Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación
CC3001 – Algoritmos y Estructuras de Datos

Comparación de Árboles Balanceados

TAREA 5

Nombre Alumno : Gabriel Iturra Bocaz

Correo : gabrieliturrab@ug.uchile.cl

Profesor : Patricio Poblete

Profesor Auxiliar : Manuel Cáceres
Sergio Peñafiel

Fecha : 6 de Junio de 2016

Santiago, Chile.

Índice

I. Introducción

El siguiente informe detalla el trabajo el desarrollo de la Tarea 5 del curso de Algoritmos y Estructuras de Datos (CC3001). Como quinta tarea del curso, se pretende que el alumno se interiorice con la implementación del TDA Árboles de Búsqueda Binaria y la de los Árboles Balanceados (que también son Árboles Binario de Búsqueda).

El problema propuesto es implementar tres clases de Árboles de Búsqueda Binaria, que son, la implementación usual que sigue los conceptos básicos y la de los Árboles Balanceados, AVL y los Rojo Negro, todos bajo una interface llamada ArbolBinario, con dos métodos obligatorios dentro de la interface void insertar(int x) y boolean buscar(int x), comparando cuál de ellas es la implementación más eficientes (Costo de tiempo de Ejecución) para distintos casos.

Como estrategia de solución al desafío se utilizó los conceptos de la materia de cómo funcionan los métodos buscar(int x) e insertar(int x), y también el material visto en clases auxiliares que fue subido a material docentes para la elaboración del código de las clases pedidas. En cuanto al análisis de cada una de las implementaciones, se estudió distintos escenarios que eran tuplas (n,d), donde $n \in \{20000, 40000, \ldots, 200000\}$ y $d \in \{AZAR, ASC\}$ y se midió tres indicadores que fueron: Tiempo promedio de Inserción, Altura de los Árboles al terminar las inserciones y el Tiempo promedio de Búsqueda.

II. Análisis del Problema

II.1. Descripción del Problema

Un Árbol de Búsqueda de Binaria (ABB) es una estructura de datos que permiten guardar llaves y realizar búsquedas dentro de él. Su complejidad promedio en tiempo de búsqueda e inserción es O(log n). Sin embargo, dependiendo como los datos son insertados un ABB puede degenerarse en una lista enlazada en el peor caso, provocando que los costos de tiempo sean O(n).

Para enfrentar este problema, existe otro tipo de ABB llamado Árboles Balanceados que se encargan de "regular" las inserciones de las llaves, preocupando que la cantidad de nodos sea los más similar posible en cada subárbol, lo que garantiza los costos de tiempo sean O(log) incluso en el peor caso.

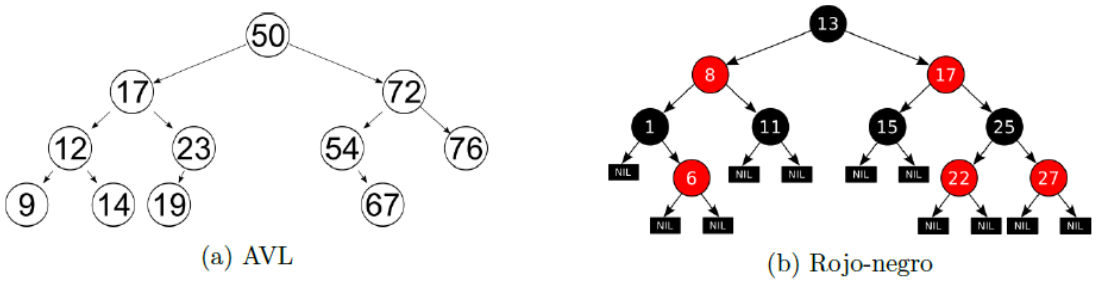
Los Árboles Balanceados que se consideraron para la tarea son:

- AVL: Es un tipo de ABB tal que cada vez que un nodo es insertado (de la misma forma que un ABB), se revisa que se cumpla la condición | altura(n.izq) – altura(n.der) | < = 1 para cado uno de los n nodos del árbol. Sí se viola la condición, esta se restaurada mediante rotaciones (Simples o Dobles).
- Rojo-Negro: Es un tipo ABB, donde los nodos tiene un bit extra llamado *color*. Cualquier nodo insertado es de color *Rojo*. Tras la inserción deben cumplirse las siguientes propiedades:

    1. La raíz del árbol es *negra.*
    2. Todas las hojas son *negra.*
    3. Sí un nodo es *rojo*, sus hijos deben ser *negros.*
    4. Para todo nodo, todos los caminos desde él hasta las hojas tienen la misma cantidad de nodos *negros.*

***En caso de fallar, los nodos son reparados vías inserciones y cambios de color*.

En la figura 1, se aprecia un ejemplo de los árboles explicados.



(a) AVL (b) Rojo-negro

Figura 1: Árboles Balanceados.

II.2. Implementación

Como se mencionó anteriormente, el trabajo consiste en la implementación de tres clases: ABB normal, ABB AVL y ABB Rojo-Negro que implementen la siguiente interfaz que se puede apreciar en la figura 2.

```
1 public interface ArbolBinario{
2   public void insertar(int x);
3   public boolean buscar(int x);
4 }
```

Figura 2: Interface ArbolBinario.

La interface genera una especie de "contrato" hacia las clases que son implementadas a ella donde los métodos de la interface deben ser obligatoriamente implementados dentro de la clase.

II.3. Experimentación

Las implementaciones fueron comparadas en distintos escenarios, que consistieron en tuplas (n,d), donde:

n ∈ {20000, 40000, 60000, 80000, 100000, 120000, 140000, 160000, 180000, 200000}

d ∈ {AZAR, ASCENDENTE}

Realizando las comparaciones en 3 indicadores, luego de las inserciones para cada tupla, que son:

- Altura final del Árbol, medida al terminar las inserciones.
- Tiempo promedio de inserción. Donde se toma el tiempo total de inserción para algún n y se divide por dicho n.
- Tiempo promedio de búsqueda. Donde se toma el tiempo total de búsqueda para algún n y se divide por 3000, que fueron el total de búsqueda para todos los casos.

II.4. Supuesto Importantes

Como principales consideraciones y supuestos dentro del trabajo de desarrollado, es importante mencionar:

- Los elementos insertados dentro de los árboles binarios son números enteros mayores o iguales a 1.
- Algunas de las implementaciones están basadas en el trabajo visto en auxiliar, como el caso del ABB AVL y ABB RojoNegro que fue visto el método de inserción en el auxiliar 6 y fragmentos de la inserción en el auxiliar 7 respectivamente.
- En la experimentación los valores n fueron alterados por indicación de los auxiliares dando rangos de libertad a cada alumno.
- La medición de tiempo fue hecha, con el método *System.currentTimeMillis()* por lo que la unidad de tiempo usada es el milisegundo.
- Para los indicadores de búsqueda, como los números a buscar dentro de cada árbol debían ser elegidos al azar, se amplió el intervalo de números que podían ser elegidos al azar, sí por cada n elementos insertados, el intervalo de números elegidos podía estar entre 0 a 1.5*n, debido a que para tener un análisis más real de tiempo promedio se consideraron números que no pudiesen estar dentro del árbol realizando así búsqueda infructuosas. Para cada escenario el número de búsquedas a realizar fue 0.1*n.
- Se extrajo un pequeño código de internet que generaba números aleatorios (tomando el supuesto que estos no se repetían) de una matriz resultados, y luego eran insertados a los árboles usando los métodos de las clases.

III. Solución al Problema

III.1. Desarrollo

La implementación de cada una de las clases es muy similar en cuanto a que los métodos que debían seguir las mismas funciones, para evitar extensión del informe, en esta sección se explican las diferencias existentes en cada clase de árbol binario de búsqueda (sin redundar tanto en el código de cada una).

Como se dijo en la sección anterior las clases deben estar bajo un interface con dos métodos obligatorios (boolean buscar(int x) e void insertar(int x)), a continuación se explica el detalle de cada clase.

III.2. Clase ArbolABB

Está posee la implementación usual de un Árbol Binario de Búsqueda Binaria, posee una clase Nodo usual donde se guarda un llave tipo entera info y dos referencia a los nodos a los hijos izquierdo izq y derecho der. Tiene un nodo cabeza raíz donde se implementan sus métodos, solo cuenta con los métodos pedidos por la interface, que se explican a continuación:

- boolean buscar(int x): este método recibe un elemento tipo int, sí ese elemento se encuentra dentro de la raíz, retorna true, que indica que elemento fue encontrado, sí el elemento es mayor al elemento de la raíz, este es buscado recursivamente en el subárbol derecho, retornando true sí lo encuentra y false si no; sí el elemento es menor se realiza mismo análisis pero en el subárbol izquierdo de la raíz, finalmente sí el elemento no es encontrado se retorna false. En la figura 3, se aprecia un fragmento del código.

```
private boolean buscar(int x, Nodo nodo){
    if (nodo.info == x){
        return true;
    }
    else if (nodo.info > x){
        if (nodo.izq != null){
            return buscar(x, nodo.izq);
        }
        else{
            return false;
        }
    }
```

Figura 3: Método buscar

- void insertar(int x): inserta nuevo elemento al árbol binario, sí la raíz es nula construye un nodo que se convierte en la nueva raíz (con el elemento dentro de él), sí el árbol no ésta vacío tiene dos caso, sí el nuevo elemento es mayor al elemento dentro de la raíz se recorre recursivamente el subárbol derecho hasta llegar a una referencia nula y se insertar, sino se sigue buscando recursivamente hasta encontrar su lugar, sí el elemento es menor a la raíz se aplica la misma recursión pero en el subárbol izquierdo. En la figura 4, se aplica un fragmento del código.

```
private Nodo insertar(int x, Nodo nodo){
    Nodo nuevo = new Nodo(x);
    if (nodo == null){
        nodo = nuevo;
    }
    else if (nodo.info > x){
        if (nodo.izq == null){
            nodo.izq = nuevo;
        }
        else{
            nodo.izq = insertar(x, nodo.izq);
        }
```

Figura 4: Método de Inserción en ABB

III.3. Clase ArbolAVL

En cuanto a su contrucción es similar a la clase ABB, sin embargo dentro de los nodos posee un bit extra llamada altura, que dice la altura del nodo respectivo. A pesar de cumplir una condición adicional que regula el balance de las inserciones, sigue siendo un ABB, por lo que método de búsqueda es el mismo que el método boolean buscar(int x) de la clase ArbolABB, la diferencia radica en el método de insercion:

- void insetar(int x): este método insertar de la misma forma que en un ABB, pero la diferencia radica que al insertarse un nuevo elemento puede perderse la condición de AVL, esto se regula con el método int altura(Nodo nodo) que devuelve la altura de un nodo (y 0 si el nodo tiene nulo, para que JAVA no devuelva la exception java.lang.NullPointerException), sí la diferencia de las altura de los subárboles de un nodo es mayor a 1 se ha perdido la condición de AVL, por lo estos nodos deben volver a la condiciones de AVL mediante rotaciones que puede ser simple o doubles y de izquierda o derecha. En la figura 5, se aprecia esta diferencia en la inserción.

```
if (altura(nuevo.izq) - altura(nuevo.der) == 2){
    if (nuevo.izq.info > x){
        nuevo = rotacionSimpleIzq(nuevo);
    }
    else{
        nuevo = rotacionDobleIzq(nuevo);
    }
```

Figura 5: Condición AVL

III.4. Clase ArbolRN

La clase Árbol Rojo Negro fue un poco más compleja de implementar que la clase AVL, debido a que posee cuatro condiciones importantes, que fueron mencionadas al principio del informe. La clase Nodo posee una referencia a su nodo padre (cuando se explica el método de inserción se menciona el porqué) y otro bit extra llamado color (definido por el alumno boolean negro = false y boolean rojo = true).

La inserción de un nuevo elemento puede desestabilizar cualquiera de las condiciones, como el Árbol Rojo Negro sigue siendo un ABB su método de búsqueda es lo mismo que en las dos clases anteriores, su mayor diferencia radica en el método de inserción:

- void insertar(int x): Este método inserta de la misma forma que en un ABB, donde le da un color rojo al nuevo nodo insertado, excepto a la raíz que por la propiedad 1 debe ser negra. Luego de insertar el nuevo nodo este puede no cumplir con las propiedades 2, 3 o 4, por lo que pasa se llama a un método void recuperarCondiciones(Nodo nodo) para que se vuelva al estado Rojo-Negro. Generalmente las propiedades 2, 3 o 4 se violan por los casos siguientes:

    - Caso 1: Cuando se inserta un nuevo nodo, y su padre y tío son nodo rojos no nulos, (ya que se viola propiedad 3) para recuperar la condición los nodos padre y tío son recoloreados a negro y los nodos abuelo y tío-abuelo pintados rojos, sí esto afecta a los nodo de más arriba se sigue recoloreando hasta llegar a la raíz que por propiedad 1 es negra. En el figura 6 se aprecia dicha condición.

```
if (nodo.padre == nodo.padre.padre.izq){
    Nodo tio = nodo.padre.padre.der;
    if (tio != null && tio.color == rojo){
        nodo.padre.color = negro;
        tio.color = negro;
        nodo.padre.padre.color = rojo;
        nodo = nodo.padre.padre;
```

Figura 6: Recoloración.

    - Caso 2: Sí el tío del nodo insertado es de color negro o nulo, en este caso es necesario realizar rotaciones, donde estas pueden ser rotaciones simples o dobles, dependiendo si las inserción es “hacia fuera” o “hacia dentro” (como se explica en el apunte para el caso de los AVL), en la figura 7 se ve un fragmento de este caso.

```
        if (nodo == nodo.padre.der){
            nodo = nodo.padre;
            rotacionIzq(nodo);
        }
    nodo.padre.color = negro;
    nodo.padre.padre.color = rojo;
    rotacionDer(nodo.padre.padre);
    }
```

Figura 7: Rotaciones y Recoloracion.

****Las rotaciones de un Árbol Rojo Negro son similares a las de un árbol AVL, salvo que como los nodos poseen una referencia al padre, estas deben cambiar evaluando algunos casos.*

III.5. Fase de Experimentación

Para esta fase había que ponerse en dos valores d, uno en que los valores eran insertados ascendentemente y otro al azar, para el primero se utilizó un ciclo for para insertar los datos variando los en los distintos valores de n y el segundo se extrajo un pequeño código de internet donde se generaba un matriz "resultados" con números aleatorios en un intervalo que era elegido por el alumno y variaba en cada escenario. En la sección siguiente se explica en detalle cómo fue la toma de datos y la evaluación de los indicadores.

IV. Resultados y Conclusiones

En esta sección se explica el detalle cómo se midieron los indicadores y las conclusiones desprendidas en base a los a datos obtenidos.

IV.1.1 Caso de inserción de Elementos Ascendentes.

Mediante un ciclo for, se fueron insertando los valores hasta completar todos los escenarios de n. En la tabla 1 podemos ver los datos obtenidos para cada uno de los árboles y en los gráficos 1 y 2 las comparaciones entre las tres clases.

N° de Llaves insertadas	Altura Árbol BB	Altura Árbol AVL	Altura Árbol RN
20.000	20.000	15	24
40.000	40.000	16	28
60.000	60.000	16	29
80.000	80.000	17	30
100.000	100.000	17	31
120.000	120.000	17	31
140.000	140.000	18	32
160.000	160.000	18	32
180.000	180.000	18	32
200.000	200.000	18	32

Tabla N1: Tabulación altura de los árboles.

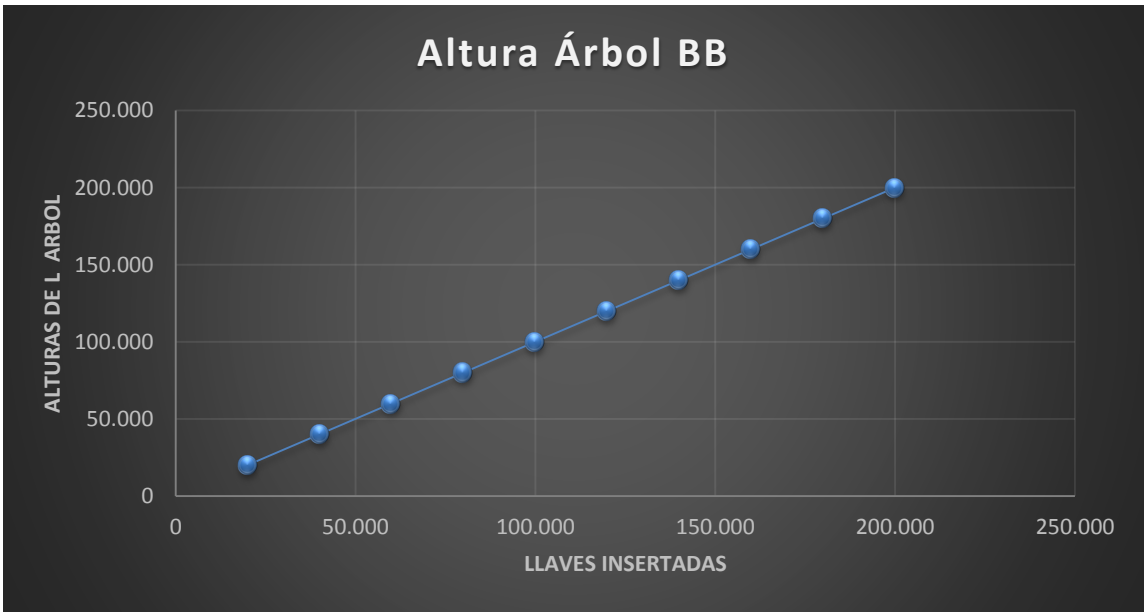


Gráfico 1: Altura obtenidas en los ABB.

****Como las alturas de los ABB es muy grande en comparación a los AVL y ABB, se hicieron dos gráficos para apreciar mejor los datos.*

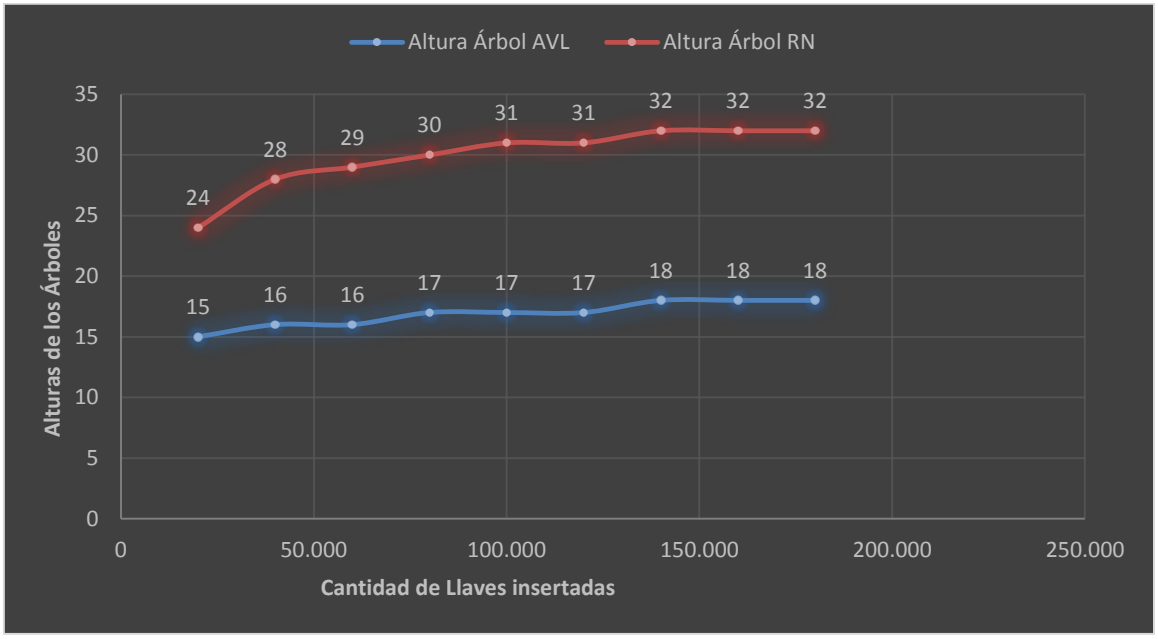


Gráfico 2: Alturas de los árboles rojo y negro.

Como la inserción de datos ascendentemente consiste en peor caso para los ABB (degenera el árbol a una lista enlazada) la altura de sus árboles es igual a la cantidad de elementos insertados, en cambio para los AVL y los ARN, se estabilizan las alturas cuando se alcanza la inserción de 140.000 llaves con 18 para los AVL y 32 para los ARN, lo que muestra una eficiencia en la implementación de los AVL y ARN por sobre los ABB en este caso, ya que las alturas de los ABB son increíblemente mayores a los AVL y ARN.

IV.1.2 Tiempo Promedio de Inserción para llaves ascendentes

Para calcular el tiempo promedio de inserción, se tomó el tiempo que se realiza por cada inserción para los distintos valores de n, y se divide es tiempo n. La unidad de tiempo son milisegundos (ms). En la tabla 2, se presentan los datos tabulados, y en el gráfico 3 las comparaciones de los tiempos promedios de las 3 clases.

N° de Llaves insertadas	T. P. Árbol BB	T. P. Árbol AVL	T. P. Árbol RN
20.000	0,23505	0,00080	0,00050
40.000	0,48297	0,00060	0,00035
60.000	0,72261	0,00055	0,00028
80.000	1,01631	0,00053	0,00028
100.000	1,13117	0,00047	0,00025
120.000	1,59570	0,00044	0,00025
140.000	1,90228	0,00045	0,00024
160.000	2,16179	0,00040	0,00023
180.000	2,51171	0,00040	0,00022

200.000	2,76598	0,0003	0,000165

Tabla 2: Tiempo Promedio de inserción de datos.

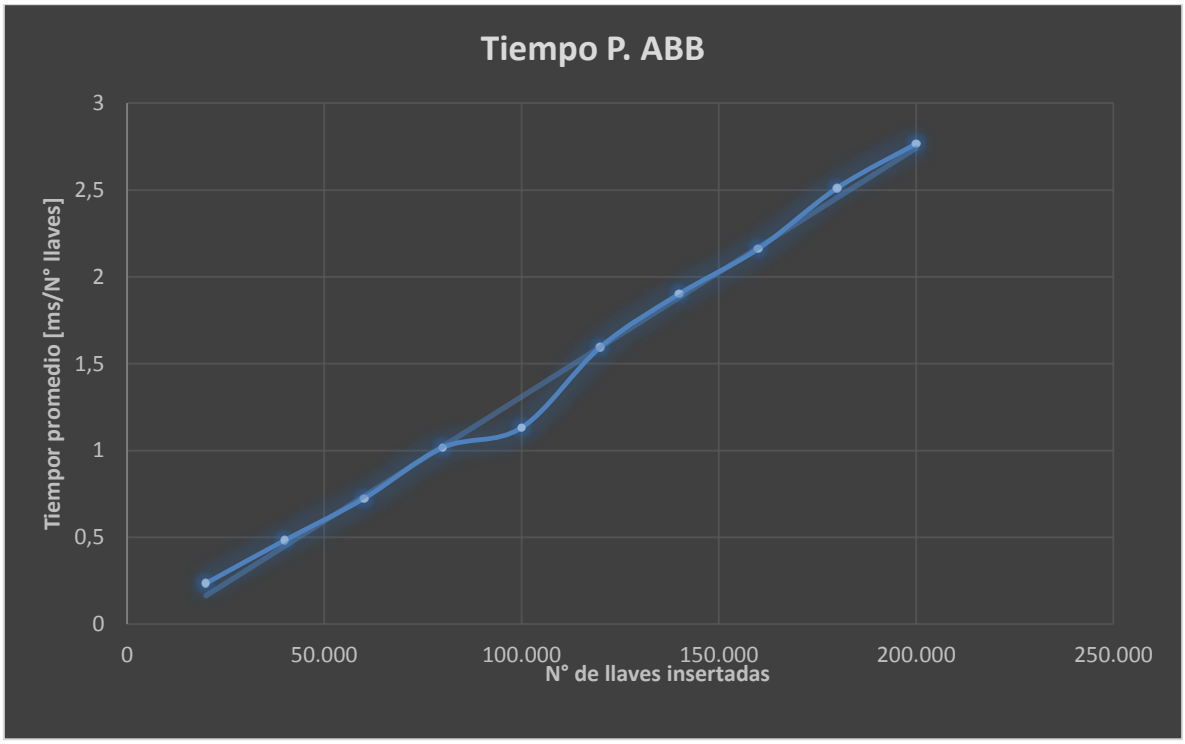


Gráfico 3: Tiempo de inserción ABB.

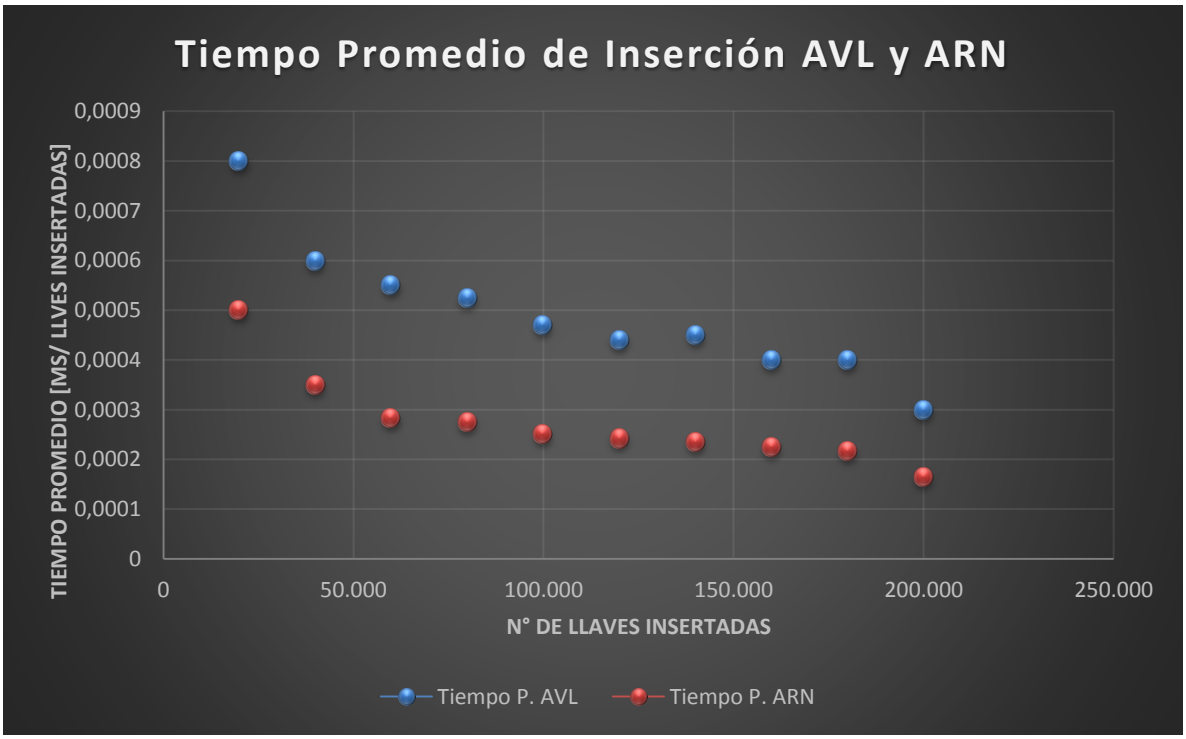


Gráfico 4: Tiempo Promedio de Inserción AVL y ARN

Para el tiempo promedio de inserción, se la misma conducta para los ABB una tendencia lineal, en cambio para los AVL y ARN esta tendencia tiende a 0, por lo que al ser mayor la cantidad de elementos a insertar el tiempo promedio disminuye, no así para el caso del ABB que este aumenta, dejando al descubierto el peor caso para los ABB, lo confirman que demoran un tiempo O(n) para este caso y no O(log n) como lo hacen los AVL y ARN.

IV.1.3 Tiempo promedio de Búsqueda para llaves ascendente

Para el indicador del tiempo promedio, se toma un n que fueron insertados al largo, el rango de números que podían ser elegidos al azar para buscar estaba entre 0 y n*1.5, pero solo se realizan n*0.1 búsquedas del rango de elementos al azar, en el gráfico 5, se puede ver el comportamiento del tiempo promedio de búsqueda para los ABB, el comportamiento de los AVL y ARN se puede apreciar en el gráfico 6.

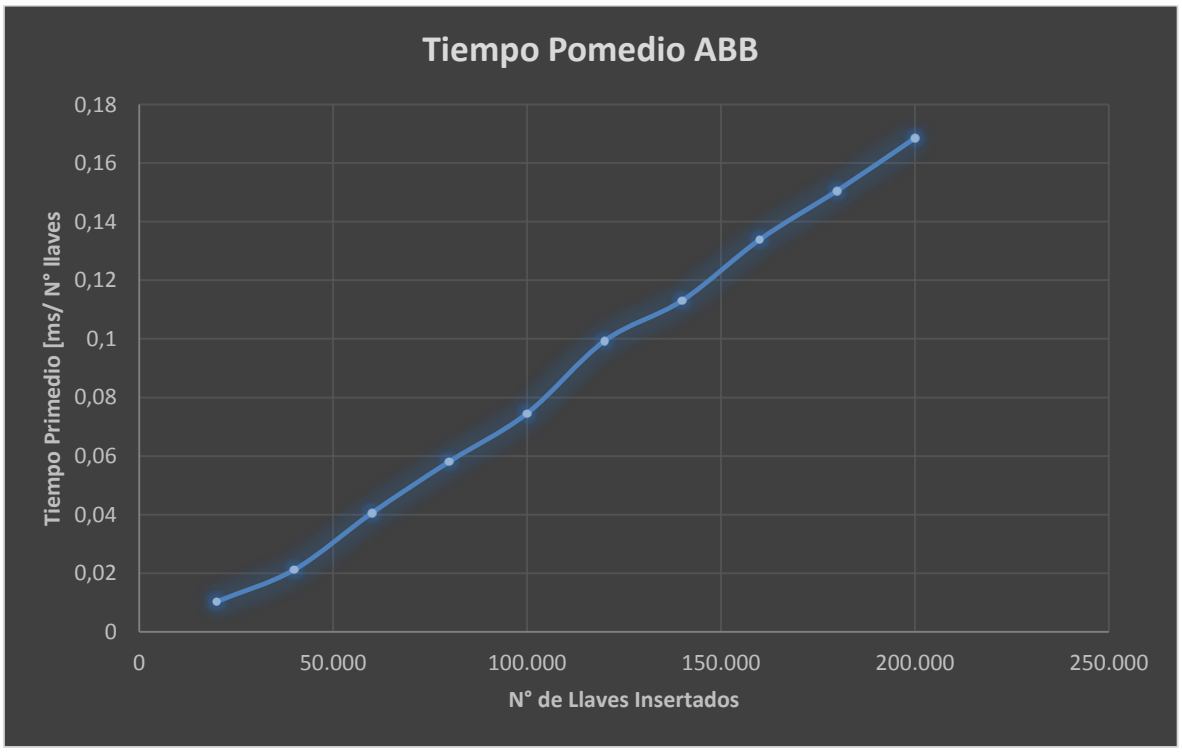


Gráfico 5: Tiempo promedio de búsqueda en un árbol ABB.

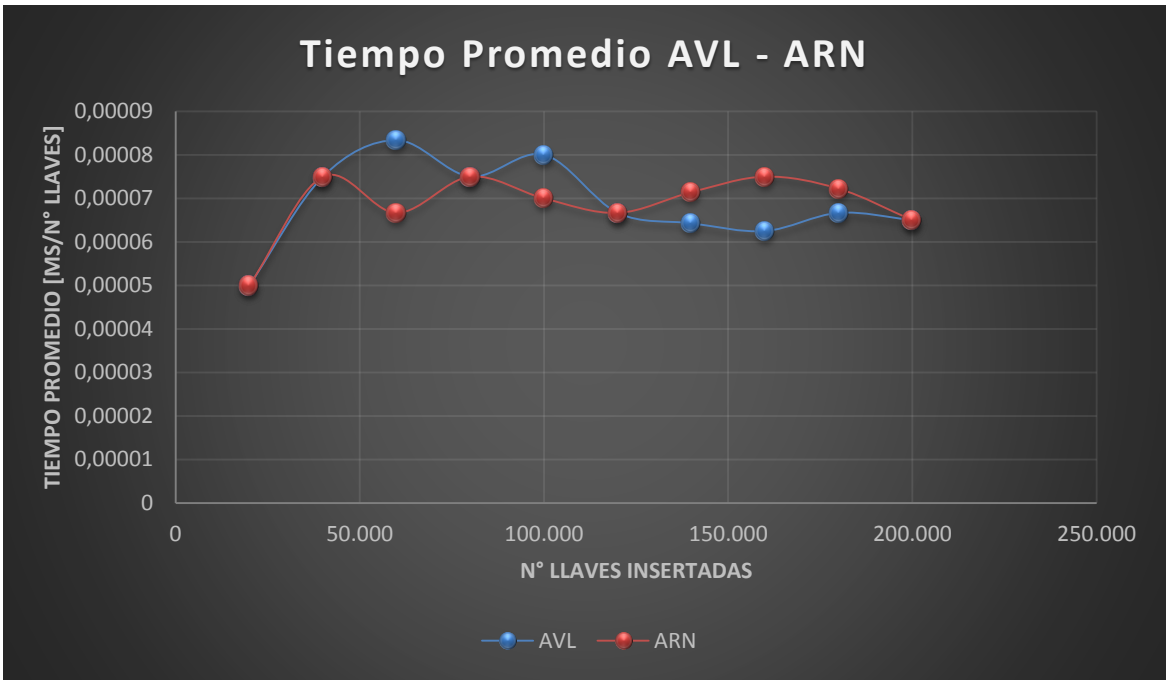


Gráfico 6: Tiempo promedio de búsqueda para árboles AVL y ARN.

Luego de observar los distintos datos para la distribución de llaves ascendentemente (para distintos valores de n), se puede concluir que el peor caso para los ABB en costo de tiempo O(n) es demasiado alto sí se compara con los AVL y ARN que se comportan de manera similar con un costo O(log n). Esto se puede ver claramente en los tres indicadores, donde para los ABB, las alturas y los tiempos promedios son excesivamente altos si se comparan con los AVL y ARN que en este caso se comportan de manera similar. Resumiendo se puede deducir que la búsqueda y la inserción tiene complejidad O(n) para los ABB (en este caso) y O(log n) para los AVL y ARN y que se encuentran dentro de las cotas teóricas que se explican en el apunte.

IV.1. Altura Árboles para llaves escogidas al azar.

Para el caso al azar, como eran necesario insertar elementos al azar, se utilizó un pequeño programa sacado de internet, que generaba una matriz con llaves al azar, mediante un ciclo for (al ir recorriendo la matriz generada) estas se insertaban dentro de árbol para, y con método altura en cada clase, se calculaba la altura final de cada árbol. El importante mencionar que para insertar elementos al azar, para cada n, se consideró un intervalo de 0 a n*10 para cada escenario, por ejemplo si n igual a 20.000 el intervalo de elementos insertados al azar era entre 0 a 200.000, así sucesivamente. En la tabla 4, se ver los datos obtenidos y en el gráfico 7 la comparación de las alturas para las distintas clases.

N° de Llaves insertadas	Altura Árbol BB	Altura Árbol AVL	Altura Árbol RN
20.000	34	21	17
40.000	33	24	19
60.000	37	24	20
80.000	41	25	20
100.000	41	25	20
120.000	43	27	21
140.000	42	28	21
160.000	42	28	21
180.000	42	28	21
200.000	42	28	21

Tabla 3: Altura Árboles para datos insertados al azar.

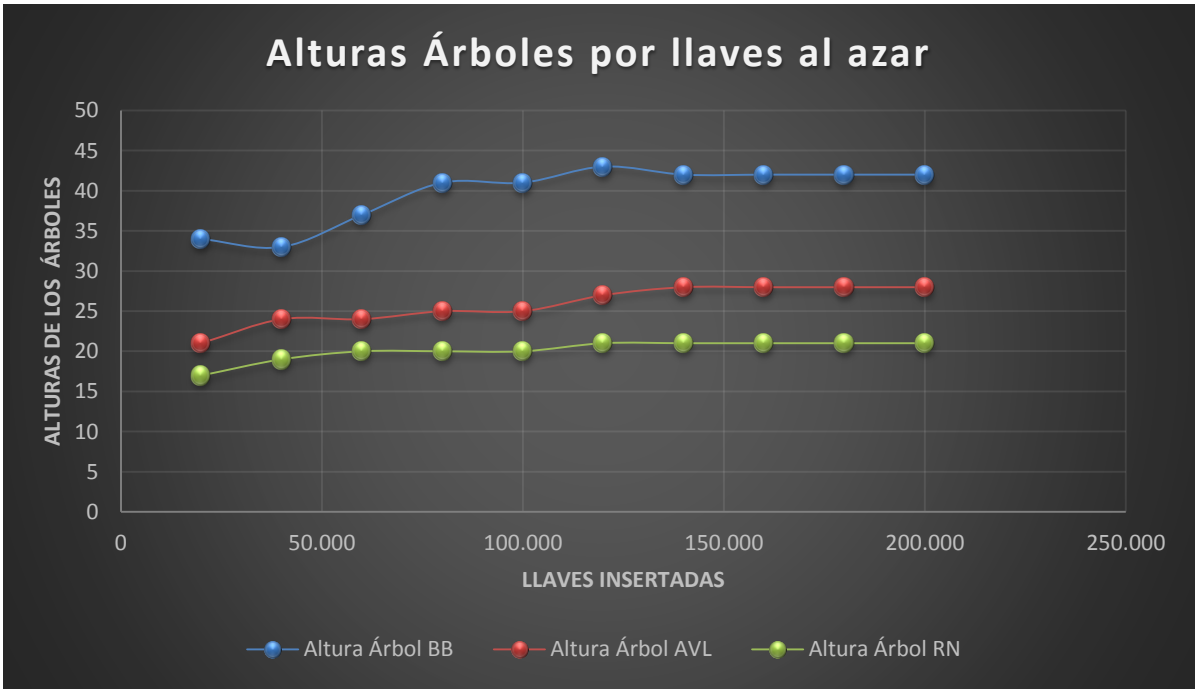


Gráfico 7: Alturas Arboles al azar.

En este caso, se puede ver los las alturas de los árboles ABB siguen siendo mayores pero una medida significativamente menor en comparación a su peor caso, el comportamiento de las gráficas es bastante similar entre sí, pero los árboles AVL y RN siguen tiendo menores alturas.

IV.2.2 Tiempo promedio de inserción con llaves escogidas al azar

Para este indicador es importante mencionar lo que se dijo en el indicador anterior sobre los intervalo donde podían existir elementos al azar, para cada n el intervalo era 0 entre 10*n y eran insertados a los tres tipos de árboles, en la tabla 4 se puede ver los datos obtenidos, y en el gráfico 8 sus comparaciones en las clases.

N° de Llaves insertadas	T.P. Árbol BB	T.P. Árbol AVL	T.P. Árbol RN
20.000	0,00095	0,00120	0,00060
40.000	0,00083	0,00097	0,00045
60.000	0,00082	0,00103	0,00047
80.000	0,00075	0,00097	0,00047
100.000	0,00082	0,00090	0,00050
120.000	0,00080	0,00090	0,00048
140.000	0,00075	0,00089	0,00050
160.000	0,00081	0,00085	0,00050
180.000	0,00086	0,00089	0,00052
200.000	0,00081	0,00089	0,00057

Tabla 4: Datos tiempo promedio de inserción

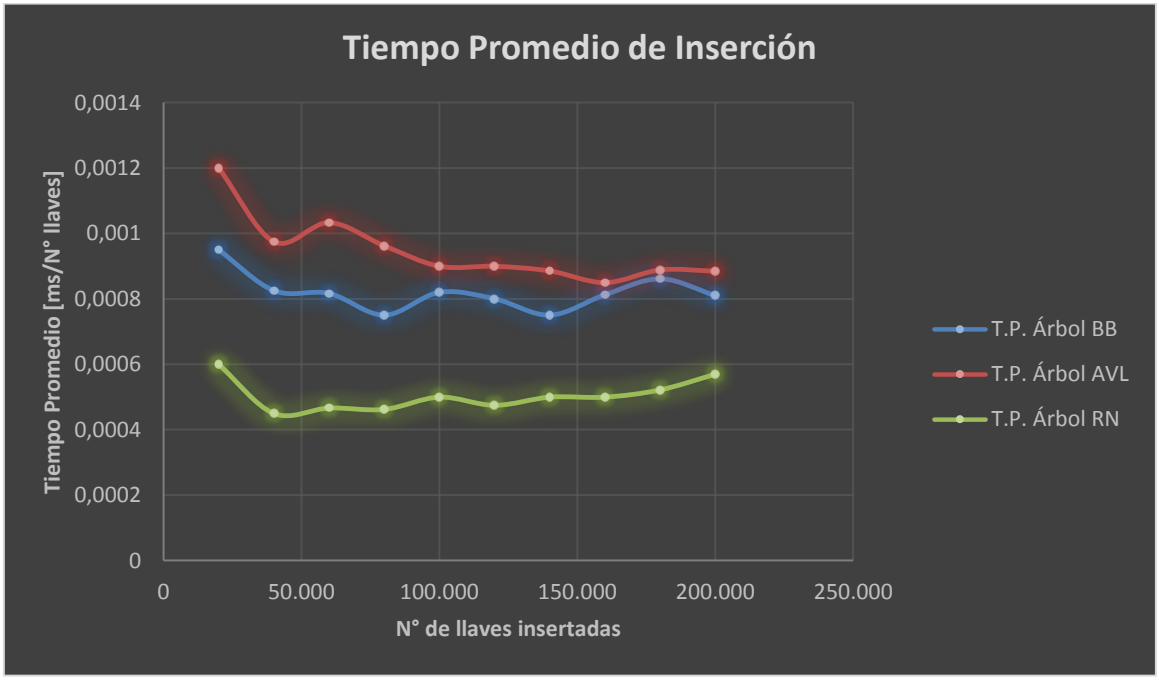


Gráfico 8: Tiempo promedio de inserción caso al azar.

Sí el comportamiento es similar para cada uno de los árboles, cada uno de ellos adquieren tiempos promedios distintos dejando al AVL con el mayor tiempo promedio, seguido del ABB y al final el ARN

IV.2.3 Tiempo promedio de búsqueda para llaves escogidas al azar

El método de búsqueda es similar al del caso anterior, salvo que ahora fueron insertados valores, para cada valor de n, los resultados puede verse en el gráfico 9.

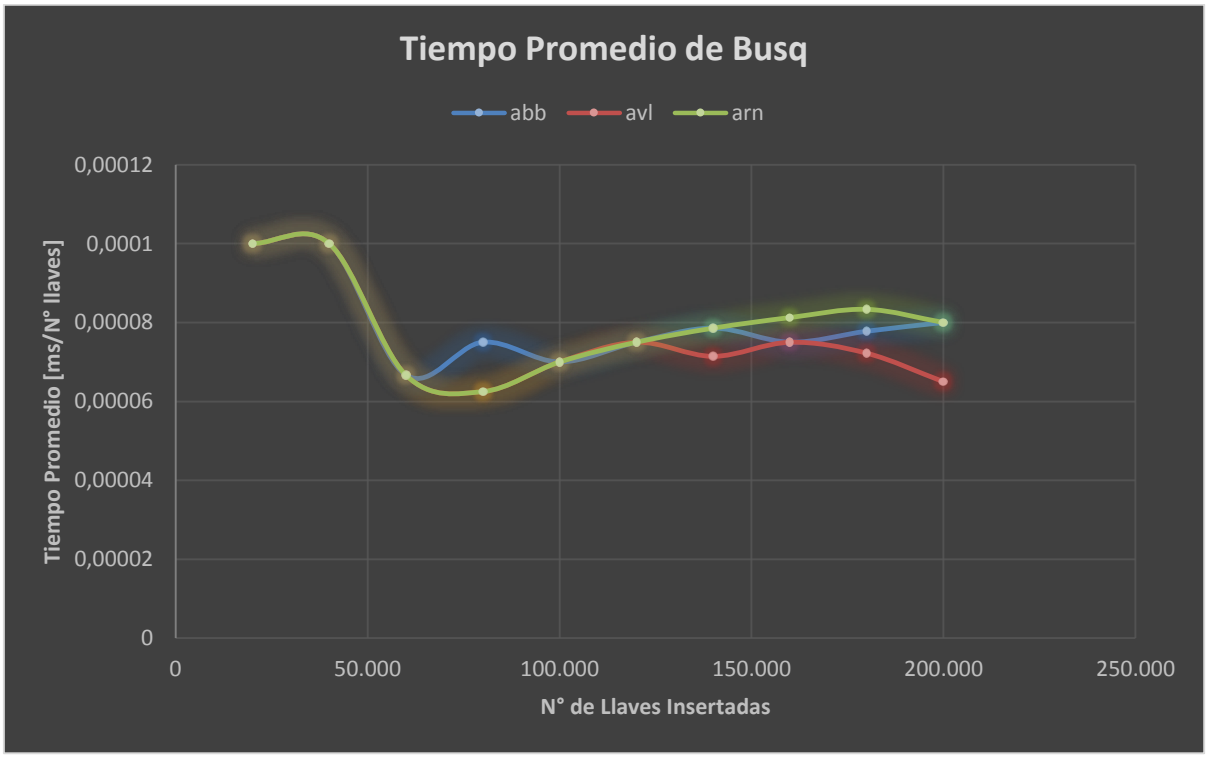


Gráfico 9: Tiempo promedio de búsqueda caso al azar.

Para la distribución al azar, n los distintos valores se n se vio que los tres árboles se comportan de manera similar, pero con pequeñas diferencias, como en el indicador de las alturas los ABB tiene la mayor altura luego de la inserción en comparación a los AVL y los ARN, en cambio el tiempo promedio de inserción es mayor para los AVL en comparación con los otros dos, sin embargo el tiempo promedio de búsqueda es similar para los tres tipos de árboles, lo que se condice que para este caso, las 3 implementaciones siguen la complejidad O(log n) tanto para inserción y búsqueda pero con pequeñas variaciones de tiempo.

****Algunos posible errores en las implementaciones puede que para el ARN se consideró un método iterativo en vez de recursivo, lo que pudo a ver hecho que algunos datos fallasen al medirlos, puede que al estar usando el computador con otras funciones los datos se hayan tomados erróneos. En cuanto a la toma de datos, se puso los datos que alumno obtuvo en cuanto a los supuestos que siguió, si bien puede a ver errores en ellos, se intentó ser riguroso cuidando los casos, el tipo de árbol, y los valores al azar que matriz tomaba en cada ejecución.*