Punto 1: Sistema de Clasificación Deportiva

Problema:

Definir para un sistema de clasificación de resultados deportivos (ordenar equipos o atletas por puntuación o tiempo) y un conjunto de datos de al menos 200 elementos, cuál sería el método de ordenamiento por el que usted recomendaría ordenar la data para las consultas propuestas.

Solución:

1. Introducción

La eficiente clasificación de resultados deportivos es fundamental en el mundo del deporte, ya sea para ligas, competiciones individuales o torneos multitudinarios. Con el crecimiento de la cantidad de datos generados en eventos deportivos, la selección del método de ordenamiento más adecuado se vuelve esencial para analizar y presentar información precisa y relevante sobre equipos o atletas.

Este experimento tiene como objetivo determinar el método de ordenamiento más eficiente para un sistema de clasificación deportiva por puntaje y tiempo. Se explorará cómo diferentes algoritmos de ordenamiento pueden afectar el rendimiento y la velocidad de clasificación de grandes conjuntos de datos relacionados con resultados deportivos.

2. Marco Teórico

Métodos de Ordenamiento: Definición y Propósito

Los métodos de ordenamiento son algoritmos diseñados para organizar una lista de elementos en un orden específico, ya sea ascendente o descendente, según un criterio definido, como el valor numérico o alfabético. Su propósito principal es organizar datos de manera estructurada para facilitar su búsqueda, acceso y análisis, contribuyendo a la eficiencia en la manipulación y presentación de información en sistemas de gestión y clasificación.

Breve Descripción de Métodos de Ordenamiento Utilizados

1. MergeSort:

    - Es un algoritmo de ordenamiento basado en la estrategia "dividir y conquistar". Divide la lista en mitades, ordena cada mitad recursivamente y luego combina las mitades ordenadas.

- Ofrece estabilidad en la clasificación y es eficiente en grandes conjuntos de datos, aunque su complejidad espacial puede ser un inconveniente.

- En el peor caso, promedio y mejor caso, MergeSort tiene un orden de complejidad de tiempo de O(n log n), donde 'n' es el número de elementos a ordenar.

2. HeapSort:

   - Clasifica datos utilizando una estructura de datos llamada heap (montículo). Convierte la lista en un heap para luego extraer sucesivamente el elemento máximo y reorganizar el heap.

   - Es eficiente y estable, con un rendimiento constante independientemente del estado de los datos, adecuado para grandes conjuntos de datos.

   - En todos los casos (peor, promedio y mejor), HeapSort tiene un orden de complejidad de tiempo de O(n log n).

3. TimSort:

   - Es un algoritmo híbrido que combina MergeSort e InsertionSort. Divide los datos en "runs" (secuencias ordenadas) y aplica operaciones de fusión.

   - Destaca por su estabilidad y eficiencia en datos parcialmente ordenados o con patrones preexistentes.

   - En el peor caso, TimSort tiene un orden de complejidad de tiempo de O(n log n). Sin embargo, en la práctica, su rendimiento tiende a ser más eficiente debido a adaptaciones que lo hacen más rápido en datos parcialmente ordenados o con ciertas características.

Importancia de la Selección del Método de Ordenamiento

La elección del método de ordenamiento adecuado es crucial en los sistemas de clasificación deportiva. Se busca un algoritmo estable y eficiente que maneje grandes volúmenes de datos, se adapte a actualizaciones frecuentes en la clasificación y tenga una complejidad algorítmica moderada para permitir respuestas ágiles en tiempo real.

La comparación y evaluación de algoritmos como MergeSort, HeapSort y TimSort proporcionarán información valiosa para determinar el método de ordenamiento más apropiado para la clasificación precisa y ágil de datos deportivos.

3. Justificación

La elección del método de ordenamiento en un sistema de clasificación deportiva tiene un impacto directo en la eficiencia del sistema. Diferentes algoritmos de ordenamiento pueden ofrecer ventajas o desventajas en términos de tiempo de procesamiento, uso de recursos computacionales y adaptabilidad a conjuntos de datos específicos.

La comparación y evaluación de distintos métodos de ordenamiento como *MergeSort*, *HeapSort* y *TimSort*, todos ellos reconocidos por su estabilidad, resulta crucial en la determinación del más idóneo para ordenar datos en sistemas de clasificación deportiva. Esta estabilidad es fundamental en competiciones que comparan puntuaciones y consideran el tiempo, ya que garantiza que elementos con valores iguales mantengan su orden relativo original, brindando resultados precisos y coherentes.

Considerando la inherente variabilidad en los resultados deportivos, la elección del método de ordenamiento adecuado cobra un papel crucial. Se busca un algoritmo capaz de manejar eficientemente grandes volúmenes de datos, así como de adaptarse a actualizaciones frecuentes en la clasificación. Esto permitirá una respuesta ágil y precisa a medida que se generan y actualizan los datos de resultados deportivos en tiempo real, optimizando la gestión y presentación de información en sistemas de clasificación.

Es importante resaltar que la elección de *MergeSort, HeapSort* y *TimSort* se debe no solo a su estabilidad, sino también a su escalabilidad en conjuntos de datos extensos y a su capacidad para trabajar con tipos variables de datos numéricos. Métodos como Insertion Sort o BubbleSort, aunque sencillos, no son eficientes en conjuntos de datos grandes, lo que los hace menos adecuados para la gestión de datos deportivos a gran escala. Asimismo, algoritmos como Bucket Sort dependen de una distribución uniforme de los datos, condición poco común en competiciones deportivas, limitando su eficacia. Por otra parte, tanto RadixSort como CountingSort presentan restricciones en la distribución y estructura específica de los números, lo que los hace menos idóneos para manejar la variabilidad en los resultados deportivos. Además de la estabilidad y la escalabilidad, se buscaba un método de ordenamiento con una complejidad algorítmica no excesivamente alta, ya que la agilidad en el procesamiento y presentación de datos en tiempo real es fundamental en competiciones en curso.

Este experimento servirá como base para optimizar el rendimiento de sistemas de clasificación deportiva, ayudando a seleccionar el algoritmo de ordenamiento más idóneo para garantizar tiempos de respuesta rápidos, precisión en la presentación de resultados y una experiencia de usuario satisfactoria.

El análisis y comparación de diferentes métodos de ordenamiento en este contexto podrían brindar valiosas perspectivas sobre cómo mejorar la eficiencia y la capacidad de respuesta en sistemas de clasificación deportiva, beneficiando a organizaciones deportivas, aficionados y profesionales involucrados en el análisis de datos deportivos.

4. Descripción del Experimento:

Se llevará a cabo un análisis de rendimiento utilizando datos reales de los tiempos registrados por distintos corredores en la maratón de Boston. El objetivo es evaluar y comparar los tiempos de ordenamiento de los métodos *MergeSort*, *HeapSort* y *TimSort*. Se trabajará con 8 conjuntos de datos que contienen registros variados, y se estudiarán diferentes tamaños de estos conjuntos para analizar el rendimiento de cada método de ordenamiento en función del tamaño de los datos.

El propósito principal es determinar cómo varía el tiempo de ejecución de cada algoritmo de ordenamiento al enfrentarse a conjuntos de datos idénticos pero con tamaños distintos, utilizando como base los registros temporales de la maratón de Boston.

El experimento también tiene en cuenta la posible influencia de factores variables, como la diferencia en los datos, en el tiempo de ordenamiento, reconociendo que cada intervalo de tiempo medido podría afectarse o variar.

Objetivos

- Evaluar y comparar el rendimiento de los métodos de ordenamiento MergeSort, HeapSort y TimSort en términos de eficiencia y velocidad en la clasificación de datos deportivos para determinar las posiciones finales de diferentes deportistas o equipos en competencias.

- Observar cómo la variación en el tamaño de los conjuntos de datos afecta el tiempo de ordenamiento de cada algoritmo.

- Ofrecer información precisa para seleccionar el método de ordenamiento más idóneo en sistemas de clasificación deportiva, considerando la agilidad en el procesamiento de los datos por cada uno de los métodos.

Preparación del Entorno

1. Se utilizó un notebook de Jupyter en un entorno local de Python para llevar a cabo las pruebas de rendimiento.

2. Se programaron los métodos *MergeSort*, *HeapSort* y *TimSort*.

3. Se procedió a instalar las bibliotecas necesarias, como pandas para el manejo de datos y matplotlib para la visualización gráfica. Además, se cargó el conjunto de datos de los finalistas de la Maratón de Boston de los años 2015, 2016 y 2017 desde Kaggle, enfocándose en el dataset correspondiente al año 2016.

4. Se realizó un proceso de preprocesamiento de los datos simplificando los tiempos que antes se encontraban en formato horas:minutos:segundos a segundos y extrayendo únicamente las columnas relevantes, como los tiempos registrados por

los corredores en distintas etapas de la carrera (5k, 10k, 15k, 20k, half, 25k, 30k, 35k, 40k) y un identificador representativo de cada corredor (Bib).

5. Se generaron listas de tuplas en formato (Bib, 5k), (Bib, 10k), ..., utilizando los datos extraídos del dataset. Estas tuplas se diseñaron considerando una posible implementación futura que permita acceder al identificador (Bib) de cada corredor para obtener información adicional, como posiciones y tiempos de carrera.

6. Se llevaron a cabo pruebas con diferentes tamaños para cada método de ordenamiento. Se comenzó con 861 elementos y se incrementó en pasos de 861 hasta llegar a 26481 elementos, lo que representó 31 tamaños de conjuntos de datos diferentes.

7. Se calculó el tiempo promedio que cada método de ordenamiento tardó en ordenar listas de diferentes tamaños, lo que permitió evaluar y comparar su rendimiento en condiciones variables de tamaño de datos.

IMPORTANTE: Pruebas y gráficas con métodos de ordenamiento con Python

Se hicieron las pruebas teniendo en cuenta las siguientes características de Python:

- Interpretación y Tipado Dinámico: Es un lenguaje interpretado y de tipado dinámico, lo que significa que el intérprete de Python realiza tareas adicionales durante la ejecución para inferir tipos y llevar a cabo otras operaciones dinámicas. Esto puede resultar en un menor rendimiento en comparación con lenguajes compilados estáticamente como C o C++.

- Gestión de Memoria: Usa un recolector de basura para gestionar la memoria, lo que implica un sobrecoste en términos de tiempo de ejecución. En comparación con lenguajes de bajo nivel, el manejo automático de la memoria puede resultar en una menor eficiencia.

Las cuales podrían sesgar los resultados.

¿Cómo se intentó disminuir el sesgo?

Disminuir el sesgo en pruebas de rendimiento es crucial para obtener resultados más confiables y representativos. En el contexto de las pruebas de métodos de ordenamiento en Python, tuvimos en cuenta varias estrategias para mitigar estos sesgos inherentes:

1. Variabilidad del hardware y entorno de ejecución: Ejecutar las pruebas en diferentes computadoras puede ayudar a capturar la variabilidad del hardware y del entorno. Al realizar pruebas en múltiples sistemas, se puede obtener una imagen más completa y representativa del rendimiento de los métodos de ordenamiento.

2. Múltiples ejecuciones: Realizar las pruebas varias veces en el mismo sistema puede proporcionar datos más consistentes.

3. Tamaños de datos variados: Probar los métodos de ordenamiento con conjuntos de datos de diferentes tamaños puede revelar cómo se comportan en diferentes escalas. Algunos algoritmos pueden ser más eficientes para tamaños pequeños de datos, mientras que otros pueden destacar en tamaños más grandes.

Esto permite obtener resultados más confiables y una comprensión más completa del comportamiento de los algoritmos en el contexto de las características específicas del lenguaje Python.

Resultados

Las representaciones gráficas y tabulares detalladas, que presentan los tiempos de procesamiento específicos obtenidos al emplear los métodos de ordenamiento *MergeSort*, *HeapSort* y *TimSort* con distintos vectores de tamaño variable, se encuentran disponibles en el notebook de Jupyter denominado sort-methods-test.ipynb. Para una síntesis informativa, nos enfocaremos en mostrar y discutir los resultados globales, es decir, los promedios obtenidos a partir de dichos datos.

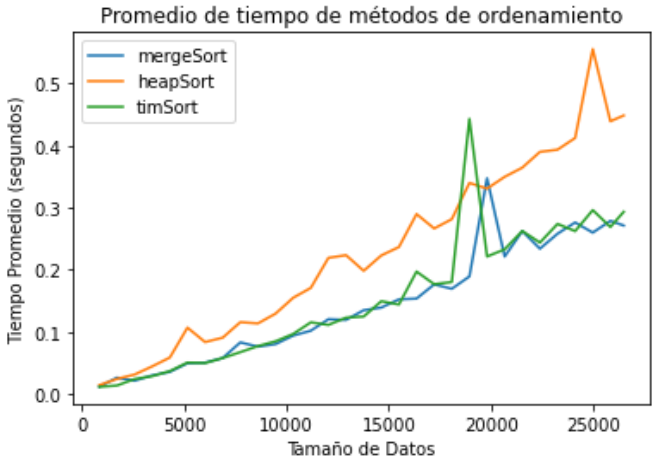
Iteración #1 - Computador #1: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

Tabla de Promedios:

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.011804	0.014349	0.011618
1	1722	0.026412	0.024366	0.013749
2	2583	0.021468	0.031437	0.024179
3	3444	0.030113	0.044613	0.028793
4	4305	0.035699	0.058815	0.036943
5	5166	0.049581	0.106968	0.050751
6	6027	0.050600	0.083726	0.049897
7	6888	0.058048	0.090809	0.058009
8	7749	0.083201	0.115811	0.067503
9	8610	0.076311	0.113767	0.077244
10	9471	0.080465	0.129571	0.084932
11	10332	0.094162	0.155079	0.096811
12	11193	0.102024	0.170992	0.115483
13	12054	0.120543	0.219383	0.111382
14	12915	0.119506	0.223689	0.123017
15	13776	0.135014	0.198472	0.124845
16	14637	0.139274	0.223260	0.149528
17	15498	0.152493	0.236793	0.144253
18	16359	0.153798	0.289870	0.197246
19	17220	0.176604	0.266678	0.176932
20	18081	0.169529	0.281674	0.180430
21	18942	0.189417	0.340077	0.443282
22	19803	0.347495	0.330917	0.221485
23	20664	0.221552	0.349873	0.233131
24	21525	0.262396	0.364635	0.262620
25	22386	0.234105	0.390041	0.243921
26	23247	0.258229	0.393606	0.274012
27	24108	0.276367	0.412465	0.262474
28	24969	0.260017	0.555616	0.296287
29	25830	0.278948	0.439340	0.269081
30	26481	0.271439	0.448685	0.293534

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.144729	0.229206	0.152367



Iteración #2 - Computador #1: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

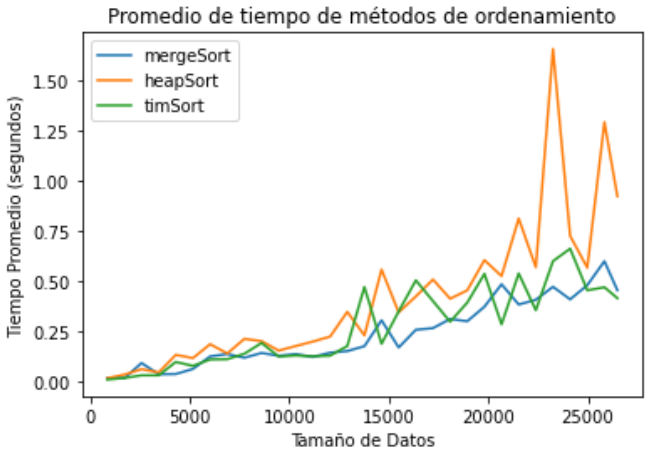
Sara Lorena Duque Ramírez
Natalia Andrea García Ríos

Tabla de Promedios:

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.014064	0.013572	0.006865
1	1722	0.014150	0.031403	0.014611
2	2583	0.089557	0.059017	0.028147
3	3444	0.033392	0.042773	0.027983
4	4305	0.034620	0.130333	0.094866
5	5166	0.060008	0.114299	0.075326
6	6027	0.122686	0.183737	0.107314
7	6888	0.134315	0.137427	0.108718
8	7749	0.116042	0.210442	0.135953
9	8610	0.139843	0.198843	0.189683
10	9471	0.125727	0.151540	0.121946
11	10332	0.134204	0.174490	0.127303
12	11193	0.119802	0.196804	0.122879
13	12054	0.141417	0.221561	0.126456
14	12915	0.149196	0.345383	0.174246
15	13776	0.173558	0.227571	0.469357
16	14637	0.303133	0.556883	0.185474
17	15498	0.167096	0.342638	0.347880
18	16359	0.255770	0.422504	0.502858
19	17220	0.263866	0.507037	0.399000
20	18081	0.309564	0.410908	0.295885
21	18942	0.298406	0.452790	0.391451
22	19803	0.370758	0.603812	0.535259
23	20664	0.483186	0.524056	0.282960
24	21525	0.381740	0.812039	0.537126
25	22386	0.405552	0.567763	0.353326
26	23247	0.470547	1.658996	0.598583
27	24108	0.407637	0.724986	0.660879
28	24969	0.478393	0.565637	0.452678
29	25830	0.598365	1.293031	0.468300
30	26481	0.453092	0.922506	0.413432

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.233861	0.413057	0.269572



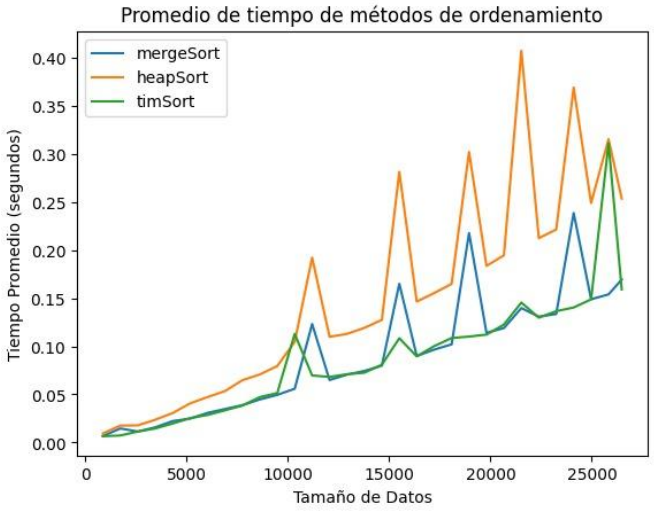
Iteración #3 - Computador #2: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

Sara Lorena Duque Ramírez
Natalia Andrea García Ríos

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.006652	0.009450	0.006536
1	1722	0.014578	0.017579	0.007292
2	2583	0.011425	0.017776	0.011326
3	3444	0.015740	0.023639	0.014583
4	4305	0.022254	0.030637	0.019707
5	5166	0.024822	0.040671	0.025335
6	6027	0.030759	0.047215	0.028610
7	6888	0.034773	0.053446	0.033439
8	7749	0.038807	0.064753	0.038396
9	8610	0.044765	0.070756	0.047219
10	9471	0.049476	0.079482	0.051351
11	10332	0.055902	0.104586	0.112927
12	11193	0.123174	0.192204	0.069698
13	12054	0.064938	0.109895	0.068161
14	12915	0.070584	0.112933	0.070996
15	13776	0.074401	0.119189	0.072555
16	14637	0.079914	0.127533	0.080838
17	15498	0.165015	0.281371	0.108420
18	16359	0.089917	0.146505	0.089564
19	17220	0.096594	0.155393	0.100217
20	18081	0.102007	0.164920	0.108589
21	18942	0.217750	0.302044	0.110123
22	19803	0.114030	0.183605	0.112184
23	20664	0.118986	0.194648	0.122627
24	21525	0.139745	0.407232	0.145490
25	22386	0.130830	0.212428	0.129757
26	23247	0.133594	0.221446	0.136543
27	24108	0.238642	0.369068	0.140367
28	24969	0.148897	0.249009	0.148804
29	25830	0.153981	0.315360	0.311237
30	26481	0.169402	0.253490	0.159164

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.089753	0.150912	0.086518



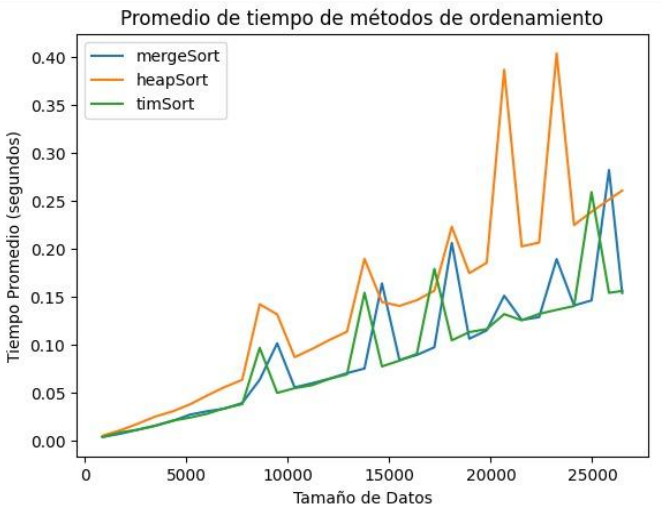
Iteración #4 - Computador #2: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

Tabla de Promedios:

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.004213	0.005502	0.003989
1	1722	0.007446	0.011009	0.008837
2	2583	0.011609	0.017752	0.011701
3	3444	0.015772	0.025283	0.015767
4	4305	0.020729	0.030689	0.021164
5	5166	0.027391	0.038061	0.024245
6	6027	0.030992	0.047397	0.028376
7	6888	0.033694	0.056217	0.034170
8	7749	0.039546	0.063669	0.038341
9	8610	0.063843	0.142411	0.097010
10	9471	0.101733	0.131897	0.050059
11	10332	0.055754	0.087289	0.055004
12	11193	0.060140	0.095828	0.058108
13	12054	0.064919	0.105306	0.064727
14	12915	0.070649	0.113894	0.069401
15	13776	0.075452	0.189683	0.154376
16	14637	0.164120	0.144803	0.077662
17	15498	0.084378	0.140684	0.083666
18	16359	0.089585	0.146736	0.090764
19	17220	0.097602	0.156510	0.179302
20	18081	0.206388	0.223158	0.104837
21	18942	0.106500	0.174811	0.113414
22	19803	0.115207	0.185730	0.116529
23	20664	0.151325	0.386712	0.132147
24	21525	0.126123	0.202715	0.125700
25	22386	0.129037	0.206703	0.132413
26	23247	0.189478	0.403770	0.136608
27	24108	0.141193	0.224921	0.140566
28	24969	0.146409	0.238757	0.259205
29	25830	0.282257	0.251428	0.154282
30	26481	0.154199	0.260907	0.156329

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.092506	0.145491	0.088345



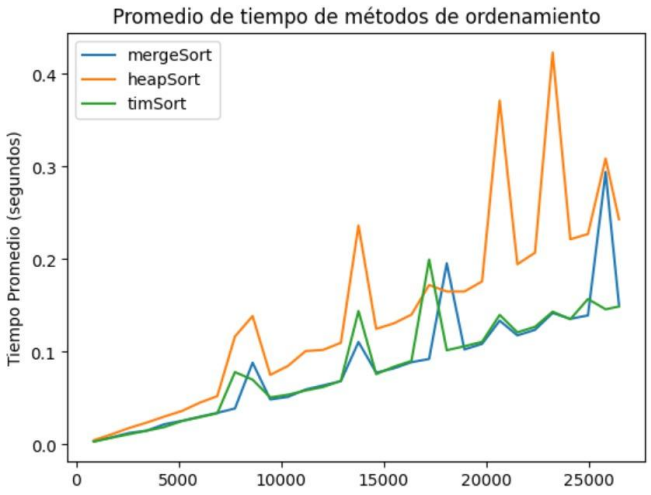
Iteración #5 - Computador #3: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

Tabla de Promedios:

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.003356	0.004587	0.003193
1	1722	0.007690	0.010975	0.007271
2	2583	0.012460	0.017804	0.011138
3	3444	0.014968	0.023675	0.014886
4	4305	0.021902	0.030198	0.018854
5	5166	0.025581	0.036268	0.025380
6	6027	0.029980	0.045107	0.029500
7	6888	0.034200	0.052296	0.033744
8	7749	0.038919	0.116664	0.078188
9	8610	0.088322	0.138664	0.070035
10	9471	0.048692	0.075185	0.050871
11	10332	0.051406	0.084821	0.053759
12	11193	0.059369	0.100839	0.058347
13	12054	0.063850	0.102219	0.062124
14	12915	0.068407	0.109813	0.068734
15	13776	0.110684	0.236354	0.144128
16	14637	0.077581	0.124858	0.075998
17	15498	0.082286	0.130794	0.083988
18	16359	0.088879	0.140238	0.090253
19	17220	0.092378	0.172221	0.199756
20	18081	0.195630	0.165170	0.101785
21	18942	0.102611	0.165173	0.106114
22	19803	0.108666	0.176109	0.110841
23	20664	0.133698	0.371320	0.139961
24	21525	0.117814	0.194596	0.121125
25	22386	0.123802	0.207138	0.127070
26	23247	0.142041	0.423163	0.143426
27	24108	0.135667	0.221596	0.135257
28	24969	0.139404	0.227280	0.157115
29	25830	0.294230	0.308776	0.146017
30	26481	0.150977	0.243164	0.148976

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.085982	0.143776	0.084446



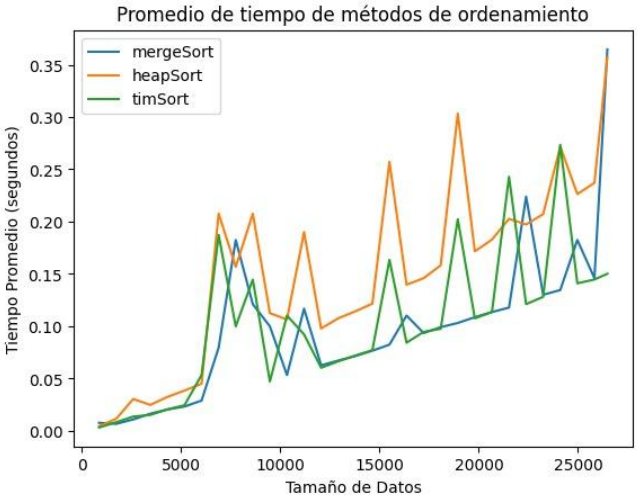
Iteración #6 - Computador #3: Tiempo promedio de ordenamiento respecto al tamaño de conjunto de datos y método de ordenamiento utilizado

Tabla de Promedios:

	Tamaño de Datos	mergeSort	heapSort	timSort
0	861	0.007639	0.004431	0.003146
1	1722	0.006646	0.011437	0.008184
2	2583	0.010839	0.030420	0.013626
3	3444	0.016377	0.024748	0.015234
4	4305	0.020364	0.032419	0.020439
5	5166	0.023213	0.038474	0.024483
6	6027	0.028898	0.045006	0.053327
7	6888	0.079489	0.207605	0.187330
8	7749	0.182436	0.156800	0.099844
9	8610	0.121077	0.207573	0.144618
10	9471	0.099885	0.112467	0.047149
11	10332	0.053291	0.106421	0.110207
12	11193	0.116772	0.190034	0.092142
13	12054	0.062630	0.097837	0.060266
14	12915	0.067159	0.107548	0.066437
15	13776	0.071464	0.114203	0.071930
16	14637	0.076573	0.121605	0.077158
17	15498	0.082390	0.257203	0.163482
18	16359	0.110200	0.139709	0.084222
19	17220	0.093466	0.145898	0.094591
20	18081	0.099111	0.158239	0.097551
21	18942	0.103094	0.303252	0.202258
22	19803	0.108636	0.171622	0.107542
23	20664	0.113538	0.182818	0.113542
24	21525	0.117818	0.202685	0.242856
25	22386	0.223923	0.197321	0.121089
26	23247	0.130080	0.207194	0.128138
27	24108	0.134569	0.271763	0.273465
28	24969	0.182473	0.226325	0.140999
29	25830	0.145935	0.237026	0.144684
30	26481	0.364660	0.357205	0.150123

Tabla de Promedios Globales:

	mergeSort	heapSort	timSort
Promedio Global	0.098537	0.150558	0.101938



Nuestros resultados experimentales revelaron que, de las seis iteraciones realizadas, en cinco de ellas *TimSort* demostró una eficiencia superior en comparación con *MergeSort* y *HeapSort*. Sin embargo, es importante destacar que *MergeSort* también exhibió un rendimiento muy eficaz, no quedando rezagado en términos de eficiencia.

Sara Lorena Duque Ramírez
Natalia Andrea García Ríos

5. Conclusiones del Experimento sobre Métodos de Ordenamiento en Clasificación Deportiva

1. Eficiencia y Rendimiento

Se observó que *TimSort* mostró una eficiencia superior en cinco de las seis iteraciones realizadas, seguido por *MergeSort*, que también exhibió un rendimiento eficaz. *HeapSort*, aunque se desempeñó de manera satisfactoria, no alcanzó la misma eficiencia que los otros dos métodos en la mayoría de los casos.

2. Estabilidad y Adaptabilidad

La elección de *MergeSort*, *HeapSort* y *TimSort* se basó en su estabilidad y escalabilidad en conjuntos de datos extensos. Estos métodos ofrecen adaptabilidad a diferentes tipos de datos numéricos, lo que los hace más apropiados para manejar la variabilidad inherente en los resultados deportivos.

3. Limitaciones de Otros Algoritmos

Se identificaron limitaciones en métodos como *InsertionSort*, *BubbleSort*, *BucketSort*, *RadixSort* y *CountingSort*, ya sea por su ineficiencia en grandes conjuntos de datos, dependencia de distribuciones uniformes o restricciones en la estructura específica de los números.

4. Complejidad Algorítmica

Se consideró la complejidad algorítmica como un factor clave, buscando métodos de ordenamiento con una complejidad moderada para asegurar la agilidad en el procesamiento de datos en tiempo real.

5. Aplicación Práctica en Sistemas de Clasificación Deportiva

Los resultados experimentales proporcionan una base sólida para seleccionar el método de ordenamiento más adecuado en sistemas de clasificación deportiva, optimizando tiempos de respuesta, precisión en la presentación de resultados y mejorando la experiencia de usuario.

6. Perspectivas de Mejora y Utilidad Práctica

La comparación y análisis de métodos de ordenamiento en este contexto ofrecen valiosas perspectivas para mejorar la eficiencia y capacidad de respuesta en sistemas de clasificación deportiva, beneficiando a organizaciones deportivas, aficionados y profesionales involucrados en el análisis de datos deportivos.

En resumen, los resultados del experimento resaltan a *TimSort* como el método de ordenamiento óptimo en sistemas de clasificación deportiva. Además, se cataloga como la mejor opción gracias a su estabilidad y capacidad para mantener una eficiencia consistente sin importar el tamaño de los datos (escalabilidad) y su relativa facilidad de implementación. Estas características convierten a TimSort en la elección preferida para gestionar conjuntos de datos variables (tanto en tipo como tamaño) y diversos en competiciones deportivas.

TimSort demostró una eficiencia sobresaliente y una notable adaptabilidad a conjuntos de datos extensos, manteniendo un rendimiento sólido. Aunque *MergeSort* también se mostró confiable y eficiente, la consistencia de *TimSort* fue evidente en la mayoría de los casos evaluados.

Por otro lado, si bien *HeapSort* mostró efectividad en el proceso de ordenamiento, su desempeño ligeramente inferior en comparación con *TimSort* y *MergeSort* lo coloca en una posición secundaria en este contexto específico. Estos descubrimientos proporcionan un marco fundamental para la implementación de métodos de ordenamiento en entornos deportivos, apuntando continuamente a mejorar la eficiencia y precisión en la clasificación de datos, donde *TimSort* destaca como una elección sólida y confiable para optimizar los sistemas de clasificación deportiva.