

PREGUNTAS SECCIÓN 2

Sesión práctica 4

Jonathan Arias Busto




UO283586

71780982-Y

Escuela de Ingeniería Informática - EII

INFORMACIÓN SOBRE ORDENADOR USADO PARA MEDIR TIEMPOS

Procesador	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
RAM instalada	16,0 GB (15,8 GB usable)

- ▼  Adaptadores de pantalla
 -  Intel(R) UHD Graphics
 -  NVIDIA GeForce GTX 1650

Se trata de un portátil con el modo de energía en “High performance”.

Cambiar la configuración del plan: High performance

TABLAS TIEMPOS DIFERENTES ALGORITMOS

Lo primero de todo es saber que algoritmos son los que estamos estudiando en esta práctica. Los diferentes algoritmos son: inserción, selección, burbuja y mediana 3.

Para todos estos algoritmos vamos a usar 3 configuraciones distintas (opciones), siendo estas:

- Ordenado: tenemos un vector con los elementos ya ordenados.
- Inverso: vector ordenado de forma inversa.
- Aleatorio: vector con elementos añadidos de forma aleatoria.

ALGORITMO DE INSERCIÓN

N	t ordenado	t inverso	t aleatorio
1000	42	23	26
2000	47	28	30
4000	110	72	57
8000	175	173	116
16000	355	296	260
32000	690	766	583
64000	1386	2238	1482
128000	3207	7349	4724
256000	5977	26199	15173
512000	11611	101385	55390
1024000	24329	374266	214785
nº veces: 100000 nº veces: 1000 nº veces: 1000			

Esta es la tabla correspondiente al método de inserción, como se puede ver la cantidad de veces que tenemos que ejecutar el algoritmo depende del tipo de opción que seleccionemos. En el caso de este algoritmo necesitaremos ejecutar más veces el caso ordenado puesto que es mucho más rápido que para los otros casos. Podemos ver como en los otros casos el algoritmo es mucho más lento. La tabla de estos tiempos en mili segundos es la siguiente:

N	t ordenado	t inverso	t aleatorio
1000	0,00042	0,023	0,026
2000	0,00047	0,028	0,03
4000	0,0011	0,072	0,057
8000	0,00175	0,173	0,116
16000	0,00355	0,296	0,26
32000	0,0069	0,766	0,583
64000	0,01386	2,238	1,482
128000	0,03207	7,349	4,724
256000	0,05977	26,199	15,173
512000	0,11611	101,385	55,39
1024000	0,24329	374,266	214,785

De esta forma todas las medidas están en la misma medida de tiempo (ms). Podemos ver como en el caso de la configuración de vector ordenado el algoritmo es muy rápido en comparación a cuando el vector está desordenado.

Siguiendo la implementación del algoritmo, podemos ver como cuando el vector está ordenado el bucle while de dentro del for nunca va a ser verdadera, por lo tanto se ahorra muchas comprobaciones haciendo que el algoritmo sea mucho más rápido. Esto ya no va a ocurrir en el caso de que el vector este desordenado, como se puede ver en el caso inverso y aleatorio. La gráfica correspondiente a este algoritmo es la siguiente:



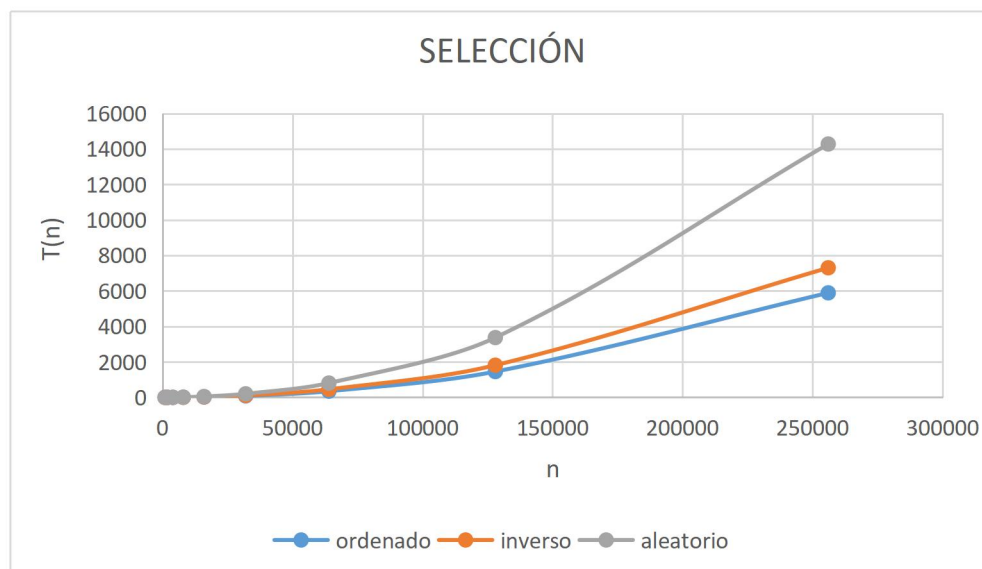
ALGORITMO DE SELECCIÓN

N	t ordenado	t inverso	t aleatorio
1000	20	20	32
2000	40	46	93
4000	150	189	323
8000	546	696	1257
16000	2164	2747	5033
32000	8724	11119	20194
64000	34962	45707	80904
128000	145508	181498	337181
256000	589292	730537	1428142
	nº veces: 100	nº veces: 100	nº veces: 100

Esta es la tabla correspondiente al algoritmo de selección. En este caso podemos ver como en todos los casos en número de repeticiones del algoritmo para la toma de tiempos es de 100, por lo que en este caso todos los casos son parecidos o próximos en cuanto a tiempos de ejecución se trata. Analizando la tabla podemos ver como el caso más difícil de ejecutar para el ordenador fue el caso aleatorio. La tabla pasada a mili segundo es la siguiente:

N	t ordenado	t inverso	t aleatorio
1000	0,2	0,2	0,32
2000	0,4	0,46	0,93
4000	1,5	1,89	3,23
8000	5,46	6,96	12,57
16000	21,64	27,47	50,33
32000	87,24	111,19	201,94
64000	349,62	457,07	809,04
128000	1455,08	1814,98	3371,81
256000	5892,92	7305,37	14281,42

Dada esta tabla en la que todos los tiempos siguen la misma unidad de tiempo (ms) podemos hacer la gráfica correspondiente al algoritmo:



Analizando el código de este algoritmo podemos ver como hay dos bucles for, estos se van a ejecutar independientemente de si la lista esta ordenada o no, de ahí que los tiempos estén tan parejos para las distintas configuraciones de la lista. Por tanto la complejidad de este algoritmo para el caso bueno, medio y malo es de una complejidad cuadrática.

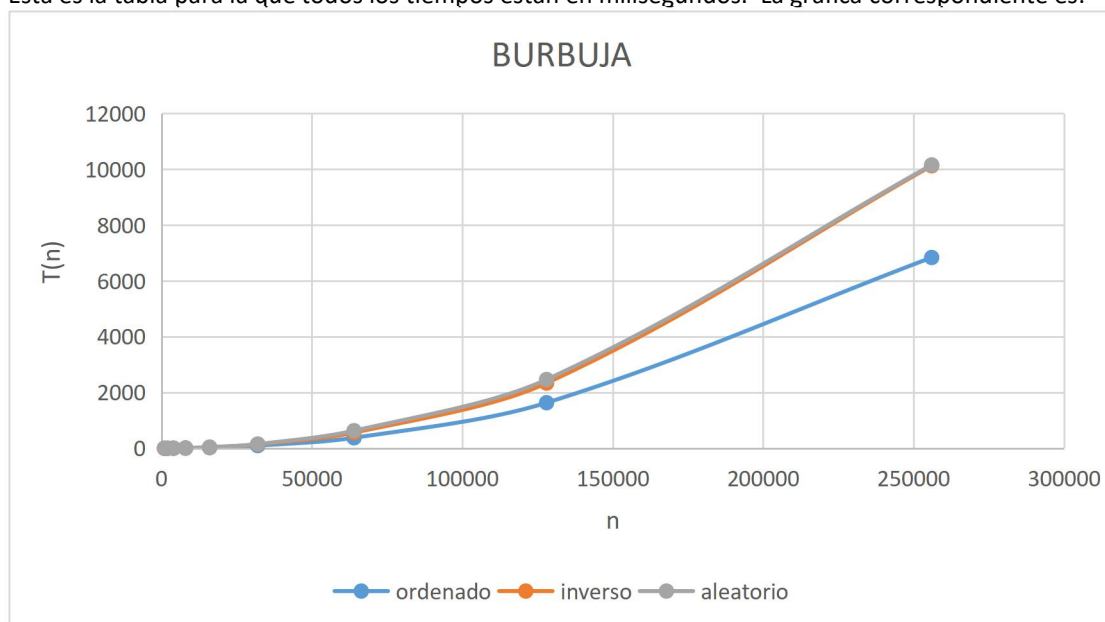
ALGORITMO BURBUJA

N	t ordenado	t inverso	t aleatorio
1000	23	31	38
2000	42	64	62
4000	154	248	259
8000	597	854	898
16000	2334	3366	4016
32000	9575	14080	14972
64000	37496	56052	62990
128000	163272	233610	246468
256000	683834	1012751	1015719
	nº veces: 100	nº veces: 100	nº veces: 100

Esta es la tabla correspondiente al algoritmo burbuja. En este caso como en el anterior algoritmo tenemos el mismo número de repeticiones para todas las diferentes configuraciones del vector. Podemos ver que los tiempos mas elevados los tendremos para las configuraciones en inverso y aleatorio, estando estos dos muy parejos en tiempos. La tabla calculada a ms es la siguiente:

N	t ordenado	t inverso	t aleatorio
1000	0,23	0,31	0,38
2000	0,42	0,64	0,62
4000	1,54	2,48	2,59
8000	5,97	8,54	8,98
16000	23,34	33,66	40,16
32000	95,75	140,8	149,72
64000	374,96	560,52	629,9
128000	1632,72	2336,1	2464,68
256000	6838,34	10127,51	10157,19

Esta es la tabla para la que todos los tiempos están en milisegundos. La gráfica correspondiente es:



Haciendo el análisis del código podemos ver como en el caso del algoritmo por selección, tenemos dos bucles for que se van a ejecutar independientemente de si el vector esta ordenado o no. Por tanto la complejidad de este algoritmo para el caso bueno, medio y malo es de una complejidad cuadrática.

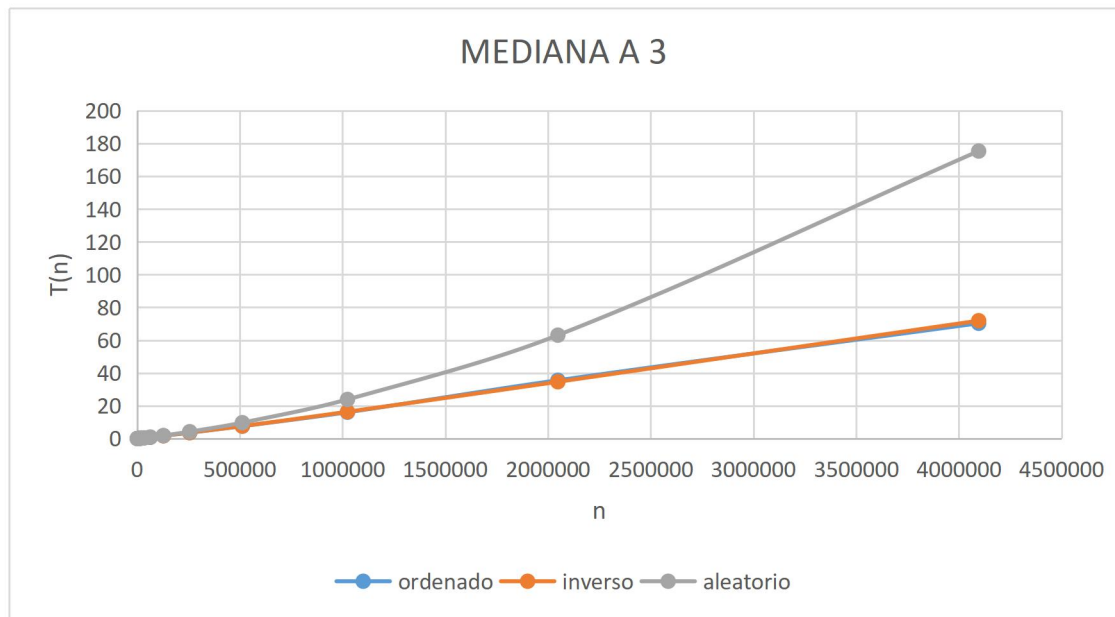
ALGORITMO MEDIANA 3

N	t ordenado	t inverso	t aleatorio
1000	22	19	27
2000	23	23	19
4000	39	42	41
8000	94	89	95
16000	186	192	198
32000	389	388	397
64000	815	849	868
128000	1718	1684	1835
256000	3588	3711	4217
512000	7575	7606	9711
1024000	16067	16347	23884
2048000	35550	34637	63080
4096000	70315	71909	175369
	nº veces: 1000	nº veces: 1000	nº veces: 1000

Esta es la tabla correspondiente al algoritmo rápido con la implementación de pivote mediana 3. En este caso también tendremos el mismo número de ejecuciones del algoritmo para todas las configuraciones del vector. Este algoritmo es algo más especial que los anteriores, ya que se trata de un algoritmo recursivo y de la forma que está implementado es mucho más rápido que los dos anteriores algoritmos. Esta tabla pasada a ms queda de la siguiente forma:

N	t ordenado	t inverso	t aleatorio
1000	0,022	0,019	0,027
2000	0,023	0,023	0,019
4000	0,039	0,042	0,041
8000	0,094	0,089	0,095
16000	0,186	0,192	0,198
32000	0,389	0,388	0,397
64000	0,815	0,849	0,868
128000	1,718	1,684	1,835
256000	3,588	3,711	4,217
512000	7,575	7,606	9,711
1024000	16,067	16,347	23,884
2048000	35,55	34,637	63,08
4096000	70,315	71,909	175,369

Con estos datos podemos construir la gráfica, que está en la próxima hoja.



Como se puede ver en este ejemplo, la configuración más exigente es la configuración de aleatorio.