INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

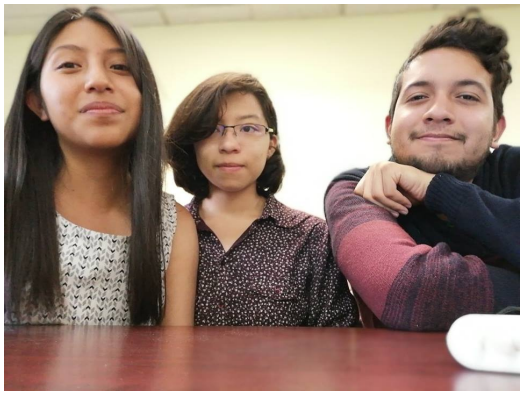
Práctica 1 - Pruebas a posteriori (Algoritmos de Ordenamiento)

Unidad de aprendizaje: Análisis de Algoritmos

Grupo: 3CM3

Alumnos(a): "La naranja mecánica"
Nicolás Sayago Abigail
Parra Garcilazo Cinthya Dolores
Ramos Díaz Enrique

Profesor(a): Edgardo Adrián Franco Martínez



12 de Septiembre 2018

Índice

1. Planteamiento del problema

Existen diversos métodos de ordenamiento, en este documento se analizaran 5, se observara y comparará el comportamiento de cada uno, para determinar el mejor de todos.

Este es el llamado Análisis a priori, en donde se tomaran resultados experimentales en una plataforma determinada para determinar la complejidad temporal y espacial de los siguientes algoritmos: Burbuja Simple, Burbuja Optimizada, Inserción, Selección, Shell y Árbol de Búsqueda Binario.

2. Plataforma Experimental

Especificaciones de Hardware:

- CPU: Intel Core-i5 6500 3.2 GHz
- Memoria: RAM DDR4 5.9 GB 2133 MHz

Compilador: GCC version 7.3.0 desde la Terminal

Sistema Operativo: Linux Ubuntu 18.04.1 LTS x64

3. Actividades y Pruebas

3.1. Burbuja simple

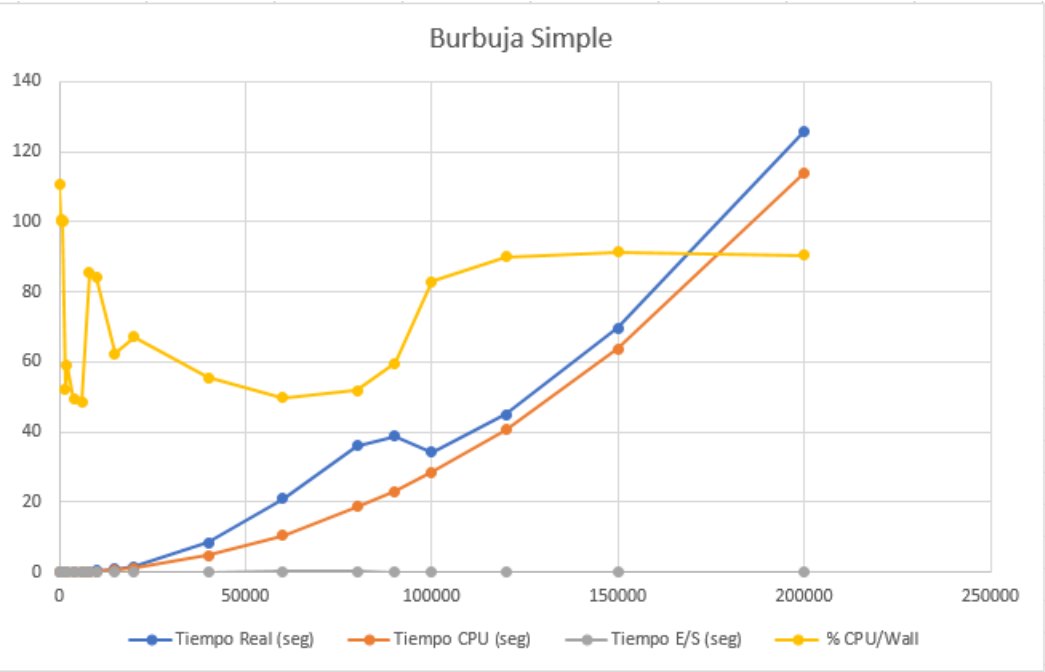
3.1.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc BurbujaSimple.c tiempo.c -o BurbujaSimple
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./BurbujaSimple 20 <numeros10millones.txt
n = 20

BurbujaSimple
real (Tiempo total)  0.000001907348632812500000000000000000 s
user (Tiempo de procesamiento en CPU) 0.000000000000000000000000000000000000 s
sys (Tiempo en acciónes de E/S)  0.000005999999999999929001237575221 24 s
CPU/Wall   314.572799999996277620084583759307861 33 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
------------------------------
```

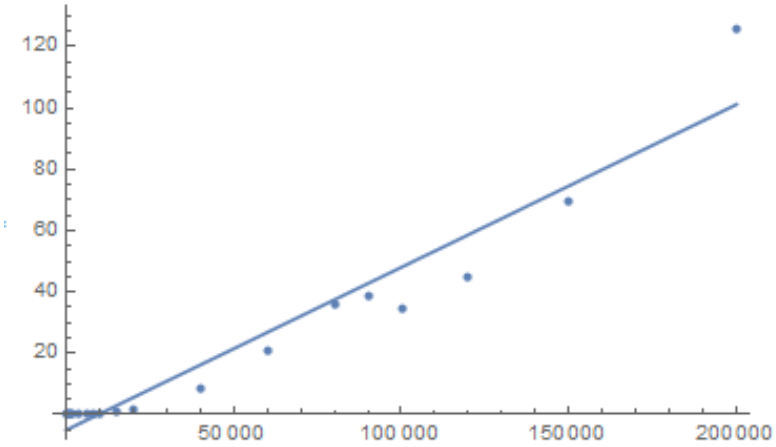
3.1.2. Análisis Temporal

Algoritmo Burbuja Simple	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	3.79086E-05	4.2E-05	0	110.7929358
500	0.000477076	0.00048	0	100.6129895
800	0.001088142	0	0.001091	100.2626131
1,000	0.001725912	0.001728	0	100.1209741
1,500	0.007205963	0.002712	0.001045	52.1373747
2,000	0.012443066	0.005839	0.001488	58.88420274
4,000	0.074679852	0.034934	0.001831	49.23014608
6,000	0.156076193	0.075393	0	48.30525311
8,000	0.171564102	0.14566	0.000789	85.36109719
10,000	0.288758993	0.243166	0	84.21071058
15,000	0.918097019	0.566284	0.00666	62.40560507
20,000	1.580190897	1.046458	0.013206	67.05923962
40,000	8.452590942	4.667328	0.028097	55.55012696
60,000	20.97909689	10.331053	0.107508	49.75696073
80,000	36.00872087	18.579333	0.096787	51.86554686
90,000	38.71759105	23.103081	0.005671	59.68540752
100,000	34.32539296	28.490575	0.006625	83.02075385
120,000	45.06316805	40.596333	0.006702	90.1024867
150,000	69.61539507	63.610683	0.027423	91.4138402
200,000	125.895797	113.9256	0.016438	90.50503806

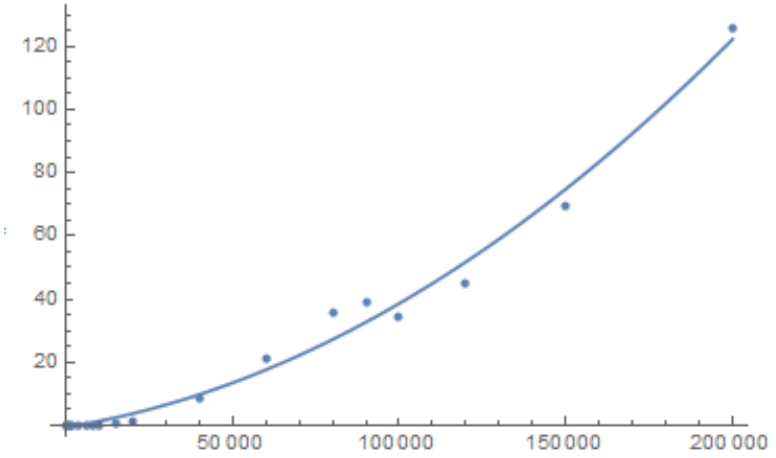
3.1.3. Gráfica de comportamiento



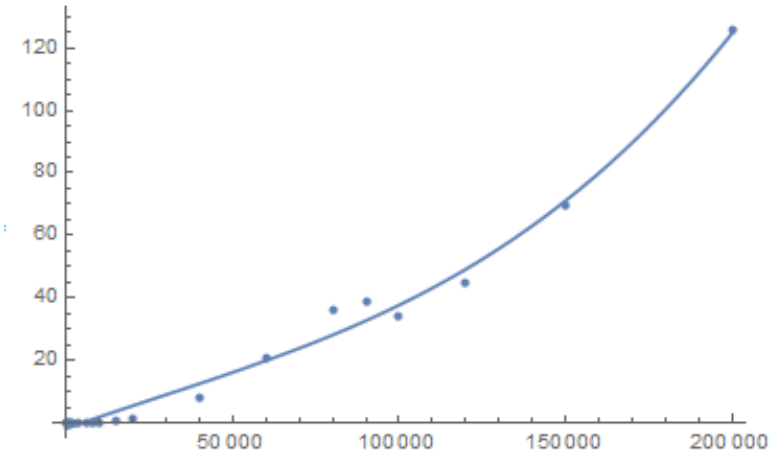
3.1.4. Aproximaciones Polinomiales



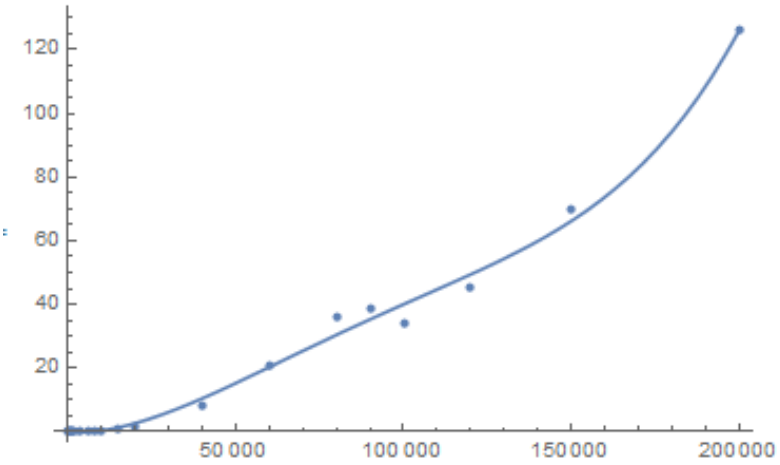
Grado 1: $-5.00391 + 0.000530695x$



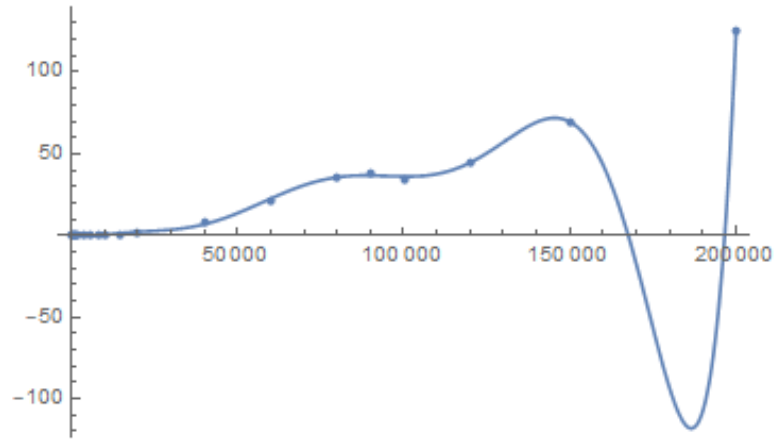
Grado 2: $-0.519656 + 0.0001691x + 2.22402 \times 10^{-9}x^2$



Grado 3: $-1.67871 + 0.000378947x - 9.51029 \times 10^{-10}x^2 + 1.11022 \times 10^{-14}x^3$



Grado 4: $-0.141273 - 0.0000406877x + 1.08914 \times 10^{-8}x^2 - 9.16426 \times 10^{-14}x^3 + 2.70052 \times 10^{-19}x^4$



Grado 8: $0.516742 - 0.000566326x + 9.99756 * 10^{-}8x^2 - 5.90874 * 10^{-}12x^3 + 1.70318 \times 10^{-16}x^4 - 2.50739 \times 10^{-21}x^5 + 1.94326 \times 10^{-26}x^6 - 7.5416 \times 10^{-32}x^7 + 1.15235 \times 10^{-37}x^8$

3.1.5. Evaluación de n's en Polinomios

n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	26529.7	5.56×10^6	1.38×10^9	1.67×10^12	4.44×10^24
100,000,000	53064.45	2.22×10^7	1.109×10^10	2.69×10^13	1.14×10^27
500,000,000	265342.28	5.56×10^8	1.38×10^12	1.68×10^16	4.49×10^32
1,000,000,000	530689.57	2.22×10^9	1.11×10^13	2.69×10^17	1.15×10^35
5,000,000,000	2.65×10^6	5.56×10^10	1.38×10^15	1.68×10^20	4.5×10^40

3.2. Burbuja Optimizada

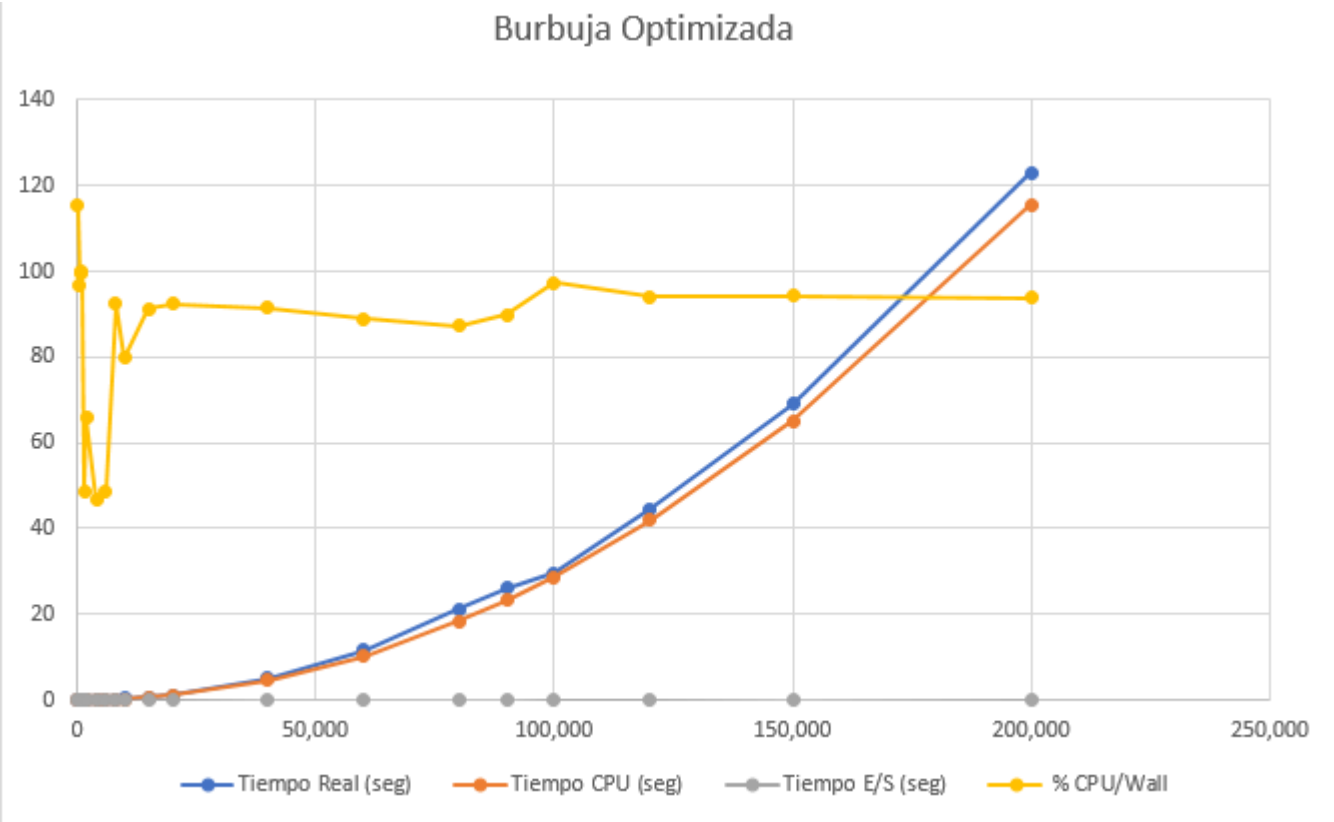
3.2.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc BurbujaOptimizada.c tiempo.c -o BurbujaOptimizada
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./BurbujaOptimizada 20 <numeros10millones.txt
n = 20

BurbujaOptimizada
real (Tiempo total)  0.000000953674316406250000000000000000 s
user (Tiempo de procesamiento en CPU) 0.000005000000000000013114509478384666 s
sys (Tiempo en acciónes de E/S)  0.000000000000000000000000000000000000 s
CPU/Wall   524.288000000001375155989080667495727540 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
------------------------------
```

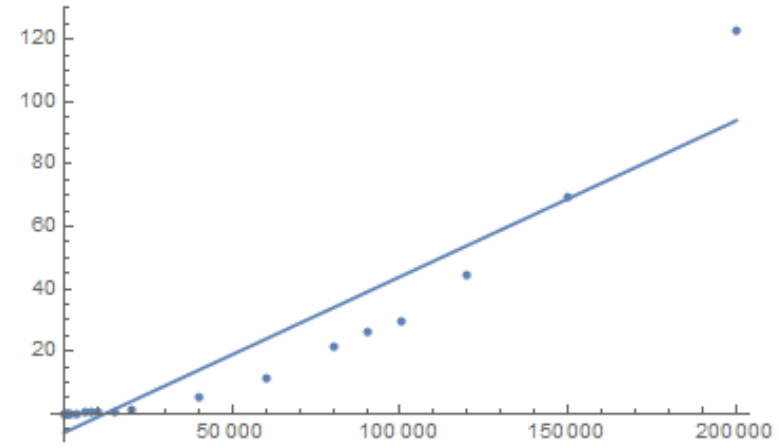
3.2.2. Análisis Temporal

Algoritmo Burbuja Simple Optimizada	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	2.59876E-05	0.00003	0	115.43956
500	0.000544071	0.000527	0	96.8623229
800	0.001171112	0.001166	0	99.5634866
1,000	0.001971006	0.001971	0	99.9996756
1,500	0.008566141	0.004172	0	48.7033769
2,000	0.010462046	0.005508	0.00138	65.8379844
4,000	0.06998992	0.030378	0.002312	46.706726
6,000	0.165845156	0.080588	0	48.5923147
8,000	0.156778097	0.144978	0	92.4733765
10,000	0.296716928	0.236724	0	79.7810901
15,000	0.615751028	0.558922	0.00263	91.1978989
20,000	1.121236086	1.034315	0.001162	92.3513801
40,000	4.966567039	4.540853	0	91.4284044
60,000	11.57947397	10.308512	0	89.0240094
80,000	21.15674901	18.486901	0	87.380632
90,000	25.99217796	23.336727	0	89.7836535
100,000	29.47539186	28.659385	0.000119	97.2319694
120,000	44.48436809	41.876134	0.003294	94.144145
150,000	69.18493414	65.204462	0	94.24662
200,000	123.0501759	115.45371	0.016621	93.8400349

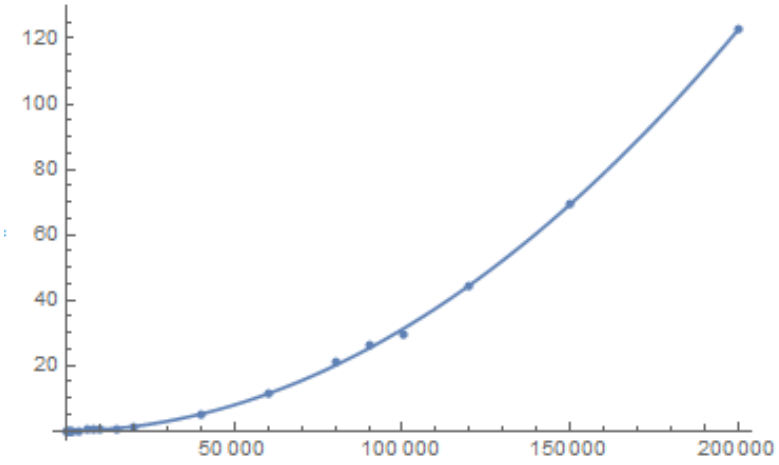
3.2.3. Gráfica de comportamiento



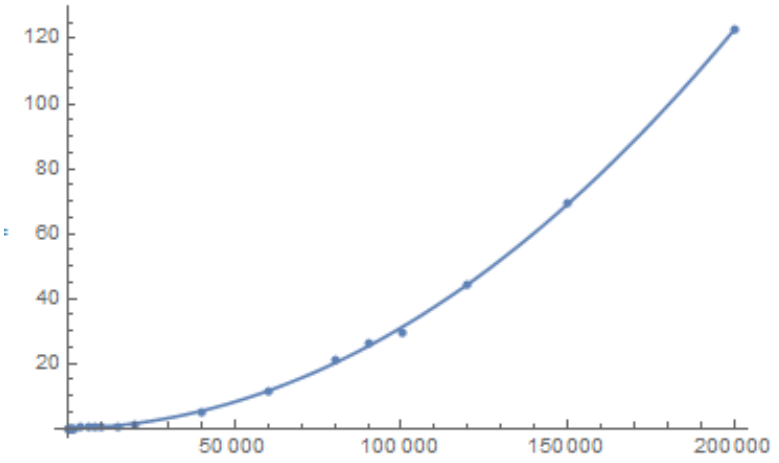
3.2.4. Aproximaciones Polinomiales



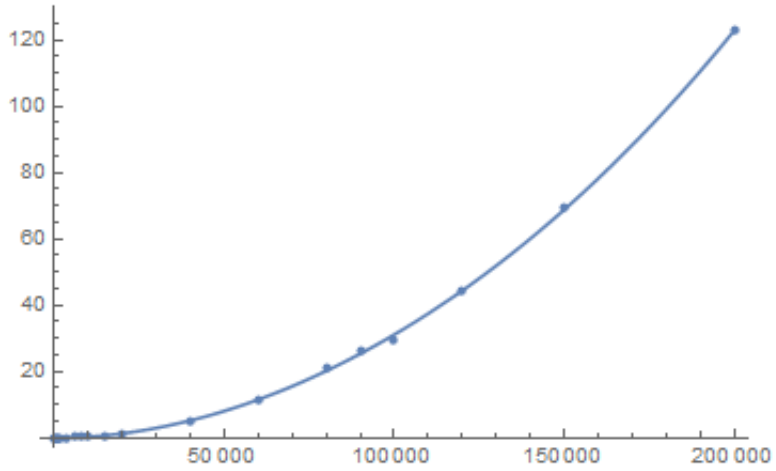
Grado 1: $-6.14724 + 0.000500917x$



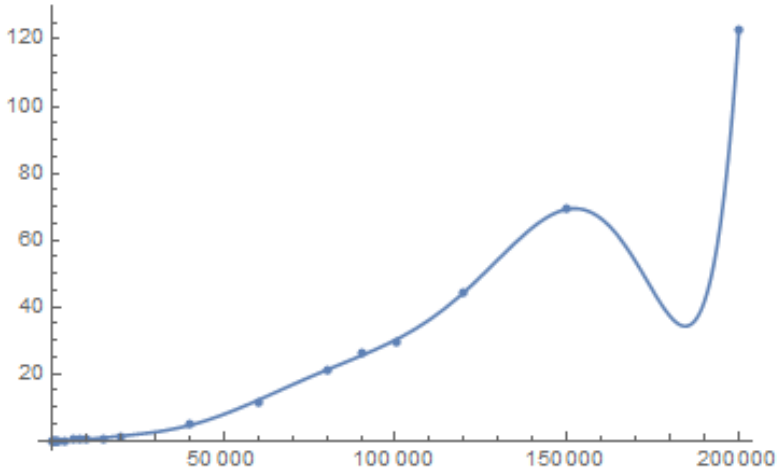
Grado 2: $-0.0288546 + 7.55172 \times 10^{-6}x + 3.03448 \times 10^{-9}x^2$



Grado 3: $-0.0998419 + 0.000020404x + 2.84002 \times 10^{-9}x^2 + 6.79964 \times 10^{-16}x^3$



Grado 4: $-0.0214544 - 9.91421 \times 10^{-7}x + 3.44382 \times 10^{-9}x^2 - 4.55857 \times 10^{-15}x^3 + 1.37689 \times 10^{-20}x^4$



Grado 8: $0.139882 - 0.000148344x + 2.83418 \times 10^{-8}x^2 - 1.5772 \times 10^{-12}x^3 + 4.54088 \times 10^{-17}x^4 - 6.70478 \times 10^{-22}x^5 + 5.24334 \times 10^{-27}x^6 - 2.05962 \times 10^{-32}x^7 + 3.18722 \times 10^{-38}x^8$

3.2.5. Evaluación de n's en Polinomios

n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	25039.7	7.588×10^6	9.20×10^7	8.54×10^10	1.22×10^24
100,000,000	50085.6	3.0310^7	7.08×10^8	1.37×10^12	3.16×10^26
500,000,000	250452	7.58×10^8	8.57×10^10	8.59×10^14	1.24×10^32
1,000,000,000	500911	3.03×10^9	6.82×10^11	1.37×10^16	3.18×10^34
5,000,000,000	2.5×10^6	7.58×10^10	8.50×10^13	8.60×10^18	1.24×10^40

3.3. Ordenamiento por inserción

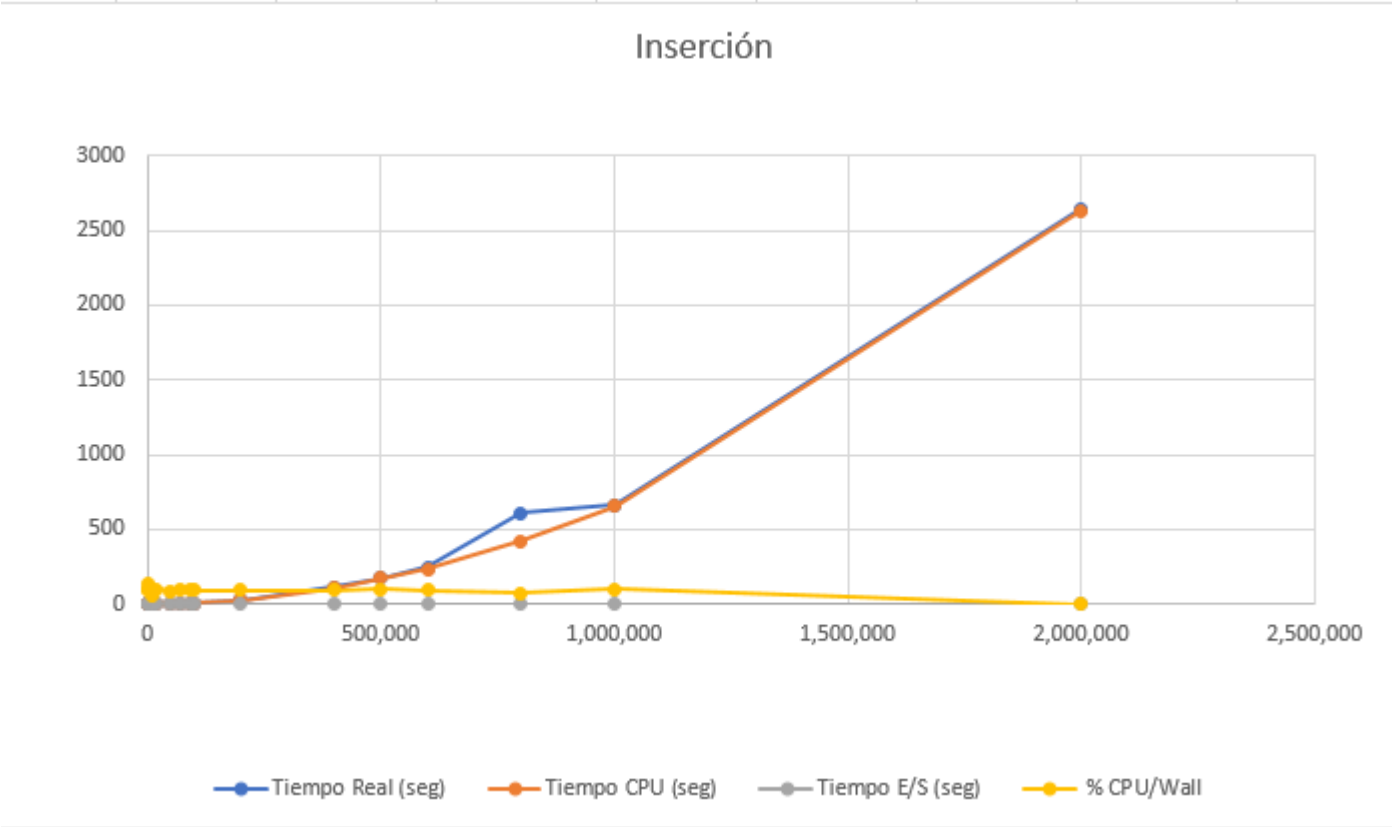
3.3.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc Insercion.c tiempo.c -o Insercion
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./Insercion 20 <numeros10millones.txt
n = 20

Insercion
real (Tiempo total)  0.000001192092895507812500000000000000 s
user (Tiempo de procesamiento en CPU) 0.000003999999999999880387346884447 20 s
sys (Tiempo en acciónes de E/S)  0.000000000000000000000000000000000000 s
CPU/Wall   335.544319999989966163411736488342285 16 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
-------------------------------
```

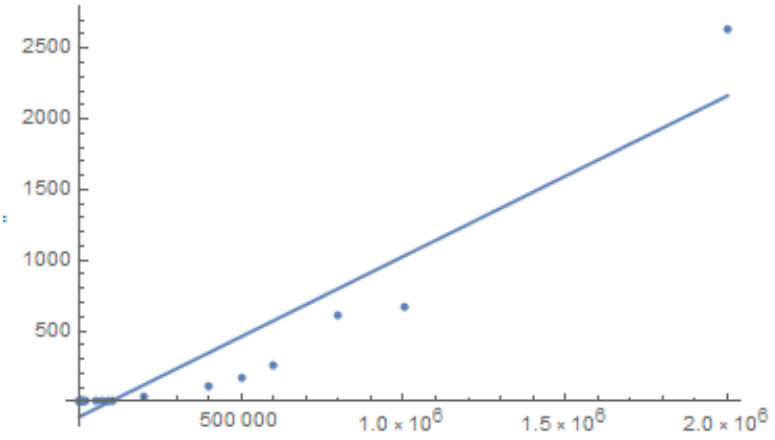
3.3.2. Análisis Temporal

Algoritmo Insercion	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	8.1062E-06	0	1.1E-05	135.6980706
500	0.00016212	0	0.000165	101.7735529
1,000	0.00067496	0.000678	0	100.449951
2,000	0.00307298	0.002874	0	93.52494139
5,000	0.01806092	0.009566	0.00758	94.93424206
8,000	0.05405307	0.045219	0	83.65667582
9,000	0.109267	0.053493	0.000638	49.54011897
10,000	0.07591796	0.06727	0.000892	89.78376224
20,000	0.27856898	0.267258	0	95.93961146
50,000	1.91829801	1.641222	0.002362	85.67928418
70,000	3.44881487	3.352152	0	97.19721491
90,000	5.7711091	5.606555	0	97.1486572
100,000	6.98440289	6.49953	0	93.05777599
200,000	27.7728181	25.817088	0.004384	92.97389958
400,000	112.682211	103.946726	0.038242	92.2816185
500,000	173.109576	170.484703	0.087342	98.53414753
600,000	249.995821	233.331495	0.23985	93.43009978
800,000	607.273253	418.971287	0.802269	69.12432813
1,000,000	664.149975	658.755994	0.004417	99.18850196
2,000,000	2643.31414	2630.21403	0.02329	0.02329

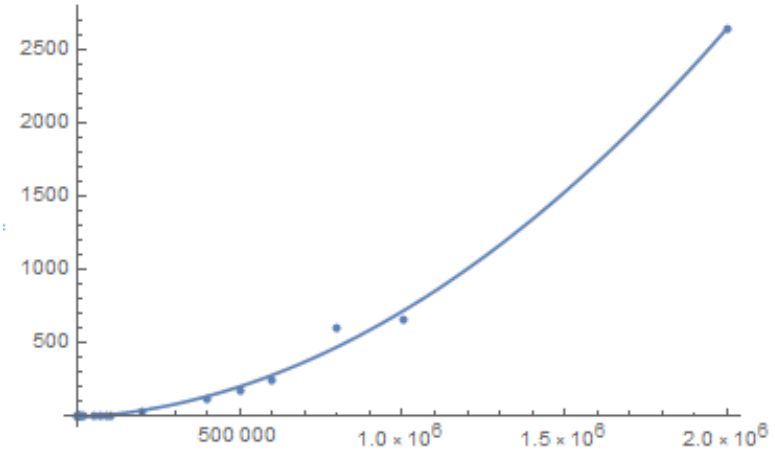
3.3.3. Gráfica de comportamiento



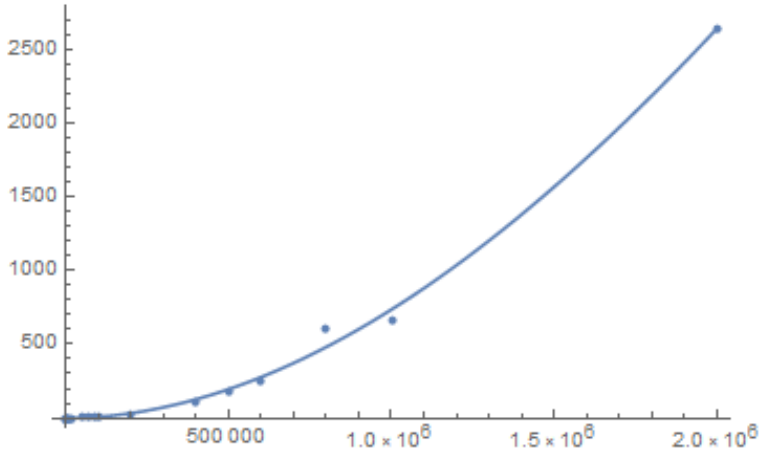
3.3.4. Aproximaciones Polinomiales



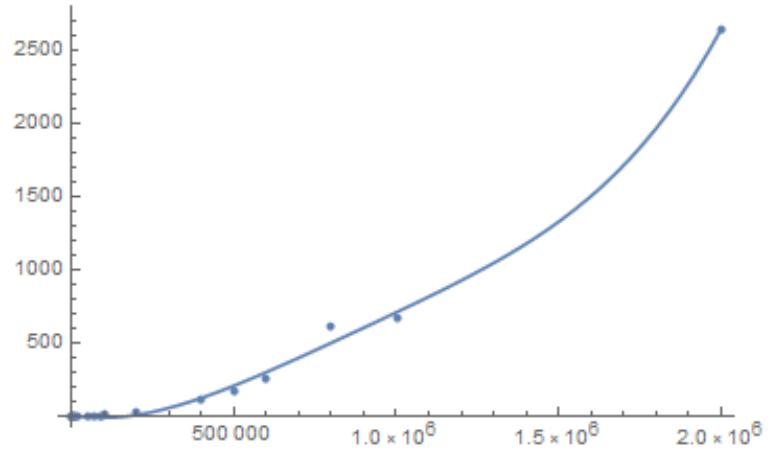
Grado 1: $-108.55 + 0.00113679x$



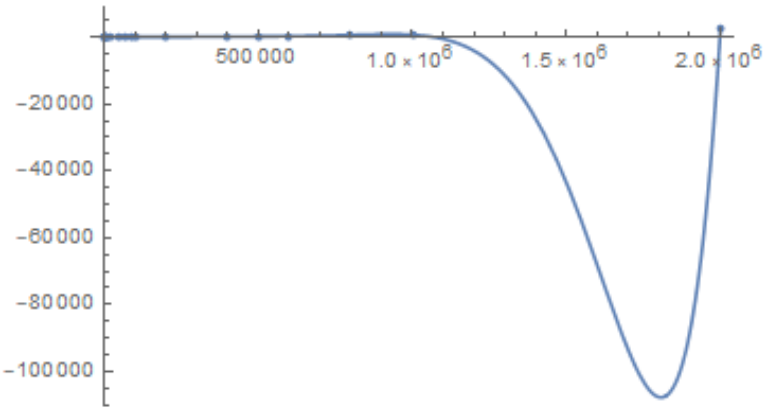
Grado 2: $-5.13494 + 0.0001174x + 6.03912 \times 10^{-10}x^2$



Grado 3: $-1.40767 + 0.0000152309x + 7.89524 \times 10^{-10}x^2 - 6.82285 \times 10^{-17}x^3$



Grado 4: $7.34056 - 0.000400925x + 2.35585 \times 10^{-9}x^2 - 1.76108 \times 10^{-15}x^3 + 5.06409 \times 10^{-22}x^4$



Grado 8: $0.42113 - 0.000104528x + 3.68568 \times 10^{-9}x^2 - 3.03438 \times 10^{-14}x^3 + 1.42669 \times 10^{-19}x^4 - 3.43268 \times 10^{-25}x^5 + 4.27728 \times 10^{-31}x^6 - 2.55327 \times 10^{-37}x^7 + 5.5625 \times 10^{-44}x^8$

3.3.5. Evaluación de n's en Polinomios

n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	56730.9	1.51×10^6	-6.55×10^6	2.95×10^9	1.97×10^18
100,000,000	113570	6.05×10^6	-6.03×10^7	4.89×10^10	5.31×10^20
500,000,000	568286	1.51×10^8	-8.33×10^9	3.14×10^13	2.15×10^26
1,000,000,000	1.13×10^6	6.04×10^8	-6.74×10^10	5.04×10^14	5.53×10^28
5,000,000,000	5.68×10^6	1.50×10^10	-8.50×10^12	3.16×10^17	2.17×10^34

3.4. Ordenamiento por selección

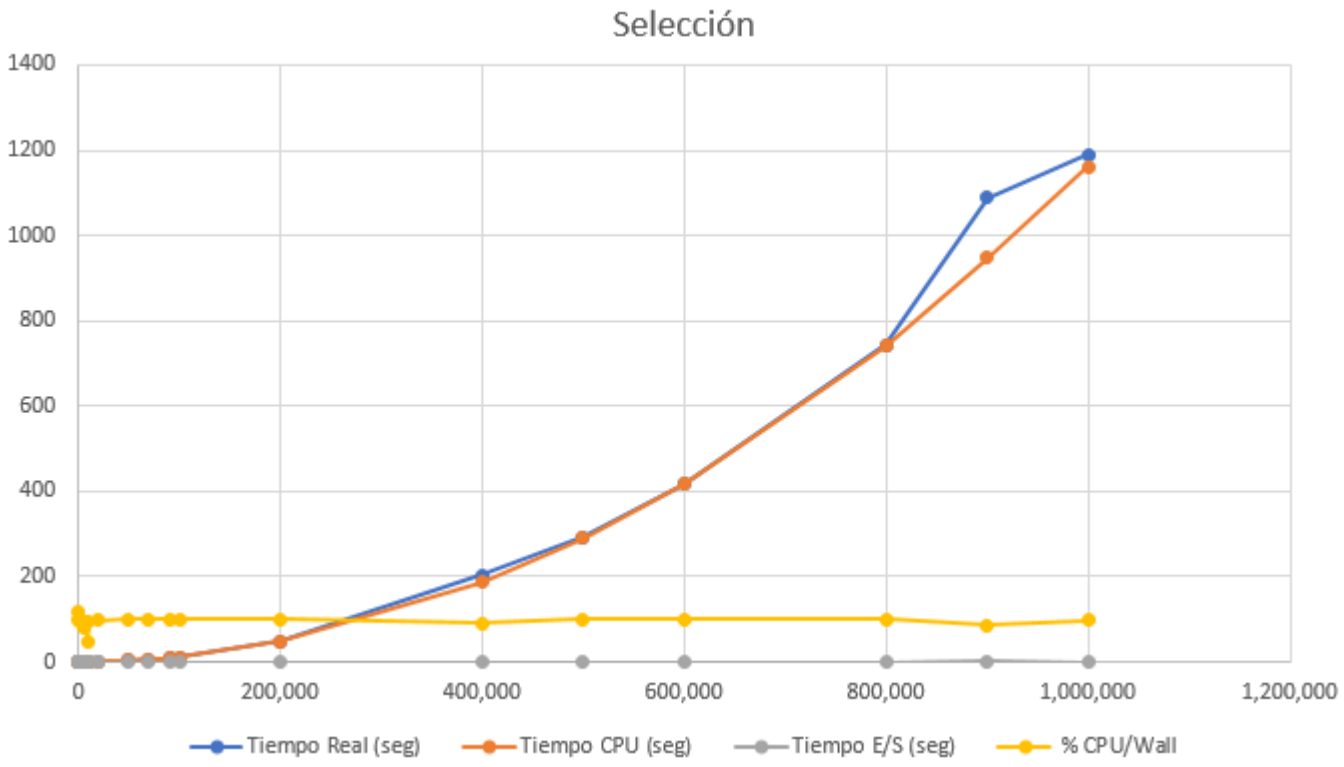
3.4.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc Seleccion.c tiempo.c -o Seleccion
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./Seleccion 20 <numeros10millones.txt
n = 20

Seleccion
real (Tiempo total)  0.000001907348632812500000000000000000 s
user (Tiempo de procesamiento en CPU) 0.000004000000000000097227781381548 08 s
sys (Tiempo en acciónes de E/S)  0.000000000000000000000000000000000 s
CPU/Wall   209.715200000005097535904496908187866 21 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
------------------------------
```

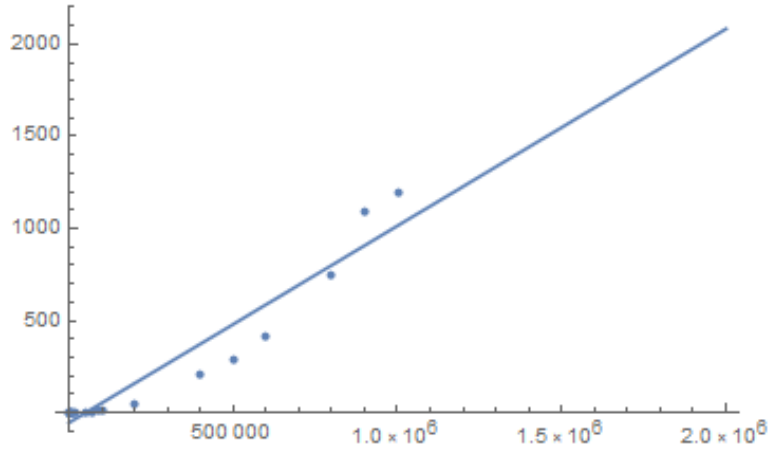
3.4.2. Análisis Temporal

Algoritmo Selección	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	1.5974E-05	1.9E-05	0	118.9429493
500	0.000365019	0.000368	0	100.8167127
1,000	0.001303196	0	0.001268	97.29925854
2,000	0.005572081	0.005224	0	93.75313036
5,000	0.038476944	0.030952	0	80.44297911
8,000	0.080535173	0.073645	0.00402	96.43612437
9,000	0.198409081	0.097388	0	49.08444702
10,000	0.127197027	0.115482	0.004239	94.12248276
20,000	0.48895812	0.481849	0.000926	98.73544991
50,000	2.950371981	2.937647	0	99.56869911
70,000	5.861546993	5.833201	0.009381	99.67645072
90,000	9.651067972	9.610695	0.01151	99.70093494
100,000	11.82698393	11.789417	0	99.68236256
200,000	48.65380311	48.508206	0	99.70074876
400,000	204.3738148	186.007273	0	91.01326076
500,000	291.7892001	290.469773	0.004077	99.54921222
600,000	418.247813	417.169571	0	99.74220021
800,000	744.585923	740.692717	0.001739	99.47736496
900,000	1088.344782	948.118081	0.865546	87.19512811
1,000,000	1190.107606	1161.63258	0.144266	97.61947879

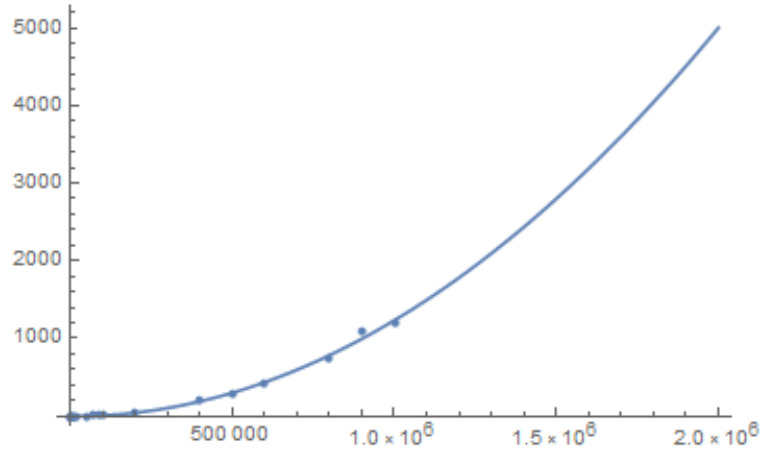
3.4.3. Gráfica de comportamiento



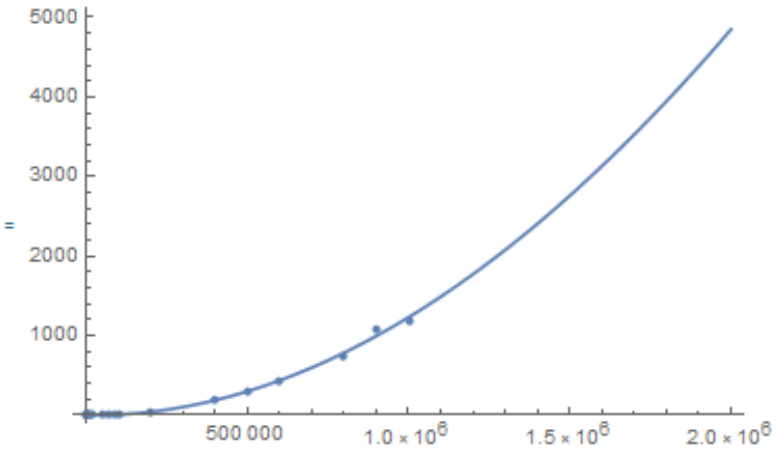
3.4.4. Aproximaciones Polinomiales



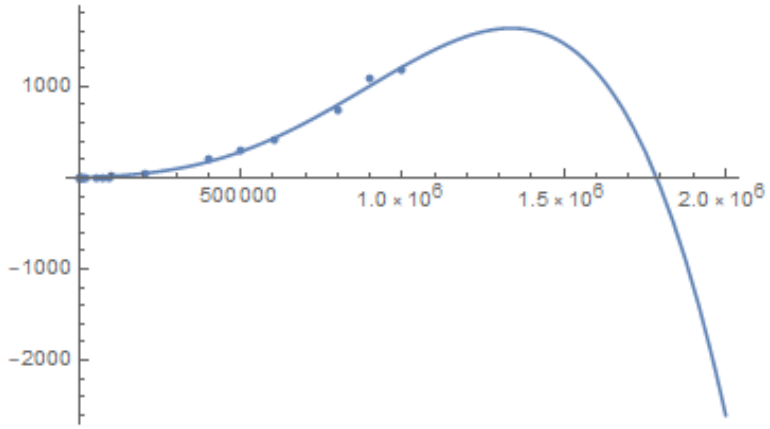
Grado 1: $-53.8503 + 0.00106898x$



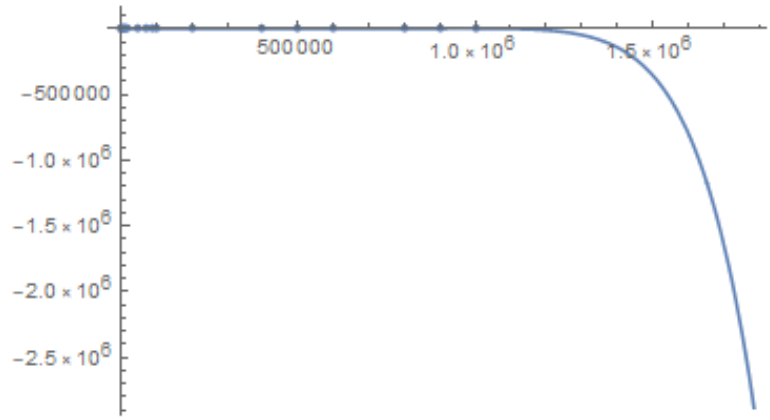
Grado 2: $0.933709 - 0.0000347738x + 1.26724 \times 10^{-9}x^2$



Grado 3: $1.46243 - 0.0000567496x + 1.33337 \times 10^{-9}x^2 - 4.6366 \times 10^{-17}x^3$



Grado 4: $-3.14247 + 0.000234829x - 3.72062 \times 10^{-10}x^2 + 2.91292 \times 10^{-15}x^3 - 1.55504 \times 10^{-21}x^4$



Grado 8: $-1.65689 + 0.00041812x - 1.0197 \times 10^{-8}x^2 + 1.10428 \times 10^{-13}x^3 - 5.05429 \times 10^{-19}x^4 + 1.24266 \times 10^{-24}x^5 - 1.68957 \times 10^{-30}x^6 + 1.19325 \times 10^{-36}x^7 - 3.40375 \times 10^{-43}x^8$

3.4.5. Evaluación de n's en Polinomios

n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	53395.1	3.16x10^6	-2.46x10^6	-9.35*10^9	-1.23x10^19
100,000,000	106844	1.26x10^7	-3.30x10^7	-1.52x10^11	-3.28x10^21
500,000,000	534436	3.16x10^8	-5.46x10^9	-9.68x10^13	-1.32x10^27
1,000,000,000	1.06x10^6	1.26x10^9	-4.5x10^10	-1.55x10^15	-3.39x10^29
5,000,000,000	5.34x10^6	3.16x10^10	-5.76x10^12	-9.71x10^17	-1.32x10^35

3.5. Ordenamiento Shell

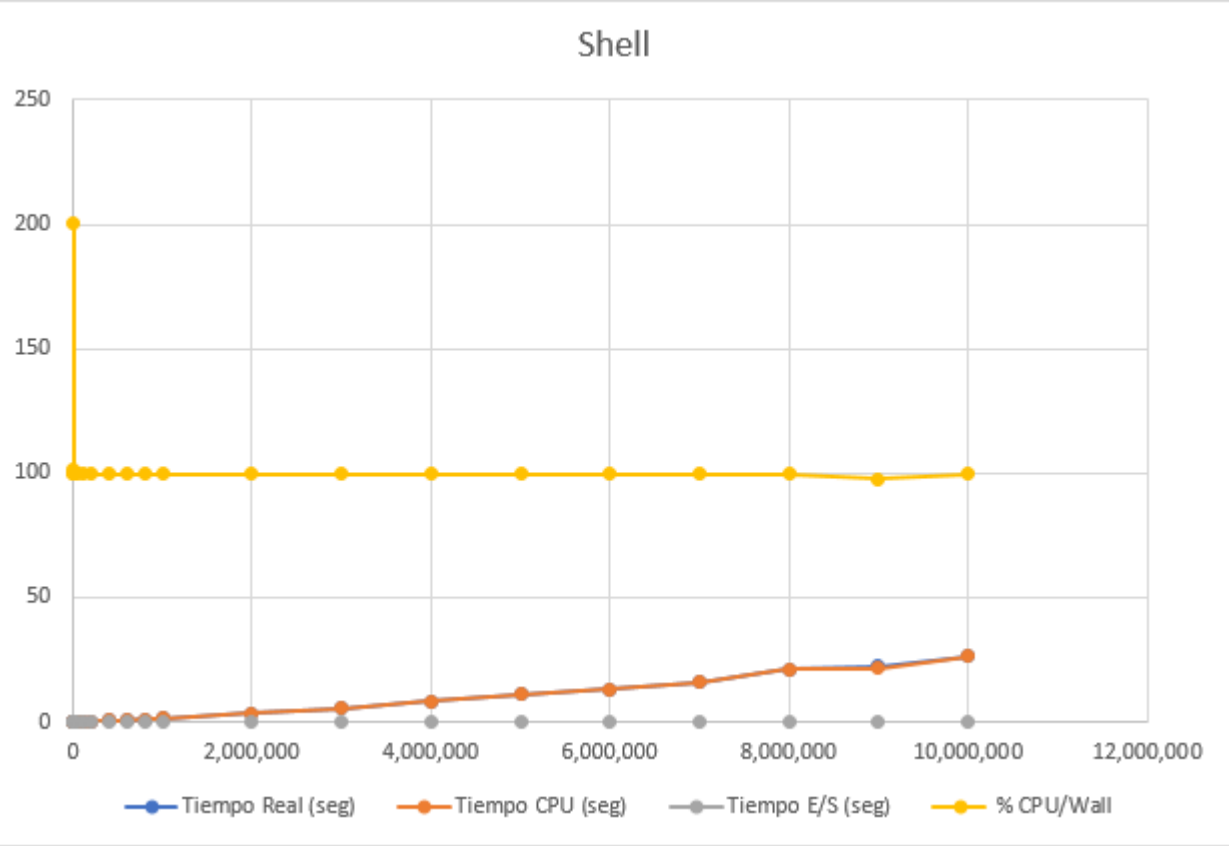
3.5.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc Shell.c tiempo.c -o Shell
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./Shell 20 <numeros10millones.txt
n = 20

Shell
real (Tiempo total)  0.000002145767211914062500000000000000 s
user (Tiempo de procesamiento en CPU) 0.000000000000000000000000000000000000 s
sys (Tiempo en acciónes de E/S)  0.000005000000000000131145094783846 6 s
CPU/Wall   233.016888888889496911360765807330608 37 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
------------------------------
```

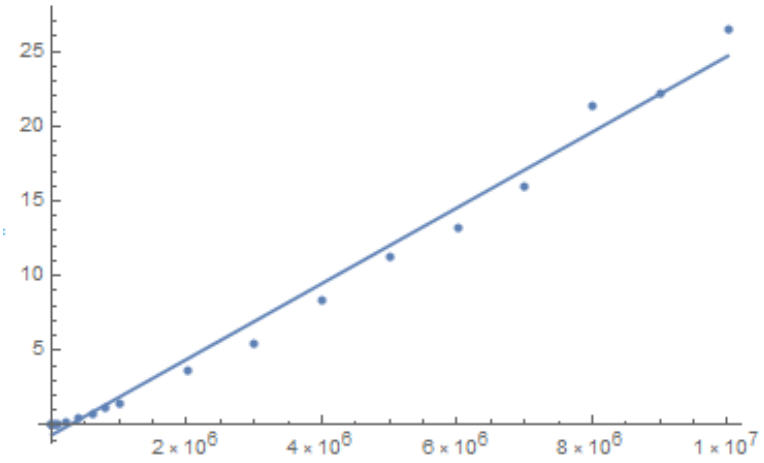
3.5.2. Análisis Temporal

Algoritmo Shell	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	1.0967E-05	2.2E-05	0	200.5971478
1,000	0.00022292	0	0.000226	101.3810379
5,000	0.00199103	0.001994	0	100.1489903
10,000	0.00467682	0.003879	0.00079	99.83281697
50,000	0.03550386	0.035344	0	99.54972707
100,000	0.092874	0.092785	0	99.90411731
200,000	0.210251	0.207508	0.002363	99.81921953
400,000	0.51499891	0.514218	0	99.84836612
600,000	0.73926806	0.737743	0	99.79370616
800,000	1.19754601	1.194927	0	99.78130233
1,000,000	1.395174	1.39131	0	99.72297524
2,000,000	3.65216112	3.643176	0	99.75397796
3,000,000	5.44203091	5.429574	0	99.7710982
4,000,000	8.38806891	8.369377	0	99.7771607
5,000,000	11.3036599	11.278542	0	99.77778953
6,000,000	13.1929722	13.164379	0	99.78326959
7,000,000	15.911361	15.858994	0	99.67088309
8,000,000	21.3013718	21.254095	0	99.77805743
9,000,000	22.200381	21.670029	0	97.61106785
10,000,000	26.474169	26.390007	0	99.68209761

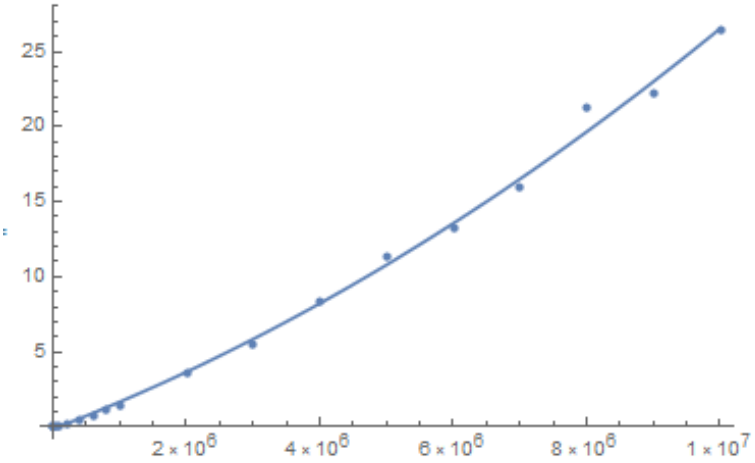
3.5.3. Gráfica de comportamiento



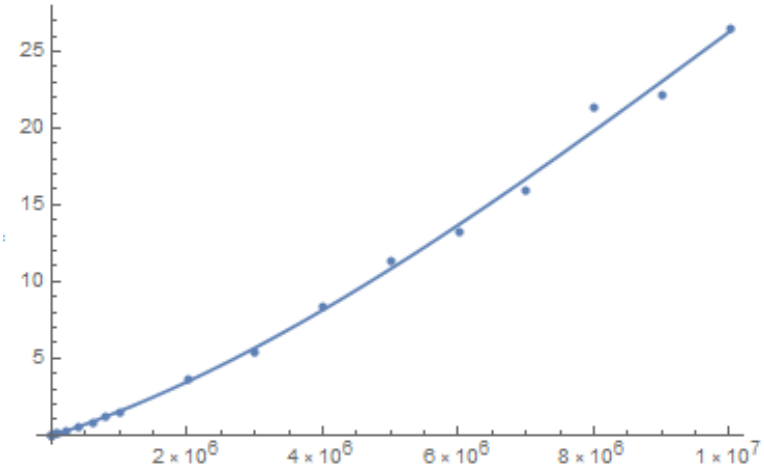
3.5.4. Aproximaciones Polinomiales



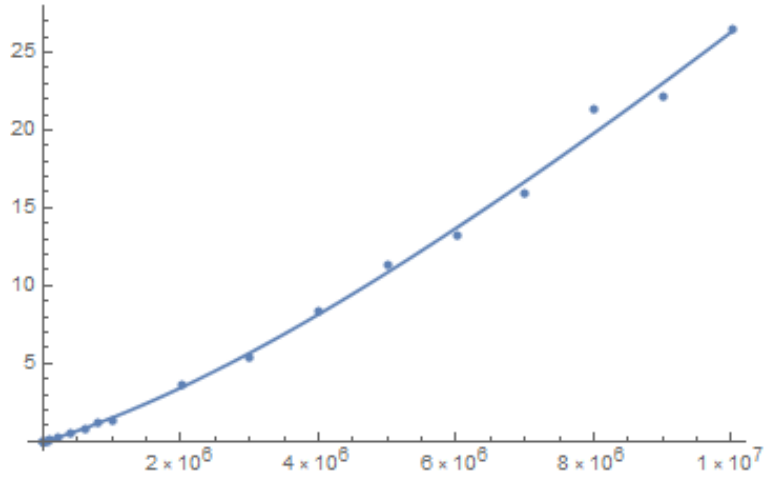
Grado 1: $-0.642248 + 2.53478 \times 10^{-6}x$



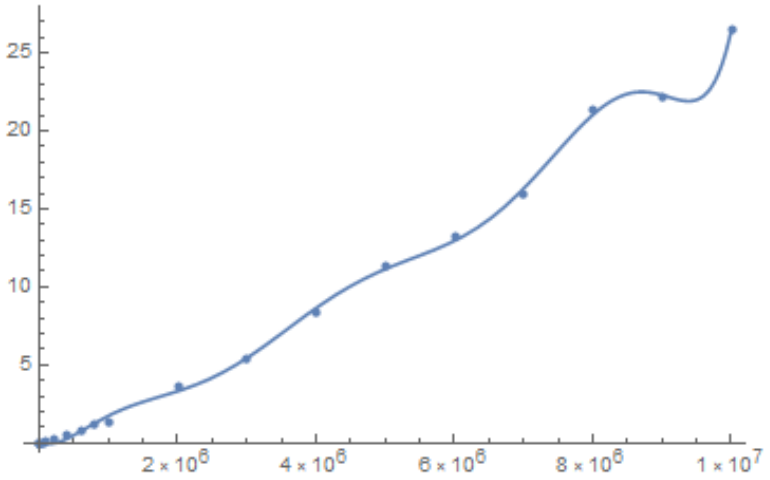
Grado 2: $-0.128825 + 1.70604 \times 10^{-6}x + 9.60803 \times 10^{-14}x^2$



Grado 3: $-0.0519355 + 1.42946 \times 10^{-6}x + 1.78285 \times 10^{-13}x^2 - 5.77542 \times 10^{-21}x^3$



Grado 4: $-0.0400139+1.35877\times10^{-6}x+2.17949\times10^{-13}x^2-1.24843\times10^{-20}x^3+3.46719\times10^{-28}x^4$



Grado 8: $0.0889081 - 2.02157 \times 10^{-6}x + 9.15851 \times 10^{-12}x^2 - 8.48395 \times 10^{-18}x^3 + 3.82072 \times 10^{-24}x^4 - 9.14639 \times 10^{-31}x^5 + 1.19178 \times 10^{-37}x^6 - 7.97176 \times 10^{-45}x^7 + 2.1411 \times 10^{-52}x^8$

3.5.5. Evaluación de n's en Polinomios

n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	126.09	325.37	-204.794	1219.23	3.73×10^9
100,000,000	252.83	1131.28	-3849.68	24502.9	1.45×10^12
500,000,000	1266.75	24873	-676642	2.01×10^7	7.75×10^17
1,000,000,000	2534.14	97786.2	-5.59×10^6	3.34×10^8	2.06×10^20
5,000,000,000	12673.3	2.41054×10^6	-7.17×10^8	2.15×10^11	8.30×10^25

3.6. Árbol binario de búsqueda

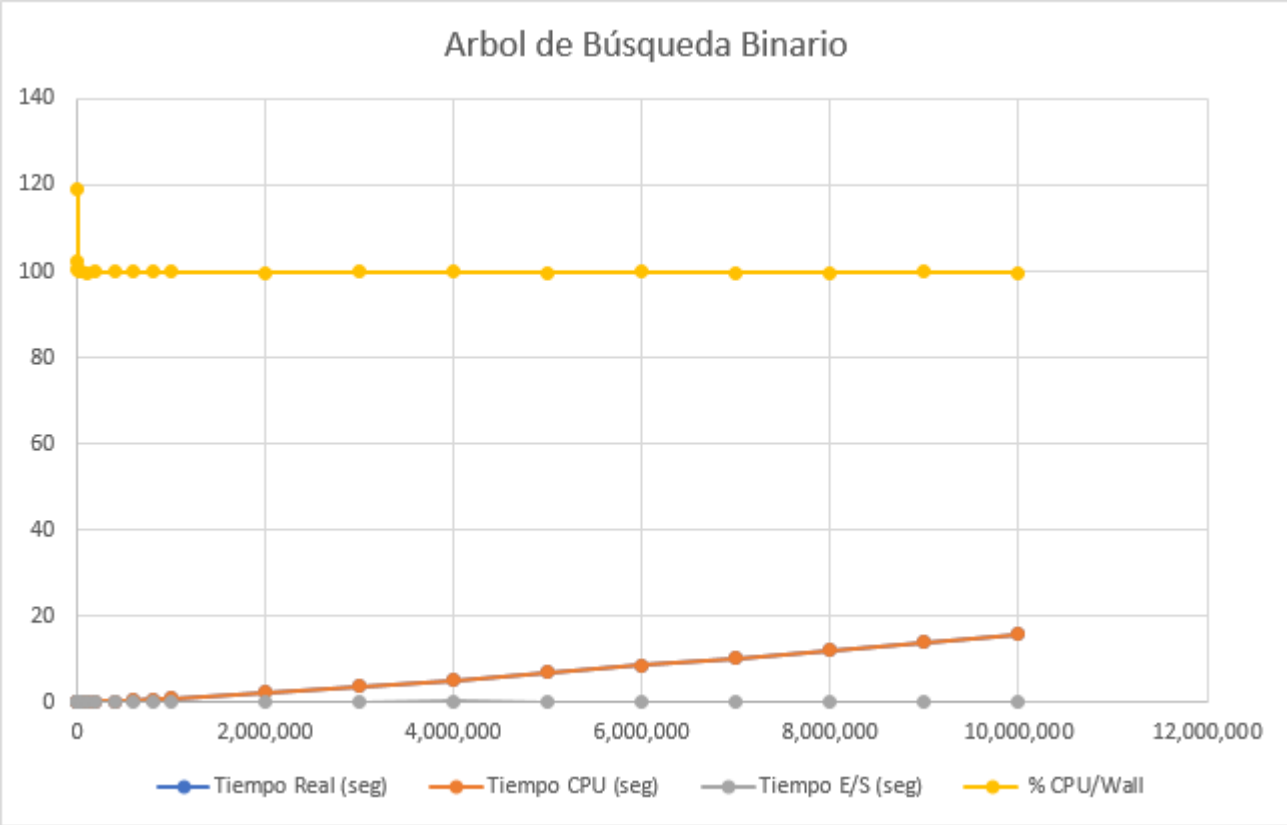
3.6.1. Ejecución del algoritmo

```
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ gcc ABB.c tiempo.c -o ABB
enrike@enrike:/media/Google_Drive/5° SEMESTRE/Analisis de algoritmos/Practica
/Practica1/Programas$ ./ABB 20 <numeros10millones.txt
n = 20

Arbol de Busqueda Binario
real (Tiempo total)  0.000004053115844726562500000000000000 s
user (Tiempo de procesamiento en CPU) 0.000007000000000000006172840016915870 s
sys (Tiempo en acciónes de E/S)  0.000000000000000000000000000000000000 s
CPU/Wall   172.70663529411916670142090879380702972 %

118245790, 178804962, 187130533, 207442309, 494387258, 525017507, 654552922,
834068396, 839426280, 846455593, 856834115, 870128690, 966245083, 984674766,
1511588122, 1799388853, 1846873530, 1920624725, 1990378944, 2083630786, -----
------------------------------
```

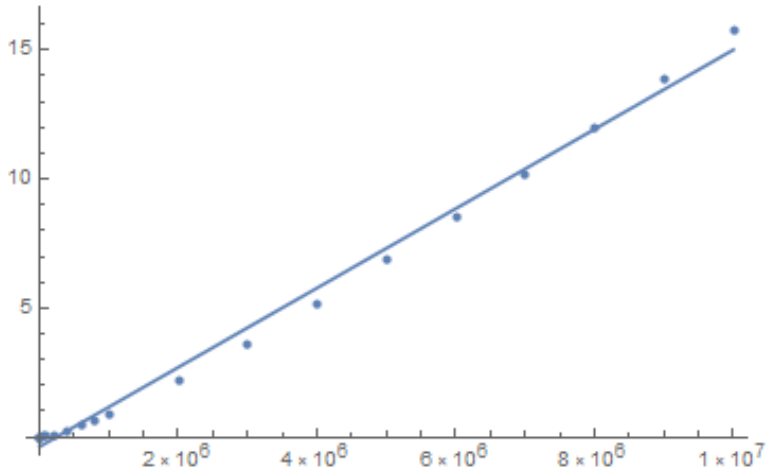
3.6.2. Análisis Temporal

Algoritmo ABB	Tiempo Real (seg)	Tiempo CPU (seg)	Tiempo E/S (seg)	% CPU/Wall
100	1.5974E-05	0.000019	0	118.9429493
1,000	0.0001719	0.000176	0	102.3852294
5,000	0.001017094	0.00102	0	100.2857496
10,000	0.002295017	0	0.002299	100.1735393
50,000	0.015423059	0.015398	0	99.8375195
100,000	0.036169	0.036002	0	99.53813519
200,000	0.08591	0.085808	0	99.88034435
400,000	0.249607086	0.245439	0.003743	99.82969787
600,000	0.449668169	0.431623	0.017142	99.79914766
800,000	0.649112225	0.641139	0.006555	99.78151319
1,000,000	0.855236	0.853512	0	99.79841198
2,000,000	2.156502962	2.145462	0.005484	99.74231605
3,000,000	3.60809207	3.599528	0	99.76264271
4,000,000	5.117367029	5.059857	0.045757	99.77033054
5,000,000	6.917185783	6.894958	0.005101	99.75240244
6,000,000	8.520278931	8.47373	0.026341	99.76282548
7,000,000	10.20669103	10.152745	0.007092	99.54094793
8,000,000	11.99638486	11.940196	0.027433	99.76029563
9,000,000	13.86554408	13.83403	0	99.77271659
10,000,000	15.75872302	15.697902	0	99.61404855

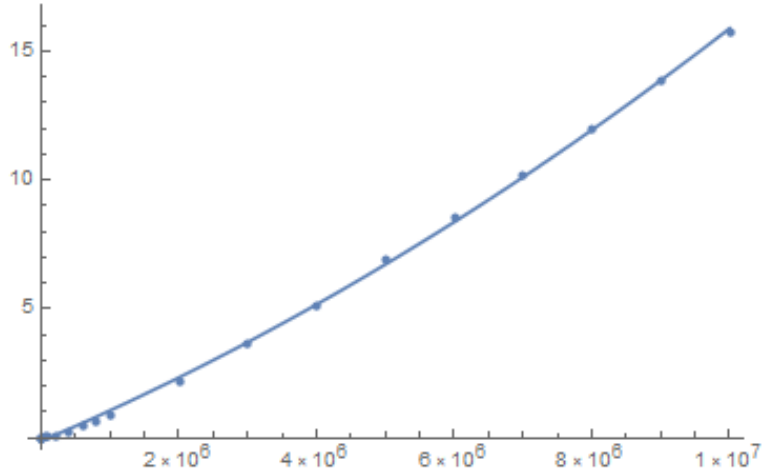
3.6.3. Gráfica de comportamiento



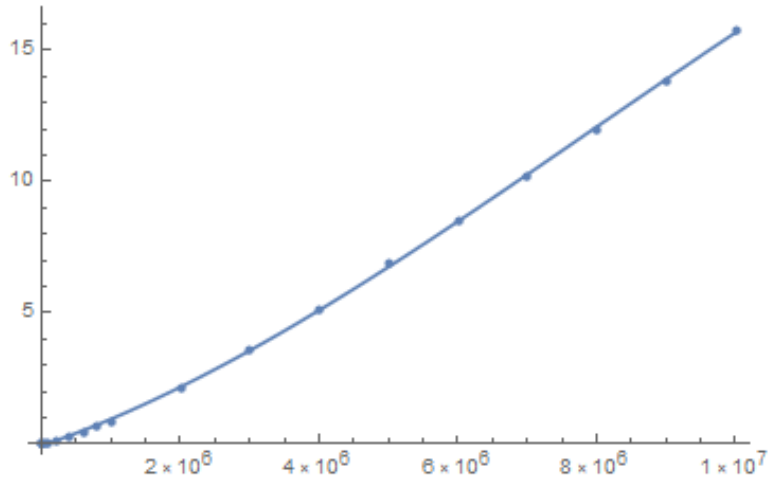
3.6.4. Aproximaciones Polinomiales



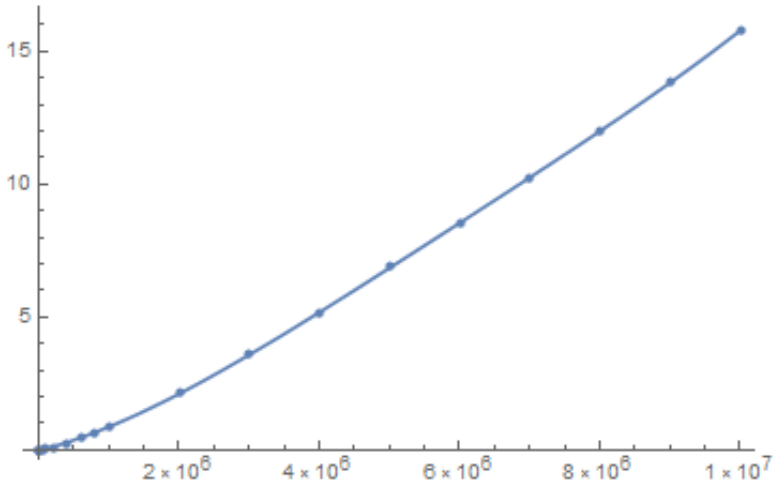
Grado 1: $-0.373011 + 1.53853 \times 10^{-6}x$



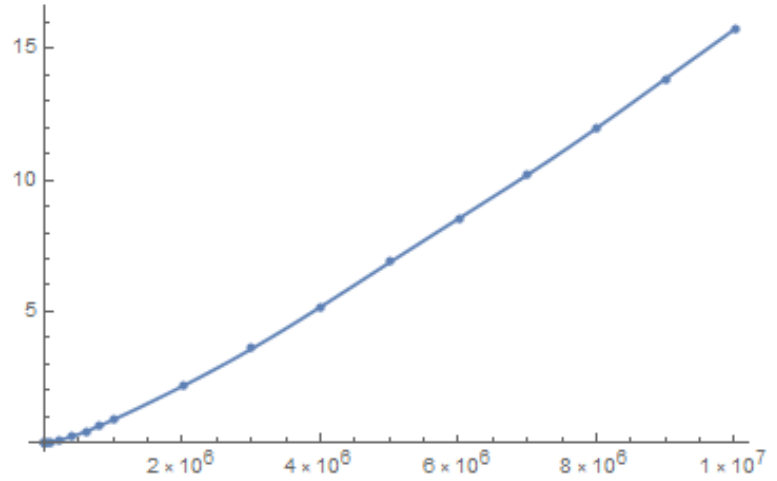
Grado 2: $-0.126987 + 1.14141 \times 10^{-6}x + 4.604 \times 10^{-14}x^2$



Grado 3: $-0.0615581 + 9.06049 \times 10^{-7}x + 1.15993 \times 10^{-13}x^2 - 4.91464 \times 10^{-21}x^3$



Grado 4: $-0.0222793+6.73151\times10^{-7}x+2.46674\times10^{-13}x^2-2.70187\times10^{-20}x^3+1.14236\times10^{-27}x^4$



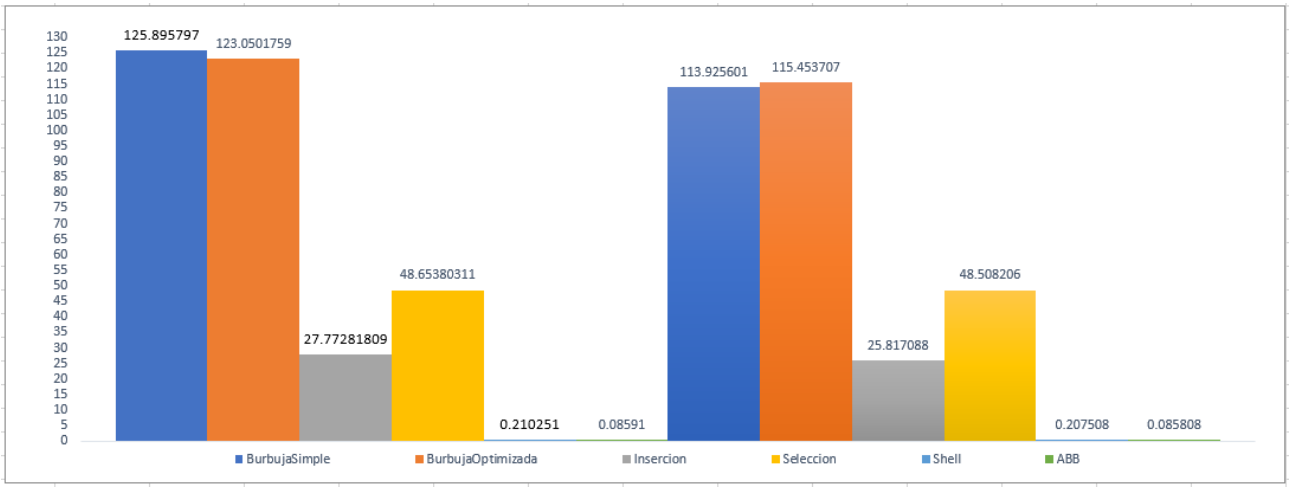
Grado 8: $0.000327011 + 2.62645 \times 10^{-7}x + 1.06888 \times 10^{-12}x^2 - 6.3221 \times 10^{-19}x^3 + 2.20885 \times 10^{-25}x^4 - 4.36496 \times 10^{-32}x^5 + 4.82736 \times 10^{-39}x^6 - 2.78724 \times 10^{-46}x^7 + 6.54442 \times 10^{-54}x^8$

3.6.5. Evaluación de n's en Polinomios

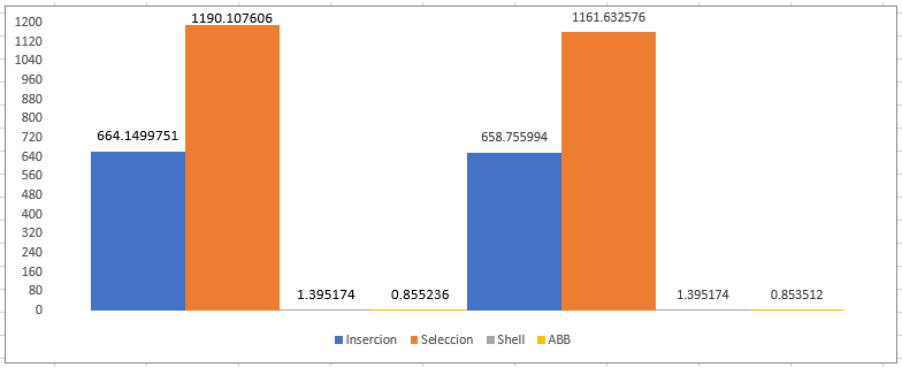
n	Grado 1	Grado 2	Grado 3	Grado 4	Grado 8
50,000,000	76.55	172.04	-279.10	4412.73	1.009×10^8
100,000,000	153.48	574.41	-3664.17	89751.3	4.19×10^10
500,000,000	768.89	12080.6	-584879	6.80×10^7	2.34×10^16
1,000,000,000	1538.16	47181.3	-4.79×10^6	1.11×10^9	2.34×10^16
5,000,000,000	7692.28	1.15×10^6	-6.11×10^8	7.10×10^11	2.347×10^16

3.7. Comparativa de tiempos reales y de CPU

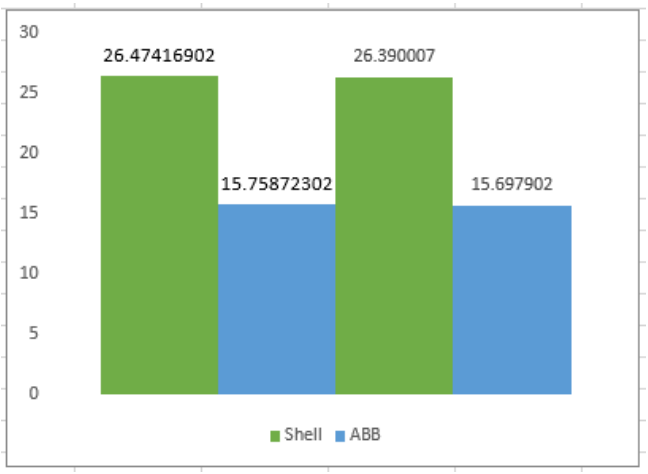
- Con n = 200,000



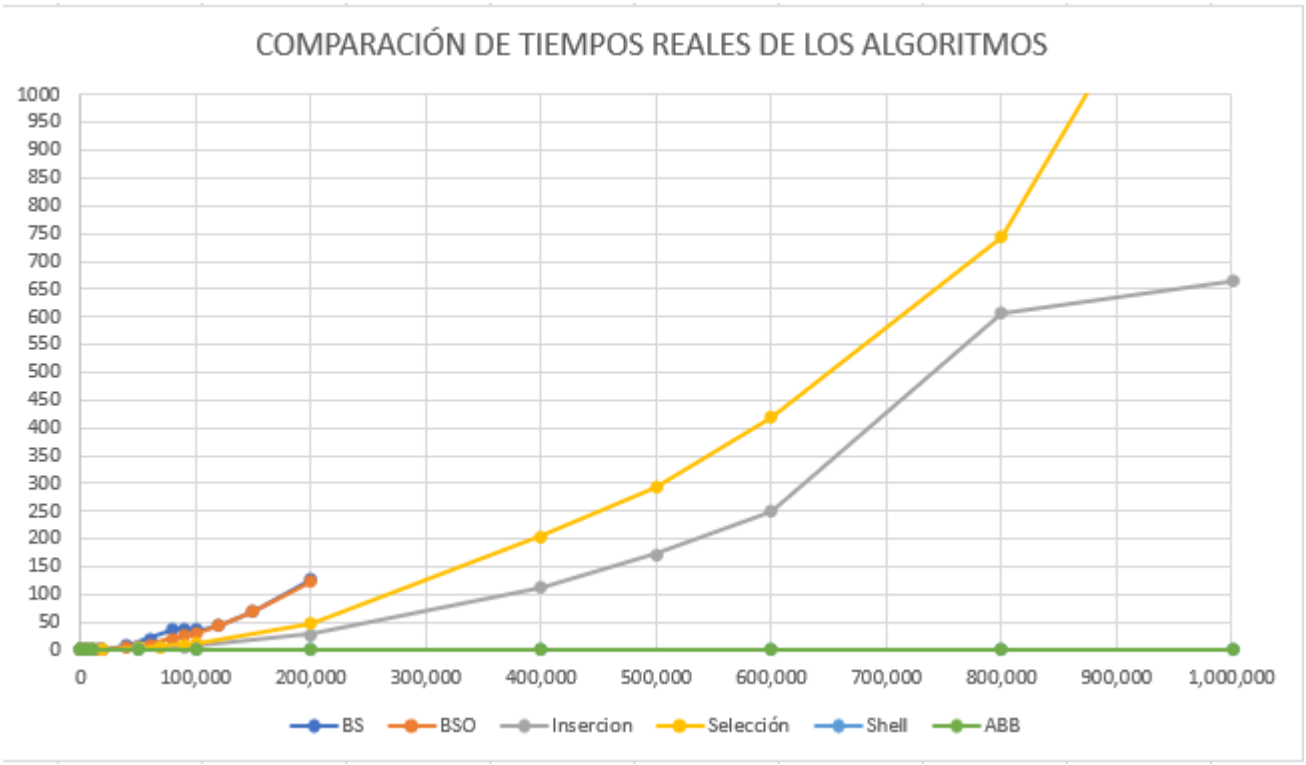
- Con n = 1,000,000



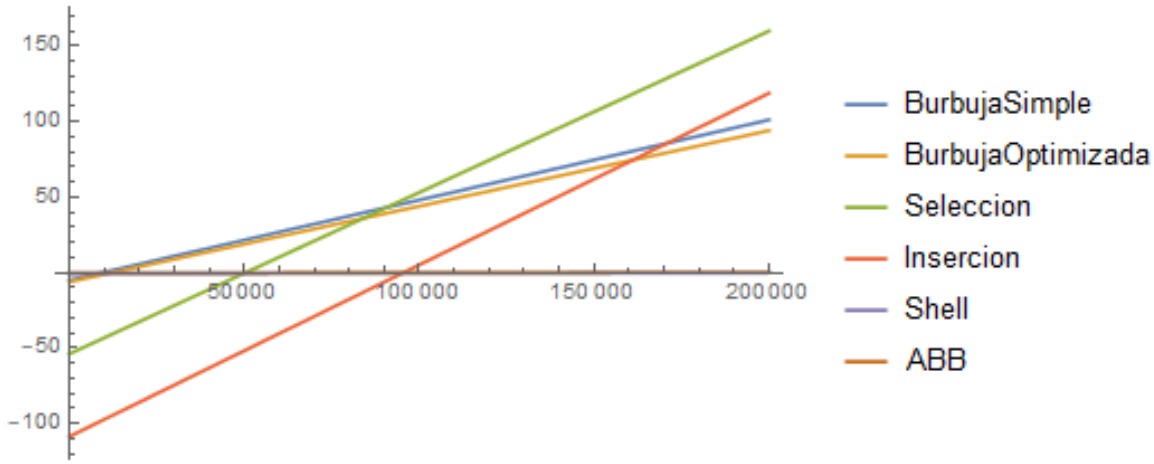
- Con n = 10,000,000



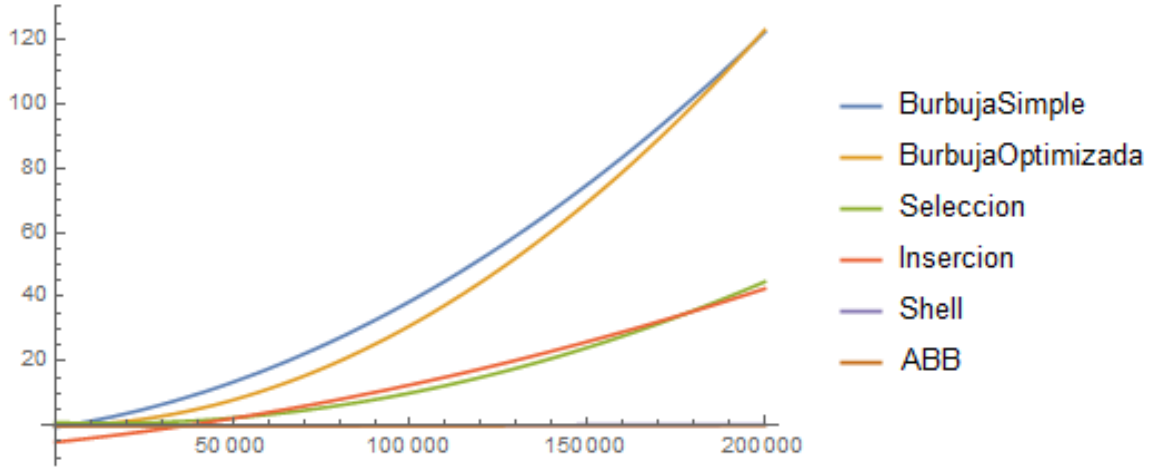
3.8. Comparativa de gráficas de comportamiento (Tiempo Real)



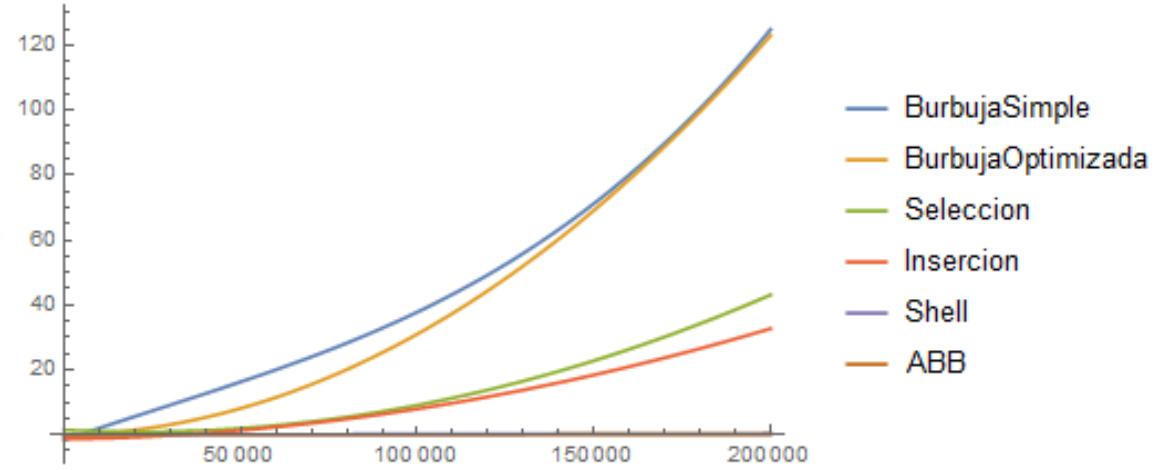
3.9. Comparativa de aproximaciones polinomiales



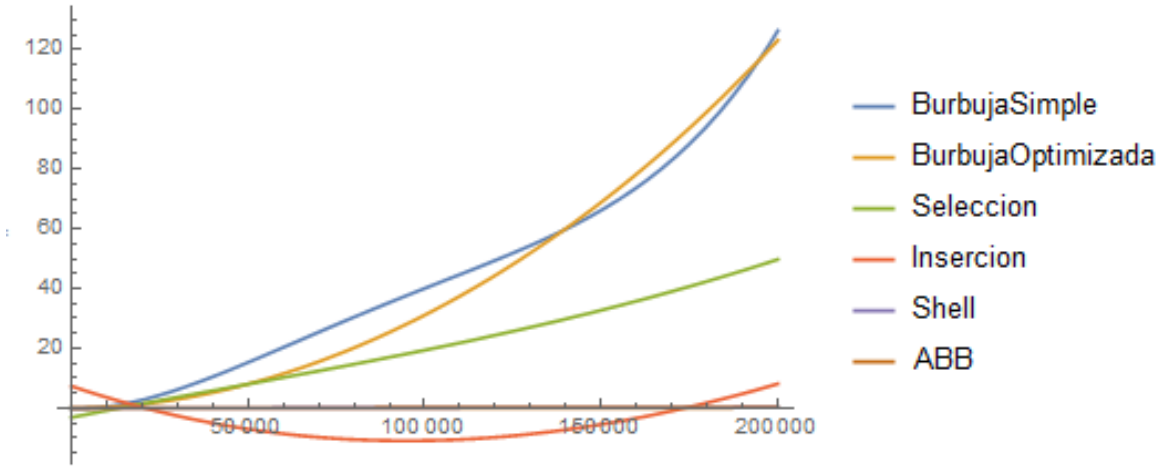
Grado 1



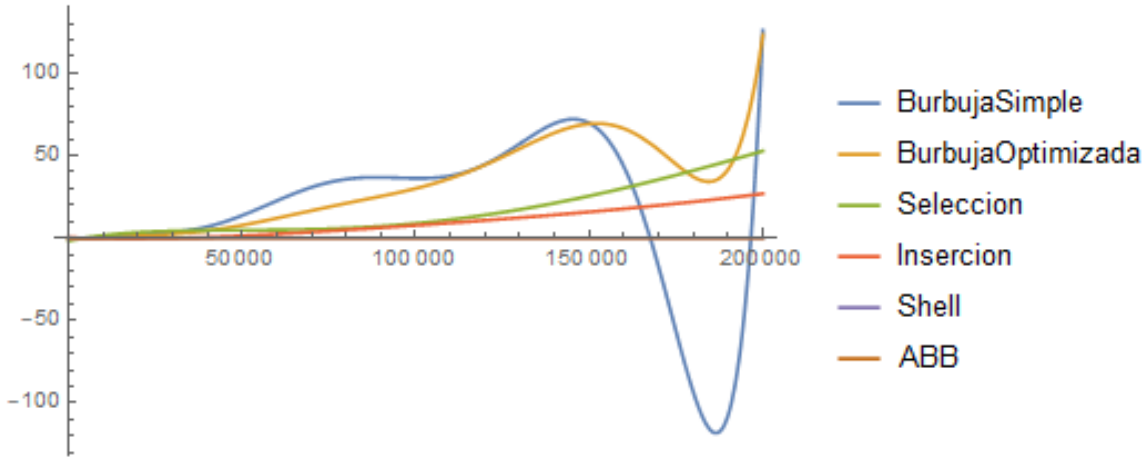
Grado 2



Grado 3



Grado 4



Grado 8

4. Cuestionario

1. ¿Cuál de los 5 algoritmos es más fácil de implementar?
   Para nosotros, el más fácil de implementar fue el de inserción, debido a que fue sencillo comprender su funcionamiento. Los algoritmos de burbuja también tienen un nivel de implementación claro.

2. ¿Cuál de los 5 algoritmos es el más difícil de implementar?
   Debido a que se requiere un análisis de apuntadores, consideramos que el árbol de búsqueda binaria podría ser el más complicado de implementar.

3. ¿Cuál algoritmo tiene menos complejidad temporal?
   Una vez realizadas las pruebas, el algoritmo con menor complejidad temporal fue el de árbol de búsqueda binaria, seguido por el de Shell.

4. ¿Cuál algoritmo tiene mayor complejidad temporal?
   En la teoría, con números muy grandes debería ser indiferente la complejidad temporal, sin embargo, los algoritmos de ordenamiento de burbuja demostraron ser los menos eficientes, ya que tan solo llegar a ordenar 200,000 números tomaba más tiempo en comparación con la misma cantidad de números con los otros algoritmos de ordenamiento.

5. ¿Cuál algoritmo tiene menor complejidad espacial? ¿Por qué?
   Existen dos que cumplen esta característica: el de burbuja simple y el de inserción, pues ambos hacen uso de únicamente un arreglo y 3 variables de control de las posiciones del arreglo, por lo que de entre todos estos consumen menor memoria.

6. ¿Cuál algoritmo tiene mayor complejidad espacial? ¿Por qué?
   En este caso observamos que el algoritmo de árbol de búsqueda binaria tuvo una mayor complejidad espacial, debido a que hace uso de estructuras del lenguaje C, apuntadores de memoria, y reasigna memoria dinámica a una variable de tipo Arbin cada vez que se realiza una inserción, siendo esta un nodo de aproximadamente tamaño n/2 en cada ramificación.

7. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?
   Si, esperábamos que los algoritmos de burbuja fuesen los que demostrasen los resultados menos favorables, así como que los algoritmos Shell y árbol binario tuviesen los mejores resultados. Esto debido al análisis previo a la práctica.

8. ¿Sus resultados experimentales difieren mucho de los del resto de los equipos? ¿A que se debe?
   No, a pesar de las diferentes capacidades de cada computadora, los resultados generales fueron similares; los algoritmos de burbuja fueron los menos favorables y el árbol binario fue el más eficiente.

9. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?
   Si, se realizaron los ordenamientos con distintos tamaños de problema (n's) según el algoritmo, todos al mismo tiempo. Todos los algoritmos fueron probados en la misma máquina al momento de realizar las gráficas comparativas.

10. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?
    Realizar una investigación previa a la práctica acerca del funcionamiento e implementación de cada algoritmo, de esta manera, cada integrante conoce el funcionamiento de cada algoritmo sin la necesidad de realizar la implementación. Realizar comentarios en el código puede ser importante, pero es más sencillo si el nombre de las variables explica el código mismo.

5. Anexos

NOTA: Para ejecutar todas las pruebas de todos los algoritmos ejecutar el archivo script.sh con el siguiente comando e una terminal dentro de la carpeta de los códigos fuente: sh script.sh

5.1. Burbuja simple

```
1  //*************************************************************
2  //   AUTORES:
3  //    Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  //    Ramos Diaz Enrique
6  // ************************************************************
7  //   Practica 1: Analisis de algoritmos de ordenamiento numerico
8  //   Compilación:
9  //   gcc tiempo.c -c
10 //   gcc BurbujaSimple.c tiempo.o -o BurbujaSimple
11 //
12 //   Ejecución: "./BurbujaSimple" (Linux)
13 //   ************************************************************
14
15 //   ************************************************************
16 //                  Librerias incluidas
17 //   ************************************************************
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include <string.h>
21 #include "tiempo.h"
22
23 //   ************************************************************
24 //                  BURBUJA SIMPLE
25 //   ************************************************************
26 //   Descripción: Ordena un arreglo de números
27 //   Recibe: El arreglo de números, y la cantidad de números que se ordenaran
28 //   Devuelve: Nada.
29 //   ************************************************************
30 void BurbujaSimple(int A[], int n)
31 {
32    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
33    uswtime(&utime0, &stime0, &wtime0);
34
35    int j, i, aux;
36    for(i=0; i<=n-1; i++)
37    {
38       for(j=0; j<=(n-2)-i; j++)
39       {
40          if (A[j] > A[j+1])
41          {
42             aux = A[j];
43             A[j] = A[j+1];
44             A[j+1] = aux;
45          }
46       }
47    }
48    uswtime(&utime1, &stime1, &wtime1);
49
50    //Cálculo del tiempo de ejecución del programa
51    printf("\nBurbujaSimple\n");
52    printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
53    printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
54    printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
55    printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));
56    printf("\n");
57
```

```
58    /*//Para imprimir los numeros de arreglo y verificar el algoritmo
59    for(i=0; i<n; i++){
60       printf("%d, ", A[i]);
61    }*/
62 }
63
64 int main(int argc, char *argv[])
65 {
66       //Obtenemos n como parametro del main y creamos una arreglo dinamico
67    int n = atoi(argv[1]);
68    int *arreglo = (int*)calloc(n,sizeof(int));
69
70    //Con este for vamos agregando los n valores del txt al arreglo
71    for(int i=0; i<n; i++)
72    {
73       fscanf(stdin, "%d", &arreglo[i]);
74    }
75
76    printf("n = %d\n", n);
77    BurbujaSimple(arreglo, n);
78    printf("-------------------------------------\n");
79
80
81 }
```

5.2. Burbuja Optimizada

```
1  //*****************************************************************
2  //   AUTORES:
3  //    Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  //    Ramos Diaz Enrique
6  // ****************************************************************
7  //   Practica 1: Analisis de algoritmos de ordenamiento numerico
8  //   gcc tiempo.c -c
9  //   gcc BurbujaOptimizada.c tiempo.o -o BurbujaOptimizada
10 //
11 //   Ejecución: "./BurbujaOptimizada" (Linux)
12 //   ****************************************************************
13
14 //   ****************************************************************
15 //                  Librerias incluidas
16 //   ****************************************************************
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <string.h>
20 #include "tiempo.h"
21 typedef int bool;
22 #define TRUE 1
23 #define FALSE 0
24
25 //   ****************************************************************
26 //                  BURBUJA SIMPLE
27 //   ****************************************************************
28 //   Descripción: Ordena un arreglo de números
29 //   Recibe: El arreglo de números, y la cantidad de números que se ordenaran
30 //   Devuelve: Nada.
31 //   ****************************************************************
32 void BurbujaOptimizada(int A[], int n)
33 {
34    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
35    uswtime(&utime0, &stime0, &wtime0);
36
37    int j, i, aux, cambios;
38    cambios = TRUE;
39    i = 0;
```

```
40      while((i < n-1) && (cambios != FALSE))
41      {
42         cambios = FALSE;
43         for(j = 0; j <= (n-2)-i; j++)
44         {
45            if (A[j] > A[j+1])
46            {
47               aux = A[j];
48               A[j] = A[j+1];
49               A[j+1] = aux;
50               cambios = TRUE;
51            }
52         }
53         i++;
54      }
55
56      uswtime(&utime1, &stime1, &wtime1);
57
58      //Cálculo del tiempo de ejecución del programa
59      printf("\nBurbujaOptimizada\n");
60      printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
61      printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
62      printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
63      printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));
64      printf("\n");
65
66      /*//Para imprimir los numeros de arreglo y verificar el algoritmo
67      for(i=0; i<n; i++){
68         printf("%d, ", A[i]);
69      }*/
70   }
71
72   int main(int argc, char *argv[])
73   {
74         //Obtenemos n como parametro del main y creamos una arreglo dinamico
75      int n = atoi(argv[1]);
76      int *arreglo = (int*)calloc(n,sizeof(int));
77
78      //Con este for vamos agregando los n valores del txt al arreglo
79      for(int i = 0; i < n; i++)
80      {
81         fscanf(stdin, "%d", &arreglo[i]);
82      }
83
84      printf("n = %d\n", n);
85      BurbujaOptimizada(arreglo, n);
86      printf("-------------------------------------\n");
87
88
89   }
```

5.3. Ordenamiento por inserción

```c
//*****************************************************************
//AUTORES:
// Nicolas Sayago Abigail
//   Parra Garcilazo Cinthya Dolores
// Ramos Diaz Enrique
//
//Practica 1: Analisis de algoritmos de ordenamiento numerico
//Compilación:
//   gcc tiempo.c -c
//   gcc Insercion.c tiempo.o -o Insercion
//
//Ejecución: "./Insercion" (Linux)
//*****************************************************************

//*****************************************************************
//Librerias incluidas
//*****************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"

//*****************************************************************
//Insercion
//*****************************************************************
//Descripción: Función que implementa el algoritmo de ordenamiento
//de Insercion
//Recibe: Un arreglo de enteros y el tamaño del arreglo
//Devuelve: Nada, pero ordena el arreglo de menor mayor
//*****************************************************************#include <stdio.h>
void Insercion(int A[], int n)
{
   double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
   uswtime(&utime0, &stime0, &wtime0);

   int j, i, temp;
   for(i = 0; i < n; i++)
   {
      j = i;
      temp = A[i];
      while(j>0 && temp<A[j-1])
      {
         A[j] = A[j-1];
         j--;
      }
      A[j] = temp;
   }
   uswtime(&utime1, &stime1, &wtime1);


   //Cálculo del tiempo de ejecución del programa
   printf("\nInsercion\n");
   printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
   printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
   printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
   printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));
   printf("\n");

   /*//Para imprimir los numeros de arreglo y verificar el algoritmo
   for(i=0; i<n; i++){
      printf("%d, ", A[i]);
   }*/
}

int main(int argc, char *argv[])
{
   //Obtenemos n como parametro del main y creamos una arreglo dinamico
```

```
68      int n = atoi(argv[1]);
69      int *arreglo = (int*)calloc(n,sizeof(int));
70
71      //Con este for vamos agregando los n valores del txt al arreglo
72      for(int i = 0; i < n; i++)
73      {
74         fscanf(stdin, "%d", &arreglo[i]);
75      }
76
77      printf("n = %d\n", n);
78      Insercion(arreglo, n);
79      printf("-------------------------------------\n");
80  }
```

5.4. Ordenamiento por selección

```
1   //*****************************************************************
2   //AUTORES:
3   // Nicolas Sayago Abigail
4   //   Parra Garcilazo Cinthya Dolores
5   // Ramos Diaz Enrique
6   //
7   //Practica 1: Analisis de algoritmos de ordenamiento numerico
8   //Compilación:
9   //   gcc tiempo.c -c
10  //   gcc Seleccion.c tiempo.o -o Seleccion
11  //
12  //Ejecución: "./Seleccion" (Linux)
13  //*****************************************************************
14
15  //*****************************************************************
16  //Librerias incluidas
17  //*****************************************************************
18  #include <stdio.h>
19  #include <stdlib.h>
20  #include <string.h>
21  #include "tiempo.h"
22
23  //*****************************************************************
24  //Seleccion
25  //*****************************************************************
26  //Descripción: Función que implementa el algoritmo de ordenamiento
27  //de Seleccion
28  //Recibe: Un arreglo de enteros y el tamaño del arreglo
29  //Devuelve: Nada, pero ordena el arreglo de menor mayor
30  //*****************************************************************#include <stdio.h>
31  void Seleccion(int A[], int n)
32  {
33     double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
34     uswtime(&utime0, &stime0, &wtime0);
35
36     int k, p, i, temp;
37     for(k = 0; k < n-1; k++)
38     {
39        p = k;
40        for(i = k+1; i < n; i++)
41        {
42           if(A[i] < A[p])
43              p = i;
44        }
45        temp = A[p];
46        A[p] = A[k];
47        A[k] = temp;
48     }
49     uswtime(&utime1, &stime1, &wtime1);
50
```

```
51      //Cálculo del tiempo de ejecución del programa
52      printf("\nSeleccion\n");
53      printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
54      printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
55      printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
56      printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));
57      printf("\n");
58
59      /*//Para imprimir los numeros de arreglo y verificar el algoritmo
60      for(i=0; i<n; i++){
61         printf("%d, ", A[i]);
62      }*/
63  }
64
65
66
67
68  int main(int argc, char *argv[])
69  {
70      //Obtenemos n como parametro del main y creamos una arreglo dinamico
71      int n = atoi(argv[1]);
72      int *arreglo = (int*)calloc(n,sizeof(int));
73
74      //Con este for vamos agregando los n valores del txt al arreglo
75      for(int i = 0; i < n; i++)
76      {
77         fscanf(stdin, "%d", &arreglo[i]);
78      }
79
80      printf("n = %d\n", n);
81      Seleccion(arreglo, n);
82
83      printf("-------------------------------------\n");
84  }
```

5.5. Ordenamiento Shell

```
1   //*****************************************************************
2   //AUTORES:
3   // Nicolas Sayago Abigail
4   //   Parra Garcilazo Cinthya Dolores
5   // Ramos Diaz Enrique
6   //
7   //Practica 1: Analisis de algoritmos de ordenamiento numerico
8   //Compilación:
9   //   gcc tiempo.c -c
10  //   gcc Shell.c tiempo.o -o Shell
11  //
12  //Ejecución: "./Shell" (Linux)
13  //*****************************************************************
14
15  //*****************************************************************
16  //Librerias incluidas
17  //*****************************************************************
18  #include <stdio.h>
19  #include <stdlib.h>
20  #include <string.h>
21  #include "tiempo.h"
22  #include <math.h>
23
24  //*****************************************************************
25  //Shell
26  //*****************************************************************
27  //Descripción: Función que implementa el algoritmo de ordenamiento
28  //Shel
29  //Recibe: Un arreglo de enteros y el tamaño del arreglo
30  //Devuelve: Nada, pero ordena el arreglo de menor mayor
```

```
31 //*************************************************************
32
33 void Shell(int arreglo[], int n)
34 {
35    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
36    uswtime(&utime0, &stime0, &wtime0);
37
38    int k = trunc(n/2);
39    int b = 0;
40    int i = 0;
41    int temp = 0;
42
43    while(k >= 1)
44    {
45       b = 1;
46       while(b != 0)
47       {
48          b = 0;
49          for(i = k;i <= (n-1);i++)
50          {
51             if(arreglo[i-k] > arreglo[i])
52             {
53                temp = arreglo[i];
54                arreglo[i] = arreglo[i-k];
55                arreglo[i-k] = temp;
56                b = b+1;
57             }
58          }
59       }
60       k = trunc(k/2);
61    }
62
63    uswtime(&utime1, &stime1, &wtime1);
64
65    //Cálculo del tiempo de ejecución del programa
66    printf("\nShell\n");
67    printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
68    printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
69    printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
70    printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 - wtime0));
71    printf("\n");
72
73    /*//Para imprimir los numeros de arreglo y verificar el algoritmo
74    for(i=0; i<n; i++){
75       printf("%d, ", arreglo[i]);
76    }*/
77 }
78
79
80 int main (int argc,char *argv[]){
81    //Creamos arreglo dinámico
82    int n = atoi(argv[1]);
83    int *arreglo = (int*)calloc(n,sizeof(int));
84    //Agregamos los n valores del txt al arreglo
85    for(int i = 0; i < n; i++)
86    {
87       fscanf(stdin, "%d", &arreglo[i]);
88    }
89
90    printf("n = %d\n", n);
91
92    Shell(arreglo,n);
93    printf("-------------------------------------\n");
94
95    return 0;
96 }
```

5.6. Árbol binario de búsqueda

✓ Archivo .C

```
1  //*************************************************************
2  //   AUTORES:
3  //    Nicolas Sayago Abigail
4  //   Parra Garcilazo Cinthya Dolores
5  //    Ramos Diaz Enrique
6  // ************************************************************
7  //   Practica 1: Analisis de algoritmos de ordenamiento numerico
8  //   Compilación:
9  //   gcc tiempo.c -c
10 //   gcc ABB.c tiempo.o -o ABB
11 //
12 //   Ejecución: "./ABB" (Linux)
13 //   ************************************************************
14
15 //   ************************************************************
16 //                  Librerias incluidas
17 //   ************************************************************
18 #include <stdio.h>
19 #include <stdlib.h>
20 #include "Arbin.h"
21 #include "tiempo.h"
22
23 //   ************************************************************
24 //                  Insertar
25 //   ************************************************************
26 //   Descripción: Inserta un arreglo de números en un ABB
27 //   Recibe: Un ABB vacio y un entero
28 //   Devuelve: Nada, pero construye el arbol binario
29 //   ************************************************************
30 void Insertar(Arbin *a, int e)
31 {
32    Arbin *apu_a = a; // Declaramos un apuntador para recorrer el árbol
33    while (*apu_a != NULL)
34    {
35       if (e > ((*apu_a) -> raiz))
36          apu_a = &((*apu_a) -> der);
37       else
38          apu_a = &((*apu_a) -> izq);
39    }
40    *apu_a = (NodoA *)malloc(sizeof(NodoA));
41    (*apu_a) -> raiz = e;          // En el nodo colocaremos el elemento a introducir en el
        ↪  árbol
42    (*apu_a) -> izq = NULL;       // Nos aseguramos de que ambos hijos estén apuntando a un
        ↪  valor NULL
43    (*apu_a) -> der = NULL;
44 }
45
46 //   ************************************************************
47 //                  GuardarRecorridoInOrden
48 //   ************************************************************
49 //   Descripción: Guarda los elementos de un arbol en un arreglo
50 //   Recibe: Un arbol binario, un arreglo y el tamaño del arreglo
51 //   Devuelve: Nada, pero coloca los elementos del arbol ordenados
52 //           dentro del arreglo
53 //   ************************************************************
54 void GuardarRecorridoInOrden(Arbin *a, int A[], int n)
55 {
56    posicion a_aux = *a; // Declaramos un apuntador auxiliar para viajar por el árbol
57    NodoA **pila = (NodoA **)malloc(n * sizeof(Arbin)); // Inicializamos una pila de nodos para
        ↪  guardar valores del recorrido
58    int tope = -1; // Tope de la pila
59    int i = 0;
60
61    do
```

```
62      {
63         while (a_aux != NULL)
64         {   // Haremos un recorrido hasta llegar a la parte más izquierda
65            pila[++tope] = a_aux; // Iremos colocando en la pila los nodos de la izquierda
66            a_aux = a_aux -> izq;
67         }
68
69         if (tope >= 0){   // Una vez llegado a la parte más izquierda, verificamos si quedan
             ↪  nodos en la pila
70            a_aux = pila[tope--]; // Sacaremos el último nodo de la pila que será la "raíz" de
                ↪  ese subárbol
71            A[i++] = a_aux -> raiz; // Ese nodo será ingresado al arreglo de números,
                ↪  posteriormente moveremos el índice un lugar más
72            a_aux = a_aux -> der;     // Ya que quitamos la "raíz", pasaremos a recorrer el lado
                ↪  derecho del subárbol
73         }
74      }
75      while (a_aux != NULL || tope >= 0); // Apuntador nulo y no tenemos más nodos que recorrer
          ↪  en la pila
76
77      free(pila);
78   }
79
80   //   ************************************************************
81   //                  OrdenaConArbolBinario
82   //   ************************************************************
83   //   Descripción: Ordena un arreglo de numeros utilizando diccionarios binarios
84   //   Recibe: Un arreglo de enteros y el tamaño del arreglo
85   //   Devuelve: Nada, pero ordena los elementos del arreglo
86   //   ************************************************************
87   void OrdenaConArbolBinario(int A[], int n)
88   {
89      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables para medición de tiempos
90      uswtime(&utime0, &stime0, &wtime0);
91
92      Arbin ArbolBinBusqueda;
93      consA(&ArbolBinBusqueda); //Asignamos el valor NULL al apuntador del ABB
94      int i, j = 0;
95
96      for(i = 0; i < n; i++)
97      {
98         Insertar(&ArbolBinBusqueda, A[i]);
99      }
100
101     GuardarRecorridoInOrden(&ArbolBinBusqueda, A, n);
102
103     destruir(&ArbolBinBusqueda);
104
105     uswtime(&utime1, &stime1, &wtime1);
106
107     //Cálculo del tiempo de ejecución del programa
108     printf("\nArbol de Busqueda Binario\n");
109     printf("real (Tiempo total)  %.35f s\n",  wtime1 - wtime0);
110     printf("user (Tiempo de procesamiento en CPU) %.35f s\n",  utime1 - utime0);
111     printf("sys (Tiempo en acciónes de E/S)  %.35f s\n",  stime1 - stime0);
112     printf("CPU/Wall   %.35f %% \n",100.0 * (utime1 - utime0 + stime1 - stime0) / (wtime1 -
          ↪  wtime0));
113     printf("\n");
114
115     /*//Para imprimir los numeros de arreglo y verificar el algoritmo
116     for(i=0; i<n; i++){
117        printf("%d, ", A[i]);
118     }*/
119  }
120
121  int main(int argc, char *argv[])
122  {
123     //Obtenemos n como parametro del main y creamos una arreglo dinamico
124     int n = atoi(argv[1]);
```

```
125     int *arreglo = (int*)calloc(n,sizeof(int));
126
127     printf("n = %d\n", n);
128     //Con este for vamos agregando los n valores del txt al arreglo
129     for(int i = 0; i < n; i++)
130     {
131        fscanf(stdin, "%d", &arreglo[i]);
132     }
133
134     OrdenaConArbolBinario(arreglo, n);
135     printf("------------------------------------\n");
136
137     return 0;
138  }
```

✓ Archivo .H

```
1   //*****************************************************************
2   //   AUTORES:
3   //    Nicolas Sayago Abigail
4   //   Parra Garcilazo Cinthya Dolores
5   //    Ramos Diaz Enrique
6   // ****************************************************************
7   //   Practica 1: Analisis de algoritmos de ordenamiento numerico
8   //   ****************************************************************
9
10  //   ****************************************************************
11  //         Librerias
12  //   ****************************************************************
13
14  #include "stdlib.h"
15
16  //Defimos la estructura de un arbol binario
17  //Consiste en una raiz, un nodo izquierdo y un bodo derecho
18  typedef struct NodoA
19  {
20     struct NodoA *izq;
21     struct NodoA *der;
22     int raiz;
23  } NodoA;
24
25  typedef NodoA *posicion;   // La posición será la dirección hacia un NodoA específico
26  typedef posicion Arbin; // El árbol binario será simplemente una posición de un NodoA
27
28  //   ****************************************************************
29  //                  consA
30  //   ****************************************************************
31  //   Descripción: Construye un arbol binario
32  //   Recibe: Un apuntador al arbol binario
33  //   Devuelve:
34  //   ****************************************************************
35  void consA(Arbin *a)
36  {
37     *a = NULL; // El apuntador enviado por el usuario se coloca en un valor NULL
38  }
39
40  //   ****************************************************************
41  //                  destruir
42  //   ****************************************************************
43  //   Descripción: Elimina de la memoria un arbol binario
44  //   Recibe: Un apuntador al arbol binario
45  //   Devuelve:
46  //   ****************************************************************
47  void destruir(Arbin *a)
48  {
49     if (*a != NULL) // Veirficamos no estar apuntando a un valor nulo en el árbol enviado
50     {
51        if ((*a) -> izq != NULL)      // Verificamos si el árbol izquierdo existe, para
          ↪  eliminarlo primero
```

```
52            destruir(&((*a) -> izq));    // Llamamos recursivamente la función por el lado
                ↪  izquierdo
53          if ((*a) -> der != NULL)      // Posteriormente eliminamos el lado derecho del árbol
              ↪  verificando que existe
54            destruir(&((*a) -> der)); // Llamamos recursivamente la función por el lado derecho
55          free(*a);                           // Liberamos el nodo una vez que ya no tiene hijos
56      }
57  }
```

5.7. Script de Compilación

```
#!/bin/bash
#Para ejecutar el script e iniciar las pruebas: sh script.sh
gcc tiempo.c -c
gcc BurbujaSimple.c tiempo.o -o BurbujaSimple
./BurbujaSimple 100 <numeros10millones.txt >>salida.txt
./BurbujaSimple 500 <numeros10millones.txt >>salida.txt
./BurbujaSimple 800 <numeros10millones.txt >>salida.txt
./BurbujaSimple 1000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 1500 <numeros10millones.txt >>salida.txt
./BurbujaSimple 2000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 4000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 6000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 8000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 10000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 15000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 20000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 40000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 60000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 80000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 90000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 100000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 120000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 150000 <numeros10millones.txt >>salida.txt
./BurbujaSimple 200000 <numeros10millones.txt >>salida.txt

gcc BurbujaOptimizada.c tiempo.o -o BurbujaOptimizada
./BurbujaOptimizada 100 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 500 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 800 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 1000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 1500 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 2000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 4000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 6000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 8000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 10000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 15000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 20000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 40000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 60000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 80000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 90000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 100000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 120000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 150000 <numeros10millones.txt >>salida.txt
./BurbujaOptimizada 200000 <numeros10millones.txt >>salida.txt

gcc Insercion.c tiempo.o -o Insercion
./Insercion 100 <numeros10millones.txt >>salida.txt
./Insercion 500 <numeros10millones.txt >>salida.txt
./Insercion 1000 <numeros10millones.txt >>salida.txt
./Insercion 2000 <numeros10millones.txt >>salida.txt
./Insercion 5000 <numeros10millones.txt >>salida.txt
./Insercion 8000 <numeros10millones.txt >>salida.txt
./Insercion 9000 <numeros10millones.txt >>salida.txt
```

```
./Insercion 10000 <numeros10millones.txt >>salida.txt
./Insercion 20000 <numeros10millones.txt >>salida.txt
./Insercion 50000 <numeros10millones.txt >>salida.txt
./Insercion 70000 <numeros10millones.txt >>salida.txt
./Insercion 90000 <numeros10millones.txt >>salida.txt
./Insercion 100000 <numeros10millones.txt >>salida.txt
./Insercion 200000 <numeros10millones.txt >>salida.txt
./Insercion 400000 <numeros10millones.txt >>salida.txt
./Insercion 500000 <numeros10millones.txt >>salida.txt
./Insercion 600000 <numeros10millones.txt >>salida.txt
./Insercion 800000 <numeros10millones.txt >>salida.txt
./Insercion 1000000 <numeros10millones.txt >>salida.txt
./Insercion 2000000 <numeros10millones.txt >>salida.txt

gcc Seleccion.c tiempo.o -o Seleccion
./Seleccion 100 <numeros10millones.txt >>salida.txt
./Seleccion 500 <numeros10millones.txt >>salida.txt
./Seleccion 1000 <numeros10millones.txt >>salida.txt
./Seleccion 2000 <numeros10millones.txt >>salida.txt
./Seleccion 5000 <numeros10millones.txt >>salida.txt
./Seleccion 8000 <numeros10millones.txt >>salida.txt
./Seleccion 9000 <numeros10millones.txt >>salida.txt
./Seleccion 10000 <numeros10millones.txt >>salida.txt
./Seleccion 20000 <numeros10millones.txt >>salida.txt
./Seleccion 50000 <numeros10millones.txt >>salida.txt
./Seleccion 70000 <numeros10millones.txt >>salida.txt
./Seleccion 90000 <numeros10millones.txt >>salida.txt
./Seleccion 100000 <numeros10millones.txt >>salida.txt
./Seleccion 200000 <numeros10millones.txt >>salida.txt
./Seleccion 400000 <numeros10millones.txt >>salida.txt
./Seleccion 500000 <numeros10millones.txt >>salida.txt
./Seleccion 600000 <numeros10millones.txt >>salida.txt
./Seleccion 800000 <numeros10millones.txt >>salida.txt
./Seleccion 900000 <numeros10millones.txt >>salida.txt
./Seleccion 1000000 <numeros10millones.txt >>salida.txt


gcc Shell.c tiempo.o -o Shell
./Shell 100 <numeros10millones.txt >>salida.txt
./Shell 1000 <numeros10millones.txt >>salida.txt
./Shell 5000 <numeros10millones.txt >>salida.txt
./Shell 10000 <numeros10millones.txt >>salida.txt
./Shell 50000 <numeros10millones.txt >>salida.txt
./Shell 100000 <numeros10millones.txt >>salida.txt
./Shell 200000 <numeros10millones.txt >>salida.txt
./Shell 400000 <numeros10millones.txt >>salida.txt
./Shell 600000 <numeros10millones.txt >>salida.txt
./Shell 800000 <numeros10millones.txt >>salida.txt
./Shell 1000000 <numeros10millones.txt >>salida.txt
./Shell 2000000 <numeros10millones.txt >>salida.txt
./Shell 3000000 <numeros10millones.txt >>salida.txt
./Shell 4000000 <numeros10millones.txt >>salida.txt
./Shell 5000000 <numeros10millones.txt >>salida.txt
./Shell 6000000 <numeros10millones.txt >>salida.txt
./Shell 7000000 <numeros10millones.txt >>salida.txt
./Shell 8000000 <numeros10millones.txt >>salida.txt
./Shell 9000000 <numeros10millones.txt >>salida.txt
./Shell 10000000 <numeros10millones.txt >>salida.txt

gcc ABB.c tiempo.o -o ABB
./ABB 100 <numeros10millones.txt >>salida.txt
./ABB 1000 <numeros10millones.txt >>salida.txt
./ABB 5000 <numeros10millones.txt >>salida.txt
./ABB 10000 <numeros10millones.txt >>salida.txt
./ABB 50000 <numeros10millones.txt >>salida.txt
./ABB 100000 <numeros10millones.txt >>salida.txt
./ABB 200000 <numeros10millones.txt >>salida.txt
./ABB 400000 <numeros10millones.txt >>salida.txt
./ABB 600000 <numeros10millones.txt >>salida.txt
```

```
./ABB 800000 <numeros10millones.txt >>salida.txt
./ABB 1000000 <numeros10millones.txt >>salida.txt
./ABB 2000000 <numeros10millones.txt >>salida.txt
./ABB 3000000 <numeros10millones.txt >>salida.txt
./ABB 4000000 <numeros10millones.txt >>salida.txt
./ABB 5000000 <numeros10millones.txt >>salida.txt
./ABB 6000000 <numeros10millones.txt >>salida.txt
./ABB 7000000 <numeros10millones.txt >>salida.txt
./ABB 8000000 <numeros10millones.txt >>salida.txt
./ABB 9000000 <numeros10millones.txt >>salida.txt
./ABB 10000000 <numeros10millones.txt >>salida.txt
```

6. Bibliografía

[1] E. A. Franco Martínez, “Practica 01 : Pruebas a posteriori (Algoritmos de Ordenamiento)” Análisis de Algoritmos, Escuela Superior de Computación, Instituto Politécnico Nacional, Ciudad de México, México, Practica01.pdf, Sep. 2018

[2] (2018) WolframAlpha. Accessed september 2018. [Online].
Available: http://www.wolframalpha.com/