如何用 python 寫 binary search tree

Binary search tree 的特點

1.若任意節點的左子樹不空，則左子樹上所有節點的值均小於它的根節點的值
2.若任意節點的右子樹不空，則右子樹上所有節點的值均大於它的根節點的值
3.任意節點的左、右子樹也分別為二元搜尋樹
4.沒有鍵值相等的節點

為什麼複雜度是 O(log(n))?

有一個排序陣列如下:

[1,3,5,8,12,13,15,16,18,20,22,30,40,50,55,67]

如果今天要找的是元素 13，以 Binary Search 搜尋先從陣列中間的數 16 開始比對（18 也可以，不過範例會以最糟的情況來看），因為 13 < 16，所以折半搜尋較小的那一半，折半後如下:

[1,3,5,8,12,13,15,16]

接著再和中間的數 8 比，因為 13 < 8，所以折半搜尋較大的那一半，折半後如下:

[12,13]

接著和中間的數 12 比，因為 13 > 12，所以折半搜尋較大的那一半，折半後最後只剩 13

[13]

以上一共比了 4 次才找到要找的 13，折半(1/2)的次數是 4 次，也就是
(** == 次方)
1/2 * 1/2 * 1/2 * 1/2 = (1/2)**4

所以從 16 個元素的陣列中找到某一元素可寫成

16 * (1/2)**4 = 1

假設今天陣列有 n 個元素，則以上可改寫為

n * (1/2)**k = 1

上面的算式等於

n * (1**k/2**k) = 1，也就是 n * 1/(2**k) = 1

左右兩邊同乘上 2^k，則為

2**k * n/(2**k) = 2k

n = 2k

所以以 2 為底 n 的對數是 log2(n) = k，所以 k 就是 log2(n)，也就是在最糟情況下，要從 n 個元素中搜尋到某一個元素，則要折半比對的次數為 k 次，也就是 log2(n)次。也就是在最壞的狀況下，演算法的執行時間不會超過 O(log(n))。

程式分析

在程式碼上面的備註

參考資料
https://lovedrinkcafe.com/python-binary-search-tree-part-1/

https://matthung0807.blogspot.com/2018/04/binary-search-ologn.html