Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

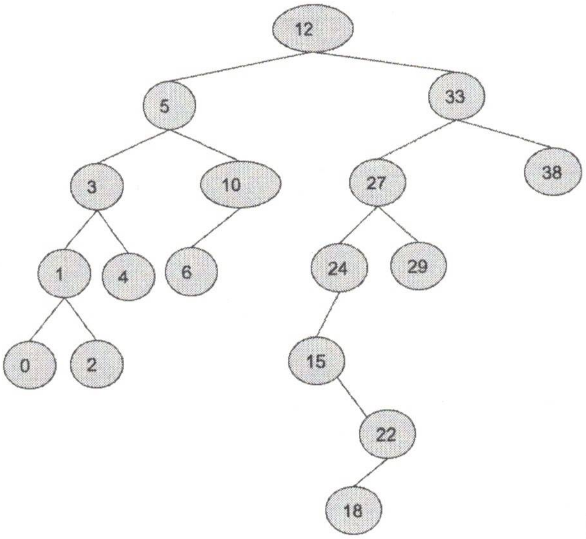
Assignment 5 (3% - 12 points)

CSI2110/CSI2510 (Fall 2021)

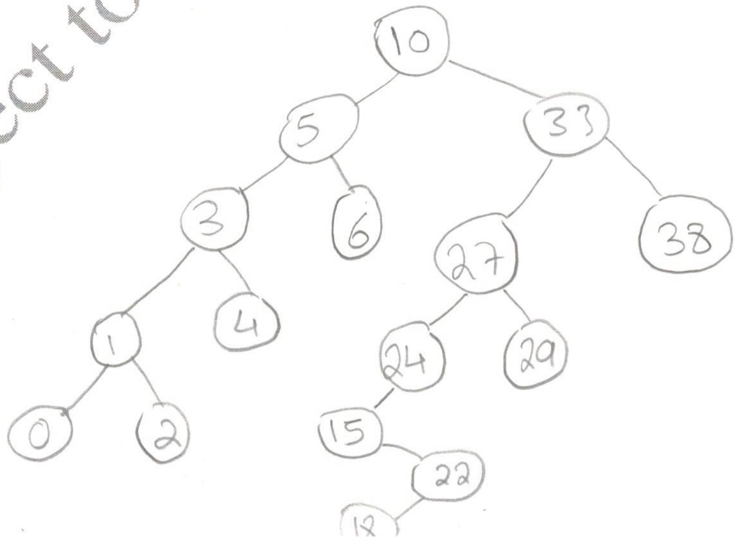
Due: Thursday Oct 21, 11:59PM

Late assignment policy : ***1min-24hs late are accepted with 30%off; no assignments accepted after 24hs late.***

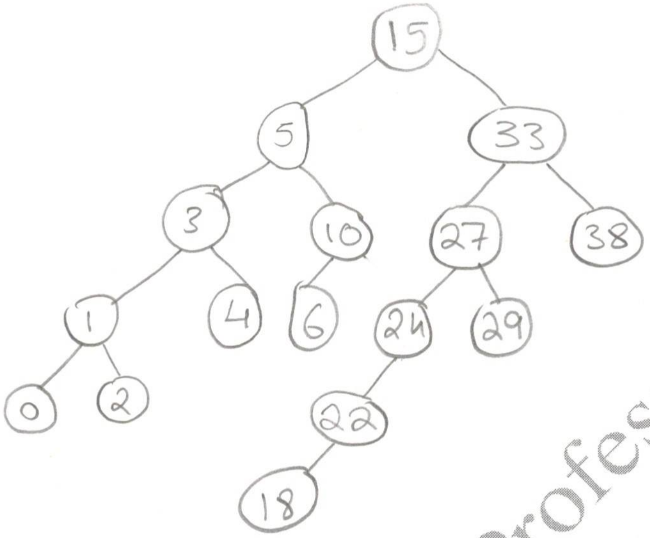
Question 1. [6 points] Below is a binary search tree.



a. [2 points] Delete 12 using an **in-order predecessor node** to fill the "hole", and show the resulting tree.



b. [2 points] Delete 12 using an **in-order successor node** to fill the "hole" in the original tree and show the resulting tree.



c. [2 points] Print the outputs of inorder traversal of (a) and (b) and explain any conclusion obtained as a result of comparing them (if the outputs are different, why?; if the outputs are the same, why?)

$\boxed{a} \Rightarrow$ (0,1,2,3,4,5,6,10,15,18,22,24,27,29,33,38)

$\boxed{b} \Rightarrow$ (0,1,2,3,4,5,6,10,15,18,22,24,27,29,33,38)

Both outputs are the same because both predeccessor and successor nodes are valid to substitute for the root, as long as the children are taken care of and put in the right place. ((right order))

Question 2. [6 points] We want to find the 3rd largest element in a binary search tree. We know the inorder traversal visits nodes in non-decreasing order. therefore we can find the 3rd largest node in the resulting sequence, but it would take *O(n)*.

a. [3 points] Write your algorithm description or give a pseudo-code to produce the $\overset{k}{3^{rd}}$ largest element in a given binary search tree, which takes *O(h)*, where the height of the tree is *h*.

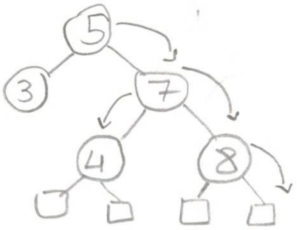
Recursively reverse traverse to find the largest element in the bst which would take a time complexity of O(h) & then traverse K element from the rightmost node to get K (3rd) largest element

So All together is O(h+3), ignoring constant -> O(h)

b. [2 points] Show why your algorithm is *O(h)*

Because with the reverse traversal I'm guaranteed that I won't be traversing all over the BST, hence not O(n), and by visiting the rightmost node and only going K steps backwards to get the K largest element

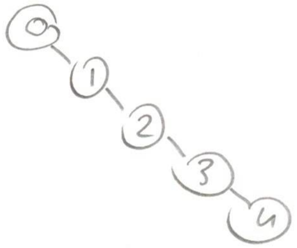
no. of nodes



c. [1 point] Give a general example of a Binary Search Tree with *n* nodes where your algorithm is not ~~*O(log n)*~~. O(h).

- An example would be a BST of 10 nodes and looking for the 10th largest element. So it would be O(h+n)

- Another example is when the BST is not balanced



in this case it it would be O(n+K)