Universidad de Granada

Algorítmica

Memoria de Prácticas

Práctica I: Eficiencia

Autores:
Elena Merelo Molina
Antonio Gámiz Delgado

17 de Marzo

Universidad de Granada

1 Análisis de la Eficiencia

1.1 Algoritmo de Inserción

```
1  inline static void insercion(int T[], int num_elem)
2  {
3    insercion_lims(T, 0, num_elem);
4  }
5
6  static void insercion_lims(int T[], int inicial, int final)
7  {
8    int i, j;
9    int aux;
10   for (i = inicial + 1; i < final; i++) {
11     j = i;
12     while ((T[j] < T[j-1]) && (j > 0)) {
13       aux = T[j];
14       T[j] = T[j-1];
15       T[j-1] = aux;
16       j--;
17     };
18   };
19 }
```

Estudiemos el peor caso que se le podría presentar al algoritmo de inserción: que el vector estuviera ordenado en orden inverso (de mayor a menor).

Como vemos en la línea 5, siempre se llama al método con los argumentos 0 y *num_elem*, por lo que a partir de ahora para nosotros, *inicial* será 0 y $final$ será n.

La mayor parte del tiempo de ejecución se emplea en el cuerpo del bucle *while* interno. Ese trozo de código se puede acotar por una constante a. Por lo tanto, las líneas 15-19 se ejecutan un número de veces dependiente del bucle externo, exactamente i veces (ya que estamos suponiendo que nos encontramos en el peor caso). El bucle externo se ejecuta exactamente n veces, por lo que nos queda:

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=1}^{i} a = a \sum_{i=0}^{n-1} \sum_{j=1}^{i} 1 = a \sum_{i=0}^{n-1} i = a \frac{n(n-1)}{2}$$

Por lo que vemos que $T(n) \in O(n^2)$, es cuadrático.

1.2 Algoritmo Selección

```
1  void seleccion(int T[], int num_elem)
2  {
3    seleccion_lims(T, 0, num_elem);
4  }
5
6  static void seleccion_lims(int T[], int inicial, int final)
7  {
8    int i, j, indice_menor;
9    int menor, aux;
10   for (i = inicial; i < final - 1; i++) {
11     indice_menor = i;
12     menor = T[i];
13     for (j = i; j < final; j++)
14       if (T[j] < menor) {
15       indice_menor = j;
16       menor = T[j];
17       }
18     aux = T[i];
19     T[i] = T[indice_menor];
20     T[indice_menor] = aux;
21   };
22 }
```

Vamos a estudiar el peor caso que se le podría presentar al algoritmo de selección, es decir, que el vector estuviera ordenado a la inversa (de mayor a menor).

Como vemos en la línea 3, siempre se llama al método con los argumentos 0 y *num_elem*, por lo que a partir de ahora para nosotros, *inicial* será 0 y $final$ será n.

Las líneas 8 y 9 son de orden $O(1)$ por lo que las descartamos. Vemos que el prácticamente todo el tiempo de ejecución es consumido por los dos bucles for de las líneas 10 y 13. El primer bucle se ejecuta n veces, mientras que el bucle interior se ejecutará $n - i$ veces por cada iteración del bucle anterior. Por lo que tenemos:

$$\sum_{i=0}^{n-1}(n-i-1) = \sum_{i=0}^{n-1}(n-1) - \sum_{i=0}^{n-1} i = (n-1)n - \frac{n(n+1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Por lo que tenemos que, $T(n) \in O(n^2)$, donde $T(n)$ es la función de eficiencia del algoritmo de selección.

1.3 Algoritmo Heapsort

```
1
2  static void heapsort(int T[], int num_elem)
3  {
4    int i;
5    for (i = num_elem/2; i >= 0; i--)
6      reajustar(T, num_elem, i);
7    for (i = num_elem - 1; i >= 1; i--)
8      {
9        int aux = T[0];
10       T[0] = T[i];
11       T[i] = aux;
12       reajustar(T, i, 0);
13     }
14 }
15
16 static void reajustar(int T[], int num_elem, int k)
17 {
18   int j;
19   int v;
20   v = T[k];
21   bool esAPO = false;
22   while ((k < num_elem/2) && !esAPO)
23     {
24       j = k + k + 1;
25
26       if ((j < (num_elem - 1)) && (T[j] < T[j+1])) j++;
27       if (v >= T[j]) esAPO = true;
28
29       T[k] = T[j];
30       k = j;
31     }
32   T[k] = v;
33 }
```

Vemos que en el primer bucle de la función *heapsort* aparece la función *reajustar*, por lo que vamos a calcular su eficiencia primero.

El cuerpo del bucle *while* consume la mayor parte del tiempo de ejecución, y se puede acotar por una constante b. Como se puede observar en la línea 22, el bucle empieza en k y termina en $\frac{n}{2}$, pero k no avanza de 1 en 1, sino de $2k+1$ en $2k+1$, por lo que como mucho se ejecutará $\log(n/2)$ veces. Así, $R(n) \in O(\log(n))$, siendo $K(n)$ la función de eficiencia de la función *reajustar*. Una vez conocida la eficiencia de la función *reajustar*, pasamos a estudiar el primer bucle de la función *heapsort*. Va desde $n/2$ hasta 0, luego se ejecuta $n/2$ veces, y cada una de esas veces ejecuta la función *reajustar*, por lo que entonces su eficiencia es $\frac{nlog(n)}{2}$. El segundo bucle se ejecuta $n-1$ veces, quedando:

$$T(n) = \frac{nlog(n)}{2} + (n-1)$$

Por lo que $T(n) \in O(n\log(n))$.

1.4 Algoritmo de Floyd

```
1  void Floyd(int **M, int dim)
2  {
3    for (int k = 0; k < dim; k++)
4      for (int i = 0; i < dim;i++)
5        for (int j = 0; j < dim;j++)
6          {
7          int sum = M[i][k] + M[k][j];
8          M[i][j] = (M[i][j] > sum) ? sum : M[i][j];
9          }
10 }
```

Como se puede observar, el cuerpo del tercer bucle *while* anidado consume la mayor parte del tiempo de ejecución, por lo que lo acotamos por una constante a. El resto de bucles va desde 0 hasta *dim*, que podemos denominar n para mayor facilidad. Por lo que evidentemente tenemos que la eficiencia de este algoritmo es an^3, es decir, $T(n) \in O(n^3)$.

1.5 Algoritmo de Hanoi

```
1  void hanoi (int M, int i, int j)
2  {
3    if (M > 0)
4      {
5        hanoi(M-1, i, 6-i-j);
6        hanoi (M-1, 6-i-j, j);
7    }
8  }
```

De la clase de teoría sabemos que la función de Hanoi tiene como función de eficienciaa:

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + 1 & n \geq 1 \\ 1 & n = 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2\overbrace{T(n-1)}^{2T(n-2)+1} + 1 && n > 1 \\ T(n) &= 2^2T(n-2) + 2 + 1 && n > 2 \\ T(n) &= 2^iT(n-i) + (2^{i-1} + \cdots + 2^2 + 2 + 1) && n > i \end{aligned}$$

Por lo que si tomamos $i = n - 1$:

$$T(n) = 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1 = \frac{2^{n-1}2 - 1}{2 - 1} = 2^n - 1$$

Por lo que tenemos que $T(n) \in O(2^n)$.

2 Eficiencia empírica

En esta práctica vamos a estudiar la eficiencia empírica de los siguientes algoritmos:

Algoritmo	Orden de Eficiencia
Burbuja	$\mathrm{O}(n^2)$
Inserción	$\mathrm{O}(n^2)$
Selección	$\mathrm{O}(n^2)$
Mergesort	$\mathrm{O}(n\log(n))$
Quicksort	$\mathrm{O}(n\log(n))$
Heapsort	$\mathrm{O}(n\log(n))$
Floyd	$\mathrm{O}(n^3)$
Hanoi	$\mathrm{O}(2^n)$

Para el estudio de la eficiencia empírica hemos escrito un script en *Python*2.7 para facilitar el trabajo de la gestión de los datos. Primeramente añadimos a cada programa unas líneas de código y de esta manera medir el tiempo que ha tardado en ejecutarse. Luego compilamos cada programa con 3 opciones de optimización distintas (sin optimizar, con -O1 y con -O2) con g++, y una vez obtenidos los ejecutables, ejecutamos cada grupo de programas 25 veces con diferentes valores de n (tamaño) para cada algoritmo y guardamos los resultados en varios archivos (los datos obtenidos se pueden ver en las tablas de abajo). Una vez hecho todo esto, ajustamos aplicando el método de mínimos cuadrados los datos con su orden de eficiencia:

1. $O(n^2)$: ajustada por la función $f(x) = ax^2 + bx + c$
2. $O(n\log(n)$: ajustada por la función $f(x) = a\log(bn) + c$
3. $O(n^3)$: ajustada por la función $f(x) = ax^3 + bx^2 + cx + d$
4. $O(2^n)$: ajustada por la función $f(x) = a2^{bn} + c$

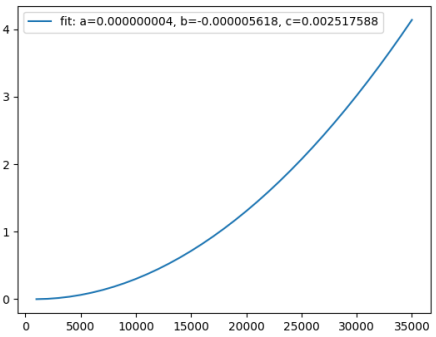
Los valores obtenidos para los coeficientes (a, b, c, d) aparecen en la legenda de las gráficas según la opción de optimización usada.

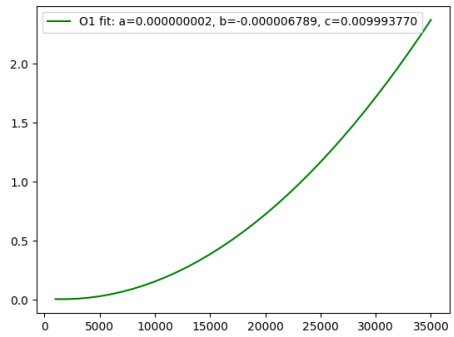
Las última gráfica de cada página muestra como varía la eficiencia del algoritmo en función de la opción de optimización usada.

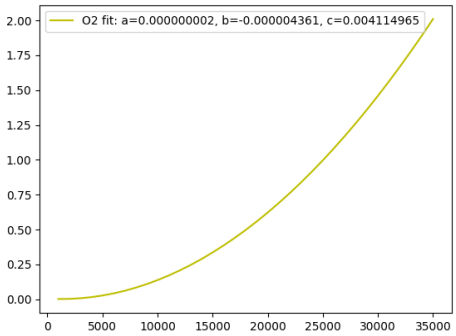
A continuación mostramos todos los datos obtenidos junto a las gráficas de los algoritmos expuestos arriba:

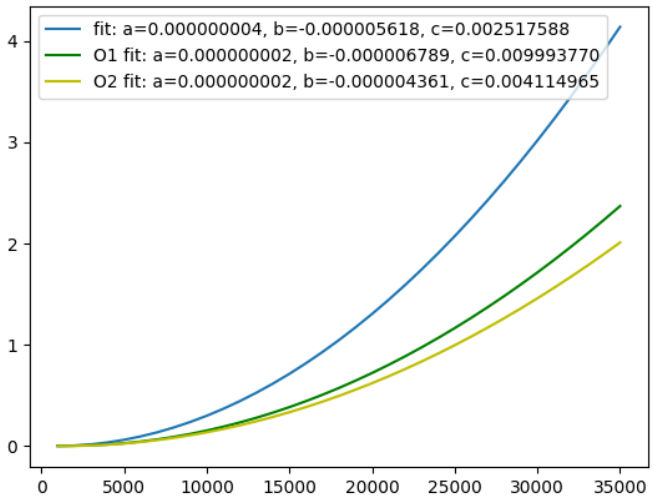
[0] Adjuntamos el código del script pero no lo explicamos ya que creemos que no es el objetivo de esta práctica.

2.1 Algoritmo burbuja (Elena)



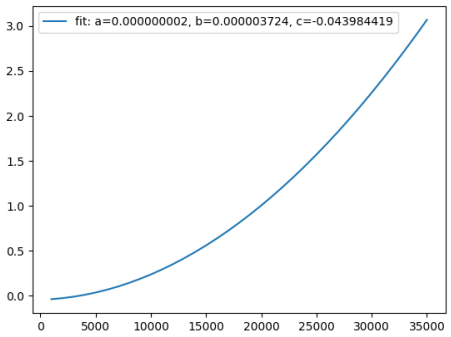


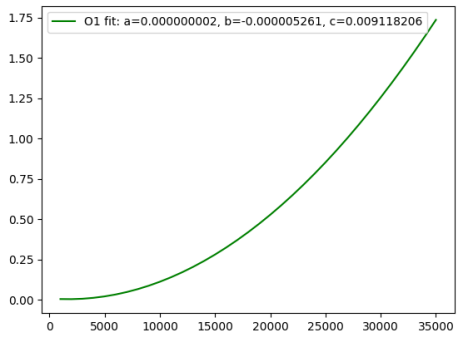


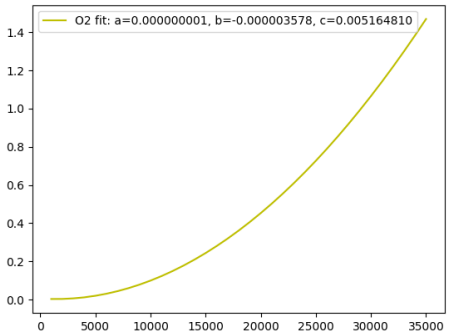


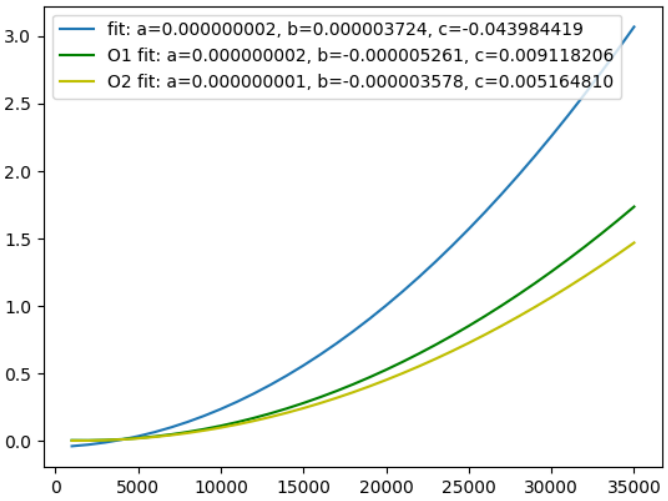
N	O-	O1	O2
1000	0.00224433	0.00102678	0.000791685
2000	0.00855251	0.00366315	0.00327288
3000	0.0197315	0.0089917	0.00777837
4000	0.0372338	0.0179805	0.0159009
5000	0.0612263	0.0307844	0.0272179
6000	0.0931501	0.0467117	0.0426332
7000	0.141121	0.0679487	0.0618318
8000	0.189752	0.0933759	0.0797291
9000	0.235398	0.122148	0.105121
10000	0.293126	0.154214	0.136209
11000	0.372834	0.193827	0.167798
12000	0.434022	0.239529	0.202906
13000	0.52755	0.278723	0.255706
14000	0.608207	0.326521	0.286101
15000	0.720389	0.387952	0.331906
16000	0.805799	0.440765	0.388973
17000	0.93492	0.508819	0.429878
18000	1.03981	0.567453	0.503668
19000	1.18039	0.642401	0.54615
20000	1.3095	0.725319	0.62702
21000	1.4486	0.800284	0.681884
22000	1.57821	0.887376	0.75987
23000	1.76802	0.970347	0.839553
24000	1.89164	1.06591	0.913394
25000	2.06779	1.17406	1.00485
26000	2.25955	1.27417	1.08369
27000	2.43821	1.36573	1.17166
28000	2.61991	1.49046	1.27354
29000	2.81486	1.59349	1.35703
30000	3.01869	1.71685	1.45953
31000	3.23556	1.84846	1.56174
32000	3.45231	1.95537	1.68971
33000	3.65181	2.0881	1.75914
34000	3.91054	2.24322	1.8894
35000	4.12783	2.3636	2.01188

2.2 Algoritmo burbuja (Antonio)



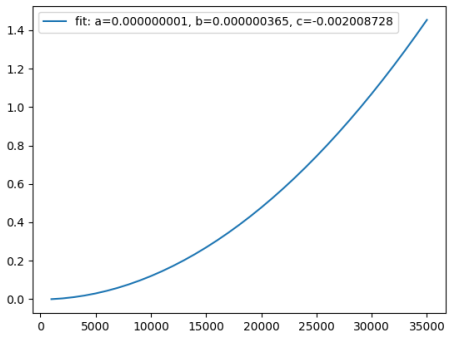


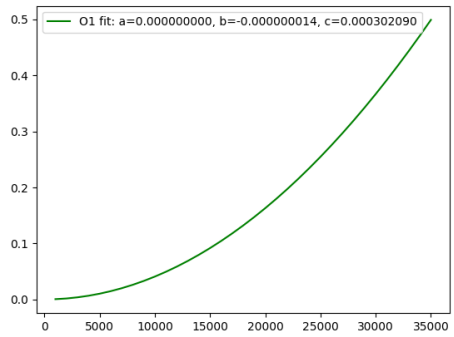


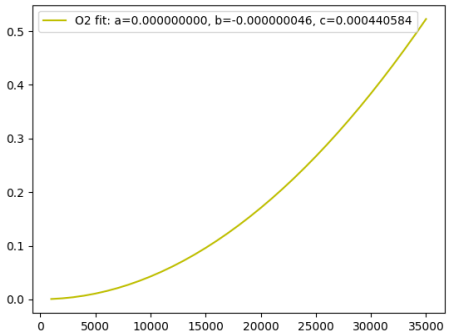


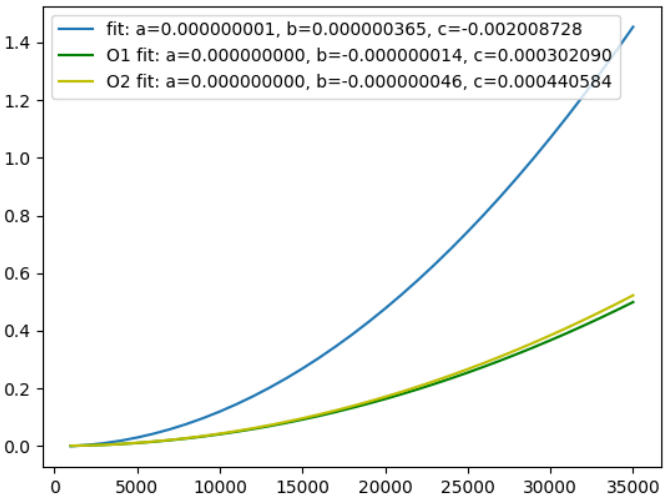
N	O-	O1	O2
1000	0.00163878	0.000677493	0.00057825
2000	0.00648541	0.00275606	0.0023118
3000	0.0144959	0.00669238	0.00550926
4000	0.0271765	0.013301	0.0112792
5000	0.0460665	0.0224053	0.0189794
6000	0.0686737	0.0349273	0.030044
7000	0.0985109	0.0502273	0.044031
8000	0.130974	0.0682356	0.0604623
9000	0.17042	0.0905028	0.0795016
10000	0.21466	0.114268	0.101166
11000	0.267826	0.140893	0.124345
12000	0.319583	0.173099	0.149805
13000	0.380808	0.204735	0.178115
14000	0.447768	0.240566	0.208851
15000	0.516807	0.284908	0.24392
16000	0.59053	0.330246	0.283318
17000	0.66985	0.370028	0.321894
18000	0.765694	0.416786	0.362117
19000	0.86069	0.472635	0.406441
20000	0.974703	0.524153	0.451943
21000	1.06859	0.58658	0.501424
22000	1.24213	0.650685	0.55244
23000	1.37154	0.711542	0.608728
24000	1.59735	0.773852	0.665455
25000	1.63384	0.845037	0.722579
26000	1.77427	0.920397	0.787619
27000	1.88275	1.00539	0.852449
28000	2.1594	1.10168	0.925192
29000	2.18708	1.15907	0.986625
30000	2.22694	1.26373	1.06383
31000	2.36625	1.33993	1.14153
32000	2.49679	1.44225	1.21892
33000	2.67365	1.53568	1.30066
34000	2.83929	1.62477	1.38277
35000	3.01601	1.74507	1.47573

2.3 Algoritmo insercion(Elena)



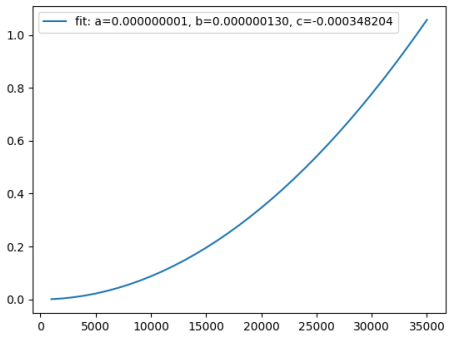


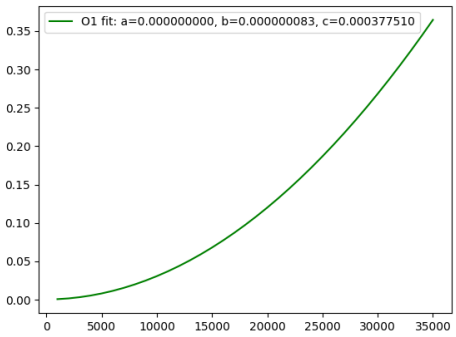


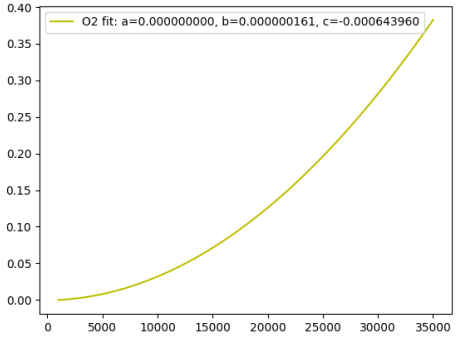


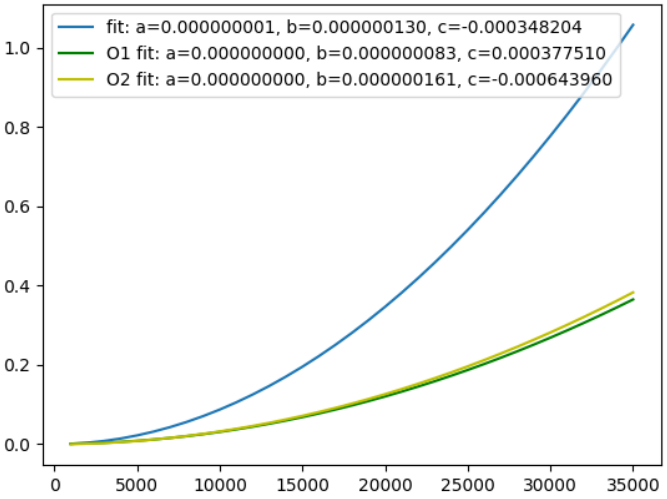
N	O-	O1	O2
1000	0.00128062	0.000394111	0.000471134
2000	0.00494392	0.00164918	0.00169287
3000	0.0107846	0.00370318	0.00386223
4000	0.0190363	0.00655303	0.00742284
5000	0.0290049	0.0104249	0.0111018
6000	0.0417463	0.0149241	0.0154514
7000	0.056829	0.0203957	0.0210657
8000	0.074506	0.0262846	0.0275941
9000	0.0937439	0.0331596	0.0347036
10000	0.120899	0.0413018	0.0427021
11000	0.146712	0.049628	0.05191
12000	0.16882	0.0588039	0.0618677
13000	0.197715	0.0691303	0.0729195
14000	0.229951	0.0795406	0.0822059
15000	0.266283	0.0931013	0.0958826
16000	0.308386	0.105894	0.109572
17000	0.339635	0.118275	0.123054
18000	0.395265	0.131365	0.13857
19000	0.430017	0.146577	0.154501
20000	0.472572	0.163316	0.172057
21000	0.520983	0.180018	0.192359
22000	0.577108	0.200415	0.204682
23000	0.661747	0.215635	0.224426
24000	0.699163	0.234647	0.244675
25000	0.745969	0.252221	0.270063
26000	0.790776	0.277755	0.285303
27000	0.85076	0.294963	0.308426
28000	0.936871	0.315499	0.33406
29000	1.0011	0.341928	0.357148
30000	1.06186	0.365663	0.382002
31000	1.14096	0.389944	0.411422
32000	1.20885	0.424888	0.43873
33000	1.28216	0.441153	0.46848
34000	1.3752	0.475582	0.488904
35000	1.47001	0.497147	0.52555

2.4 Algoritmo insercion(Antonio)



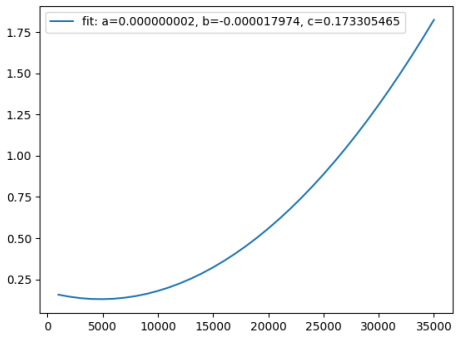


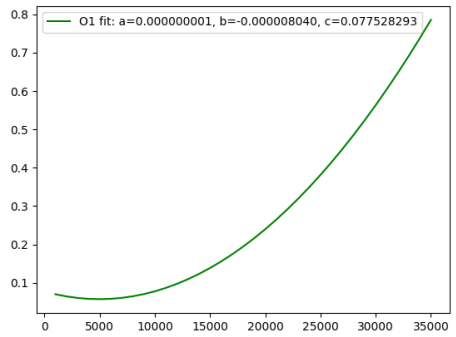


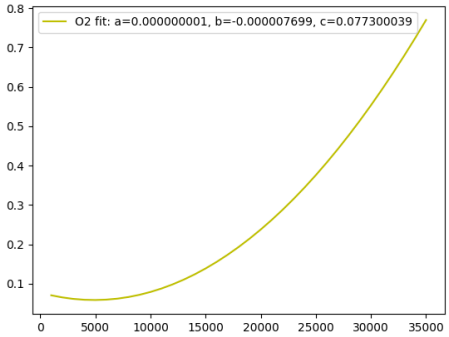


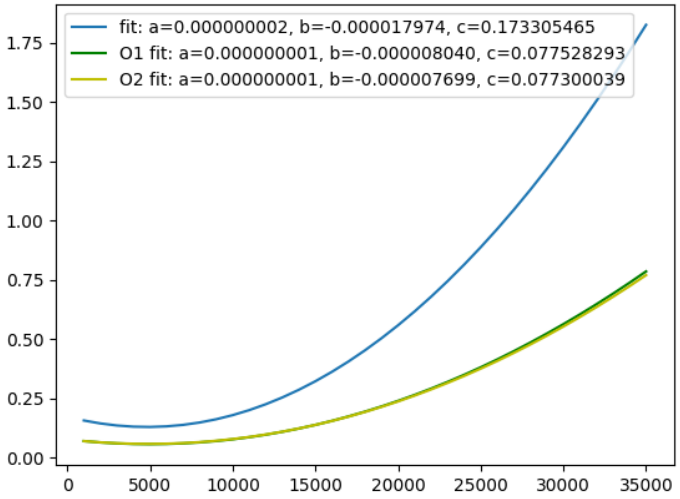
N	O-	O1	O2
1000	0.0008668	0.000303425	0.000311528
2000	0.00341429	0.00118489	0.00119304
3000	0.00796838	0.00273074	0.00288727
4000	0.0144405	0.00495307	0.00499236
5000	0.021806	0.00779367	0.00801092
6000	0.0316326	0.0111679	0.0112941
7000	0.0426453	0.015719	0.0154039
8000	0.0556216	0.0203325	0.0201064
9000	0.0702031	0.0250042	0.0253493
10000	0.0864261	0.0334055	0.0316306
11000	0.105371	0.0371124	0.0379618
12000	0.12481	0.0437312	0.044747
13000	0.147354	0.0531294	0.0528693
14000	0.169738	0.0605292	0.0612958
15000	0.195663	0.0678626	0.0703497
16000	0.22287	0.0775122	0.0801611
17000	0.248004	0.0857529	0.0916148
18000	0.276891	0.0973823	0.102555
19000	0.310587	0.108259	0.115199
20000	0.346986	0.119829	0.127033
21000	0.383003	0.131816	0.139442
22000	0.420818	0.144409	0.152354
23000	0.457508	0.158134	0.166343
24000	0.496786	0.172642	0.181194
25000	0.538522	0.185073	0.199013
26000	0.585611	0.201248	0.211649
27000	0.627596	0.21768	0.227843
28000	0.675989	0.236125	0.245768
29000	0.728514	0.249836	0.26064
30000	0.777847	0.267376	0.279988
31000	0.829541	0.284786	0.299098
32000	0.891975	0.305774	0.317623
33000	0.945496	0.324868	0.341622
34000	0.99809	0.344282	0.361573
35000	1.04677	0.366523	0.384241

2.5 Algoritmo seleccion(Elena)



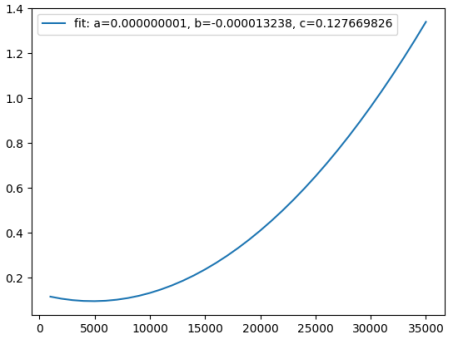


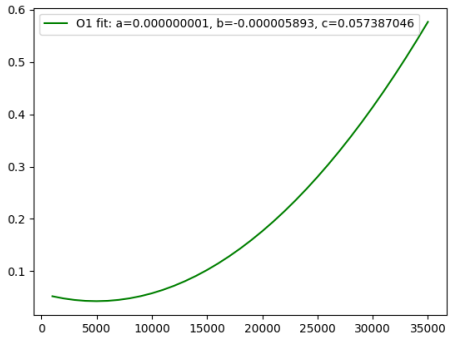


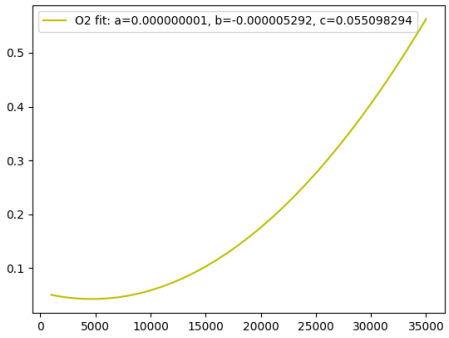


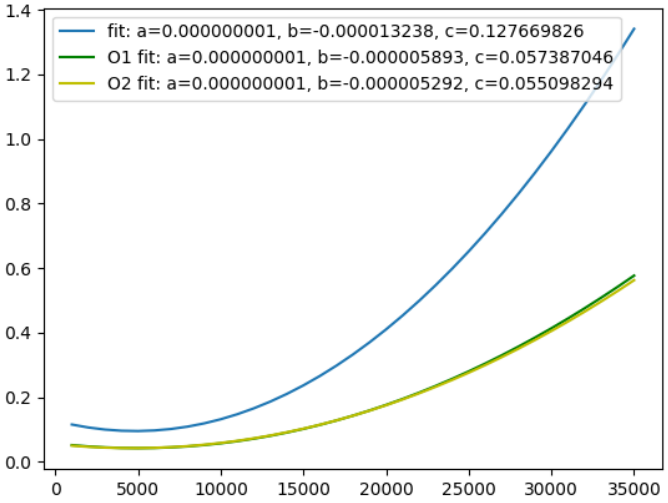
N	O-	O1	O2
1000	0.153329	0.0689078	0.0701804
2000	0.600164	0.26289	0.26924
3000	0.01341	0.00584541	0.00596033
4000	0.0234943	0.0102879	0.0107875
5000	0.0371377	0.016987	0.0159352
6000	0.0532939	0.0253317	0.02325
7000	0.0719599	0.0312657	0.0323246
8000	0.094446	0.0413923	0.0404254
9000	0.118314	0.0536091	0.0524062
10000	0.150984	0.0648981	0.0645619
11000	0.176848	0.0763845	0.0794243
12000	0.20946	0.09047	0.0926502
13000	0.247244	0.106174	0.106671
14000	0.28852	0.122478	0.123979
15000	0.328321	0.140205	0.136617
16000	0.373965	0.159223	0.158134
17000	0.42646	0.182532	0.179374
18000	0.470287	0.209179	0.205501
19000	0.526928	0.224175	0.226953
20000	0.581037	0.250155	0.246477
21000	0.643472	0.276663	0.273648
22000	0.707557	0.308111	0.310535
23000	0.769712	0.329961	0.321693
24000	0.839798	0.36345	0.355474
25000	0.909453	0.391991	0.387178
26000	0.98656	0.422566	0.412706
27000	1.06781	0.460135	0.461601
28000	1.14912	0.48845	0.478346
29000	1.23083	0.526664	0.52504
30000	1.31034	0.559554	0.547864
31000	1.39459	0.601733	0.589723
32000	1.49614	0.645822	0.630297
33000	1.58592	0.673679	0.670126
34000	1.6822	0.72405	0.712463
35000	1.78059	0.775574	0.747486

2.6 Algoritmo seleccion(Antonio)



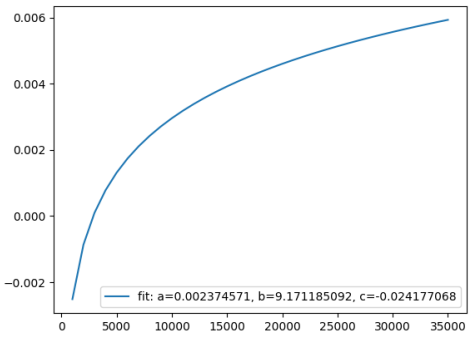


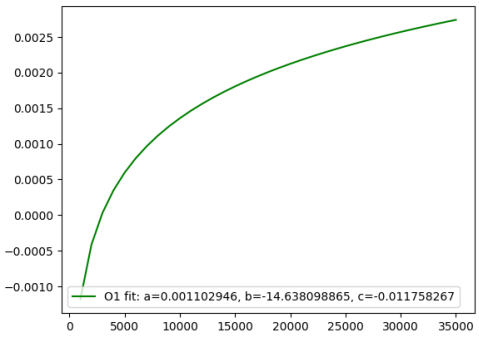


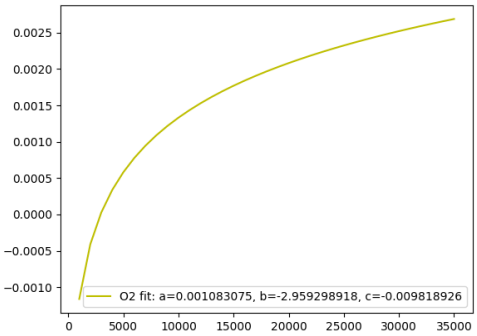


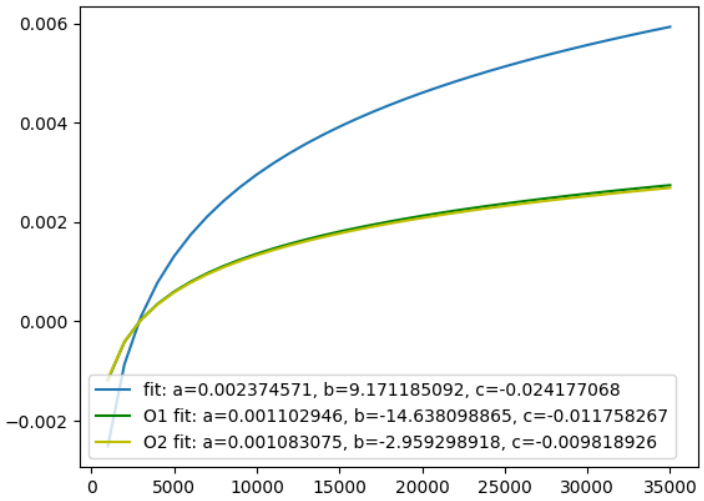
N	O-	O1	O2
1000	0.113808	0.0507492	0.0515542
2000	0.440022	0.194921	0.192643
3000	0.00994115	0.00427214	0.00442329
4000	0.0175305	0.00771941	0.00797904
5000	0.0273612	0.0121969	0.0124481
6000	0.0392684	0.0180733	0.0171079
7000	0.0530349	0.0247544	0.0235386
8000	0.0696024	0.0304388	0.0303601
9000	0.0872593	0.0399728	0.0385527
10000	0.108057	0.0471915	0.0476896
11000	0.13081	0.0563163	0.0637525
12000	0.154417	0.0666793	0.0668392
13000	0.181586	0.0784763	0.0771758
14000	0.21108	0.0922109	0.090675
15000	0.246563	0.105169	0.103795
16000	0.274096	0.12036	0.117137
17000	0.309675	0.135038	0.13268
18000	0.346677	0.152715	0.15029
19000	0.387027	0.166195	0.172956
20000	0.428833	0.183288	0.191728
21000	0.476191	0.204037	0.20652
22000	0.518642	0.220613	0.221813
23000	0.56802	0.243047	0.240955
24000	0.618507	0.262949	0.258005
25000	0.668424	0.284926	0.282445
26000	0.72458	0.315597	0.30513
27000	0.783398	0.333334	0.325479
28000	0.838477	0.367546	0.351541
29000	0.897708	0.390469	0.381924
30000	0.964555	0.416259	0.408663
31000	1.02975	0.43713	0.429654
32000	1.09034	0.47207	0.455585
33000	1.16999	0.493228	0.497087
34000	1.23219	0.528135	0.518997
35000	1.31538	0.573425	0.550583

2.7 Algoritmo mergesort(Elena)



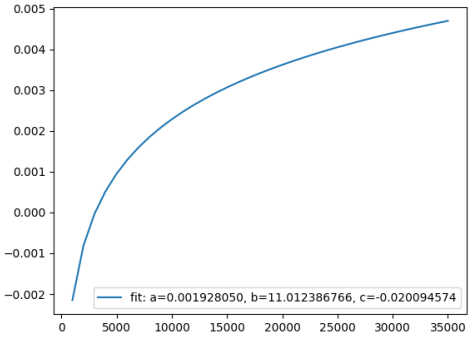


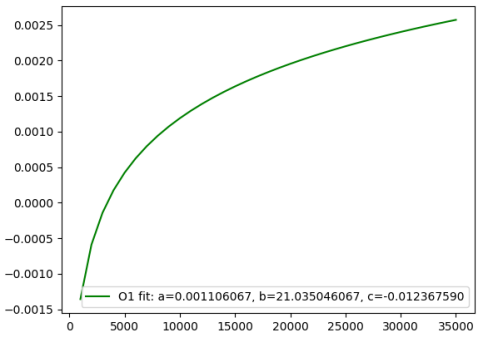


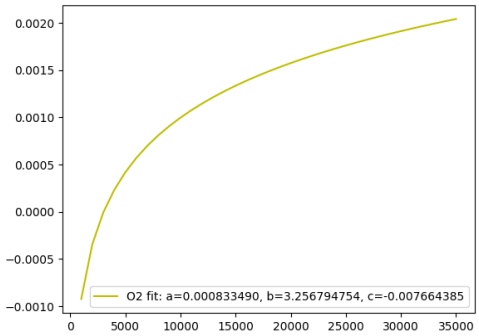


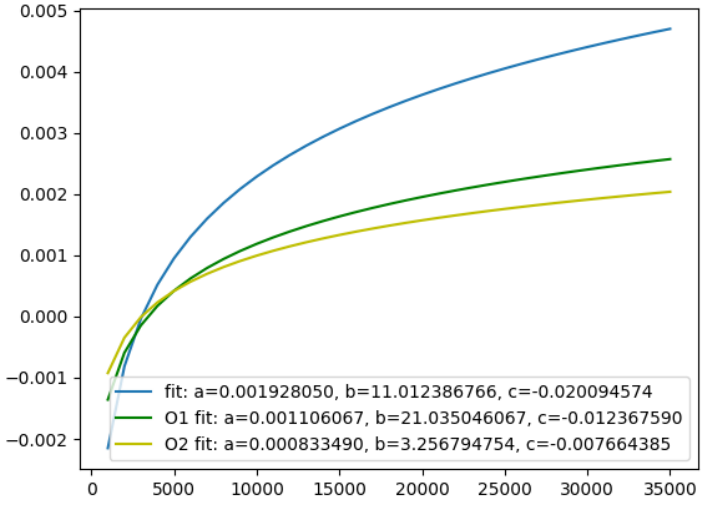
N	O-	O1	O2
1000	0.000142289	5.6138e-05	5.3256e-05
2000	0.000301	0.00014214	0.000129577
3000	0.000565186	0.000238601	0.000234651
4000	0.000656325	0.000291011	0.000289394
5000	0.000912161	0.000394628	0.000390957
6000	0.00120506	0.000520647	0.000514568
7000	0.00117797	0.000533973	0.00051628
8000	0.00141341	0.000645499	0.000621247
9000	0.00168161	0.000752588	0.00073346
10000	0.00196429	0.000865177	0.000849189
11000	0.002299	0.000993	0.001001
12000	0.002597	0.001195	0.001114
13000	0.002363	0.001145	0.001098
14000	0.002612	0.001259	0.001197
15000	0.002866	0.00136	0.001286
16000	0.003107	0.001472	0.001672
17000	0.00342	0.001594	0.001538
18000	0.00364	0.001705	0.001687
19000	0.003965	0.001841	0.001756
20000	0.004258	0.001948	0.001864
21000	0.004619	0.002085	0.002014
22000	0.004883	0.002211	0.002173
23000	0.005268	0.002372	0.002323
24000	0.005586	0.002533	0.002458
25000	0.005926	0.002623	0.00259
26000	0.00509	0.002435	0.002348
27000	0.005307	0.002543	0.002464
28000	0.005557	0.002629	0.002535
29000	0.005838	0.002783	0.002755
30000	0.006091	0.002903	0.002788
31000	0.006358	0.00301	0.003131
32000	0.006704	0.003126	0.003015
33000	0.006881	0.003241	0.003203
34000	0.007221	0.003394	0.003348
35000	0.008392	0.003499	0.003422

2.8 Algoritmo mergesort(Antonio)



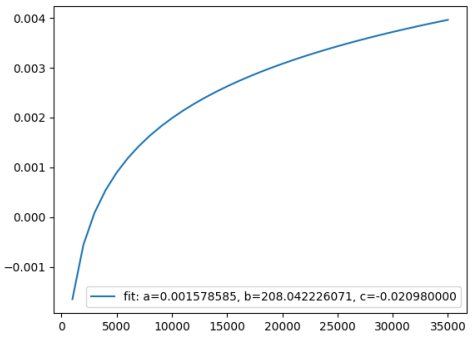


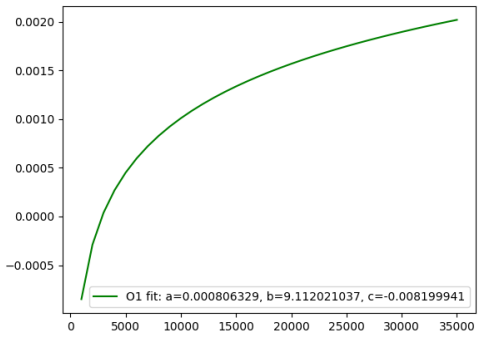


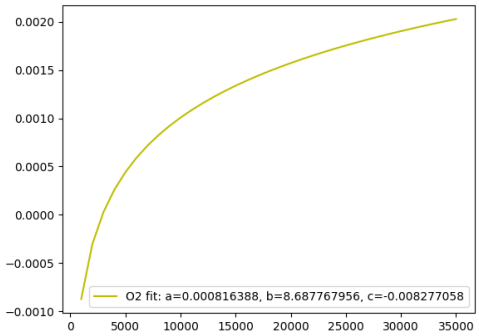


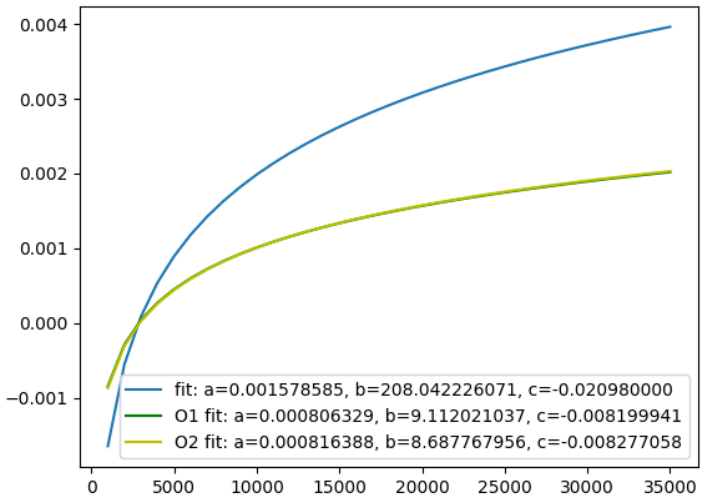
N	O-	O1	O2
1000	9.7562e-05	4.3121e-05	5.0206e-05
2000	0.000219378	9.4759e-05	9.4657e-05
3000	0.00040376	0.000172173	0.000173778
4000	0.000478964	0.000214573	0.00021195
5000	0.000672422	0.000290972	0.000283667
6000	0.000882371	0.000376491	0.000377717
7000	0.000861508	0.000395036	0.000384458
8000	0.00102842	0.000473047	0.000454253
9000	0.00124205	0.000547581	0.000538787
10000	0.00141724	0.00063134	0.000612704
11000	0.001626	0.000733	0.000785
12000	0.001873	0.00083	0.000817
13000	0.001741	0.000866	0.00077
14000	0.001882	0.000877	0.000853
15000	0.002088	0.001076	0.000928
16000	0.002283	0.001159	0.001068
17000	0.002514	0.001251	0.001073
18000	0.002668	0.001347	0.001173
19000	0.002942	0.001447	0.001405
20000	0.003065	0.001693	0.001409
21000	0.003437	0.001797	0.001555
22000	0.003711	0.001916	0.001606
23000	0.004279	0.002046	0.001739
24000	0.004591	0.002171	0.001814
25000	0.004832	0.002307	0.001987
26000	0.00456	0.002109	0.00174
27000	0.004326	0.00221	0.001877
28000	0.004546	0.002307	0.001978
29000	0.004755	0.002664	0.002051
30000	0.004972	0.002788	0.002072
31000	0.005207	0.003251	0.00222
32000	0.00544	0.003364	0.002253
33000	0.005668	0.003491	0.002364
34000	0.005887	0.003664	0.002718
35000	0.006174	0.003781	0.003063

2.9 Algoritmo quicksort(Elena)



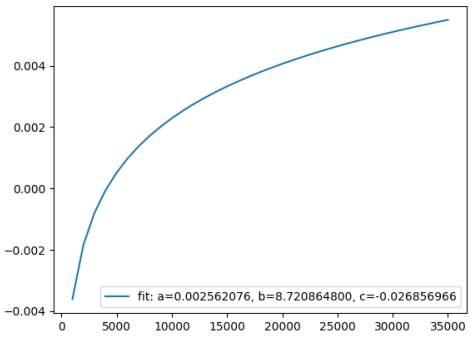


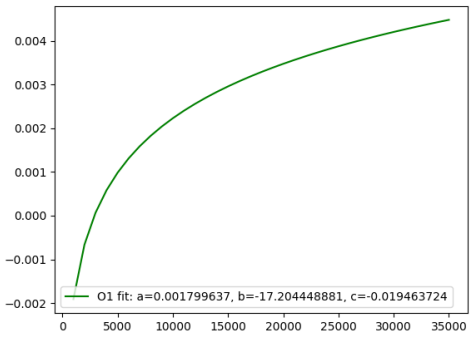


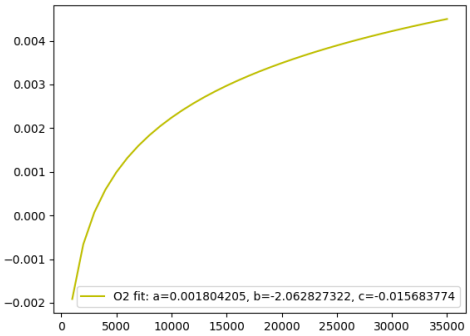


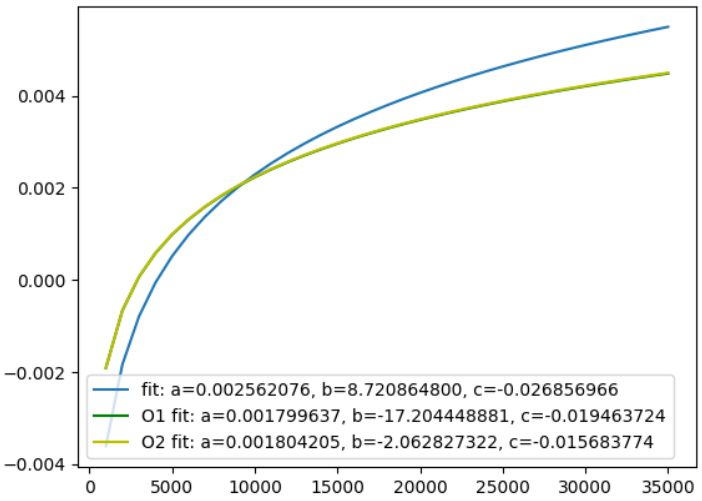
N	O-	O1	O2
1000	9.9817e-05	5.0442e-05	5.0401e-05
2000	0.000227257	0.00013266	0.000118967
3000	0.000350887	0.00017088	0.000175812
4000	0.000482895	0.00023387	0.00023271
5000	0.000646748	0.000302824	0.000304232
6000	0.000749643	0.000374281	0.000369755
7000	0.000908375	0.00044219	0.000437503
8000	0.00105866	0.000523946	0.000509061
9000	0.00117842	0.000585779	0.000581393
10000	0.00132631	0.000658304	0.000652338
11000	0.00145902	0.000734204	0.000726409
12000	0.00166884	0.000809363	0.00080204
13000	0.00169378	0.000844834	0.000853763
14000	0.00186206	0.00106827	0.000932916
15000	0.0019978	0.00101793	0.00100616
16000	0.00219878	0.0011337	0.00108273
17000	0.00232444	0.0012006	0.00119566
18000	0.00244827	0.0012454	0.00124138
19000	0.00258529	0.00131713	0.00130758
20000	0.00272364	0.00139903	0.00138977
21000	0.00299948	0.00146522	0.00147639
22000	0.00303522	0.00155317	0.0015648
23000	0.00319398	0.00158218	0.0016267
24000	0.00330492	0.00172188	0.00175563
25000	0.00351289	0.00179055	0.00186384
26000	0.00368799	0.0018612	0.0018534
27000	0.00382549	0.00194992	0.00200636
28000	0.00397248	0.00200078	0.00203078
29000	0.00418752	0.00210934	0.00210808
30000	0.00429072	0.002219	0.00222609
31000	0.00438885	0.00224658	0.0022822
32000	0.00453243	0.00237256	0.00237227
33000	0.0048428	0.0024454	0.00241964
34000	0.00491766	0.00248939	0.00255495
35000	0.00503133	0.00254678	0.00256313

2.10 Algoritmo quicksort(Antonio)



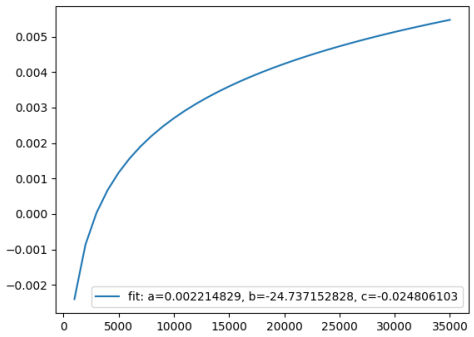


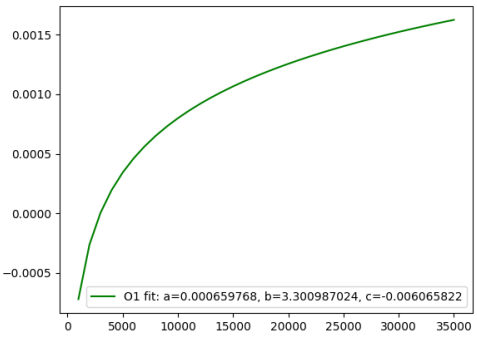


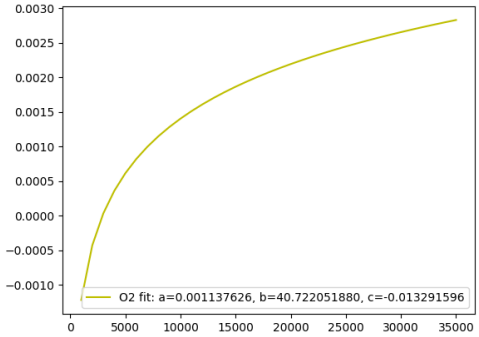


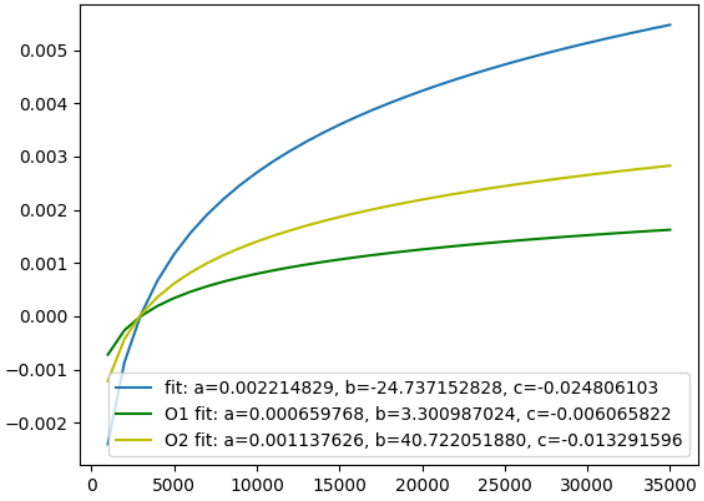
N	O-	O1	O2
1000	7.0852e-05	8.7531e-05	0.000113581
2000	0.00021565	0.000199469	0.000243369
3000	0.000242762	0.000305245	0.00038559
4000	0.000328305	0.000526492	0.000535741
5000	0.000426144	0.000659839	0.0006694
6000	0.000523048	0.000837415	0.000817214
7000	0.000616916	0.000965552	0.0009916
8000	0.000780271	0.0012482	0.00115136
9000	0.000803905	0.00129618	0.00130735
10000	0.000947855	0.00153631	0.00148174
11000	0.00108127	0.00172348	0.00162444
12000	0.00112693	0.00186451	0.00179318
13000	0.00131472	0.00195875	0.0019322
14000	0.00144836	0.00210791	0.00213215
15000	0.00157483	0.00233087	0.00226752
16000	0.00166702	0.00239326	0.00246252
17000	0.00170516	0.00257679	0.00264973
18000	0.00203383	0.00276208	0.00276352
19000	0.00213518	0.00305852	0.00295137
20000	0.00223099	0.00304744	0.0031493
21000	0.00235415	0.00329863	0.00330143
22000	0.00245928	0.00355411	0.00351015
23000	0.00261111	0.00360215	0.00363866
24000	0.0032673	0.00373966	0.00381169
25000	0.00386305	0.00390432	0.00396249
26000	0.00447933	0.00414109	0.00413046
27000	0.0056785	0.00425661	0.00433613
28000	0.00589681	0.00450222	0.00451823
29000	0.00740397	0.00464354	0.00469539
30000	0.00773386	0.00482171	0.00483742
31000	0.00793326	0.0049767	0.00503622
32000	0.00834004	0.00530646	0.0052259
33000	0.00838877	0.00536436	0.0054252
34000	0.00881353	0.00555759	0.00557981
35000	0.00921185	0.00573696	0.00579729

2.11 Algoritmo heapsort(Elena)



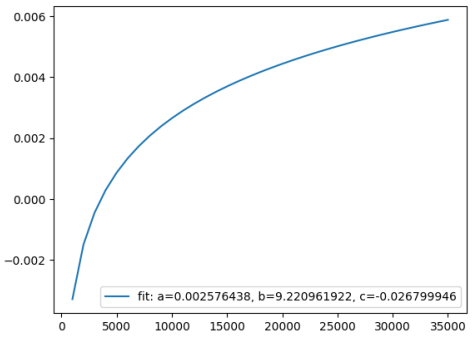


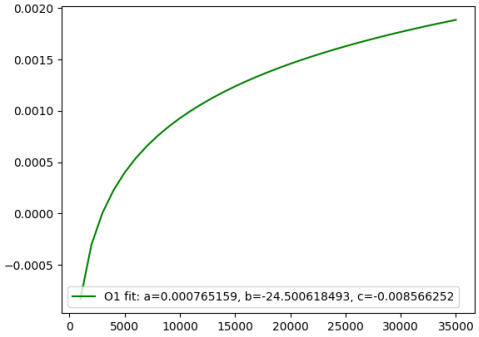


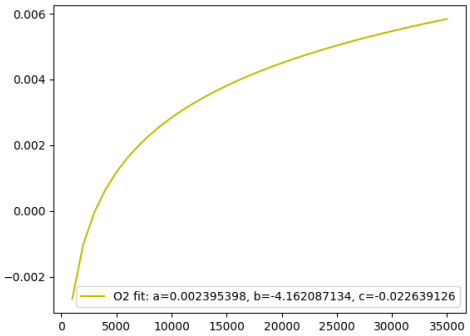


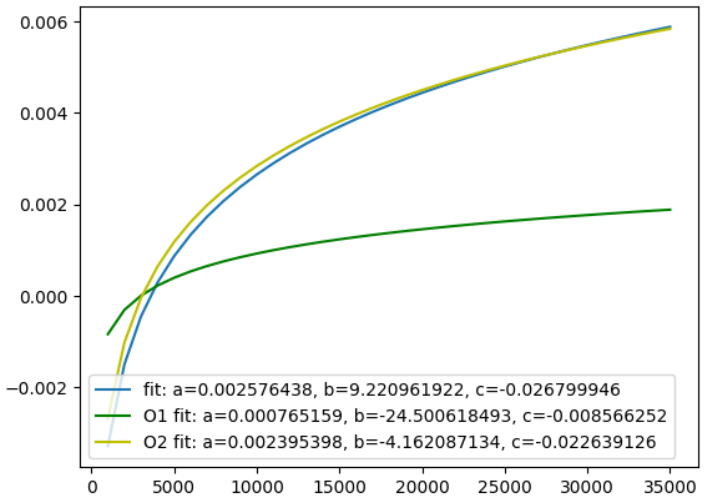
N	O-	O1	O2
1000	0.000136123	4.4751e-05	6.9021e-05
2000	0.000306375	8.0885e-05	0.00015095
3000	0.000452397	0.000128777	0.00023365
4000	0.00061982	0.000181898	0.000326824
5000	0.00080802	0.000229764	0.000416221
6000	0.000970036	0.000276247	0.000507487
7000	0.00114352	0.000337848	0.000636754
8000	0.00134811	0.000390992	0.000704134
9000	0.00154743	0.000448341	0.000804659
10000	0.00172895	0.000503122	0.000924079
11000	0.00192196	0.000550752	0.00102253
12000	0.00212514	0.000622282	0.00110433
13000	0.00233904	0.000685348	0.00120639
14000	0.00249019	0.000742696	0.00131808
15000	0.00277069	0.000802099	0.0014206
16000	0.00289043	0.00088007	0.00152127
17000	0.00318939	0.000929104	0.00164515
18000	0.00330463	0.00097859	0.00175197
19000	0.00350129	0.00103657	0.00184024
20000	0.00370489	0.00111178	0.00196114
21000	0.0038996	0.00119041	0.0020682
22000	0.00416611	0.00126033	0.00217102
23000	0.00432359	0.00129826	0.0023439
24000	0.00454306	0.00136747	0.00237447
25000	0.00483153	0.00143063	0.00250477
26000	0.00497735	0.0014979	0.0026143
27000	0.0052003	0.00156599	0.00272145
28000	0.00545135	0.00163357	0.00285597
29000	0.00554687	0.00169604	0.00294463
30000	0.00572157	0.00177656	0.00306901
31000	0.00594329	0.00186258	0.0031558
32000	0.00790022	0.00188938	0.00330005
33000	0.00660048	0.00199688	0.00340508
34000	0.00675016	0.00202066	0.00352505
35000	0.00688566	0.00212635	0.00362913

2.12 Algoritmo heapsort(Antonio)



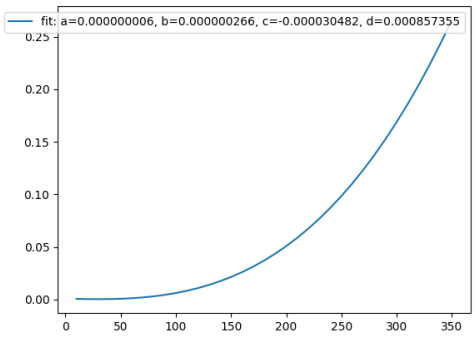


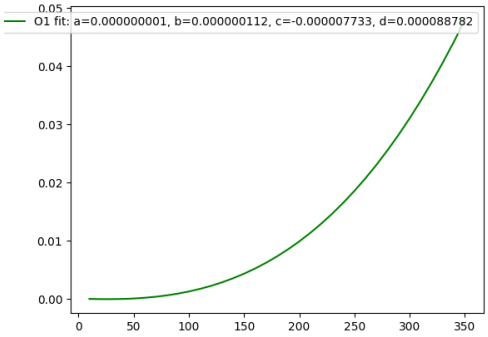


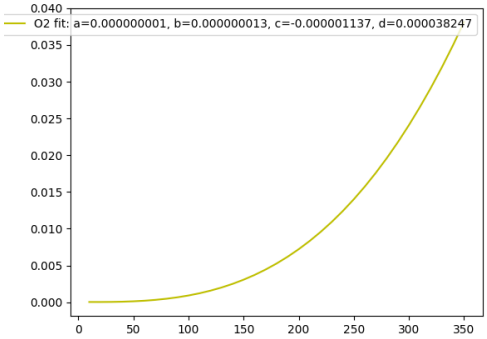


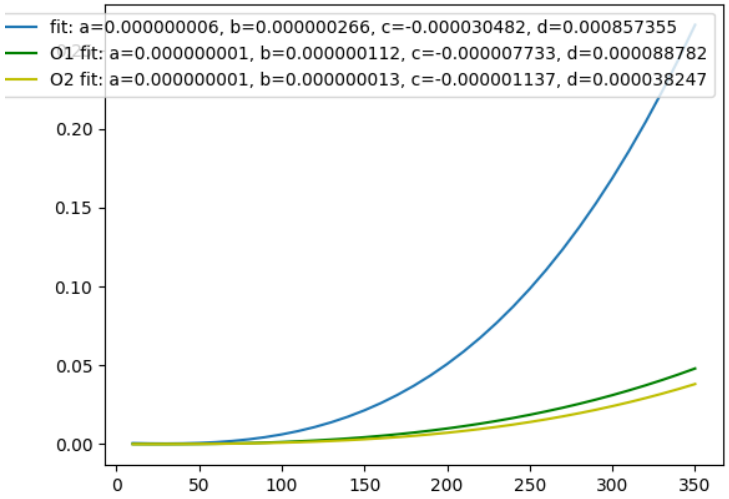
N	O-	O1	O2
1000	9.5007e-05	4.4977e-05	8.2139e-05
2000	0.000203424	9.8924e-05	0.000176791
3000	0.000320315	0.000152058	0.000372852
4000	0.000440649	0.000206437	0.000627335
5000	0.00056368	0.000273043	0.000805891
6000	0.00075261	0.000333338	0.000983016
7000	0.000893066	0.000396277	0.00118707
8000	0.00100186	0.000449124	0.00137798
9000	0.00114237	0.000538101	0.00156823
10000	0.00128021	0.00058067	0.00173922
11000	0.00155514	0.000660495	0.00194511
12000	0.00157578	0.000718106	0.00216217
13000	0.00172319	0.000787455	0.00232897
14000	0.00202401	0.000854124	0.00252672
15000	0.00201206	0.000934659	0.00272894
16000	0.00254391	0.00103014	0.00294308
17000	0.00236532	0.00107292	0.00354848
18000	0.00253443	0.00114383	0.00380026
19000	0.00266495	0.00122477	0.00389984
20000	0.00317532	0.00127798	0.00403938
21000	0.00404545	0.00134584	0.00425808
22000	0.00428085	0.00145094	0.00450159
23000	0.00490219	0.00153258	0.00473496
24000	0.00531788	0.00157331	0.00493016
25000	0.00558246	0.00166155	0.00517457
26000	0.00583703	0.00176471	0.00539419
27000	0.0060918	0.00180794	0.00563099
28000	0.00632167	0.00193519	0.00586776
29000	0.00657126	0.00196711	0.00610156
30000	0.00682382	0.00205163	0.00636649
31000	0.00706682	0.00216352	0.00659657
32000	0.00734541	0.00219583	0.00683585
33000	0.00759875	0.0023261	0.00701873
34000	0.00785531	0.00233835	0.00726229
35000	0.00810865	0.0024446	0.00751103

2.13 Algoritmo floyd(Elena)



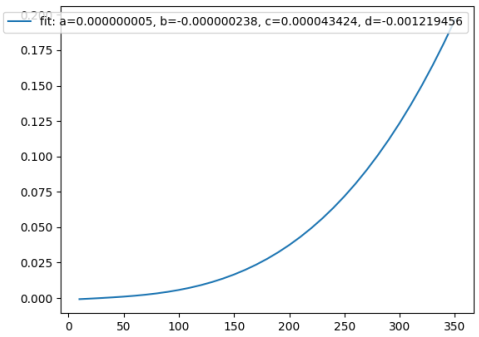


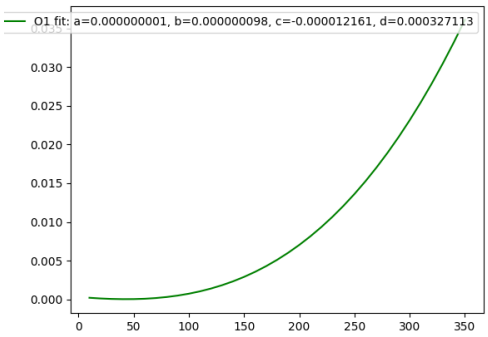


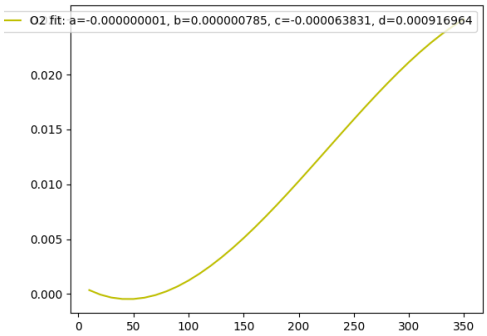


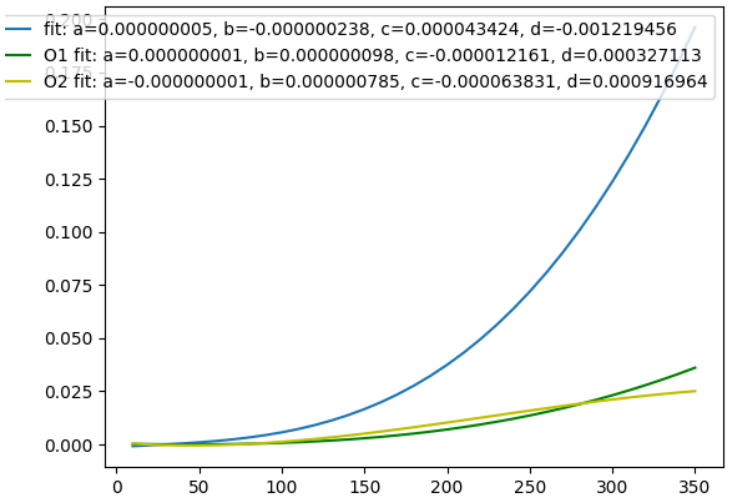
N	O-	O1	O2
10	1e-05	2e-06	2e-06
20	6.7e-05	1e-05	9e-06
30	0.000203	3.2e-05	2.6e-05
40	0.000448	8.5e-05	6.4e-05
50	0.00085	0.00018	0.000111
60	0.001523	0.000245	0.000211
70	0.002242	0.000388	0.000302
80	0.003433	0.000628	0.000468
90	0.00479	0.000887	0.000673
100	0.006563	0.001241	0.000935
110	0.008654	0.001584	0.001257
120	0.011204	0.002048	0.001619
130	0.014089	0.00262	0.00206
140	0.017503	0.003243	0.002528
150	0.021579	0.005549	0.003132
160	0.025767	0.004884	0.003714
170	0.031052	0.005792	0.004462
180	0.036781	0.006995	0.005309
190	0.043117	0.008024	0.006207
200	0.050271	0.009325	0.007302
210	0.057796	0.010893	0.00833
220	0.066049	0.013512	0.009487
230	0.075983	0.019707	0.010731
240	0.08597	0.016271	0.012288
250	0.097257	0.01794	0.013794
260	0.11489	0.020119	0.015456
270	0.12577	0.022316	0.017151
280	0.135327	0.025045	0.019537
290	0.151109	0.027848	0.022317
300	0.173293	0.030866	0.024179
310	0.184009	0.033768	0.026927
320	0.209154	0.036916	0.029534
330	0.222543	0.041274	0.032556
340	0.243091	0.044321	0.035005
350	0.264411	0.048341	0.037279

2.14 Algoritmo floyd(Antonio)



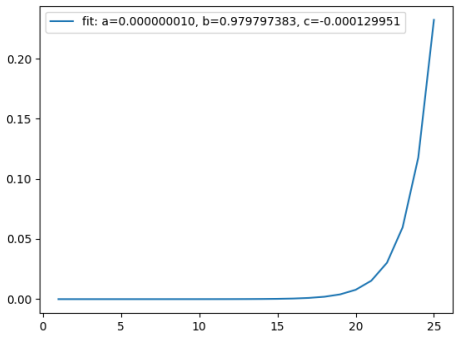


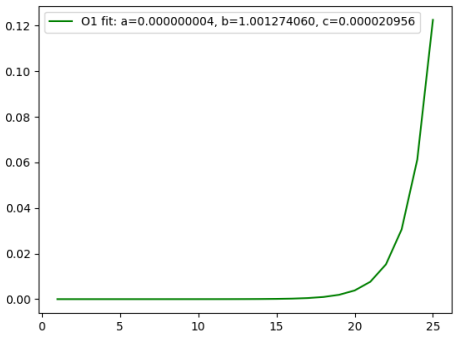


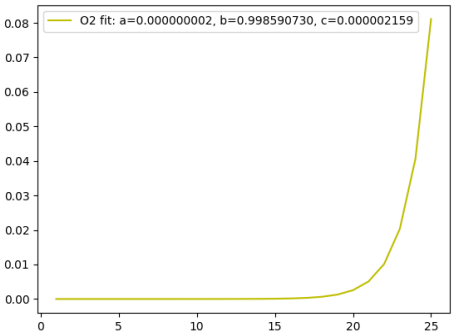


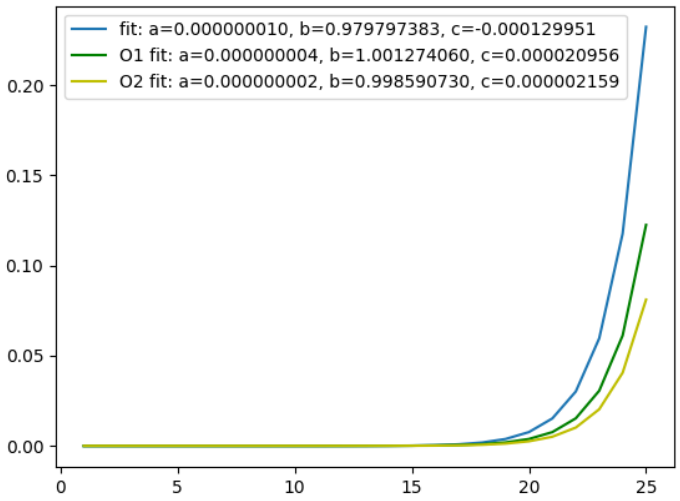
N	O-	O1	O2
10	7e-06	2e-06	1e-06
20	4.9e-05	8e-06	6e-06
30	0.000138	2.2e-05	1.8e-05
40	0.000318	6e-05	4.5e-05
50	0.00061	0.000102	7.8e-05
60	0.001095	0.000174	0.000135
70	0.001595	0.000275	0.000214
80	0.002462	0.000505	0.000333
90	0.003471	0.000626	0.000487
100	0.00476	0.000858	0.000677
110	0.006267	0.00122	0.000967
120	0.00939	0.0016	0.001387
130	0.011997	0.001919	0.002126
140	0.014861	0.002401	0.002966
150	0.018396	0.002879	0.003878
160	0.022149	0.003477	0.005903
170	0.024497	0.004126	0.007088
180	0.027264	0.00495	0.008384
190	0.03225	0.005785	0.009857
200	0.036798	0.006849	0.011657
210	0.042716	0.007822	0.013427
220	0.048946	0.009034	0.015228
230	0.056021	0.010254	0.017314
240	0.0633	0.011721	0.017356
250	0.071353	0.013761	0.013016
260	0.080501	0.015499	0.014667
270	0.090349	0.017336	0.016381
280	0.100123	0.019371	0.018226
290	0.110743	0.02118	0.020386
300	0.122328	0.023664	0.021195
310	0.136484	0.026285	0.019489
320	0.149135	0.02747	0.021226
330	0.171821	0.030105	0.023383
340	0.17891	0.033021	0.025405
350	0.194148	0.035809	0.027583

2.15 Algoritmo hanoi(Elena)



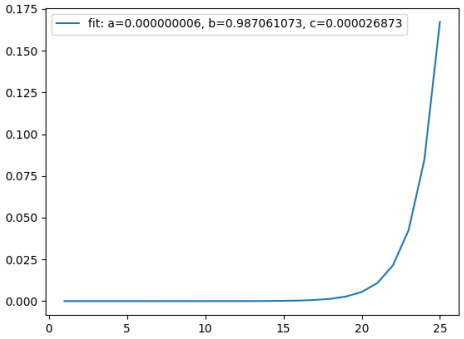


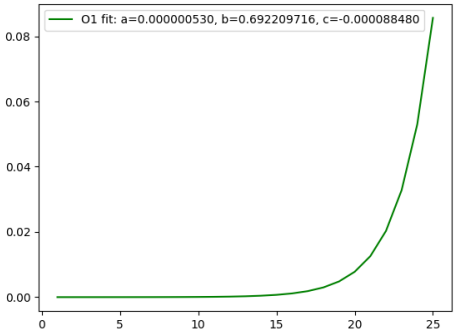


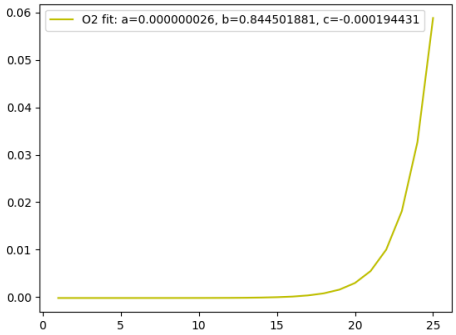


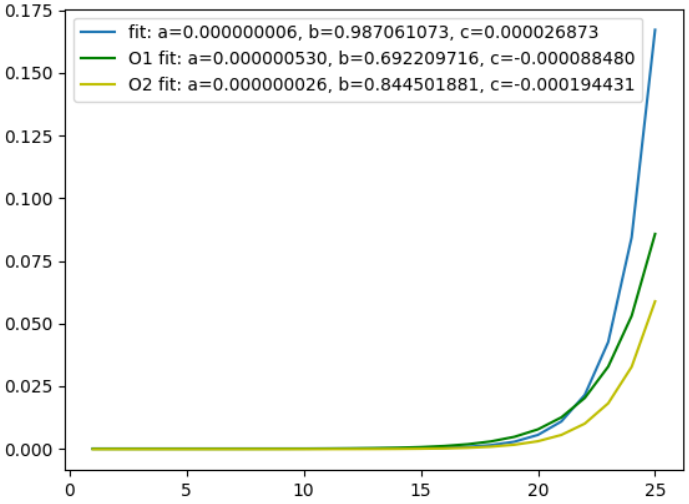
N	O-	O1	O2
1	2e-07	2.24e-07	1.64e-07
2	1.67e-07	1.79e-07	1.63e-07
3	3.33e-07	2.87e-07	2.65e-07
4	5.75e-07	3.4e-07	3.68e-07
5	6.74e-07	4.52e-07	4.45e-07
6	9.91e-07	6.66e-07	5.85e-07
7	1.647e-06	8.31e-07	8.24e-07
8	2.758e-06	1.512e-06	1.117e-06
9	4.436e-06	3.343e-06	1.8e-06
10	8.186e-06	5.609e-06	3.002e-06
11	1.5659e-05	8.349e-06	5.657e-06
12	3.2058e-05	1.5869e-05	1.0862e-05
13	6.0502e-05	3.0724e-05	2.0561e-05
14	0.000119143	6.344e-05	4.19e-05
15	0.000237536	0.000125609	8.0066e-05
16	0.000471937	0.000258071	0.000159233
17	0.000943677	0.000481852	0.000328443
18	0.00184691	0.000994402	0.000635549
19	0.00365114	0.00192413	0.00134496
20	0.00729903	0.00389515	0.00257432
21	0.0146018	0.00784956	0.00513506
22	0.0292939	0.0155307	0.0101176
23	0.0567671	0.0305533	0.0200906
24	0.12188	0.0609997	0.0408326
25	0.231254	0.122658	0.0810675

2.16 Algoritmo hanoi(Antonio)



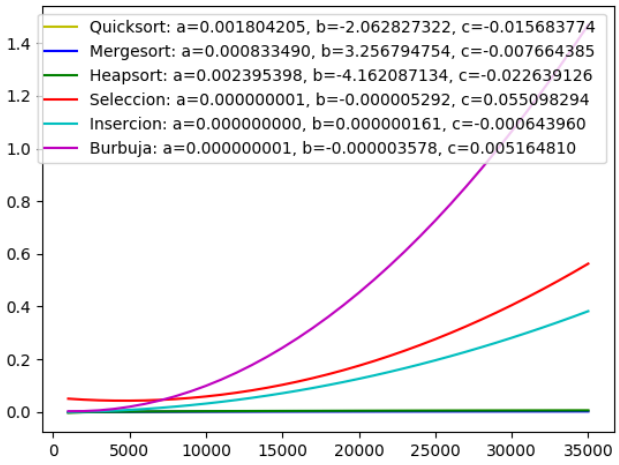






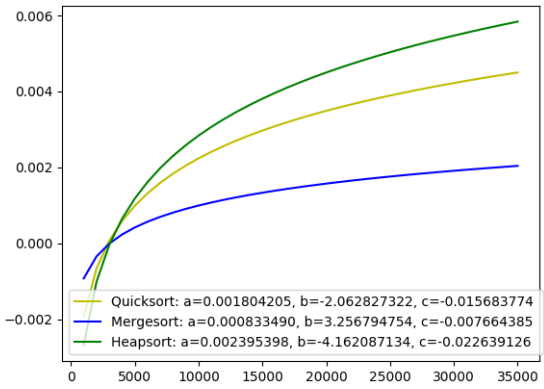
N	O-	O1	O2
1	1.2e-07	1.78e-07	1.36e-07
2	1.35e-07	1.9e-07	1.57e-07
3	2.89e-07	1.91e-07	2.03e-07
4	3.43e-07	3.49e-07	2.94e-07
5	5.8e-07	3.61e-07	3.9e-07
6	7.92e-07	5.13e-07	4.65e-07
7	1.218e-06	7.12e-07	6.87e-07
8	1.82e-06	1.061e-06	9.75e-07
9	3.074e-06	1.929e-06	1.429e-06
10	6.153e-06	3.231e-06	2.336e-06
11	1.1591e-05	6.328e-06	4.272e-06
12	2.2193e-05	1.3284e-05	7.746e-06
13	4.3379e-05	2.8738e-05	1.4845e-05
14	8.624e-05	6.3205e-05	2.92e-05
15	0.000170917	0.000125993	5.9962e-05
16	0.00034173	0.000283643	0.000115692
17	0.000681671	0.000565734	0.000237385
18	0.00136209	0.0015293	0.000459186
19	0.00285817	0.00373938	0.000997598
20	0.00556612	0.00751584	0.00200475
21	0.0110711	0.0149939	0.0041239
22	0.0223174	0.0296655	0.00922785
23	0.0420834	0.0304637	0.0183847
24	0.0841359	0.0445143	0.0347469
25	0.167375	0.0894432	0.057937

3 Algoritmos de búsqueda



En la gráfica superior hemos representado la eficiencia empírica de todos los algoritmos de ordenación estudiados. Observando el gráfico, vemos lo eficientes que son quicksort, heapsort y mergesort con respecto a los otros.

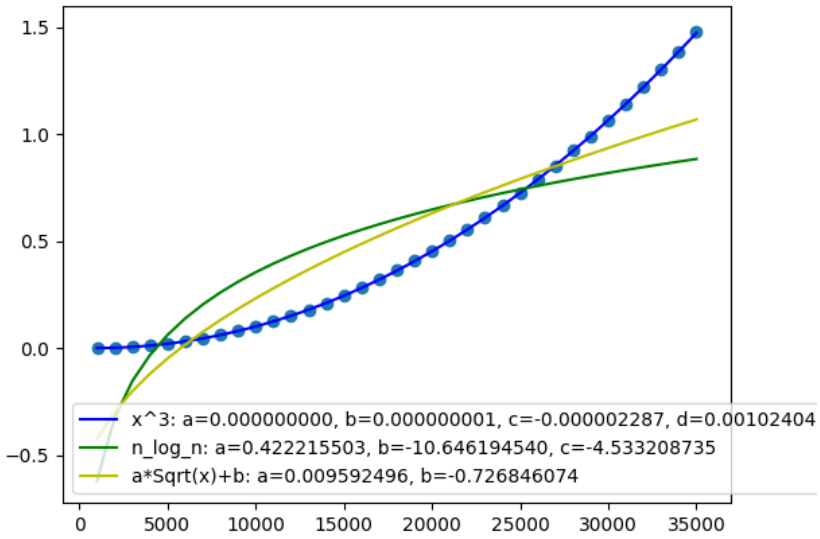
Vemos también, que, para tamaños entre 5000 y 6000, el algoritmo de selección es más lento que el de la burbuja, pero para tamaños superiores, el algoritmo de selección es notablemente más rápido (esto se debe a que el valor del UMBRAL se encuentra entre esos valores). Además observamos que el algoritmo de inserción siempre es más rápido que el de selección y el de la burbuja.



Debido a que en la anterior imagen no podemos comparar adecuadamente los algoritmos quicksort, mergesort y heapsort, hemos generado otra gráfica con ellos solos.

Observamos ahora que para tamaños menores a 2000 el algoritmo mergesort es más lento que los demás, pero a partir de ese tamaño se hace evidente que es el mejor algoritmo de ordenación de todos los vistos aquí.

4 Variación del ajuste según la función



En la gráfica de arriba hemos realizado el ajuste correspondiente al algoritmo de la burbuja optimizado con la opción $O2$, pero esta vez ajustándolo a otras funciones. Como vemos, la calidad del ajuste es pésima, la función $f(x) = x^3$ sale tan alejada de los datos que ni siquiera se muestra en el gráfico, y la función logarítimica no se acerca mucho tampoco. Sin embargo la función x^2 se ciñe perfectamente a los datos tomados. Aquí se ve la importancia de realizar un buen estudio teórico de la eficiencia antes de realizar el estudio empírico.

5 Variación de la eficiencia según el ordenador donde se ejecuta

A continuación vemos unas gráficas donde podemos ver que la eficiencia teórica de los algoritmos se mantiene sin importar el ordenador en el que sea ejecutado. Lo único que varía son los coeficientes, que serán mejores según las prestaciones del ordenador, en este caso podemos ver que el ordenador de Antonio ejecuta mejor la mayoría de los algoritmos al tener unas mejores prestaciones que el de Elena, aunque esto no pasa en todos los casos.

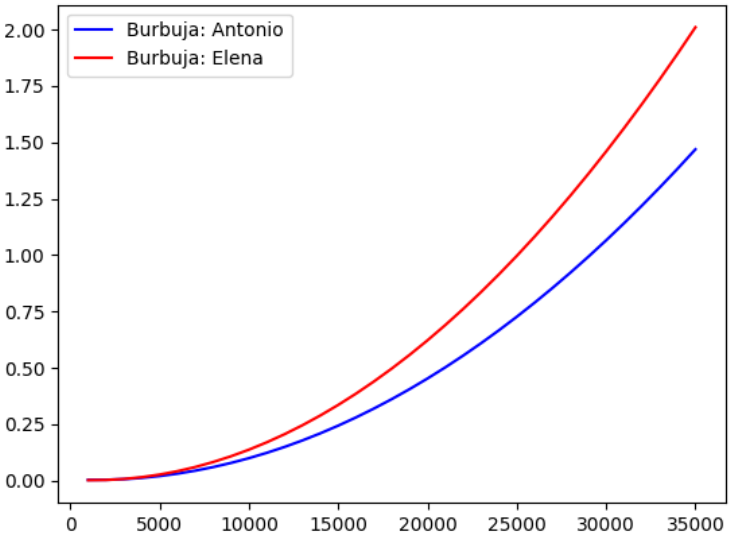


Figure 1: Burbuja

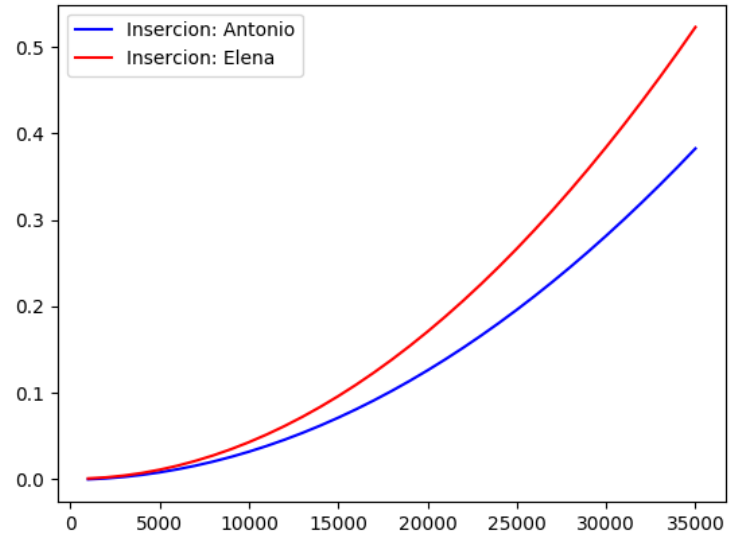


Figure 2: Inserción

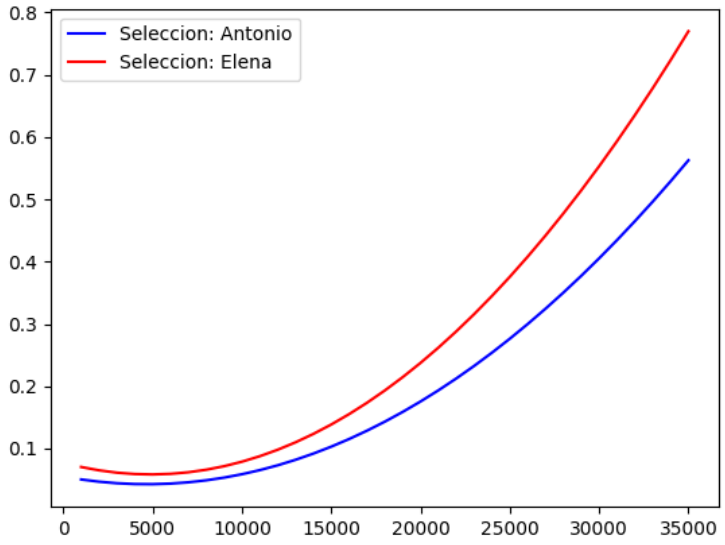


Figure 3: Selección

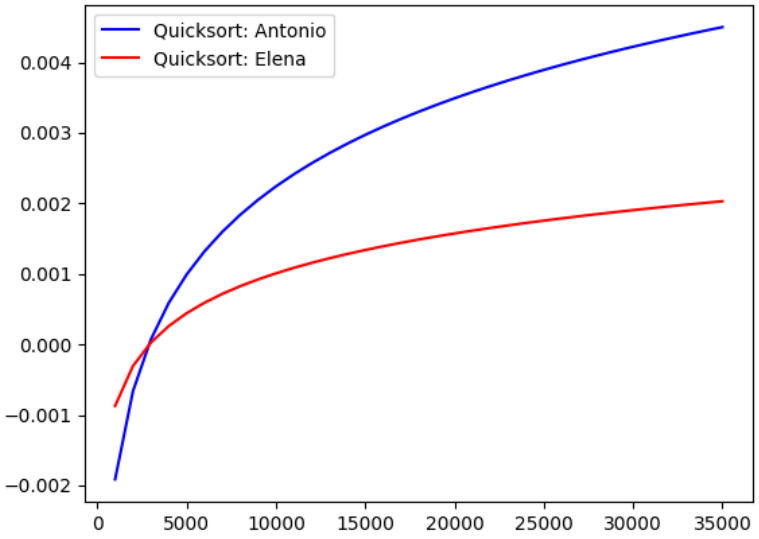


Figure 4: Quicksort

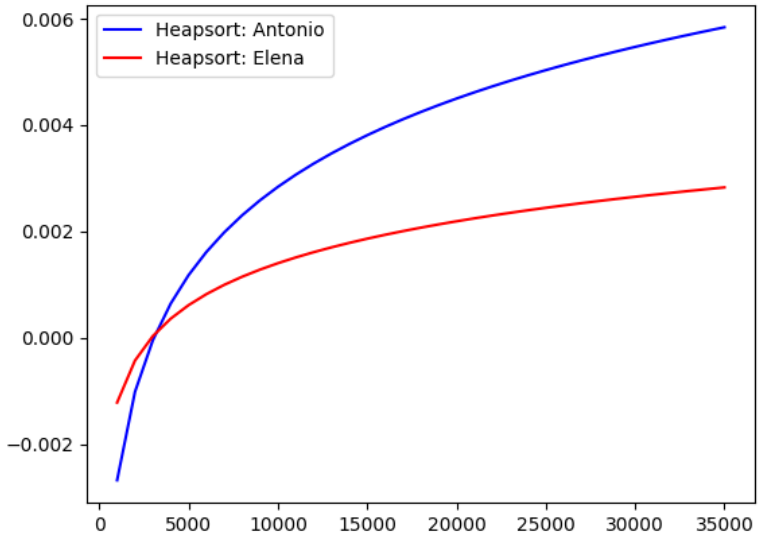


Figure 5: Heapsort

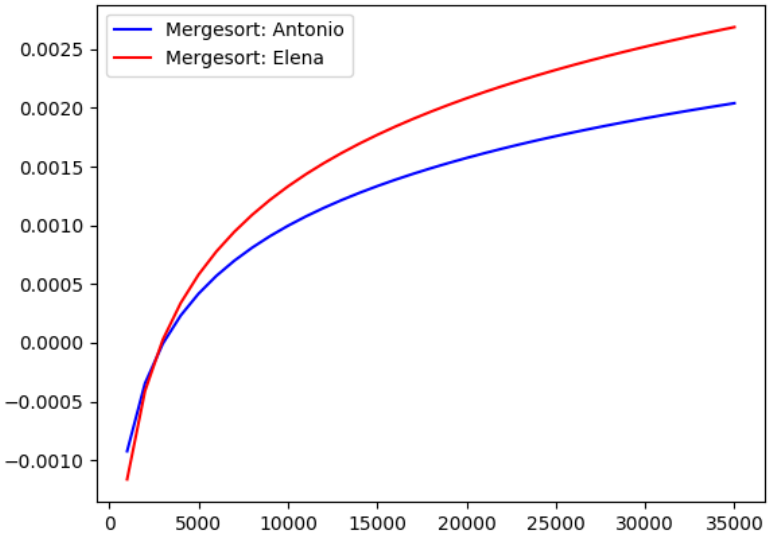


Figure 6: Mergesort

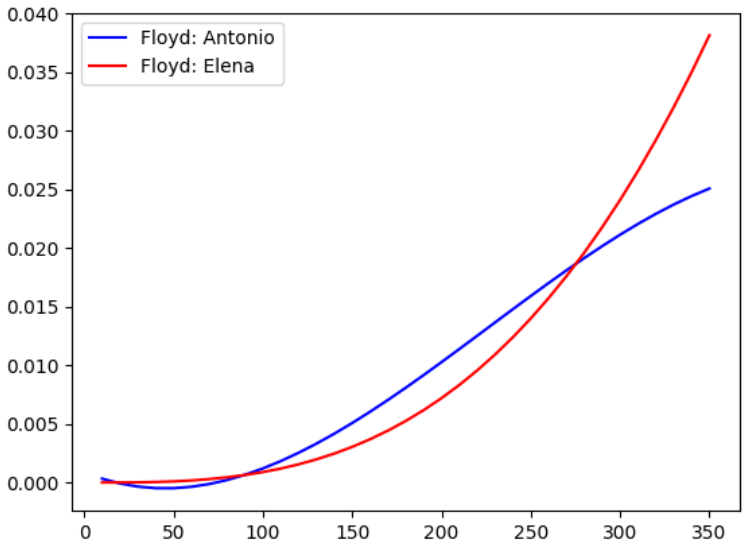


Figure 7: Floyd

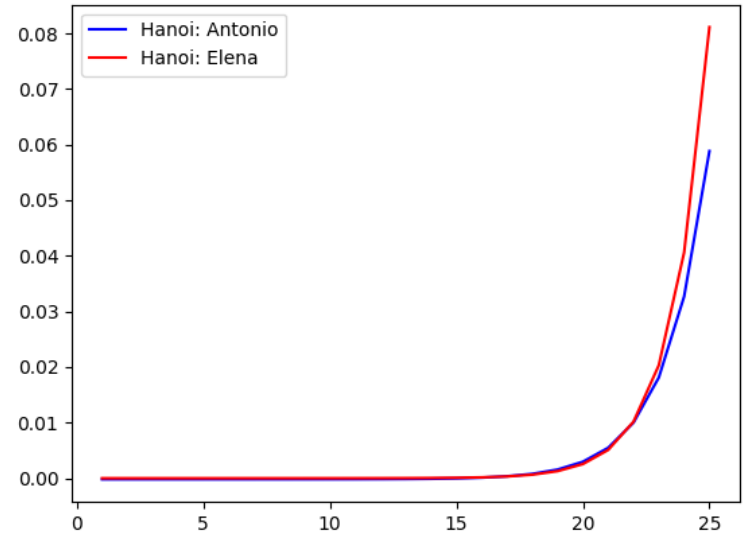


Figure 8: Hanoi