UNIVERSIDAD NACIONAL DE MOQUEGUA

Facultad de Ingeniería y Arquitectura

Escuela Profesional de Ingeniería de Sistemas e Informática

Algoritmos de ordenamiento y su eficiencia en los lenguajes de programación C++ y Python

Informe laboratorio N° 1

Estudiante:
Luis Angel Murillo Bernedo

Profesor:
Honorio Apaza Alanoca

8 de junio de 2023

Índice

1. Introducción **2**
1.1. Motivación y Contexto . . . 2
1.2. Objetivo general . . . 3
1.3. Objetivos específicos . . . 3
1.4. Justificación . . . 3

2. Marco teórico **4**
2.1. Antecedentes . . . 4
2.2. Marco conceptual . . . 5
2.2.1. C++ . . . 5
2.2.2. Python . . . 5

3. Materiales usados **6**

4. Ejercicios **7**

5. Resultados **8**
5.1. Bubble Sort . . . 8
5.2. Counting sort . . . 9
5.3. Selection sort . . . 10
5.4. Heap sort . . . 11

6. Conclusiones **12**

7. Recomendaciones **13**

Referencias **14**

1. Introducción

1.1. Motivación y Contexto

Desde el principio, el ordenamiento de datos ha sido vital para la computación por ser importante en la resolución de problemas de hace muchos años. Cuando se necesitó organizar la información para facilitar la búsqueda de información en los nacientes sistemas, el empleo de algoritmos que ordenaran una serie de datos fue indispensable para el avance en este rubro de la ciencia. Probablemente el algoritmo de ordenamiento más antiguo sea el Radix sort y se le atribuye su creación al genio informático Herman Hollerith por haber presentado en su momento un algoritmo muy similar.

Todo algoritmo presenta un problema en común con los demás y este es la eficiencia del mismo. Los métodos de ordenamiento no escapan de esta situación. Cada algoritmo tiene una forma diferente de ordenar valores, aunque pueden compartir conceptos y teorías. El problema de la eficiencia se puede dividir en dos claros referentes, los cuales son: tiempo de ejecución y uso de memoria.

Por un lado, el tiempo de ejecución, a nivel de código, puede variar de acuerdo con la manera en que ordenan los datos, pues uno es más rápido realizando la misma que otro dependiendo de la situación que se el presente. Cada algoritmo tiene escenarios ideales para su ejecución, es decir, que pueden funcionar muy rápido si se dan las condiciones óptimas. A nivel de hardware, el tiempo de ejecución de un algoritmo puede variar dependiendo de las características de la computadora en la cual es ejecutado. Si los componentes de esta son antiguos o de bajos recursos van influenciar para que el resultado se demore más tiempo en producirse. Y a su vez, si los componentes son de alto rendimiento y son modernos, repercutirán en favor de un tiempo de procesado más veloz. Otra vez, dependiendo del método empleado para ordenar, los algoritmos pueden sufrir retrasos en la creación de una respuesta si el medio, el cual emplean para realizar su trabajo, tiene un bajo rendimiento. Por ejemplo, si un algoritmo hace mucho uso de cálculos y poco de memoria, el procesador va a ser el determinante de que se demore en ejecutar menos o más tiempo, mientras que las unidades de memoria que intervienen no tendrán gran efecto en los mismo. En caso contrario de que el algoritmo haga principal uso de memoria para realizar el ordenamiento, el rendimiento del hardware de memoria va a ser el que más influya en su tiempo de ejecución.

Por otro lado, el uso de memoria es un problema que se presenta cuando el ordenador que se utiliza para realizar el trabajo no tiene grandes capacidades de almacenamiento para ser utilizado por el algoritmo. Frente a estas limitaciones, se debe encontrar la manera de que el algoritmo no necesite ocupar mucho espacio en memoria para llevar a cabo el ordenamiento.

1.2. Objetivo general

Evaluar la eficiencia de siete algoritmos de ordenamiento (bubble sort, counting sort, heap sort, insertion sort, merge sort, quick sort y selection sort) con respecto a su tiempo de ejecución en los lenguajes de programación Python y C++.

1.3. Objetivos específicos

- Comparar el tiempo de procesamiento de cada algoritmo implementado en cada lenguaje de programación.
- Probar cada uno de los alogritmos indicados en la práctica.
- Generar un informe de laboratorio sobre los resultados de la eficiencia de los algoritmos.

1.4. Justificación

El presente informe se hace a razón de evidenciar la eficiencia de los algoritmos planteados para ordenar arreglos numéricos de diferentes tamaños, pudiendo determinar en qué situaciones es preferible el uso de uno u otro de los algoritmos, es decir, ver con qué volúmenes de datos se debería trabajar cada algoritmo de ordenamiento, dependiendo de la necesidad que un programa tenga para resolver un problema específico que sea en favor de la sociedad, como la finalidad que la carrera profesional de Ingeniería de Sistemas e Informática busca cumplir. Asimismo, este trabajo contribuirá al propio entendimiento de los algoritmos mediante el análisis de los mismos, siendo a lo que hace referencia el nombre del curso Análisis y Diseño de Algoritmos.

2. Marco teórico

2.1. Antecedentes

Antes de ejecutar el proyecto, se consultaron distintas fuentes para tener un mejor entendimiento del tema, de acuerdo con las preguntas que surgieron al tener muy poco conocimiento de los algoritmos de ordenamiento dados en la práctica. En ese sentido, se encontraron cinco trabajos académicos que guardan fuerte relación con el realizado y expuesto en este informe.

En primer lugar, el trabajo especial de grado que título Un algoritmo de ordenamiento sobre multiconjuntos para optar el título de licenciado en computación por el bachiller Ricardo Cattafi en el año 2001 quien compara un algoritmo al cual denomina Rain sort con otros, los cuales son Quick sort y Bubble sort en una distribución de datos Multiconjuntos. Se realiza un análisis comparativo relacionado al número de iteraciones [2], planteando la hipótesis de que, en el peor de los casos, la complejidad se asemejará a Bubble sort, pero en el mejor de ellos será superior. Al final se llega a la conclusión de que las características de la distribución empleada que se parece a la lluvia permiten que Rain sort sea más eficiente que Quick sort en un entorno específico, pero en uno normal se encuentra, en eficiencia, entre los dos bien conocidos algoritmos de ordenamiento. La hipótesis se cumple en el peor de los casos donde presenta la misma eficiencia de Bubble sort.

En segundo lugar, el trabajo realizado por Octavio Alberto Agustín Aquino en el año 2005 se titula Algoritmos de ordenamiento describe los métodos de ordenamiento por comparaciones y por conteo, con sus más grandes ejemplares. El lenguaje que se muestra en su explicación es el C, representándose también en funciones y sumatorias. Se concluye diciendo lo interesante que es su análisis matemático y computacional [1], así como su importancia en el actual incremento de volúmenes de datos y su manejo.

En tercer lugar, está la investigación de Amalia Duch publicada el año 2007 con el título de Análisis de algoritmos. Se usa el término de coste para analizar el tiempo de ejecución y la memoria utilizada. Se usa notación matemática para representar el tal coste de los algoritmos según su tipo iterativo o recursivo en el peor, promedio y mejor de los casos [3].

En cuarto lugar, el artículo científico que lleva por título Algoritmo de ordenamiento por comparaciones Heapinsert sort fue publicado el 2010 que propone un algoritmo de ordenamiento que fusiona Heap sort e Insert sort en uno solo, ayudando el primero en la eficiencia del segundo. Se utiliza el lenguaje C para estructurar los datos y comparar el denominado Heapinsert sort con el Insert sort. Se corren las pruebas en una computadora de 64 bits con procesador AMD Athlon™ a 2.00 GHz y 1 GB de memoria sobre una versión 2002 del sistema operativo Windows XP [5]. Los resultados muestran una ventaja del doble del algoritmo presentado con Insertion sort.

Por último, otro artículo científico que lleva el nombre Comparación entre algoritmos de ordenamiento paralelizados en Java del año 2019 por Diego Gómez García y José Luis Alonzo Velázquez, donde se analiza cómo rinden diferentes algoritmos de ordenamiento que son paralelizados para la prueba en una máquina virtual de Java [4]. Todo lleva a que

la conclusión sea que todos los algoritmos exceptuando Odd Even sort reciben un beneficio en el aumento de núcleos.

2.2. Marco conceptual

2.2.1. C++

Es un lenguaje de bajo nivel, multiparadigma que permite emplear diferentes métodos de programación en la realización de un proyecto, por lo que es muy versátil en su uso para muchos tipos de tareas. Constituye una extensión de C para incluir el nuevo paradigma orientado a objetos. Su mejor cualidad es su buen rendimiento, al tener una comunicación directa con el sistema operativo y ser compatible con lenguaje ensamblador. Sirve mucho para crear funciones libres y presenta una organización muy buena orientada a eso. Este lenguaje puede ser usado para crear desde simples programas hasta sistemas operativos.

2.2.2. Python

Es a diferencia del anterior, un lenguaje de alto nivel que simplifica la escritura de código para programas sencillos, pero resulta tedioso su uso para programar código y estructuras complejas que requieren de una comunicación precisa con el sistema operativo. Su uso es más común para el desarrollo de aplicaciones web, data science, machine learning y software en general. Los programadores prefieren su uso por ser fácil de aprender, muy eficiente en la mayoría de las tareas y multiplataforma. Python es de uso gratuito y se integra fácilmente a sistemas de todo tipo, disminuyendo comúnmente el tiempo que se toma en desarrollar un proyecto.

3. Materiales usados

- Para el proyecto se utilizó mobiliario de escritorio para la redacción del informe y la prueba de los algoritmos frente a diferentes volúmenes de datos.
- Se hizo necesario un cargador para los momentos en que el nivel de la batería estuviera cercana a acabarse.
- Se contó para la ejecución de los diferentes algoritmos con una computadora portátil de características importantes para resaltar, por la naturaleza de la investigación:
    - Procesador: Core i3 de cuarta generación con cuatro núcleos a una velocidad máxima de 1.7 GHz.
    - Memoria RAM: 12 GB con tecnología de generación DDR3.
    - Unidad de estado sólido: capacidad máxima de 256 GB.

4. Ejercicios

1. Primero, se investigó trabajos académicos anteriores y similares al presente. Se identificaron cinco autores e hizo un resumen a grandes rasgos el contenido de sus investigaciones para conocer lo importante que se tiene que saber de cada uno.

2. Primero, se investigó trabajos académicos anteriores y similares al presente. Se identificaron cinco autores e hizo un resumen a grandes rasgos el contenido de sus investigaciones para conocer lo importante que se tiene que saber de cada uno.

3. Se pusieron a prueba siete algoritmos de ordenamiento:

    - Bubble sort
    - Counting sort
    - Heap sort
    - Insertion sort
    - Merge sort
    - Quick sort
    - Selection sort

4. Se comparó el tiempo de procesamiento de cada algoritmo en Python y C++.

5. Se comparó el tiempo de ejecución de todos los algoritmos en su respectivo lenguaje de programación.

5. Resultados

5.1. Bubble Sort

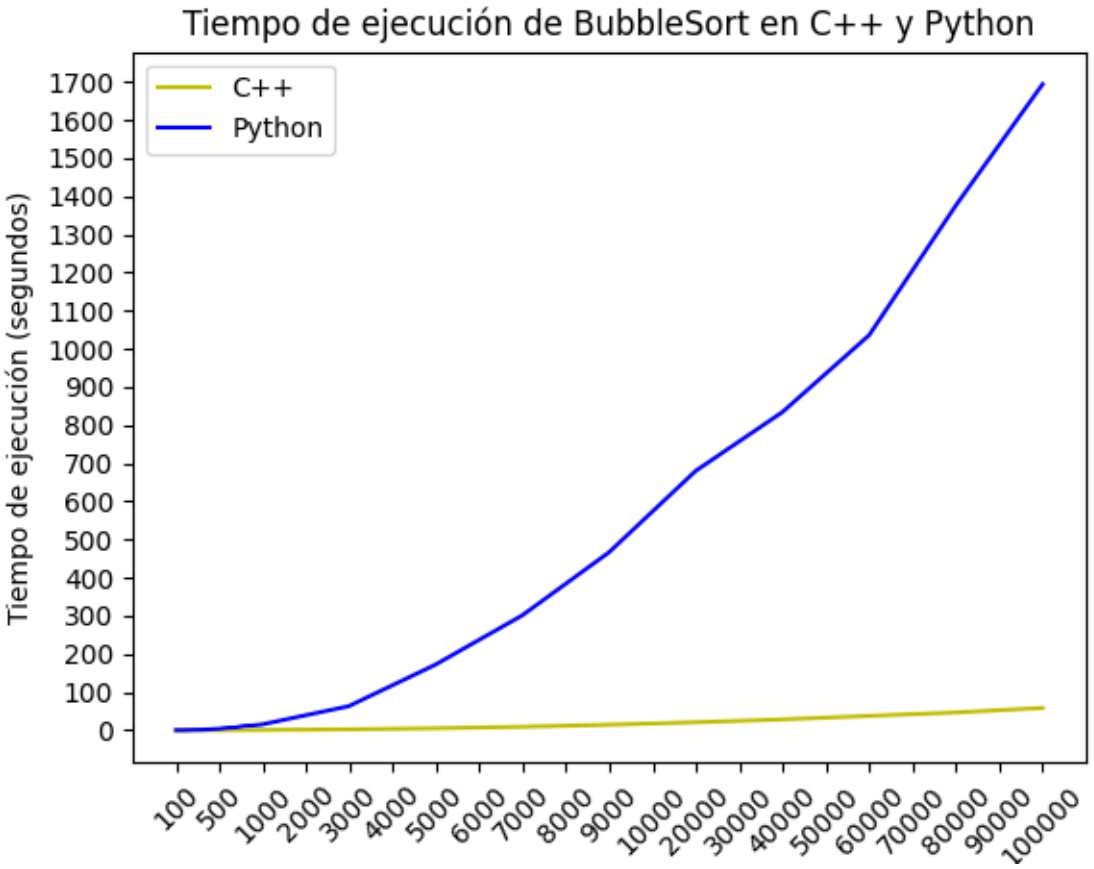

Figura 1: Comparación de Bubble sort en los lenguajes de programación C++ y Python.

5.2. Counting sort

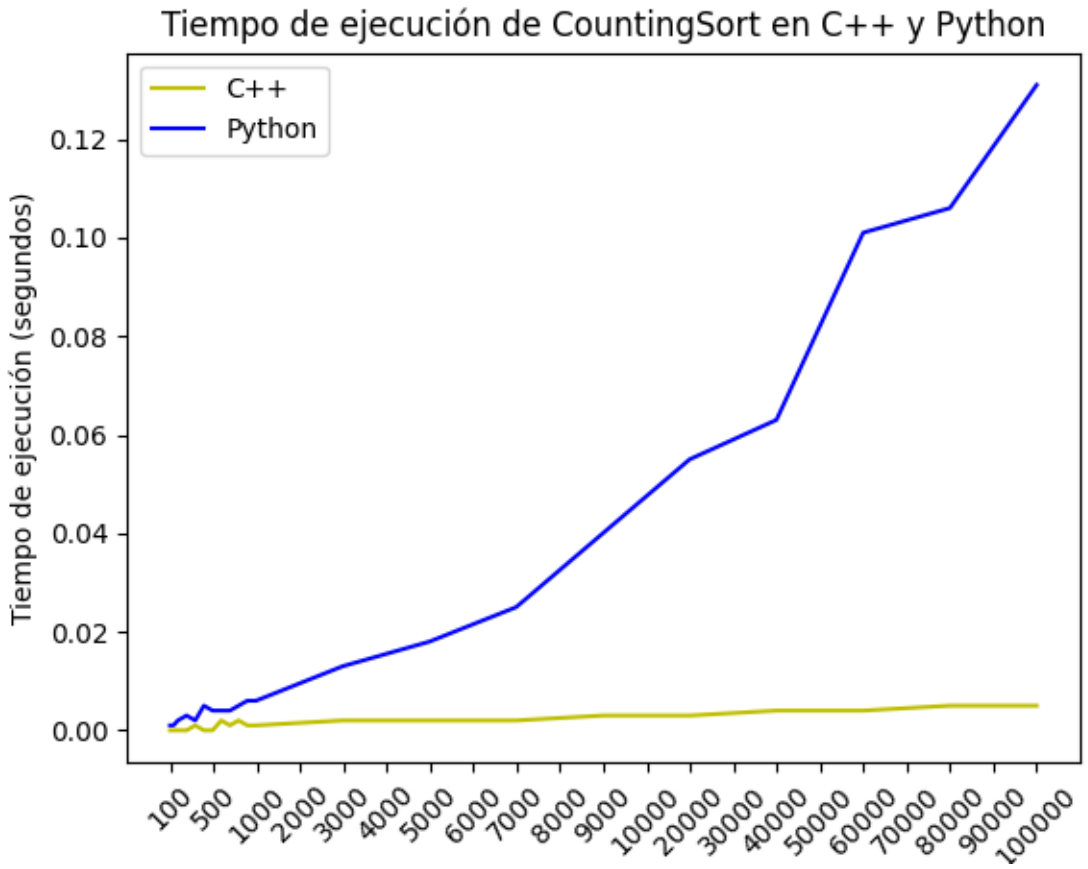

Figura 2: Comparación de Counting sort en los lenguajes de programación C++ y Python.

5.3. Selection sort

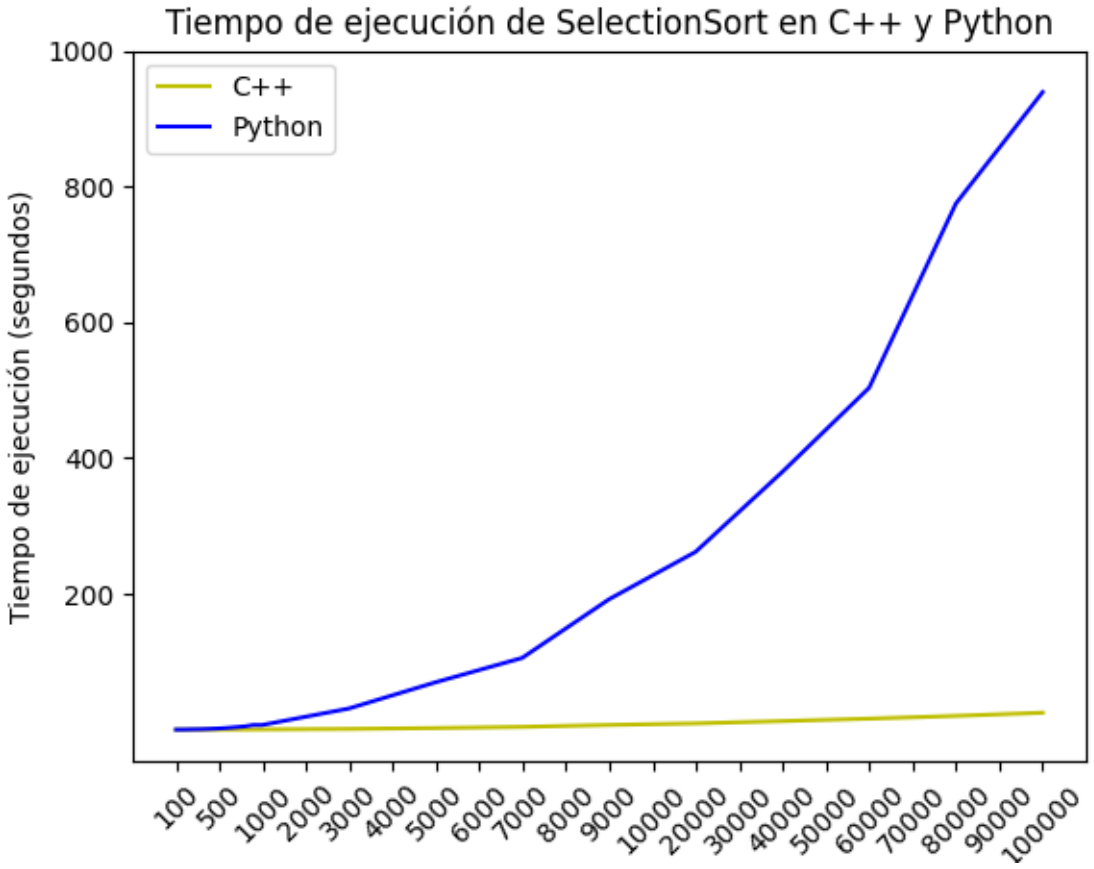

Figura 3: Comparación de Selection sort en los lenguajes de programación C++ y Python.

5.4. Heap sort

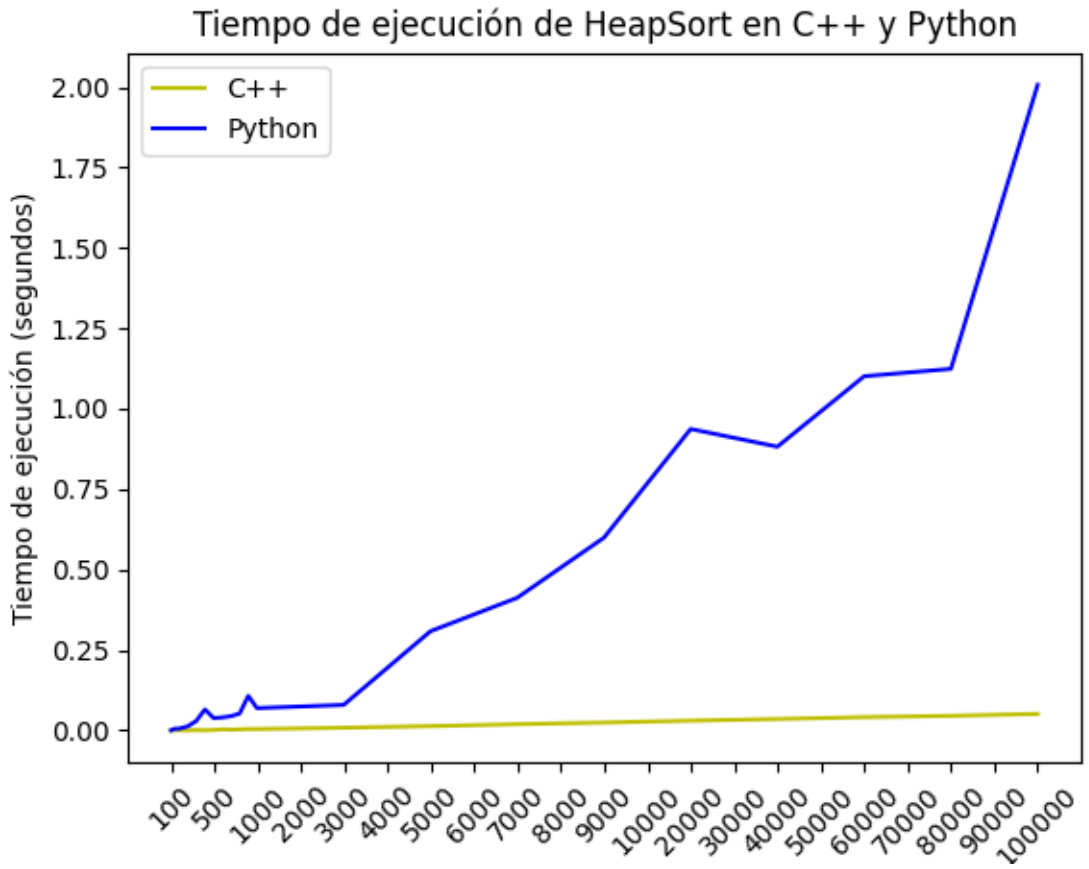

Figura 4: Comparación de Heap sort en los lenguajes de programación C++ y Python.

6. Conclusiones

1. Coclusión numero uno : El lenguaje de programación Python tiene una tendencia a demorar más que C++ en la ejecución de todos los siete algoritmos probados y considerados en esta práctica.

2. Coclusión numero dos: Unos algoritmos son más eficientes que otros al momento de realizar la tarea de ordenar series de datos numéricos, teniendo en muhos casos un crecimiento exponencial del tiempo empleado para cada volumen de datos.

3. Coclusión numero tres : El presente informe permite dar a conocer los resultados de una prueba de la eficencia de siete algoritmos de ordenamiento, representando su comportamiento en diferentes escenarios por medio de gráficas que brindan una clara visión de los diferentes resultados y a su vez permite compararlos

7. Recomendaciones

1. Recomendación numero uno: Para casos como los que se presentan en la practica se sugeriría trabajar con IDEs sofisticadas que permitan un desarrollo del proyecto más productivo y rápido.

2. Recomendación numero dos: Se recomendaría trabajar en una computadora de alto rendimiento para que, a la hora de poner a prueba los algoritmos, estos no tomen demasiado tiempo en ejecutarse y así conseguir un resultado quesirva de insumo para las gráficas.

3. Recomendación numero tres: Se debería trabajar el documento del informe con el programa LaTeX para lograr un nivel alto de calidad tipográfico y de presentación. El uso de este sistema se ha habituado estos últimos años y es un cambio positivo en cuanto a la producción de artículos científicos y, en sí, toda producción académica que se haga con completa seriedad y compromiso.

Referencias

[1] Octavio Alberto Agustın Aquino. Algoritmos de ordenamiento.

[2] Trabajo Especial de Grado. *Un Algoritmo de Ordenamiento Sobre Multiconjuntos*. PhD thesis, Universidad de Carabobo.

[3] Amalia Duch. Análisis de algoritmos. *Barcelona, Universidad Politécnica de Barcelona*, 2007.

[4] Diego Gomez Garcia and José Luis Alonzo Velázquez. Comparación entre algoritmos de ordenamiento paralelizados en java (comparison between sorting algorithms parallelized in java). *Pistas Educativas*, 41(133), 2019.

[5] Hugo Humberto Morales Peña, Angel Augusto Agudelo, and Jorge Ivan Rios Patiño. Algoritmo de ordenamiento por comparaciones heapinsert sort. *Scientia et technica*, 17(46):68–73, 2010.