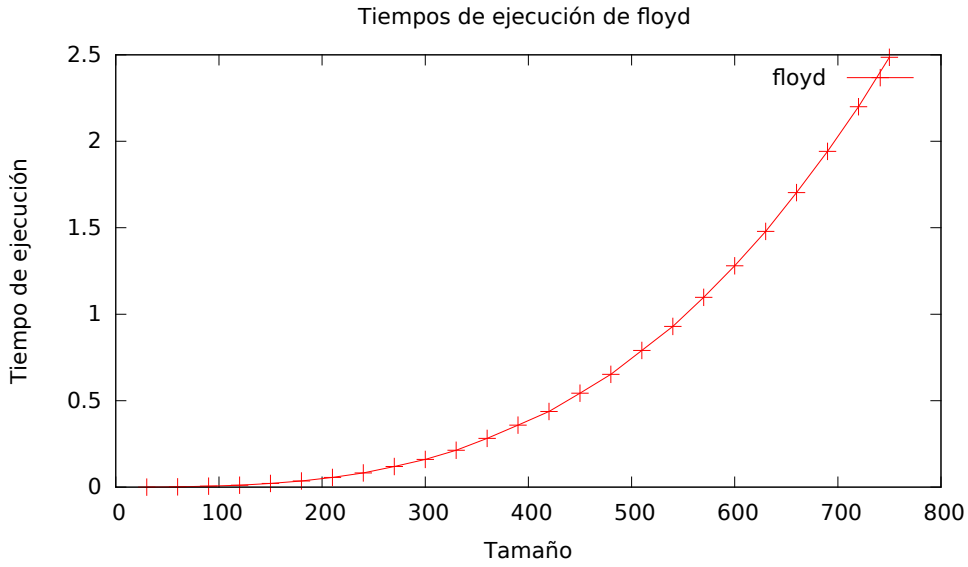

Figura 3: Algoritmo de Floyd (cúbico)

2.4. Gráfica del algoritmo de Fibonacci $(O(\frac{1+\sqrt{5}}{2})^n)$

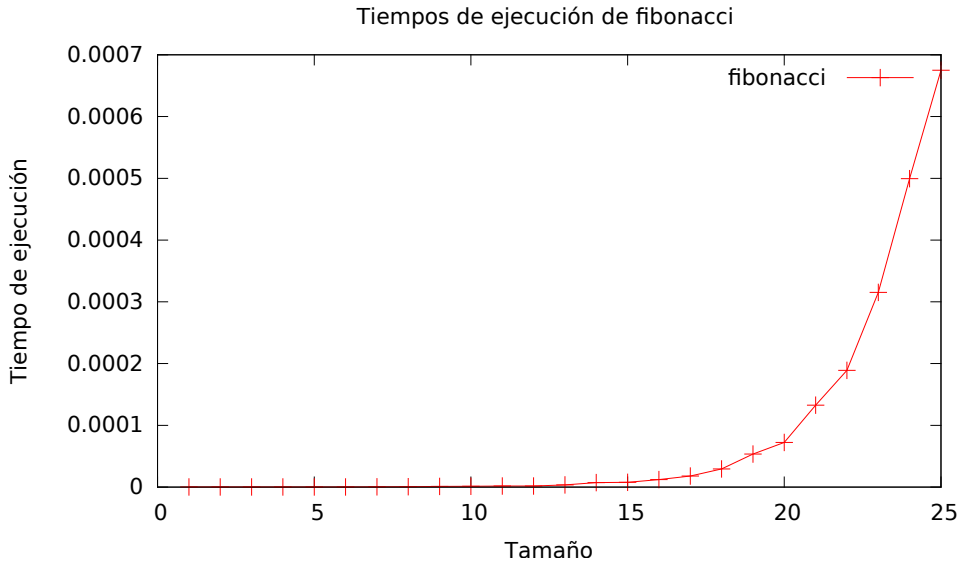

Figura 4: Algoritmo de Fibonacci

2.5. Gráfica del algoritmo de Hanoi ($O(2^n)$)

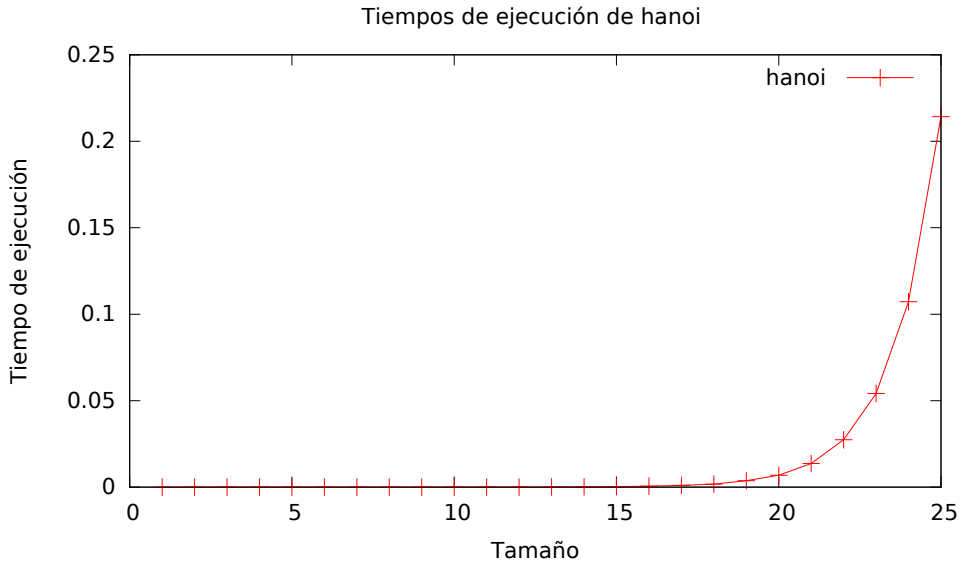

Figura 5: Algoritmo de Hanoi

2.6. Gráfica de los algoritmos de ordenación

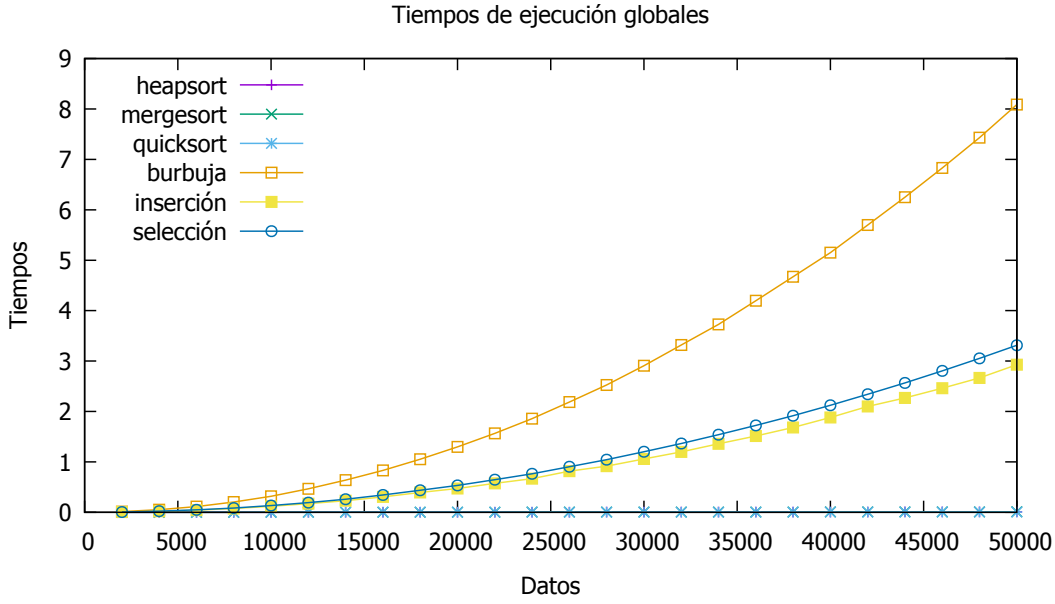

Figura 6: Algoritmos de ordenación

La diferencia en el tiempo de ejecución de los algoritmos de un orden de eficiencia y de otro provoca que los tiempos de los algoritmos de orden de eficiencia $n \cdot \log(n)$ se muestren como una línea horizontal.