Informe sobre los tiempos de ejecución de la Práctica 3

MARCOS GARCÍA BLANCO

Programación II
Grupo 03

6 de mayo de 2025

I. INTRODUCCIÓN

Este informe tiene como objetivo evaluar y comparar los **tiempos de ejecución** de tres algoritmos: búsqueda de pares que sumen un valor dado, ordenación mediante Quicksort y ordenación mediante Insertion Sort. El análisis de dichos algoritmos se realizará mediante gráficas que relacionen el tamaño de los datos de entrada con el tiempo de ejecución, analizando de esta forma su complejidad computacional.

Este estudio se realizará en un ordenador portátil con procesador AMD Ryzen 7 7735HS, 16 GB de RAM, tarjeta gráfica NVIDIA RTX 3060 y un disco duro SSD, ejecutando Ubuntu 24.04 como sistema operativo.

II. MÉTODO UTILIZADO

Este estudio se realizará de una forma experimental implementando un programa que genere vectores desde un tamaño inicial hasta otro valor final proporcionados por el usuario en línea de comandos. Estos vectores los genera siguiendo un determinado paso que se da también por línea de comandos. La medición de los tiempos de ejecución de cada algoritmo se hará calculando con la función clock() el número de ticks que pasan desde el inicio hasta el final de la ejecución. Para calcular el tiempo en segundos se divide el número de ticks de reloj por la constante CLOCKS_PER_SEC.

Una vez conseguidos los resultados experimentales, estos se compararán con las complejidades teóricas de los algoritmos. Se comprobará que los algoritmos de **Búsqueda de pares que sumen X** y **Ordenación por *Insertion Sort*** sigan una complejidad de $O(n^2)$ y que el algoritmo de **Ordenación por *Quicksort*** tenga una complejidad superlineal en el caso promedio o cuadrática en el peor de los casos. Por último, cabe destacar que se considerará que un algoritmo deja de ser gestionable temporalmente cuando su ejecución supera los cinco minutos (300 segundos).

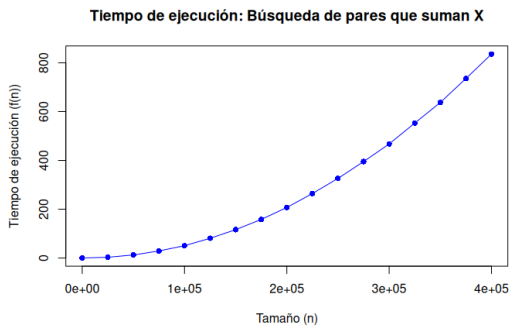
III. ANÁLISIS ALGORITMO DE BÚSQUEDA DE PAREJAS QUE SUMEN X

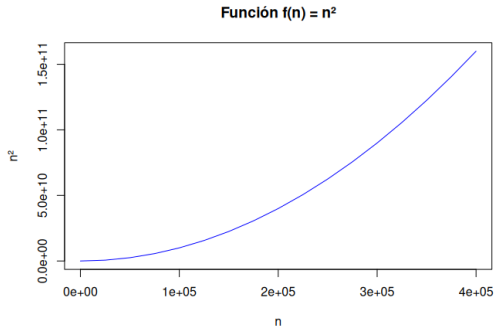
En este apartado se analizará con detalle la ejecución del algoritmo de **Búsqueda de pares que sumen X**. La implementación de este algoritmo empleando dos bucles *for* de manera anidada hace que la complejidad de este algoritmo sea cuadrática, es decir, tiene una complejidad de $O(n)$. Ejecutando el programa obtenemos los siguientes resultados:

Cuadro 1: *Tiempos de ejecución para el algoritmo de búsqueda de pares que sumen X*

n	$t(n)$ [s]	n	$t(n)$ [s]
25 000	3,177 884	225 000	264,368 281
50 000	12,494 016	250 000	326,344 256
75 000	29,038 608	275 000	395,240 936
100 000	50,569 262	300 000	467,449 335
125 000	80,957 913	325 000	552,983 633
150 000	116,270 985	350 000	637,450 043
175 000	158,212 005	375 000	735,465 946
200 000	207,130 153	400 000	835,602 250

Analizando los datos se obtiene que el algoritmo tiene una complejidad cuadrática, lo cual se corrobora con las siguientes gráficas hechas en R:





Observando las gráficas se ve claramente que la forma de ambas es prácticamente idéntica y que crecen siguiendo la misma tendencia. También cabe destacar que, a partir de vectores de 250000 componentes, este algoritmo deja de ser gestionable temporalmente hablando debido a que su ejecución supera los 300 segundos.

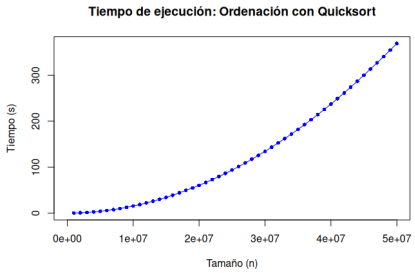
IV. Análisis algoritmo de ordenación *Quicksort*

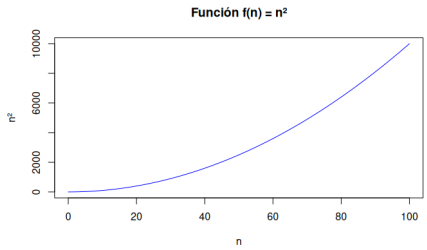
En este apartado se analizará la ejecución del algoritmo de ordenación **Quicksort**. Este es un algoritmo con una complejidad computacional superlineal, es decir $O(n * log(n))$ en el caso promedio y con complejidad cuadrática $(O(n))$ en el peor de los casos. Para comprobar la ejecución se ejecuta el programa comenzando con un vector de tamaño 1000000 y terminando en tamaño 50000000. Ejecutando el programa con estos datos de tamaño obtenemos los siguientes resultados de tiempo de ejecución:

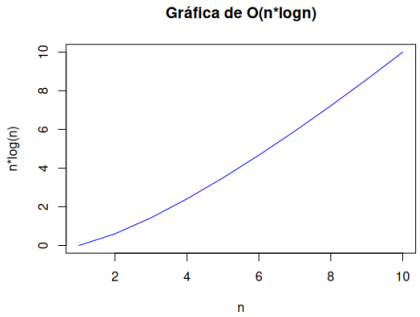
Cuadro 2: *Tiempos de ejecución del algoritmo de ordenación Quicksort [1]*

n	$t(n)$ [s]	n	$t(n)$ [s]	n	$t(n)$ [s]	n	$t(n)$ [s]
1 000 000	0,221 735	14 000 000	29,958 018	27 000 000	109,049 808	40 000 000	237,181 856
2 000 000	0,736 572	15 000 000	34,312 845	28 000 000	117,170 945	41 000 000	249,126 489
3 000 000	1,544 681	16 000 000	38,967 597	29 000 000	125,642 117	42 000 000	261,372 045
4 000 000	2,629 921	17 000 000	44,033 576	30 000 000	134,319 502	43 000 000	273,928 417
5 000 000	4,087 215	18 000 000	49,554 238	31 000 000	143,338 205	44 000 000	286,794 328
6 000 000	5,762 493	19 000 000	54,825 613	32 000 000	152,572 888	45 000 000	299,978 913
7 000 000	7,714 524	20 000 000	60,542 929	33 000 000	162,223 395	46 000 000	313,306 090
8 000 000	10,052 837	21 000 000	66,592 762	34 000 000	171,981 447	47 000 000	326,847 333
9 000 000	12,602 906	22 000 000	72,923 815	35 000 000	182,056 260	48 000 000	340,612 507
10 000 000	15,487 576	23 000 000	79,551 722	36 000 000	192,571 832	49 000 000	354,619 923
11 000 000	18,712 045	24 000 000	86,498 846	37 000 000	203,216 719	50 000 000	368,862 074
12 000 000	22,183 721	25 000 000	93,712 843	38 000 000	214,189 812		
13 000 000	25,925 497	26 000 000	101,212 458	39 000 000	225,627 371		

Fijándonos en la tabla, este algoritmo, en términos de complejidad temporal, deja de ser tratable a partir de vectores de 46000000 componentes. Para analizar la complejidad del algoritmo, haremos en R las gráficas de estos valores de tiempo y las compararemos con la gráfica de la función $f(n) = n * log(n)$ y $f(n) = n$:







Al observar las gráficas, se puede apreciar que la correspondiente a los tiempos de ejecución de Quicksort crece de manera menos pronunciada que la de $f(n) = n^2$, aunque no sigue exactamente el mismo patrón que $f(n) = nlog(n)$, sino que crece ligeramente más rápido. Esto concuerda con lo que se había anticipado: la complejidad de Quicksort es, en el caso promedio, del orden de $O(nlog(n))$, mientras que en el peor caso puede alcanzar un comportamiento cuadrático, es decir, $O(n^2)$.

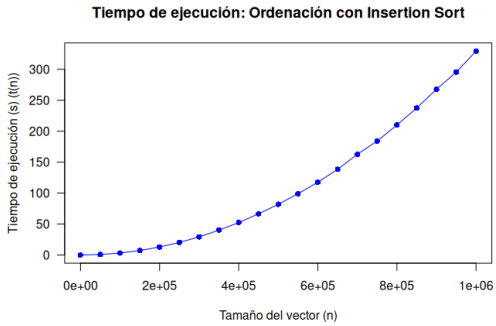
V. Análisis algoritmo de ordenación *Insertion Sort* [2]

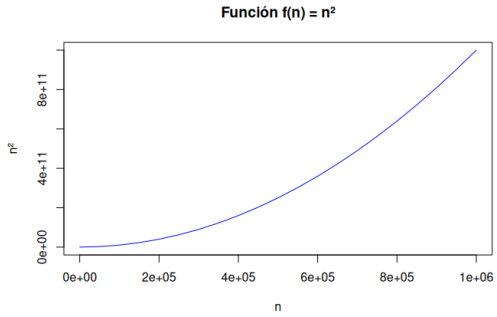
En este apartado se analizará la ejecución del algoritmo de ordenación **Insertion Sort**, el cual teóricamente tiene una complejidad computacional $O(n)$ tanto en el peor de los casos como en un caso promedio. Al igual que en el caso del algoritmo de búsqueda de pares que sumen un número X, emplearemos las mismas técnicas para comprobar que la ejecución corrobora dicha complejidad. Comenzando la ejecución con tamaño 50000 y terminándola en 1000000 con paso de 50000, obtenemos los siguientes resultados:

Cuadro 3: *Tiempos de ejecución para el algoritmo de ordenación con Insertion Sort*

n	$t(n)$ [s]	n	$t(n)$ [s]
50 000	0,804 802	550 000	99,020 914
100 000	3,226 710	600 000	117,616 480
150 000	7,279 441	650 000	138,414 102
200 000	13,006 728	700 000	162,659 688
250 000	20,325 323	750 000	184,072 679
300 000	29,353 373	800 000	209,987 758
350 000	40,136 722	850 000	237,536 299
400 000	52,409 473	900 000	267,633 650
450 000	66,506 360	950 000	295,398 063
500 000	81,914 765	1 000 000	329,320 819

Para corroborar la complejidad cuadrática, haremos la representación del tamaño del vector con respecto al tiempo de ejecución y la gráfica de $O(n)$:





Con estas gráficas hechas en R también se observa claramente que la forma y la velocidad de crecimiento coinciden para ambas gráficas, por lo que se puede concluir que este algoritmo de ordenación sigue una complejidad cuadrática. Por otro lado, el algoritmo deja de ser gestionable en términos de complejidad temporal a partir de vectores de tamaño 1000000.

VI. Conclusiones

Para concluir, se compararon los resultados de ejecución de los tres algoritmos. El primero en volverse ineficiente es el de búsqueda de pares, que con vectores de 250000 elementos supera el límite de tiempo aceptable. En contraste, el algoritmo más eficiente es Quicksort, que sigue siendo tratable hasta tamaños de 45000000 elementos, superando ampliamente a los otros dos. Por otro lado, el algoritmo Insertion Sort alcanza su límite con vectores de 1000000 elementos. En términos de complejidad temporal, el orden de eficiencia de mejor a peor es: Quicksort, Insertion Sort y búsqueda de pares.

REFERENCIAS

[1] Programiz. (2023) Algoritmo de ordenamiento rápido (quick sort). [En línea]. Último acceso: 2025-04-29. Disponible en: https://www.programiz.com/dsa/quick-sort.

[2] Programiz. (2023) Algoritmo de ordenamiento por inserción (insertion sort). [En línea]. Último acceso: 2025-04-29. Disponible en: https://www.programiz.com/dsa/insertion-sort.