ALGORITMIA

PRÁCTICA 2

Héctor Lavandeira Fernández
UO277303 | UNIVERSIDAD DE OVIEDO – CURSO 2021/22

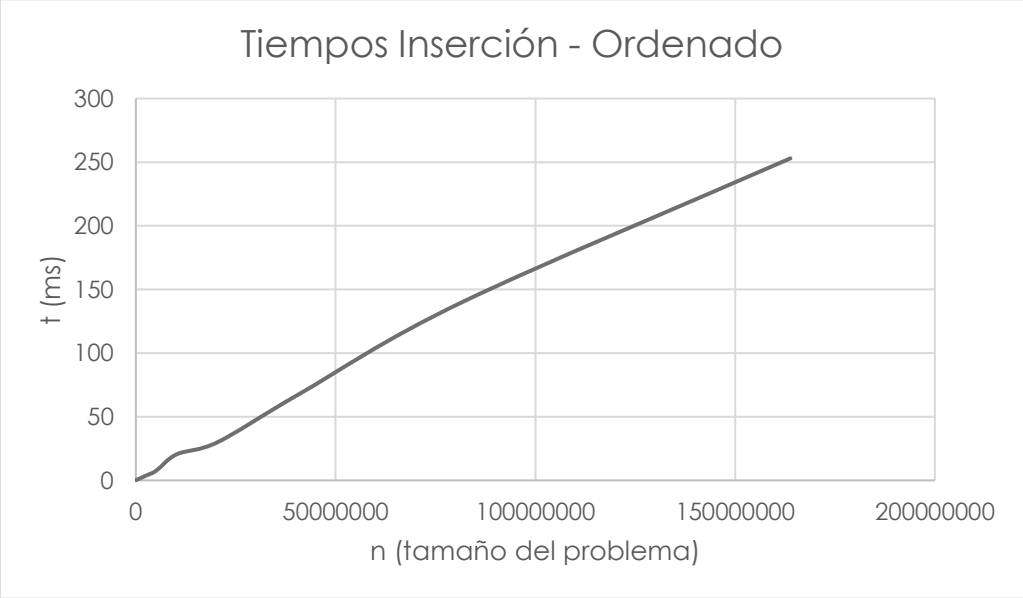
Características principales del ordenador:

Procesador:	i5-8250U
Memoria RAM:	8GB

- **ALGORITMO DE INSERCIÓN**

Mediciones de tiempo utilizando un **vector ordenado**:

n	*tiempo_ordenado (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	181	500	0,362
20000	69	500	0,138
40000	62	500	0,124
80000	121	500	0,242
160000	194	500	0,388
320000	305	500	0,61
640000	535	500	1,07
1280000	973	500	1,946
2560000	1889	500	3,778
5120000	3833	500	7,666
10240000	10257	500	20,514
20480000	15014	500	30,028
40960000	33985	500	67,97
81920000	70200	500	140,4
163840000	126502	500	253,004



Este es el caso mejor del algoritmo. Al observar la gráfica, se puede ver como los tiempos representados forman una recta, exceptuando un pequeño pico con valores de tamaño del problema pequeños. Por esto, podemos deducir que la complejidad es O(n).

Mediciones de tiempo utilizando un **vector desordenado aleatoriamente**:

n	*tiempo_aleatorio (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	183	1	183
20000	579	1	579
40000	561	1	561
80000	2100	1	2100
160000	8746	1	8746
320000	36437	1	36437
640000	149471	1	149471
1280000	676115	1	676115

Nota: el último tiempo se midió utilizando un tamaño de problema de 1.280.000, ya que para cualquier n mayor (el siguiente sería 2.560.000), el tiempo de espera era excesivo (más de 20 minutos).

Mediciones de tiempo utilizando un **vector ordenado inversamente**:

n	*tiempo_inverso (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	376	1	376
20000	1060	1	1060
40000	1020	1	1020
80000	4174	1	4174
160000	18185	1	18185
320000	74921	1	74921
640000	311653	1	311653
1280000	1281987	1	1281987

Nota: Al igual que con el caso anterior, los tiempos de espera eran excesivos para continuar midiendo con un tamaño de problema mayor.

Tanto en el caso del vector aleatorio como en el del vector inverso, las representaciones de los tiempos parecen tener una complejidad no lineal. Debido a cómo crecen los valores de tiempo para ambas mediciones, y la tendencia que siguen las dos representaciones en los gráficos, podemos confirmar que la complejidad del algoritmo para ambos casos es $O(n^2)$.

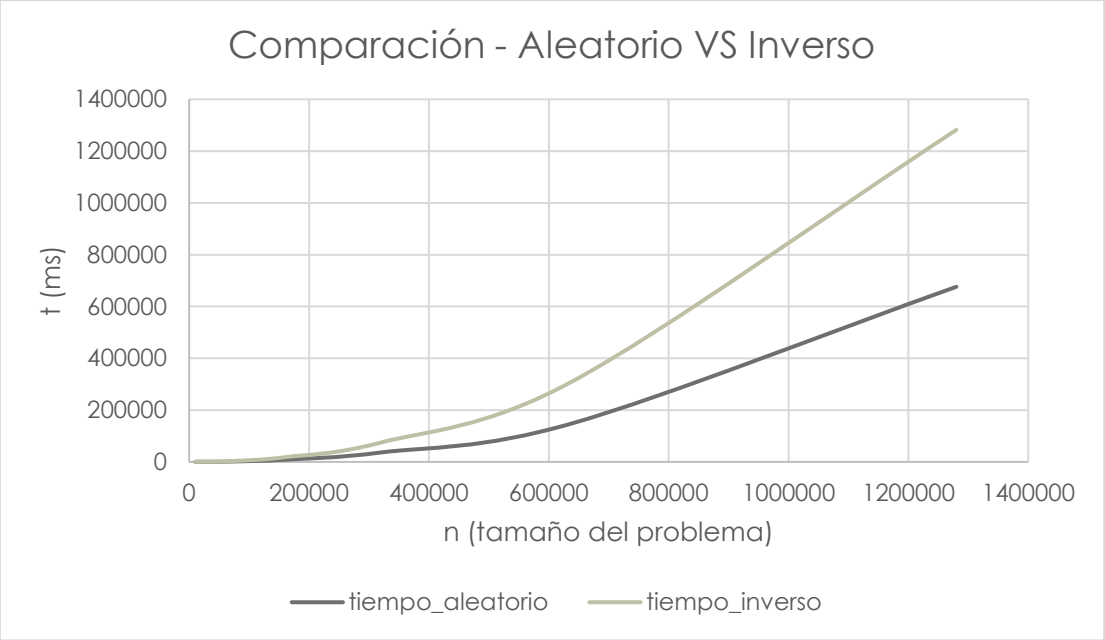
Por tanto, los resultados obtenidos coinciden con los esperados, ya que, como se ha visto en las clases de teoría, la complejidad de este algoritmo es O(n) para su caso mejor y $O(n^2)$ para sus casos peor y medio.

Vamos a comparar los tiempos resultantes de ambas mediciones para ver si obtenemos una constante:

n	*tiempo_aleatorio*	*tiempo_inverso*	*tiempo_aleatorio / tiempo_inverso*
10000	183	376	0,486702128
20000	579	1060	0,546226415
40000	561	1020	0,55
80000	2100	4174	0,503114518
160000	8746	18185	0,480945834
320000	36437	74921	0,486338944
640000	149471	311653	0,479607127
1280000	676115	1281987	0,527396144
			0,507541389

Como podemos observar, al dividir los tiempos de ejecución de ambos casos, obtenemos una constante ≈ 0,5, por lo que podemos confirmar que tienen la misma complejidad.

Comparación del algoritmo con vector aleatorio y con vector inverso:



Comparación de los tres casos:

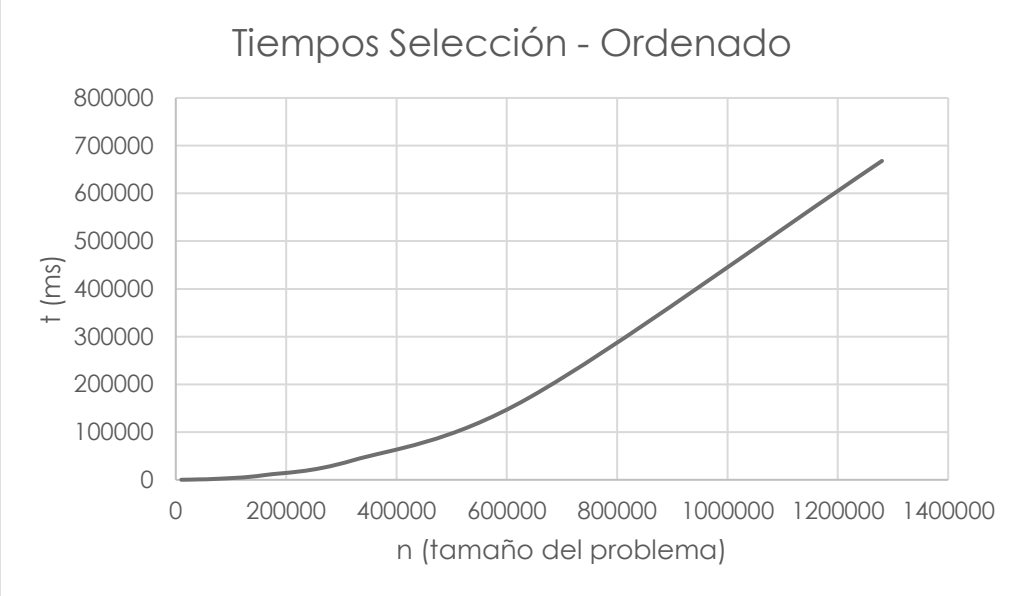
En este último gráfico, la representación de los tiempos con el vector ordenado parece una recta paralela al eje x. Esto se debe a que los otros dos casos tienen tiempos mucho mayores.

- **ALGORITMO DE SELECCIÓN**

Nota: Sólo se han tomado tiempos hasta un tamaño de problema igual a 640.000 o 1.280.000, ya que, para cualquier tamaño mayor, el tiempo era excesivamente largo.

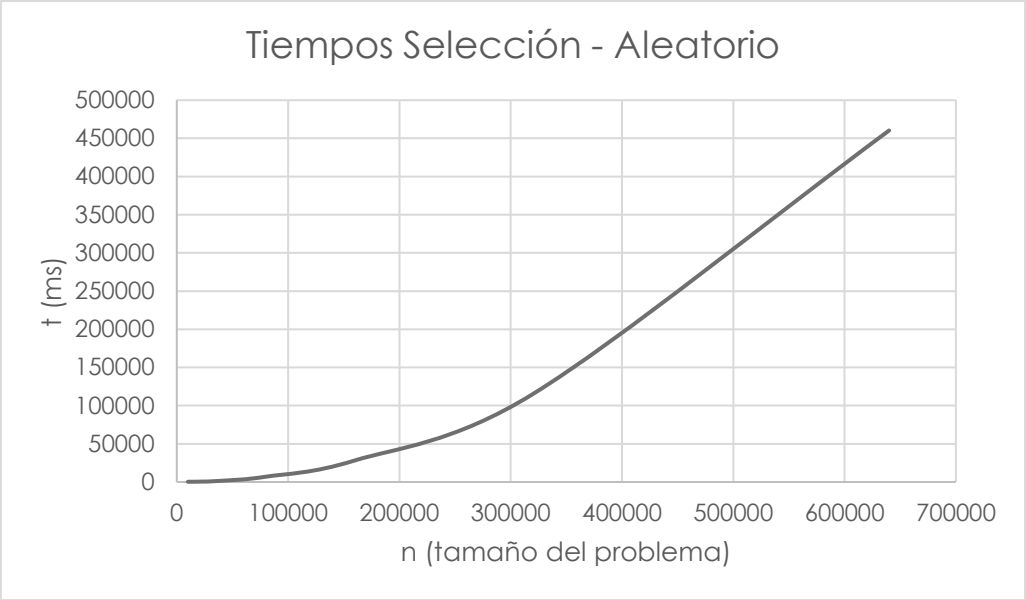
Mediciones de tiempo utilizando un **vector ordenado**:

n	*tiempo_ordenado (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	96	1	96
20000	241	1	241
40000	796	1	796
80000	2346	1	2346
160000	9717	1	9717
320000	40451	1	40451
640000	171748	1	171748
1280000	668073	1	668073



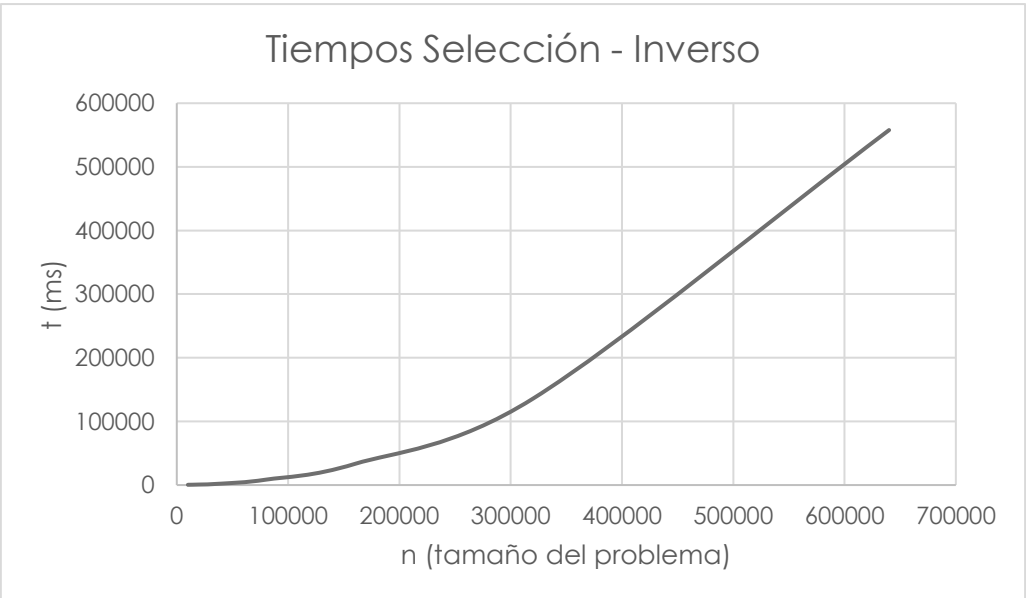
Mediciones de tiempo utilizando un **vector desordenado aleatoriamente**:

n	*tiempo_aleatorio (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	311	1	311
20000	461	1	461
40000	1560	1	1560
80000	6912	1	6912
160000	28207	1	28207
320000	115200	1	115200
640000	460248	1	460248

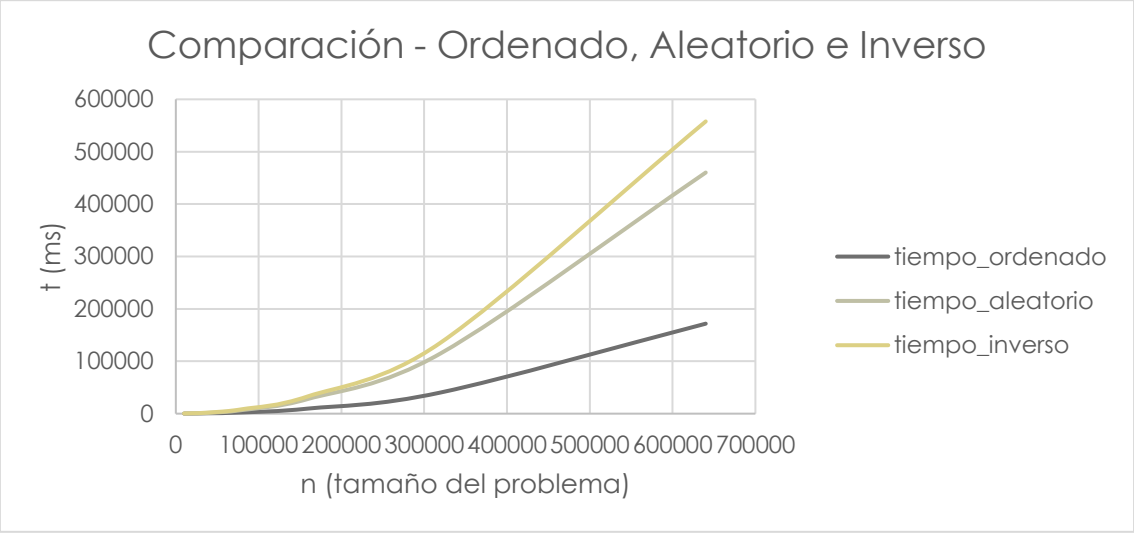


Mediciones de tiempo utilizando un **vector ordenado inversamente**:

n	*tiempo_inverso (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	420	1	420
20000	812	1	812
40000	2212	1	2212
80000	8483	1	8483
160000	33244	1	33244
320000	135743	1	135743
640000	557778	1	557778



Comparación de los tres casos:



Como podemos observar en el gráfico, los casos de vector aleatorio e inverso tienen ambos una representación similar. Aunque el caso de vector ordenado tenga tiempos más bajos, es visible que tiene la misma tendencia que los otros dos casos.

Los valores de tiempo, en cualquiera de los tres casos, crecen demasiado despacio como para decir que se trata de una complejidad cúbica o mayor, y como es obvio que no siguen una tendencia lineal, podemos confirmar que su complejidad es cuadrática: $O(n^2)$.

Al comparar el caso de vector ordenado con los otros, obtenemos los siguientes resultados:

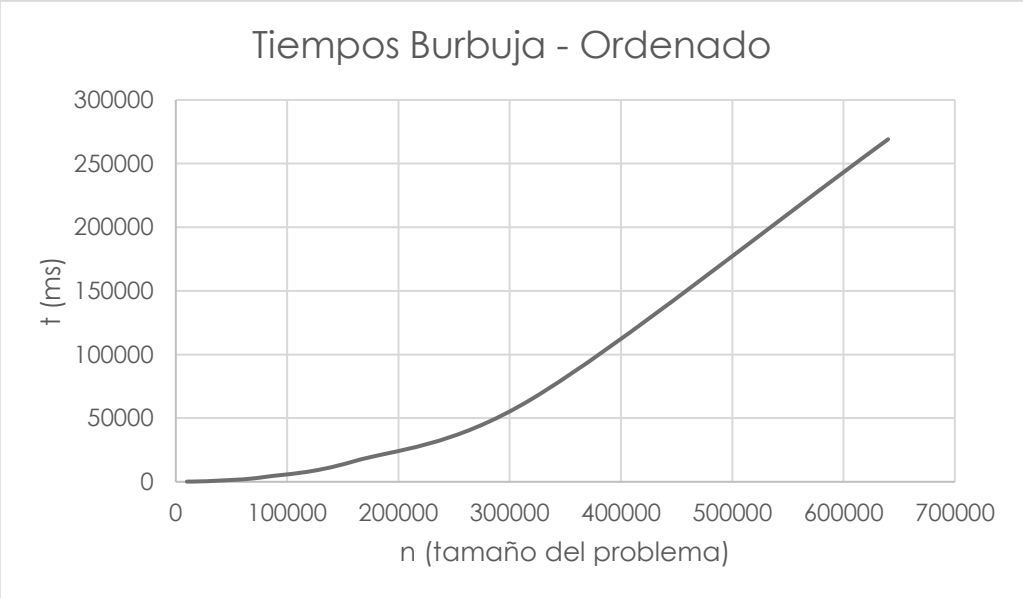
n	*ordenado*	*aleatorio*	*ordenado / aleatorio*	*inverso*	*ordenado / inverso*
10000	96	311	0,308681672	420	0,228571429
20000	241	461	0,522776573	812	0,29679803
40000	796	1560	0,51025641	2212	0,359855335
80000	2346	6912	0,339409722	8483	0,276553106
160000	9717	28207	0,344488957	33244	0,292293346
320000	40451	115200	0,351137153	135743	0,29799695
640000	171748	460248	0,373164033	557778	0,307914618
			0,351137153		0,29679803

Al dividir los tiempos de ejecución en ambos casos, obtenemos valores similares con ambas comparaciones: una mediana de ≈ 0,35 al comparar con aleatorio, y una de ≈ 0,29 al comparar con inverso. Por esto, podemos confirmar que los tres casos para este algoritmo tienen la misma complejidad, es decir, cuadrática.

- **ALGORITMO DE BURBUJA**

Mediciones de tiempo utilizando un **vector ordenado**:

n	*tiempo_ordenado (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	134	1	134
20000	286	1	286
40000	987	1	987
80000	3806	1	3806
160000	16092	1	16092
320000	65124	1	65124
640000	269094	1	269094

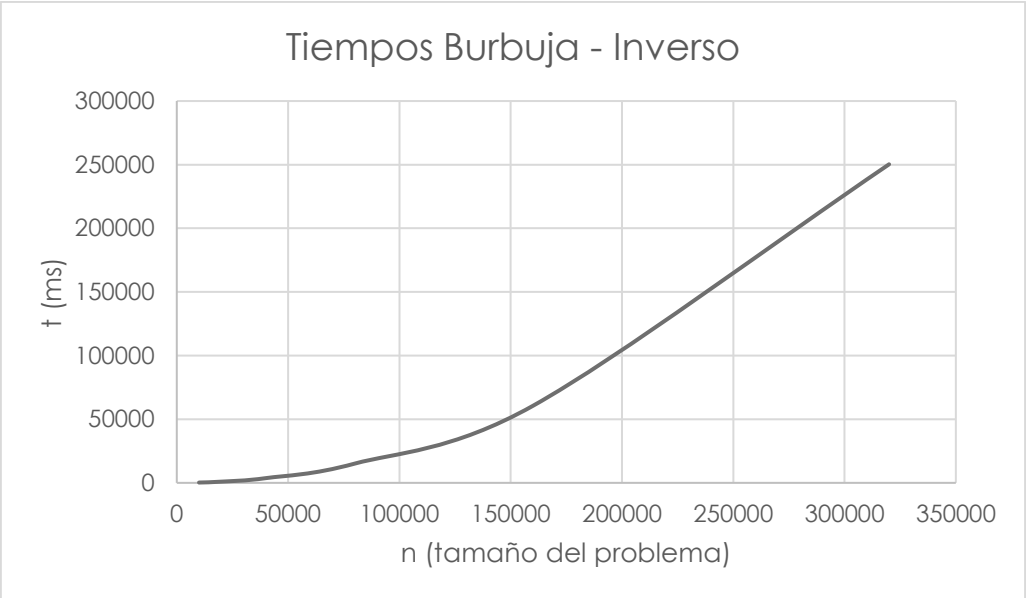


Mediciones de tiempo utilizando un **vector desordenado aleatoriamente**:

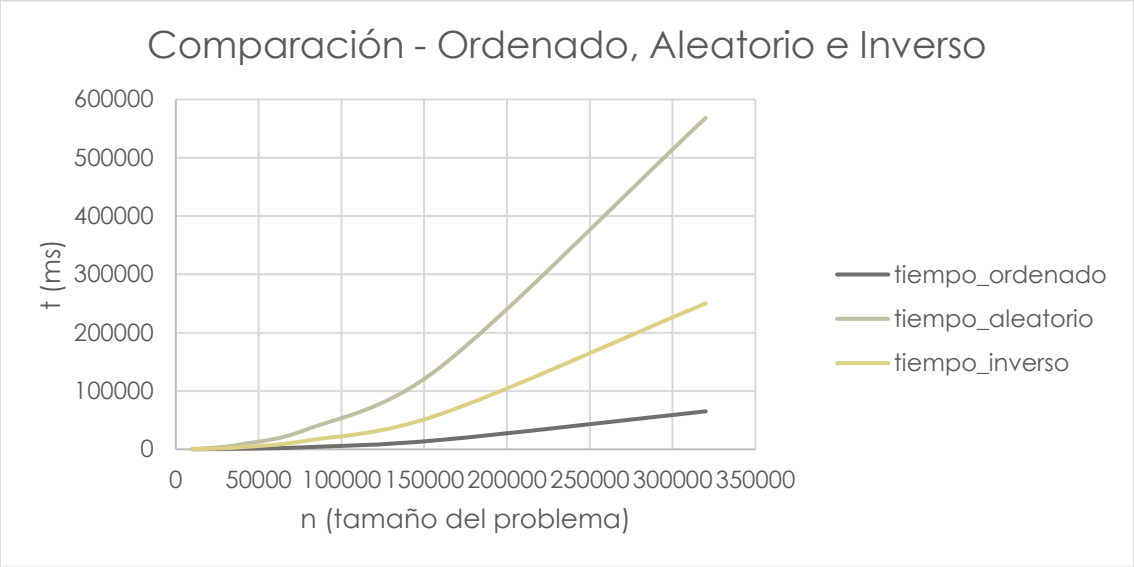
n	*tiempo_aleatorio (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	777	1	777
20000	2180	1	2180
40000	8856	1	8856
80000	35060	1	35060
160000	141478	1	141478
320000	568507	1	568507

Mediciones de tiempo utilizando un **vector ordenado inversamente**:

n	*tiempo_inverso (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	269	1	269
20000	952	1	952
40000	3724	1	3724
80000	15167	1	15167
160000	60546	1	60546
320000	250293	1	250293



Comparación de los tres casos:



Como podemos observar en el gráfico, en los tres casos los tiempos se representan con una tendencia no lineal, aunque para cada caso los valores de tiempo sean mayores o menores, lo que hace que algunos parezcan casi una línea recta.

n	*ordenado*	*aleatorio*	*inverso*	*ordenado / aleatorio*	*ordenado / inverso*
10000	134	777	269	0,172458172	0,498141264
20000	286	2180	952	0,131192661	0,300420168
40000	987	8856	3724	0,111449864	0,265037594
80000	3806	35060	15167	0,10855676	0,25093954
160000	16092	141478	60546	0,113742066	0,265781389
320000	65124	568507	250293	0,114552679	0,260191056
				0,114147373	0,265409492

Al comparar el caso de vector ordenado con los otros dos, obtenemos unas medianas de ≈ 0,11 con el caso de vector aleatorio, y una de ≈ 0,26 con el caso de vector inverso.

Estos resultados coinciden con los esperados, ya que, para el algoritmo de burbuja, tanto el caso peor, como el mejor y el medio, tienen una complejidad cuadrática: $O(n^2)$.

- **ALGORITMO QUICKSORT CON PIVOTE CENTRAL E INICIAL**

Antes de tomar y estudiar los tiempos del algoritmo con pivote utilizando mediana a 3, vamos a explicar y comparar las versiones RapidoFatal y RapidoCentral:

- RapidoCentral utiliza como pivote el elemento central del vector. Esto supone que las complejidades para los casos mejor, peor y medio sean lineales, ya que está sacando provecho de la recursividad del método gracias al pivote elegido.

- RapidoFatal elige como pivote el elemento inicial del vector. Como podemos ver en el código, el pivote es el elemento posicionado más a la izquierda del vector (línea 20):

```
18    private int particion(int iz, int de) {
19        int i, pivote;
20        pivote = this.elements[iz];
21        i = iz;
22        for (int s = iz + 1; s <= de; s++)
23            if (this.elements[s] <= pivote) {
24                i++;
25                intercambiar(i, s);
26            }
27        intercambiar(iz, i);
28        return i;
29    }
```

Al elegir como pivote el primer elemento del vector, se está desaprovechando la recursividad, ya que una de las particiones queda vacía, lo que hace que aumente mucho el número de llamadas recursivas.

El hecho de que aumente tanto el número de llamadas recursivas puede hacer que el método lance una excepción del tipo StackOverflow, ya que la pila acaba llenándose.

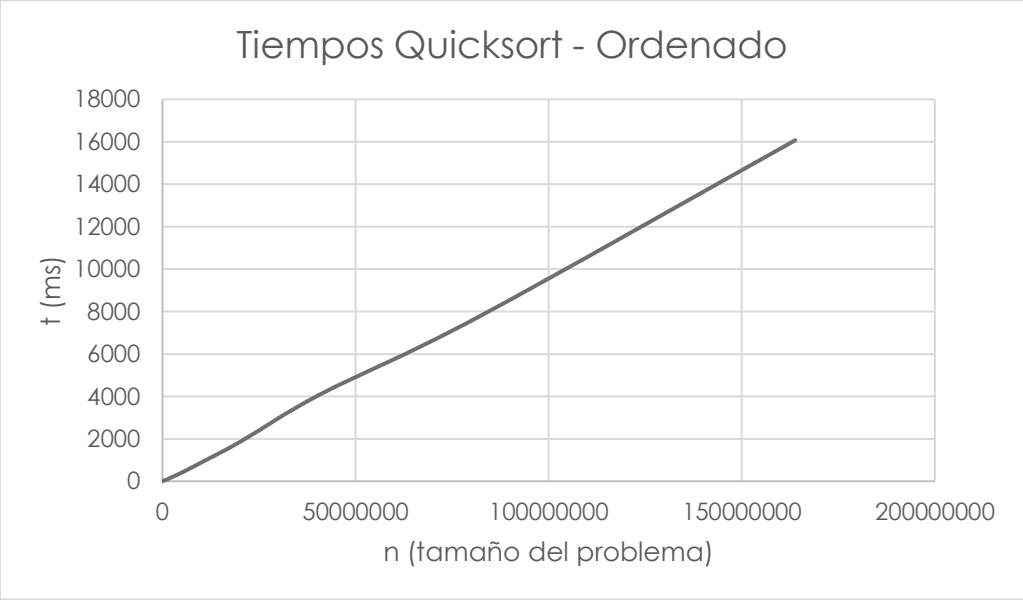
Este algoritmo funciona mal cuando se intenta ordenar un vector que ya está ordenado (inversamente o no), ya que una de las particiones quedará vacía. Sin embargo, si el vector está desordenado, es mucho más probable que ninguna partición quede vacía, lo que aumentará el rendimiento del algoritmo.

- **ALGORITMO QUICKSORT CON PIVOTE UTILIZANDO MEDIANA A 3**

Nota: Para las mediciones de este algoritmo, se han utilizado todos los tamaños de problema (n) posibles hasta que el programa agotara la memoria disponible y lanzara una excepción.

Mediciones de tiempo utilizando un **vector ordenado**:

n	*tiempo_ordenado (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	98	50	1,96
20000	114	50	2,28
40000	243	50	4,86
80000	438	50	8,76
160000	710	50	14,2
320000	1502	50	30,04
640000	2303	50	46,06
1280000	4648	50	92,96
2560000	2058	10	205,8
5120000	4191	10	419,1
10240000	8992	10	899,2
20480000	18975	10	1897,5
40960000	41067	10	4106,7
81920000	77550	10	7755
163840000	160741	10	16074,1

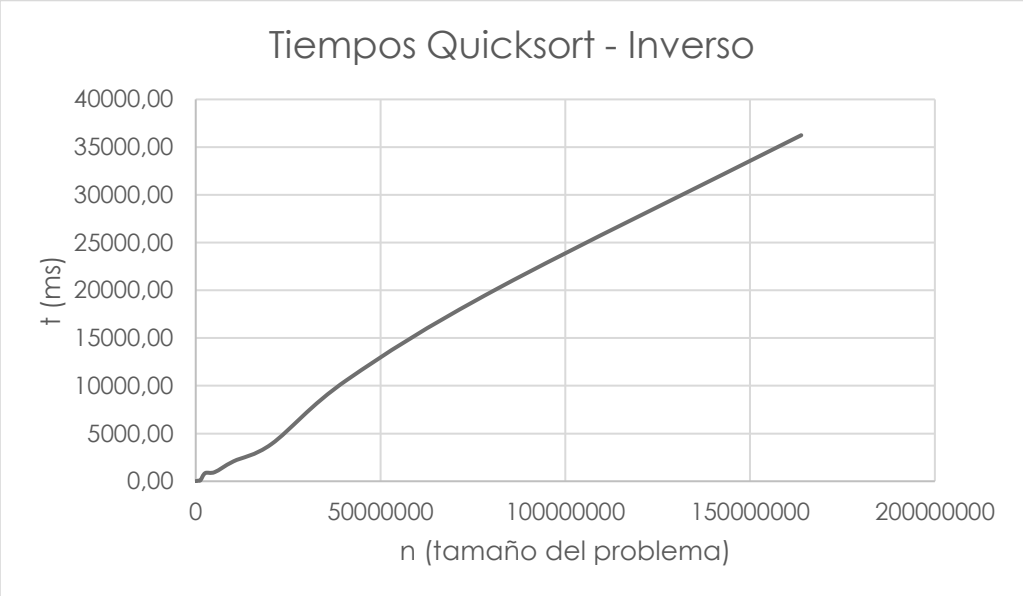


Mediciones de tiempo utilizando un **vector desordenado aleatoriamente**:

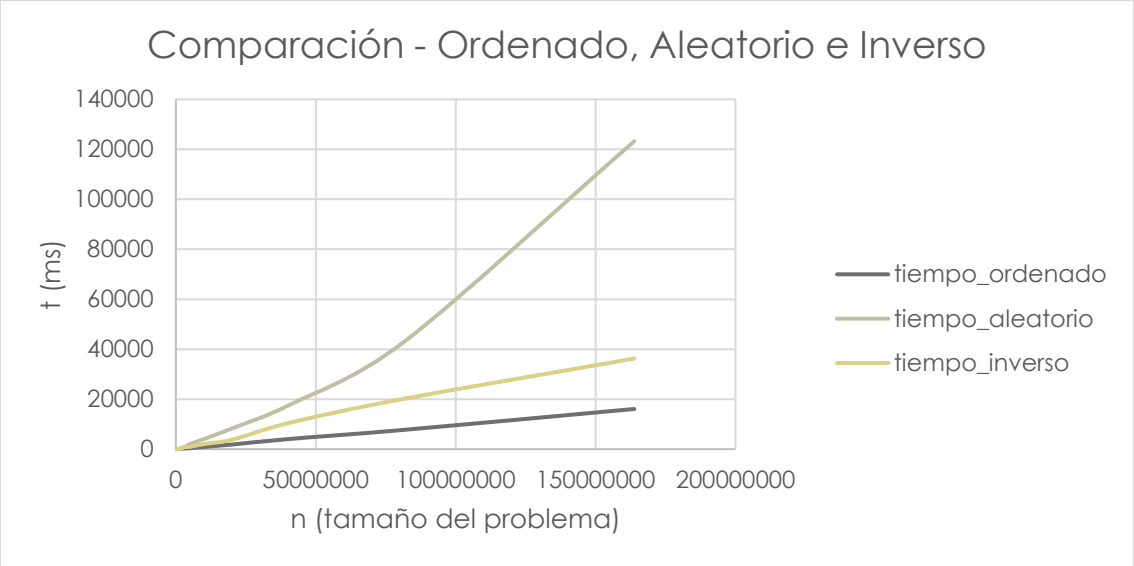
n	*tiempo_aleatorio (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	92	15	6,13
20000	81	15	5,4
40000	103	15	6,87
80000	188	15	12,53
160000	267	15	17,8
320000	1297	15	86,47
640000	1284	15	85,6
1280000	2572	15	171,47
2560000	6563	15	437,53
5120000	2028	1	2028
10240000	4127	1	4127
20480000	8463	1	8463
40960000	17893	1	17893
81920000	43247	1	43247
163840000	123232	1	123232

Mediciones de tiempo utilizando un **vector ordenado inversamente**:

n	*tiempo_inverso (ms)*	*repeticiones*	*tiempo / repeticiones (ms)*
10000	98	15	6,53
20000	89	15	5,93
40000	98	15	6,53
80000	218	15	14,53
160000	410	15	27,33
320000	439	15	29,27
640000	903	15	60,2
1280000	1572	15	104,8
2560000	854	1	854
5120000	954	1	954
10240000	2091	1	2091
20480000	3891	1	3891
40960000	10627	1	10627
81920000	20237	1	20237
163840000	36248	1	36248



Comparación de los tres casos:



Como se puede observar en el gráfico, los tres casos parecen seguir una tendencia lineal. En este caso no se puede asegurar a simple vista, ya que, según se ha visto en las clases de teoría, la complejidad de este algoritmo es $O(n \cdot \log n)$ en los tres casos (utilizando la mediana a tres como pivote).

En el caso en el que se usa un vector aleatorio, parece que hay algunos tiempos que no hacen coincidir del todo la representación de estos con la complejidad del algoritmo.