Murillo García Sebastián 191838

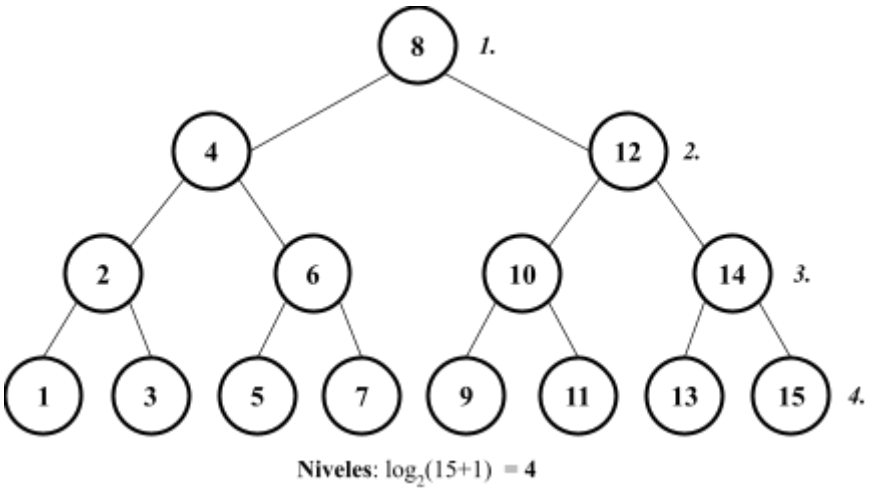
Altura esperada de un Árbol Binario de Búsqueda.

"¿Cuál es la altura esperada de un árbol binario de búsqueda?" A esta pregunta se pueden tener respuestas como: **log(n)**, tal vez **n**, o incluso un "quién sabe" si es que no se tiene un control estricto en la inserción de los datos. En este documento se expondrán los resultados obtenidos en tres casos a considerar: el mejor, el peor y el caso promedio con diferente cantidad de datos insertados en el árbol. De esta manera, se tendrá una idea más clara de la altura esperada de un árbol binario de búsqueda en relación con el tamaño de entrada de datos. Para analizar los resultados se tienen diferentes gráficas que muestran el comportamiento de la altura esperada en relación con el tamaño de entrada de datos insertados y algunas tablas para la comparación numérica.

El mejor de los casos.

En el mejor de los casos, al insertar los datos en un árbol binario de búsqueda, se va a tener a un árbol muy parecido a un AVL. Esto es debido a que la inserción de los datos en la estructura va a permitir que el árbol se mantenga lleno y relativamente balanceado, es decir, que para cada nodo más o menos queremos que tengan la misma cantidad de nodos en su lado izquierdo y su lado derecho (y si lo pensamos más a fondo esto mismo va a permitir que los factores de equilibrio de todo nodo en el árbol sea de -1, 0 o 1). De esta manera, se esperaría un árbol con una altura de log(n), pero ¿cuál es el criterio real? El criterio es el siguiente: si tomamos en cuenta que un árbol que está **lleno** tiene 'n' niveles con 'm' elementos, además de que el **nivel 1** es la raíz, ¿cuántos niveles tiene un árbol binario que está lleno? La altura de un árbol que está lleno es de $log_2(m + 1)$. Aquí tenemos un ejemplo de un árbol binario de búsqueda lleno y el número de elementos con este criterio:

Figura A: Árbol binario de búsqueda de cuatro niveles lleno.



Para intentar insertar una cantidad n de datos (de un arreglo dado) en un árbol binario de búsqueda de tal manera que al final de la inserción se tenga un árbol relativamente balanceado, se programó un método llamado "**mitades**" que ordena los elementos del arreglo a insertar para cumplir el objetivo. Este método tiene una lógica parecida al método de merge sort en el sentido de que trabaja con mitades de un arreglo, pero este método en específico inserta en una pila el dato que está justo en las mitades para que al momento de la inserción la estructura quede balanceada.

```
private void mitades(T[] arr, int min, int max, ArrayStack<T> pila){
        int mit;

        if(min >= max)
            return;
        mit = (min + max)/2;
        pila.push(arr[mit]);
        mitades(arr, min, mit, pila);
        mitades(arr, mit+1, max, pila);
}

public void mitades(T[] arr, T[] res){
        ArrayStack<T> pila = new ArrayStack<T>();
        ArrayStack<T> aux = new ArrayStack<T>();
        int i;

        mitades(arr, 0, arr.length-1, pila);

        while(!pila.isEmpty())
            aux.push(pila.pop());

        i = 0;
        while(!aux.isEmpty()){
            res[i] = aux.pop();
            i++;
        }
}
```

Si se da como parámetro un arreglo como el siguiente:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

El método devuelve un arreglo de la siguiente manera:

8	4	2	1	3	6	5	7	12	10	9	11	14	13	15

Y la inserción de los datos de este arreglo en un árbol binario de búsqueda tendría como resultado una estructura idéntica a la de la *Figura A*.

Los resultados para el mejor de los casos se muestran en las *Figuras* B y C.

Figura B: Altura de BST completos.

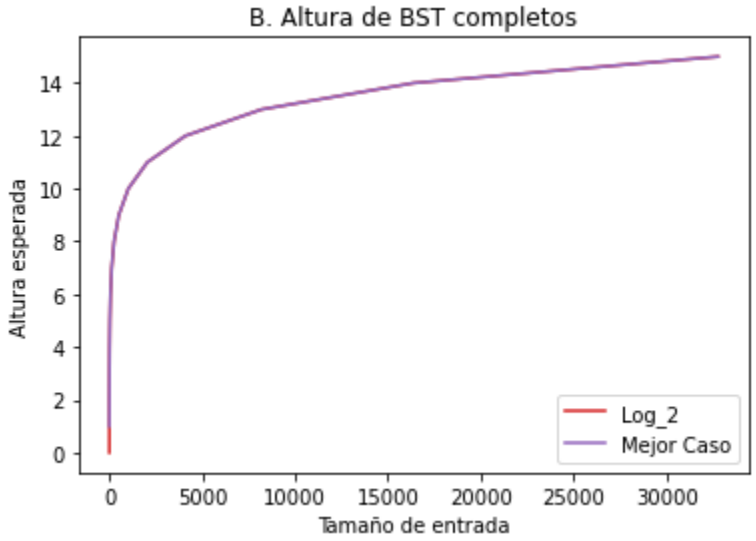


Tabla B : Valores numéricos de la Figura B.

Cantidad de elementos en el árbol (n)														
1	3	7	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767
Valores de $\log_2(n)$														
0.0	1.5	2.8	3.9	4.9	5.9	6.9	7.9	8.9	9.9	10.9	11.9	12.9	13.9	14.9
Altura esperada del árbol														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Podemos observar que dada una cantidad **n** de datos a insertar en un árbol binario de búsqueda, si se cumple que se tienen árboles completamente llenos, la altura esperada del árbol prácticamente es de **$\log_2(n)$** y este es uno de los **mejores casos**. También es por eso que las dos parecen estar sobrepuestas. Tiene mucho sentido que sea así, puesto que se cumpliría que para cada nodo se tendría la misma cantidad de nodos en su lado izquierdo y en su lado derecho; esto mantendría un árbol completamente lleno y balanceado [muy parecido a un árbol AVL].

Lo importante que hay que destacar de este diseño es que los árboles medidos fueron **árboles completamente llenos** y con el **arreglo de datos más adecuado**. Esto hace que estemos considerando el caso más afortunado de todos, y sería difícil encontrarnos con esta suerte si no tenemos extremadamente controlada la inserción de los datos en la estructura. Por lo tanto, aunque sea una belleza este resultado, no nos sirve de mucho, pues parece alejarse de la realidad.

Figura C : Altura de BST incompletos.

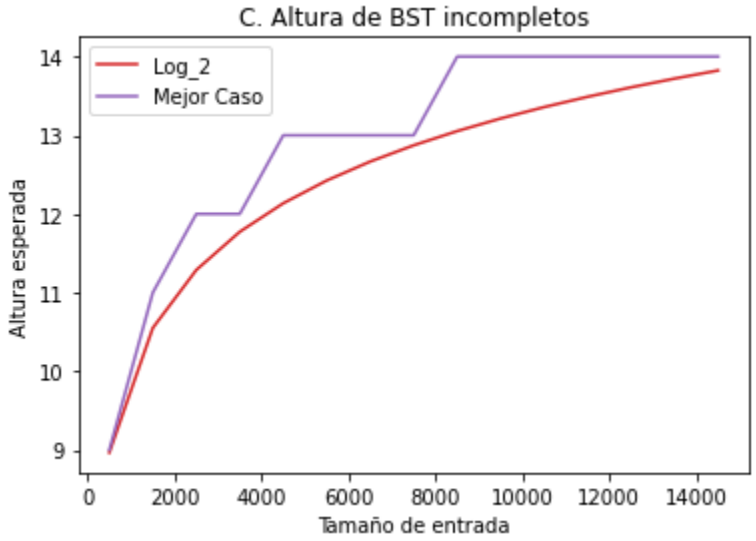


Tabla C : Valores numéricos de la Figura C.

Cantidad de elementos en el árbol (n)														
500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10.5 k	11.5 k	12.5 k	13.5 k	14.5 k
Valores de $\log_2(n)$														
8.9	10.5	11.2	11.7	12.1	12.4	12.6	12.8	13.0	13.2	13.3	13.4	13.6	13.7	13.8
Altura esperada del árbol														
9	11	12	12	13	13	13	13	14	14	14	14	14	14	14

Aquí podemos observar algo interesante: ya que estamos tratando con **árboles incompletos** [en su último nivel, los anteriores están completos] pero ordenados de tal manera que después de la inserción tenemos a un **árbol balanceado**, la altura esperada del árbol binario de búsqueda va a estar **muy cerca** y **casi tocando en algunos puntos** a los valores de **$\log_2(n)$** y este es otro de los **mejores casos** a considerar. ¿Cuándo la altura casi toca a $\log_2(n)$ y cuándo solamente está muy cerca? Esto depende completamente de la cantidad de datos. Pensemos un poco: si en la *Figura B* se utilizó una cantidad de datos tal que formaba árboles completos y la altura siempre fue de **$\log_2(n)$**, entonces podemos decir que la altura casi toca a los valores de $\log_2(n)$ cuando más lleno está el árbol [estos puntos los señalamos con color púrpura]. Por otro lado, podemos observar que la gráfica parece una escalera, ¿por qué es así? Se ve de esa manera debido a que llega un punto en el cual se suma un nivel más al árbol por la inserción de un solo elemento [esos puntos los señalamos con color naranja], mientras que el valor de $\log_2(n)$ puede tomar valores decimales. Los niveles son estrictos en el sentido de que, por ejemplo, un BST relativamente balanceado de **nivel 4** puede estar compuesto de entre 8 a 15 elementos [se puede

verificar en la *Figura A*]. Por lo tanto, mientras que el árbol esté incompleto en su último nivel, los valores de la altura del árbol van a estar **muy cerca de $\log_2(n)$** y cada vez se van a ir acercando a ese valor mientras más elementos se inserten en la estructura [estos puntos están de color negro en la tabla]. Es interesante ver cómo en este caso la gráfica obtenida es la parte entera (o como dijimos en clase: la parte "truncada" del valor) de la curva de $\log_2(n)$. Si se tuviera la capacidad de analizar la altura con un incremento de una unidad en la entrada de datos hasta 32,768 (este es el valor de 2^{15}), la gráfica resultante se vería perfectamente como una escalera que quiere parecer $\log_2(n)$, los puntos exactos donde las dos gráficas se tocan (cuando la entrada es de 2^n estrictamente) y en qué puntos se aumenta un nivel más a la estructura ($2^n + 1$).

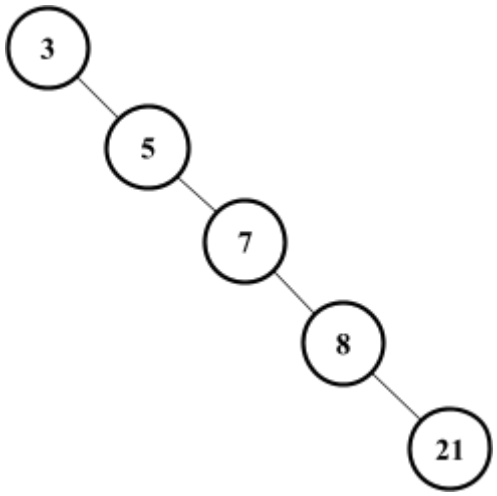
Este caso es un poco más apegado a una realidad que todavía parece un poco afortunada, pues puede ser difícil encontrarse con datos que estén ordenados de tal manera que se cumpla con que el árbol está balanceado excepto por su último nivel. Incluso aunque se pudiera implementar el código de **mitades** para lograrlo, es más eficiente simplemente utilizar un árbol AVL como la estructura indicada.

El peor de los casos.

En el peor de los casos, un árbol binario de búsqueda es **tan ineficiente** como una lista. ¿Cuándo tenemos este caso como resultado? En dos situaciones: cuando los datos se insertan en un orden estrictamente de menor a mayor, o de mayor a menor. En otras palabras, cuando los datos se insertan en la estructura de un arreglo que está completamente ordenado o que esté inversamente ordenado. Esto lo podemos ver en un ejemplo:

Ejemplo: Inserción de un arreglo ordenado de 5 elementos en un BST:

3	5	7	8	21



Después de ver este ejemplo podemos inferir rápidamente la altura de un árbol binario de búsqueda cuando nos encontramos en cualquiera de las dos situaciones antes mencionadas. Dada una cantidad **n** de datos ordenados o inversamente ordenados insertados en un árbol binario de búsqueda, ¿cuál es la altura esperada del árbol? La respuesta es simple: **n**. Esto lo podemos ver en la siguiente figura:

Figura D : Altura de BST en su peor caso.

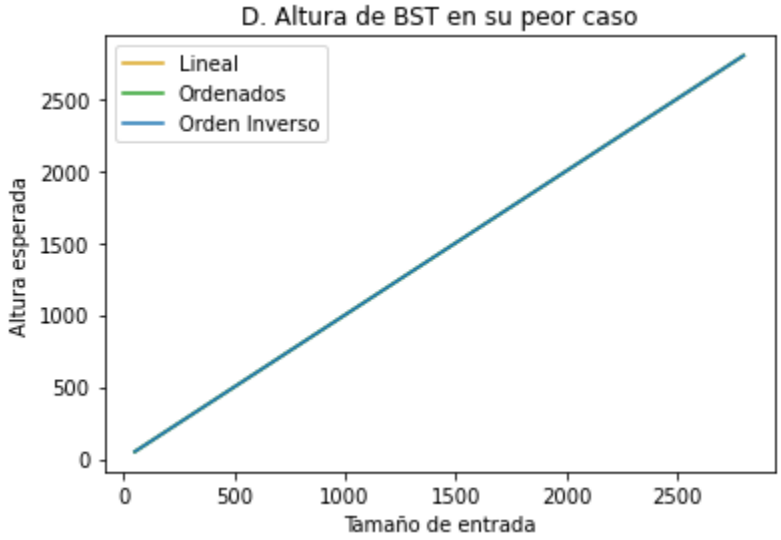


Tabla D : Valores numéricos de la Figura D.

Cantidad de elementos en el árbol (n)														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800
Valores de la función lineal														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800
Altura esperada del árbol con datos ordenados														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800
Altura esperada del árbol con datos inversamente ordenados														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800

Como podemos observar, dada una cantidad **n** de datos a insertar en un árbol binario de búsqueda en cualquiera de sus dos peores casos, la altura del árbol va a ser **siempre** de **n**. Esto se debe a que una vez que los datos están ordenados (ya sea de mayor a menor o viceversa), cada vez que se inserte un elemento más en el árbol, este permanecerá **siempre** como el **hijo derecho** de la única hoja del árbol si los datos están ordenados de manera normal. De otra forma, si los datos están ordenados de manera inversa, el nodo insertado será **siempre** el **hijo izquierdo** de la hoja de la estructura. De esta manera,

cada vez que se inserta un dato se va a "crear" un nuevo nivel, y la estructura parece más una lista (como en el *Ejemplo 1*). Esto es completamente ineficiente y no tiene sentido implementar un árbol binario de búsqueda en este caso, pues se pueden utilizar arreglos o listas que tendrían exactamente la misma cantidad de elementos y donde el recorrido sería el mismo.

Figura E : Altura de BST en uno de sus peores casos vs. en su mejor caso.

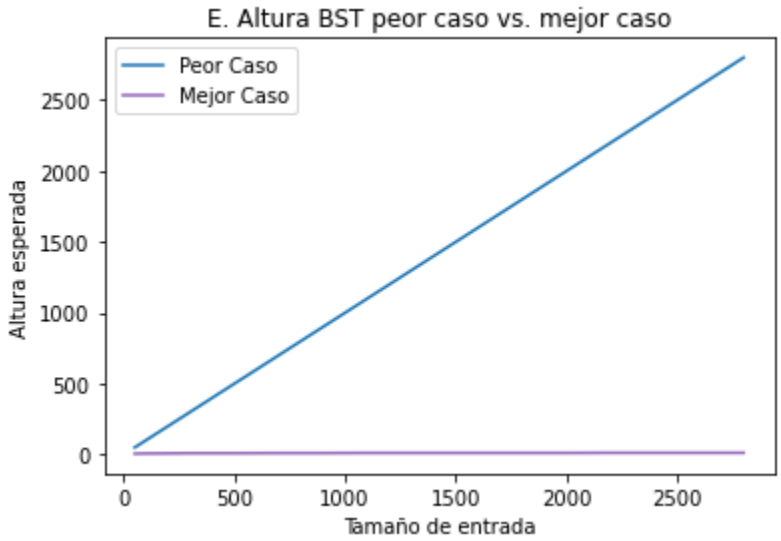


Tabla E : Valores numéricos de la Figura E.

Cantidad de elementos en el árbol (n)														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800
Altura esperada del árbol en el peor de los casos														
50	100	300	500	700	900	1100	1300	1500	1700	1900	2000	2200	2500	2800
Altura esperada del árbol en el mejor de los casos														
6	7	9	9	10	10	11	11	11	11	11	11	12	12	12

Como podemos observar, la diferencia entre la altura esperada del árbol cuando está en su mejor de los casos en comparación con el peor de sus casos es **abismal**. Es muy interesante ver cómo es que se desperdician muchas de las desventajas en el recorrido de la estructura cuando analizamos este versus de los dos casos.

El caso promedio.

Para el caso promedio, ¿cuál es la altura esperada? Aquí la respuesta más rápida después de pensar y analizar la situación en la que nos encontramos podría ser un "no tengo idea". El planteamiento está complicado, pues ¿qué tal si los datos de pura suerte están acomodados de tal manera como para que el árbol resulte que esté relativamente balanceado y la altura parece acercarse a $\log_2(n)$?, ¿o qué tal que los datos están más ordenados y la altura parece acercarse a n? ¿Cómo lo podemos determinar? Para ello, se implementó un generador de arreglos con datos completamente aleatorios y se insertaron 20 diferentes arreglos aleatorios en 20 diferentes árboles por cada tamaño de entrada para sacar un promedio de la altura. De esta manera se intentó profundizar en el análisis del caso promedio de la altura esperada de un árbol binario de búsqueda. Los resultados fueron los siguientes:

Figura F : Altura de BST en su caso promedio.

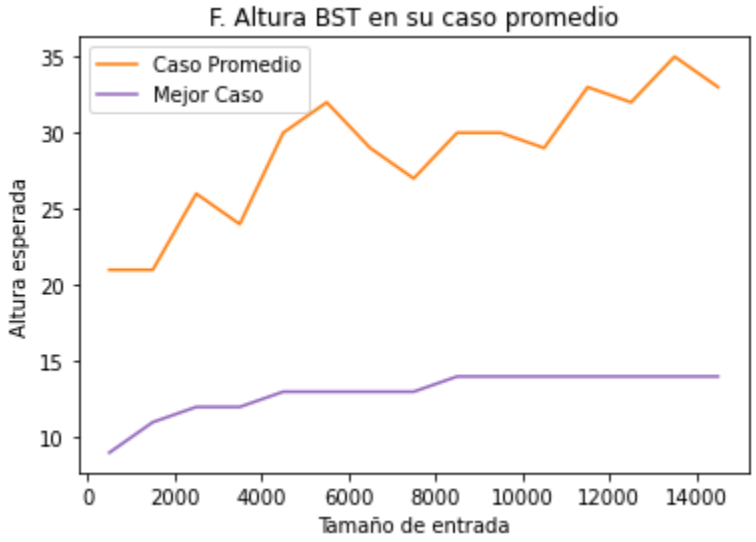


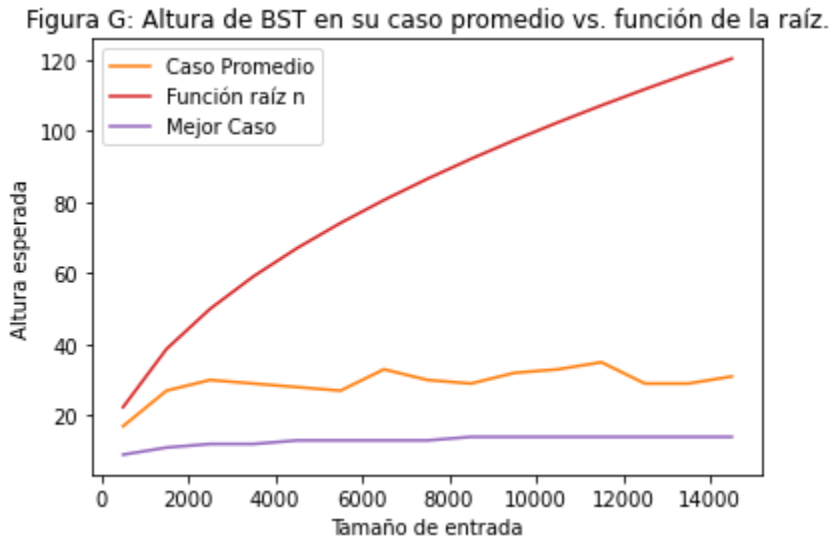
Tabla F : Valores numéricos de la Figura F.

Cantidad de elementos en el árbol (n)														
500	1500	2500	3500	4500	5500	6500	7500	8500	9500	10.5 k	11.5 k	12.5 k	13.5 k	14.5 k
Altura esperada del árbol en el caso promedio														
21	21	26	24	30	32	29	27	30	30	29	33	32	35	33
Altura esperada del árbol en el mejor de los casos														
9	11	12	12	13	13	13	13	14	14	14	14	14	14	14

Como podemos observar, los valores son fluctuantes y si bien vemos que la altura tiende a crecer conforme aumenta el tamaño de entrada, no se pudo determinar completamente la altura esperada en el caso promedio. El resultado depende completamente del acomodo de los datos a insertar en el árbol, pero como estos son completamente aleatorios, entonces la altura está condicionada por la aleatoriedad de los datos. Entonces ¿no podemos determinar nada?

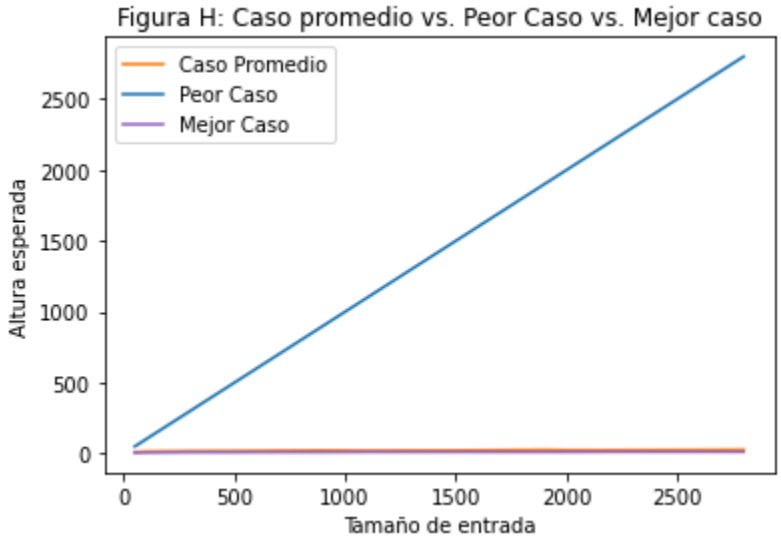
En un principio se pensó que tal vez la altura podía estar determinada por un valor cercano a la **raíz de n**, pero al graficar esta función se pudo observar que mientras más grande era la cantidad de datos a insertar, se hacía más inmensa la brecha entre la función de la raíz y la del caso promedio. Inmediatamente llegué a la conclusión de que este no era el resultado. Esto lo podemos ver en la siguiente figura:

Figura G : Altura de BST en su caso promedio vs. función de la raíz.



Pero después de analizar esta gráfica, podemos observar algo interesante: parece que la altura del árbol binario de búsqueda en su caso promedio tiene un **comportamiento logarítmico fluctuante**. Que me perdonen los grandes matemáticos y programadores por estar diciendo estas aseveraciones, pero parece que esto es así. Si comparamos la altura esperada de un árbol binario de búsqueda del **caso promedio** con el **mejor de los casos** y con el **peor de los casos**, podemos ver los valores de la altura esperada del caso promedio se acercan muchísimo más (abismalmente más) al mejor de los casos que al peor de los casos. Esto lo podemos ver en la siguiente figura:

Figuras H : Altura de BST en su caso promedio vs. mejor caso vs. peor caso.

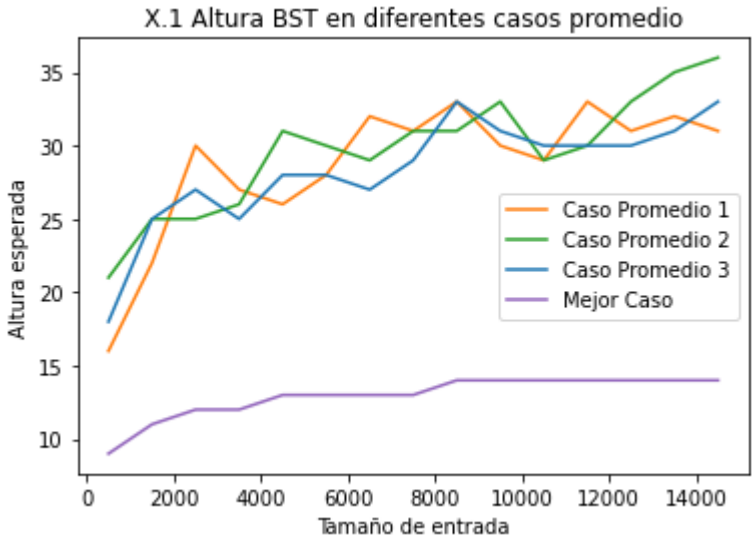


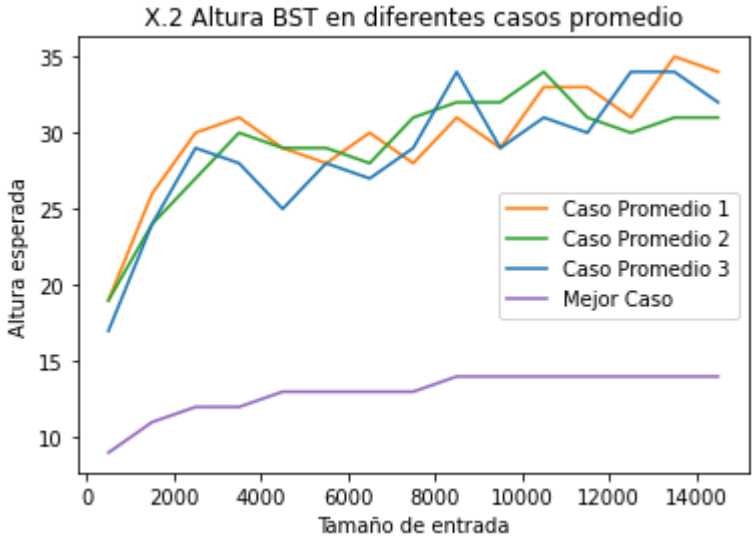
¿Seguimos sin respuesta? En efecto. A pesar de todas las pruebas e intentos para determinar un número o algo parecido para obtener la altura esperada de un árbol binario de búsqueda en su **caso promedio no se pudo obtener un resultado concreto**. Lo que sí podemos decir es que la altura esperada en el caso promedio no parece alejarse demasiado del mejor de los casos y este es un resultado importante. Parece ser que la altura se mantiene dentro de unos valores fluctuantes **cercanos entre sí** y **cercanos con el mejor de los casos**. Si bien el caso promedio no se toca con el mejor de los casos, es de reconocer que la altura no llega a límites tan extremos como lo es el peor de los casos. Debido a esto mismo podría decirse que en el caso promedio (con datos completamente aleatorios) la estructura de árbol binario de búsqueda todavía es conveniente.

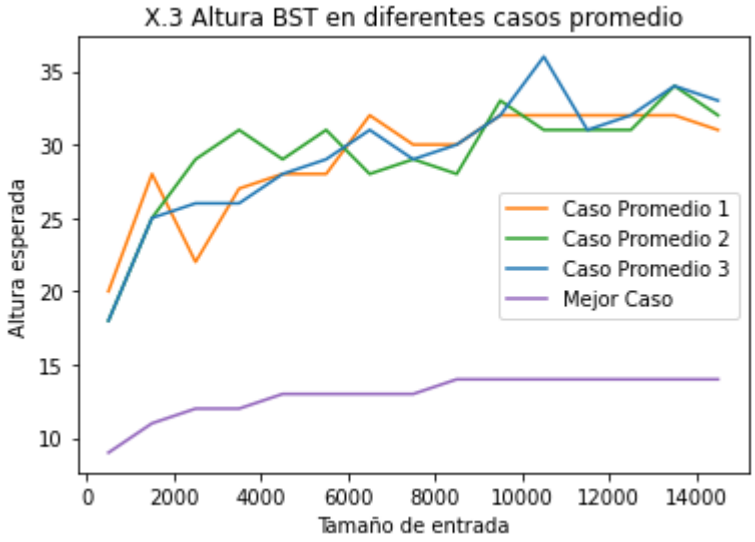
Este comportamiento del caso promedio puede deberse a que entre más elementos tenga un árbol, más oportunidades existen de que los datos se distribuyan de manera más uniforme y conveniente en los niveles del árbol. ¿Por qué decimos esto? Pensemos en la cantidad de elementos que pueden estar en el último nivel de un árbol que tiene una altura de 13, por ejemplo. Sabemos que la cantidad de elementos que tiene un árbol completo de 13 niveles es de 8191 por la fórmula: $\mathbf{2^n - 1}$ (con n como la cantidad de niveles). Por lo que al tener 8192 elementos se crea un nivel más y ahora el árbol tendría una altura de 14. Ahora bien, ¿cuántos elementos tiene un árbol completo de 14 niveles? La respuesta es 16383 elementos. Entonces, por una simple resta sabemos que en el nivel 14 hay 8192 elementos que se pueden distribuir por las hojas del nivel 13. Esto puede afectar muchísimo porque los elementos insertados en la estructura pueden encontrar un lugar para asentarse entre una cantidad muy grande del mismo nivel (y eso pensando que los otros niveles están completos, pero si no estuvieran, entonces tal vez tienen más posibilidades). Claro, esta es una suposición que espero que esté correcta, pero es muy interesante ponerse a pensar en este **caso promedio**.

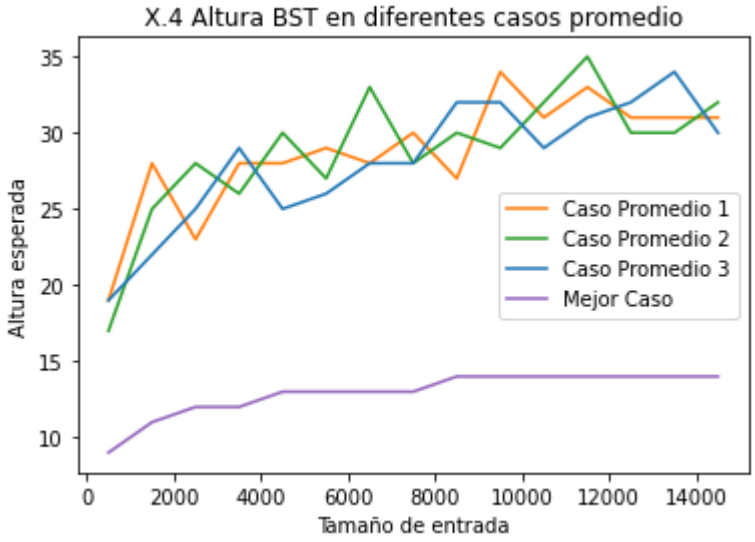
Por último, mostraremos 4 diferentes gráficas con distintas alturas del caso promedio para hacer notar ese parecido en el comportamiento y su poca lejanía tanto entre las mismas pruebas del caso promedio como entre cada una de ellas con el mejor de los casos. Nótese también que en las cuatro gráficas la escala tuvo como máximo 35 niveles de altura. Muy interesante.

Figuras X : Distintas alturas en el caso promedio.






X.3 Altura BST en diferentes casos promedio
Altura esperada
Tamaño de entrada
Caso Promedio 1
Caso Promedio 2
Caso Promedio 3
Mejor Caso
10
15
20
25
30
35
0
2000
4000
6000
8000
10000
12000
14000


X.4 Altura BST en diferentes casos promedio
Altura esperada
Tamaño de entrada
Caso Promedio 1
Caso Promedio 2
Caso Promedio 3
Mejor Caso
10
15
20
25
30
35
0
2000
4000
6000
8000
10000
12000
14000