Tarea 2. Analisis de Algoritmos

LUIS DANIEL ROA GONZÁLEZ, A01021960, Campus Santa Fe
KATIA YARETH BELLIDO LÓPEZ, A01023638, Campus Santa Fe
CONSTANZA GÓMEZ SÁNCHEZ, A01026717, Campus Santa Fe
CHRISTOPHER LUIS MIRANDA VANEGAS, A01022676, Campus Santa Fe
MIGUEL MONTERRUBIO BANDERA, A01022153, Campus Santa Fe

Additional Key Words and Phrases: árboles binarios, AVL, AB, inserción, búsqueda, eliminación, nodos, ramas, hojas

ACM Reference Format:
Luis Daniel Roa González, Katia Yareth Bellido López, Constanza Gómez Sánchez, Christopher Luis Miranda Vanegas, and Miguel Monterrubio Bandera. 2020. Tarea 2. Analisis de Algoritmos. 1, 1, Article Análisis de Algoritmos (March 2020), 10 pages. https://doi.org/10.1145/1122445.1122456

1 RESUMEN

Los árboles son una estructura de almacenamiento de datos que cuentan con una mayor optimización para inserción, búsqueda y eliminación. En este documento buscamos comparar los tiempos en los cuales se realizan los métodos anteriormente mencionados en dos tipos de árbol: Arbol *AVL*, un tipo de árbol binario autobalanceable que utiliza la memoria interna (RAM), y el Arbol *B*, un árbol que almacena los datos en bloques y es más eficiente cuando se usa el Disco Duro.

Para realizar el análisis, compararemos el tiempo que toma la creación de los dos tipos de árboles con 10, 100, 1 000, 10 000, 100 000 y 1 000 000 de elementos, así como la búsqueda y eliminación de 10 valores en cada uno de los casos. Usaremos la librería chrono de c++ para medir el tiempo que tarda cada uno de estos casos.

2 INTRODUCCIÓN

En la ciencia computacional se define a un árbol como "`un conjunto de nodos y líneas`", donde los nodos son valores que se encuentran dentro del árbol y las líneas son pares de nodos conectados. En los árboles existen diferentes tipos de nodos, el nodo principal, que se conoce como "`raíz`", la cual no tiene líneas de entrada solo de salida; los nodos padre e hijos, que pueden tener o no, más nodos; y los nodos hoja, que es el caso de que un nodo no tenga ningún subárbol.

Existen diferentes tipos de árboles binarios que cuentan con diferentes métodos, como la inserción, la búsqueda ordenada y la eliminación de valores dentro de los mismos árboles. Los árboles que destacaremos en este proyecto

Authors' addresses: Luis Daniel Roa González, A01021960, Tecnológico de Monterrey, Campus Santa Fe; Katia Yareth Bellido López, A01023638, Tecnológico de Monterrey, Campus Santa Fe; Constanza Gómez Sánchez, A01026717, Tecnológico de Monterrey, Campus Santa Fe; Christopher Luis Miranda Vanegas, A01022676, Tecnológico de Monterrey, Campus Santa Fe; Miguel Monterrubio Bandera, A01022153, Tecnológico de Monterrey, Campus Santa Fe.

XXXX-XXXX/2020/3-ARTAnálisis de Algoritmos $15.00
https://doi.org/10.1145/1122445.1122456

serán los tipo *AVL* y tipo *B*. Ambos árboles se destacan por mantener un buen orden y por ser buenos en sus métodos al implementarse de cierta forma.

En el árbol *AVL* se asume que todo funcionará en memoria interna y será mejor cuando se cumpla el factor de balanceo. El factor de balanceo implica que la diferencia de altura entre los subárboles derecho e izquierdo no puede ser mayor a un nodo; y esto aplica para todos los nodos del árbol. De acuerdo con *Geeks for Geeks*, la altura del árbol *AVL* siempre tendrá una cota de *O(Logn)*, donde 'n' es la cantidad de nodos en el árbol. Para que los árboles *AVL* cumplan su factor de balanceo, existen métodos que se deben aplicar al insertar y eliminar valores; estos métodos se conocen como rotaciones simples o rotaciones dobles, ya sea a la izquierda o a la derecha.

En el árbol *B*, a diferencia del *AVL*, funciona mejor el realizar búsqueda de memoria en el disco; esto significa que los nodos se deberán guardar en archivos. En este tipo de árbol es importante tener en cuenta la altura, pues no queremos que sea mucha; entre menos altura tenga este tipo de árbol mejor será para los métodos de búsqueda, inserción y eliminación, debido a la reducción de accesos al disco. Las características más importantes de este tipo de árbol es que cada nodo debe tener un mínimo de 'n' valores (según lo que el programador desee) y el doble como máximo, además sabemos que la cantidad de hijos que tendrá un nodo será "2n+1". Así como los demás árboles de búsqueda, el *B* cumplirá una cota de *O(Logn)*, la única diferencia va a ser que un árbol *B* crecerá hacia arriba desde la raíz.

3 DESARROLLO

A continuación analizaremos como estos códigos, el árbol *B* y el árbol *AVL*, se implementaron y corrieron, tanto en el Raspberry Pi como en la computadora de la persona que se encargó de correrlo. Se analizarán los tiempos que se tarda cada uno en realizar las operaciones de insertar, buscar y eliminar al implementar la libreria de *chrono* y *ctime*, las cuales pertenecen al estándar de C++ desde el 2011.

Para poder correrlas en el Raspberry Pi, se le ingresaron los datos de una cuenta de VNC y se controló de manera remota.

Al correr los códigos, se compilaron y, posteriormente, corrieron como se esperaría de cualquier otro programa de *C++*. Para asegurarnos que estuviera corriendo en el estándar especificado, se le añadió **-std=c++17**. Por ende, las librerías que no estuvieran dentro de ese estándar no se permitirían en el funcionamiento de nuestros programas.

Para que exista un mejor análisis, los datos del Raspberry Pi se encontrarán en la tabla 1, mientras que los datos de la computadora que se utilizó para comprobar y comparar el funcionamiento, se encontraran en la tabla 2.

Características	Descripción
Modelo	Raspberry Pi 3 B+
Sistema Operativo	Raspbian Buster
Memoria RAM	1GB
Procesador	Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
Memoria (SD)	128GB

Table 1. Raspberry Pi 3 B+

Características	Descripción
Modelo	Inspiron 5567
Sistema Operativo	Windows 10 Home versión 1909
Memoria RAM	16GB
Procesador	Intel Core i7-7500U CPU @ 2.70GHz 2.90GHz
Memoria (HDD)	2TB

Table 2. Dell Inspiron

3.1 Árbol B

Durante la ejecución del código **ArbolB.cpp** se obtuvieron los tiempos de los tres métodos para cada uno de los casos, tanto en el Raspberry Pi, como en la Dell, nos sorprendió ver que hay operaciones que son mas rápidas en el Raspberry Pi que en la Dell. A continuación, se podrán ver las tablas de tiempos de ambos dispositivos. Es importante recalcar que los tiempos dentro del código están en microsegundos, pero los que estaremos enseñando aquí ya fueron convertidos a milisegundos. Los tiempos de ejecución del Raspberry Pi se encuentran en la tabla 3.

Cantidad	Insert	Search	Delete
10	.857	.747	.669
100	6.388	.789	.891
1,000	70.575	.799	.640
10,000	1,240.728	.691	.667
100,000	14,595.55	.858	.690
1,000,000	123,549.825	3.553	2.699

Table 3. Tiempos Árbol B, Raspberry Pi

Como podemos notar, los tiempos de búsqueda y de eliminación se mantienen consistentes dentro de los tiempos del Raspberry Pi.

A continuación se puede ver la tabla con los tiempos de la computadora Dell:

Cantidad	Insert	Search	Delete
10	.967	.432	.634
100	1.420	.408	.295
1,000	209.411	.371	2.634
10,000	250.528	.379	.554
100,000	2,353.433	.393	.273
1,000,000	28,100.979	.338	.267

Table 4. Tiempos Árbol B, Dell

Al observar los tiempos de ejecución de estas operaciones dentro de la Dell, tenemos que recordar que están ocurriendo mas procesos de fondo, debido a que viene precargada con drivers específicos para que corra. Tomando esto en cuenta, los tiempos de la computadora Dell fueron mas veloces respecto a las operaciones de búsqueda y de eliminación, pero esto es debido a la mayor capacidad que posee esta misma.

Así mismo, la comparación de tiempos entre estas dos computadoras es muy diferente debido a las especificaciones técnicas que tiene cada una de estas. Puede que si se hubiera usado una Raspberry Pi 4, los tiempos de la Raspberry Pi no hubieran sido tan altos, pero esto es debido a la mejora en la memoria RAM de los modelos nuevos.

3.2 Árbol AVL

Durante la ejecución del código **ArbolAVL.cpp** se obtuvieron los tiempos finales de los tres métodos para cada uno de los casos, tanto en el Raspberry Pi, como en la Dell. Aunque había una gran posibilidad que los tiempos en el Raspberry Pi fueran más lentos que los tiempos en la Dell, fue sorprendente observar que hay operaciones que demostraron lo contrario. A continuación, se podrán ver las tablas de tiempos de ambos dispositivos. Es importante recalcar que los tiempos dentro del código están en microsegundos, pero los que estaremos enseñando en este documento ya fueron convertidos a milisegundos. Los tiempos de ejecución del Raspberry Pi se encuentran en la tabla 5; mientras que los tiempos de ejecución de la DELL se encuentra en la tabla 6.

Cantidad	Insert	Search	Delete
10	0.681	0.702	0.693
100	6.688	0.690	0.684
1,000	300.031	4.360	3.9
10,000	10,257.027	0.822	0.809
100,000	1,660,184.625	0.821	0.886
1,000,000	149,667,549.746	N/A	N/A

Table 5. Tiempos Árbol AVL, Raspberry Pi

Se puede observar como los métodos aumentan sus tiempos debido al incremento de población en los árboles, sin embargo hay diferentes casos, como en el de la población de 1,000; en el que los tiempos finales se ven afectados significativamente, permitiéndonos deducir que el código tuvo un número difícil (parte del peor de los casos) para los métodos; tanto en el Raspberry Pi como en la DELL.

Por otro lado el tiempo que le tomó ejecutar el método de inserción para una cantidad de 1,000,000 de números es un tiempo aproximado. Esto se debe a que nuestro código no fue capaz de insertar esta cantidad de números en el árbol cuando lo corrimos en la Raspberry Pi, y nos vimos obligados a detener el código. Es por esto que no pudimos determinar el tiempo que le tomaría al árbol *AVL* ejecutar los métodos de buscar y eliminar en el árbol de 1,000,000.

Cantidad	Insert	Search	Delete
10	0.244	0.732	0.684
100	3.821	3.242	0.369
1,000	264.249	0.632	0.399
10,000	15,064.104	10.278	0.876
100,000	1,660,184.625	0.821	0.886
1,000,000	21,419,563.035	0.556	0.537

Table 6. Tiempos Árbol AVL, Dell

Al igual que en el Árbol *B*, debemos tomar en cuenta que hay diferentes factores además del código que pueden afectar los tiempos, como las características que tienen las computadoras como su memoria RAM.

3.3 Comparaciones de Eliminar y Búsqueda

Las siguientes tablas y gráficas demuestran la comparación de los tiempos que le toma a cada árbol en ejecutar el método de eliminar. Este método fue de una población de 10 números aleatorios, representados del 1 al 10; y el tiempo se tomó en milisegundos.

	Delete 10 AVL-Pi	Delete 10 B-Pi
1	0	0,241
2	0,417	0,557
3	0,863	1,112
4	1,319	1,675
5	1,762	1,99
6	3,383	2,272
7	3,847	2,859
8	4,317	3,149
9	4,77	3,7
10	5,232	4,272

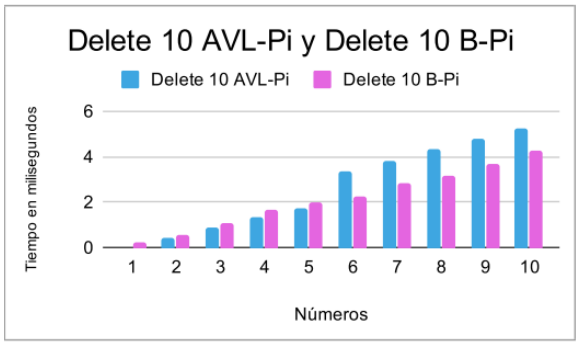

Fig. 1. Árbol AVL vs árbol AB, eliminación de 10 números

	Delete 100 AVL-Pi	Delete 100 B-Pi
1	0	0,287
2	0,43	1,04
3	0,879	2,49
4	1,34	3,11
5	1,799	3,402
6	2,255	3,983
7	2,723	4,539
8	3,174	4,84
9	3,628	5,118
10	4,08	5,397

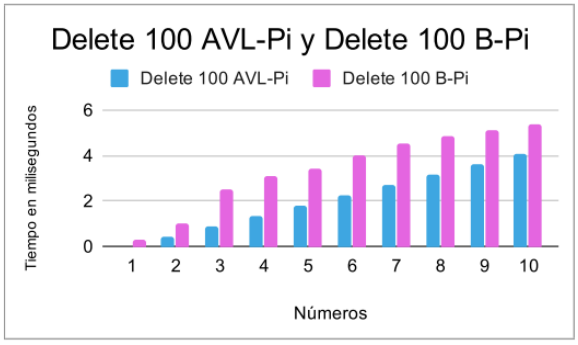

Fig. 2. Árbol AVL vs árbol AB, eliminación de 100 números

	Delete 1,000 AVL-Pi	Delete 1,000 B-Pi
1	0,001	0,001
2	0,041	0,127
3	0,079	0,187
4	0,12	0,245
5	0,158	0,321
6	0,196	0,424
7	0,233	0,504
8	0,27	0,596
9	0,308	0,648
10	0,346	0,694

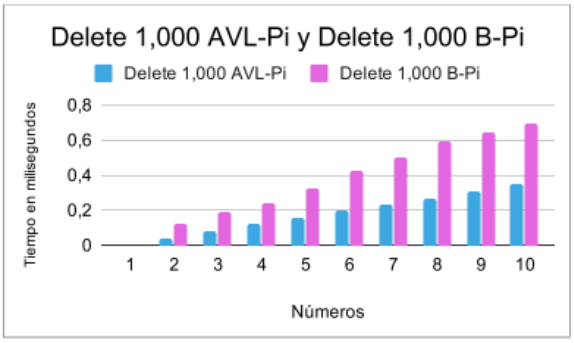

Fig. 3. Árbol AVL vs árbol AB, eliminación de 1,000 números

	Delete 10,000 AVL-Pi	Delete 10,000 B-Pi
1	0	0,002
2	0,061	0,16
3	0,118	0,223
4	0,171	0,272
5	0,223	0,398
6	0,28	0,464
7	0,337	0,571
8	0,39	0,63
9	0,467	0,727
10	0,521	0,808

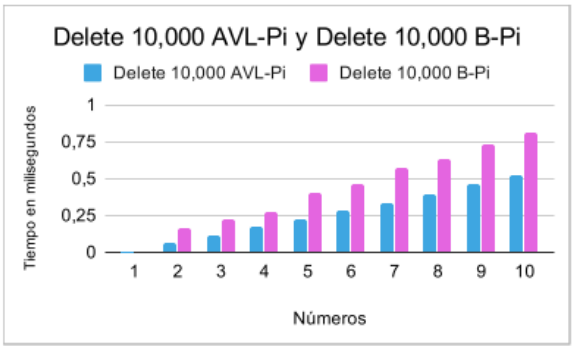

Fig. 4. Árbol AVL vs árbol AB, eliminación de 10,000 números

	Delete 100,000 AVL-Pi	Delete 100,000 B-Pi
1	0	0,048
2	0,06	0,105
3	0,114	0,168
4	0,167	0,264
5	0,225	0,376
6	0,279	0,424
7	0,337	0,471
8	0,406	0,563
9	0,465	0,614
10	0,52	0,66

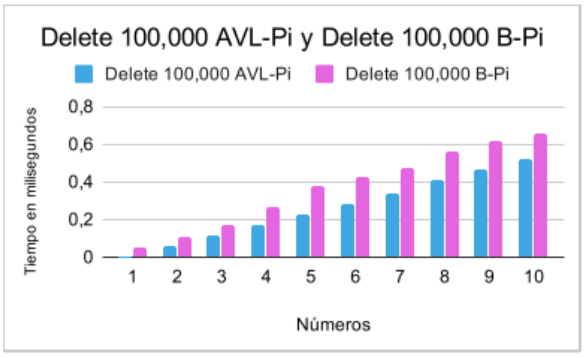

Fig. 5. Árbol AVL vs árbol AB, eliminación de 100,000 números

A continuación, el otro método que comparamos fue el de buscar. Al igual que en las tablas anteiores usamos una población de 10 números aleatorios, representados del 1 al 10; y el tiempo se tomó en milisegundos.

	Buscar 10 AVL-PI	Search 10 B-PI
1	0	0,004
2	0,559	0,29
3	1,025	0,63
4	1,534	0,934
5	2,062	1,305
6	4,452	1,619
7	4,936	1,894
8	5,505	2,174
9	5,959	2,495
10	6,415	2,798

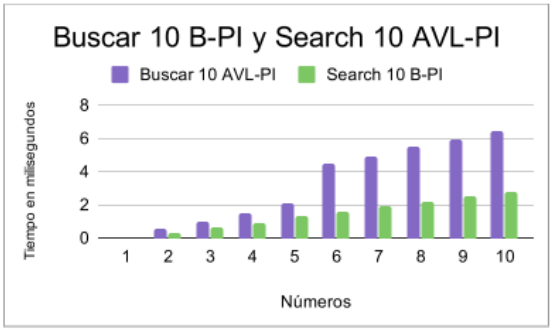

Fig. 6. Árbol AVL vs árbol AB, búsqueda de 10 números

	Buscar 100 AVL-PI	Search 100 B-PI
1	0,001	0,012
2	0,392	0,28
3	0,941	0,575
4	1,465	0,866
5	1,927	1,151
6	2,398	1,456
7	2,84	1,8
8	3,391	2,142
9	3,862	2,416
10	4,311	3,822

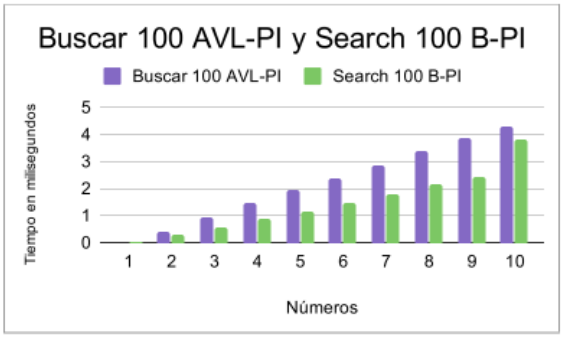

Fig. 7. Árbol AVL vs árbol AB, búsqueda de 100 números

	Buscar 1,000 AVL-PI	Search 1,000 B-PI
1	0,001	0,005
2	0,043	0,065
3	0,082	0,119
4	0,121	0,175
5	0,159	0,228
6	0,197	0,311
7	0,234	0,37
8	0,271	0,429
9	0,369	0,483
10	0,348	0,536

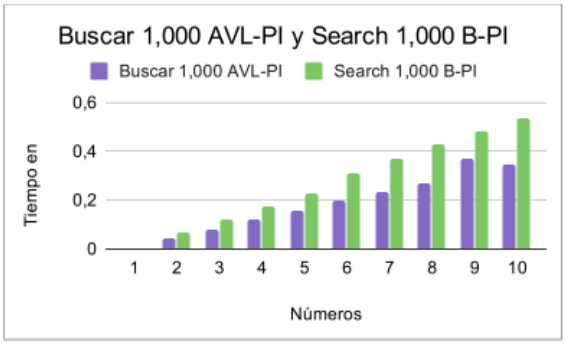

Fig. 8. Árbol AVL vs árbol AB, búsqueda de 1,000 números

	Buscar 10,000 AVL-PI	Search 10,000 B-PI
1	0	0,017
2	0,063	0,1
3	0,121	0,169
4	0,174	0,235
5	0,227	0,302
6	0,302	0,369
7	0,361	0,436
8	0,413	0,501
9	0,471	0,568
10	0,523	0,635

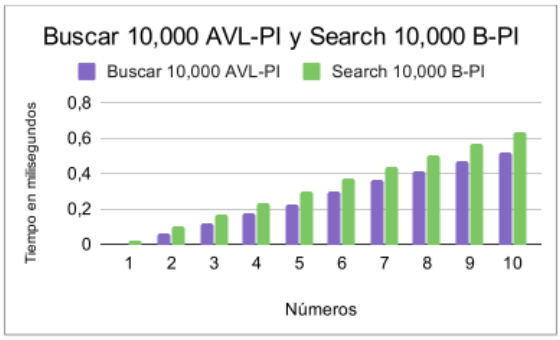

Fig. 9. Árbol AVL vs árbol AB, búsqueda de 10,000 números

	Buscar 100,000 AVL-PI	Search 100,000 B-PI
1	0,001	0,01
2	0,064	0,076
3	0,119	0,132
4	0,174	0,189
5	0,233	0,249
6	0,299	0,304
7	0,383	0,359
8	0,442	0,417
9	0,496	0,477
10	0,549	0,535

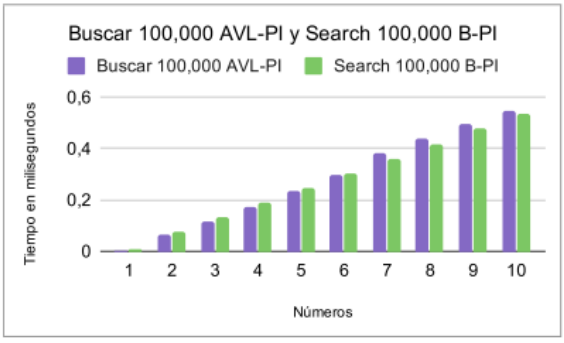

Fig. 10. Árbol AVL vs árbol AB, búsqueda de 100,000 números

Por último, podemos observar la tabla y la gráfica de comparación del tiempo que le tomó a cada árbol ejecutar el método de inserción, al igual que en las comparaciones anteriores, el tiempo se tomó en milisegundos.

Hay que tener en consideración que, como fue mencionado previamente, el tiempo que le tomó al árbol*AVL* en insertar 1,000,000 de númersos es un tiempo aproximado.

	Insertar AVL-Pi	Insertar B-Pi
10	0,681	0,857
100	6,688	6,388
1000	300,031	70,575
10000	10257,027	1240,728
100000	1660184,625	14595,55
1000000	149667549,746	123549,825

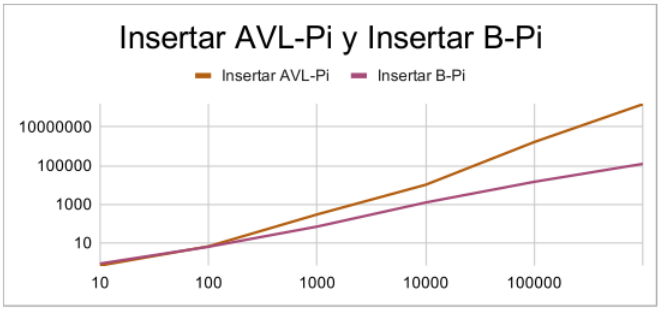

Fig. 11. Árbol AVL vs árbol AB, método insertar

Debido a que el tiempo que le tomo al árbol *AVL* es mucho mayor que el del árbol *B*, en el momento de graficarla los rangos variaban tanto que no nos permitian hacer una gráfica para observar la comparación del método en cada caso. Es por esto que los valores del tiempo en la gráfica fueron convertidos a escala logarítmica, y de esta manera podemos observar mejor la diferencia en los valores.

4 CONCLUSIONES

Al insertar los valores se puede ver que es más eficiente el árbol *B*, en especial al tratarse de datasets más grandes. Esto ocurre gracias a que un árbol binario termina con muchos más niveles, por lo que la inserción en árboles de gran tamaño debe recorrer, en el peor de los casos, todos estos niveles. Además, después de cada inserción y eliminación, el árbol debe analizar si debe rebalancearse.

Como se puede ver en las tablas anteriores, la búsqueda y la eliminación en general fueron más eficientes en el árbol *B*. Sin embargo, al trabajar con datasets más grandes, empiezan a ser más eficientes en el árbol *AVL*. Esto se debe al método que usa el árbol *B*, que almacena varios números en una misma "hoja", y al recorrer el árbol puede llegar a ser un proceso más lento.

Por otro lado, debemos considerar que el haber usado un muestreo de números aleatorios para el método de buscar y eliminar, no nos permite conocer con exactitud que árbol es mas eficaz. Esto se debe a que la busqueda de un número puede llegar a ser el mejor de los casos para el árbol *AVL* y el peor para el árbol *B*, o viceversa. Por ejemplo, un número se puede encontrar en el primer nivel del árbol *AVL* y puede estar en un nivel mayor en el árbol *B*, por esto nos va a dar como resultado que el árbol *AVL* busca y/o elimina un número en menor tiempo que el árbol *B*.

5 REFERENCIAS

[1] Clark F. (2020). Balanced binary search trees. Washington Recuperado de: 'http://courses.washington.edu/css502/zander/Notes/06AVL-Btree.pdf

[2] SA. (2020). ÁRBOLES. 20/03/2020, UAEH. Recuperado de: Recuperado de:https://www.uaeh.edu.mx/docencia/P_Presentaciones/icbi/asignatura/Cap6ARBOLES.pdf
[3] GeeksForGeeks. (2020). AVL Tree | Set 1 (Insertion). 20 de Marzo 2020, de GeeksForGeeks. Recuperado de: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/
[4] GeeksForGeeks. (2020).Introduction of B-Tree . 20 de Marzo 2020, de GeeksForGeeks. Recuperado de: https://www.geeksforgeeks.org/introduction-of-b-tree-2/